2025

Web Almanac

HTTP Archive's annual
state of the web report

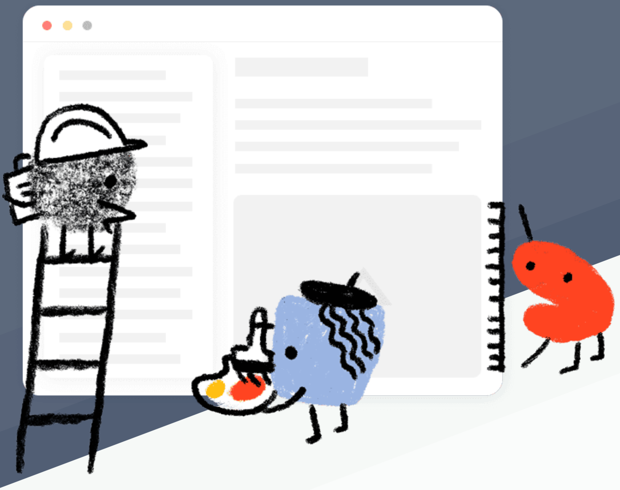


Table of Contents

Introduction

Foreword ii

Part I. Page Content

Chapter 1: Fonts 1
Chapter 2: WebAssembly 47
Chapter 3: Third Parties 61
Chapter 4: Generative AI 77

Part II. User Experience

Chapter 5: SEO 103
Chapter 6: Accessibility 165
Chapter 7: Performance 217
Chapter 8: Privacy 253
Chapter 9: Security 277
Chapter 10: PWA 337

Part III. Content Publishing

Chapter 11: CMS 355
Chapter 12: Ecommerce 389

Part IV. Content Distribution

Chapter 13: Page Weight 407
Chapter 14: CDN 477
Chapter 15: Cookies 509

Appendices

Methodology 531
Contributors 541

Foreword

The web is today an integral part of our modern digital infrastructure. Web applications can now be found in almost any technological environment that touches our daily life, from mobile apps to endpoints used in satellite communications. In the context of the Web Almanac, we want to understand the trends and evolution of the web. We, as web enthusiasts, are happy to present the 6th edition of one of the major technical reports on the web—the 2025 Web Almanac.

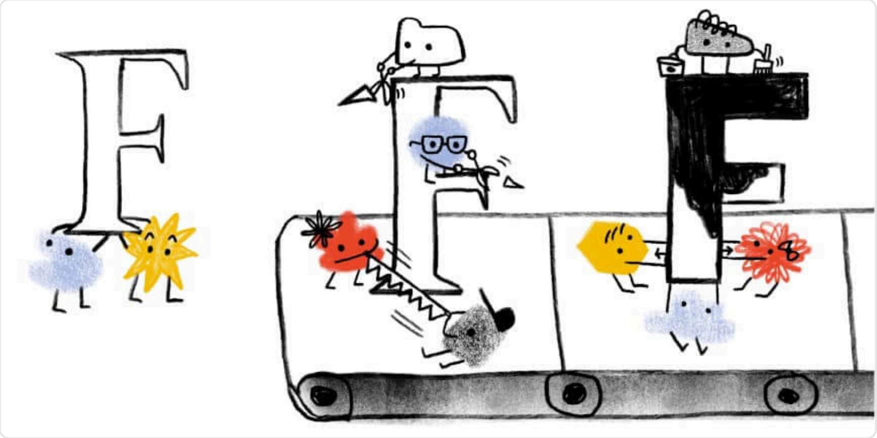
The Web Almanac is an open-source project, contributed to by experts (researchers, developers, consultants, and editors) from all over the globe. It has been another wonderful edition experiencing the exciting contributions of this community. It is the engagement of this community that has made this report possible.

This community-driven project analyzes 17.2 million websites and processes 244 TB of open-source data provided by the HTTP Archive. In this report, we analyze various aspects (such as accessibility, performance, and security), understand how the web evolves, and gain insights into structural, technical, and societal developments.

We hope our report helps the community understand the current state of the web better, and we encourage you to explore this extensive work.

—Nurullah Demir, 2025 Web Almanac Editor-in-Chief

Part I Chapter 1
Fonts



Written by Charles Berret
Reviewed by Bram Stein and José Solé
Analyzed by Charles Berret and Ivan Ukhov
Edited by Chris Lilley

Introduction

When the HTTP Archive first began gathering data on web typography in 2011, the use of custom web fonts was in the low single digits. Only about 3-5% of websites at the time delivered self-hosted fonts via `@font-face` or used services like Typekit, Fonts.com, or Google Web Fonts. Every other site in 2011 was limited to the small handful of “web-safe” system fonts (Arial, Courier, Times, etc.) that were available on every user’s device.

Web fonts quickly rose to become the norm when designers realized that custom typography could distinguish their visual identity from other, more generic sites. By 2015, web fonts were used by over half of websites. They reached around 75% adoption by 2020.

Today, the question is less about whether a website uses web fonts, but rather which specific typefaces it displays and whether they use font features to display their full expressive potential.

At the same time, font delivery methods on the web have been shifting. An increasing number of sites are choosing to self-host their font files (serving fonts from their own servers) rather than relying exclusively on third-party CDNs. Many other sites use a mix of both approaches. Last year's data showed a clear rise in exclusive self-hosting alongside a decline in sites that combined self-hosting with external services, and we investigate if this trend continues in 2025.

We also look at font providers (is Google Fonts dominance continuing) as well as how fonts are being loaded onto sites, as well as which fonts are used in, how non-Latin languages are supported, as well as some newer, more advanced topics.

In short, web fonts are nearly universal on the web, but more sites are taking font delivery into their own hands instead of depending on an external provider. We explore these patterns and describe the current state of web typography in detail below.

Webfont usage

Web font usage continues to grow, although its growth has naturally slowed as the spread of web fonts reaches saturation on the web (for all but the largest CJK character sets).

88%

Figure 1.1. Percent of mobile pages using web fonts.

In 2025, web fonts were found on roughly 88% of websites, up slightly from about 87% in 2024. (The metric here is the share of pages that request web font files, which effectively represents the percentage of websites using web fonts.)

This widespread usage highlights how far web typography has come since the days when developers were limited to core system fonts, though a notable minority (around 12% of sites) still stick to those older defaults.

Hosting and services

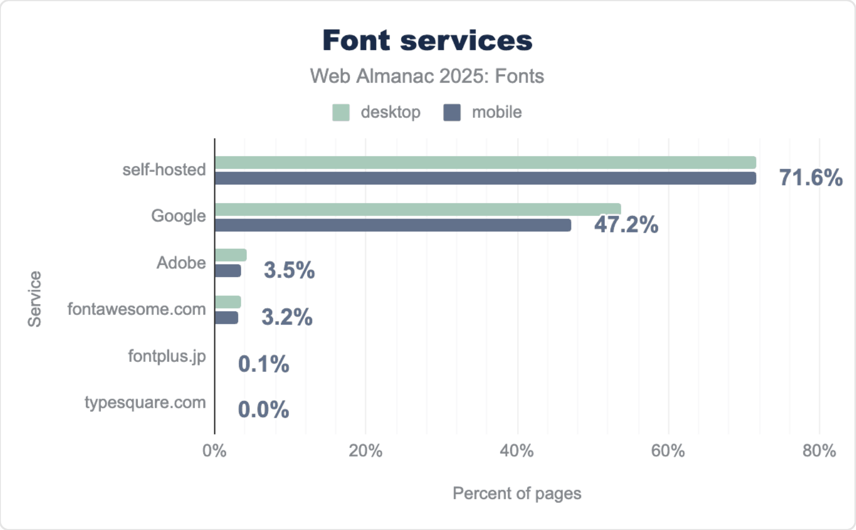


Figure 1.2. Font services.

This year's Web Almanac crawl confirms that self-hosting is the most widespread means of delivering web fonts. About 72% of websites (both desktop and mobile) serve at least one font file from their own origin, either by self-hosting exclusively or by using a mix of self-hosted and externally hosted fonts. That figure is just over one percentage point higher than last year, and about 4-5 points higher than in 2022, indicating a steady increase. In other words, the rise in self-hosting that became apparent in 2022 is still continuing.

Within that 72%, we can break down two significant groups:

- **Exclusively self-hosted fonts**: About 31.5% of desktop sites and 36.4% of mobile sites use only self-hosted fonts (no third-party font services at all). Averaging those, roughly one-third of all sites rely solely on self-hosting in 2025 (up from about 30% last year). This marks a real shift: more sites have decided to host all their fonts themselves, removing external providers from the critical path.

- **Mixed hosting (self-hosting + a service)**: Another 36.1% of desktop sites and 31.7% of mobile sites use a combination of self-hosted fonts and a font service on the same page. In most of these cases the service is Google Fonts, with perhaps an icon CDN

> or another source added in. This “hybrid” approach was slightly more common last year (around 38.5% of desktop and 32.7% of mobile pages in 2024), so it has dipped a bit as some sites fully switch over to self-hosting.

Combining those groups, about 72% of sites self-host in some capacity. The remaining ~28% of sites rely exclusively on third-party services for fonts (with Google Fonts being by far the most likely choice in that category).

Web font services

Even with the growth of self-hosting, hosted font CDNs still appear on roughly half of all websites, primarily because of Google’s massive footprint. This year’s crawl found Google Fonts on about 54% of desktop sites and 47% of mobile sites. (This counts any site that uses Google’s API to fetch fonts, whether Google is the only source or used alongside others.) Google’s reach on desktop dropped about 3 percentage points from 57% in 2024, and is down about 6 points from 2022, reflecting the shift toward self-hosting. On mobile, Google’s share dipped only slightly (about 0.3 points since last year, and about 4 points since 2022). The overall pattern is the same as noted in 2024: Google Fonts is still the default choice for many, but its dominance is slowly diminishing as more sites self-host.

Other font hosting services are much smaller by comparison. Adobe Fonts (formerly Typekit) grew modestly again. In 2025 around 4.2% of desktop sites and 3.5% of mobile sites use Adobe Fonts (roughly 3.8% combined), up from about 4.1% last year. This is a small slice of the web, but it represents real growth (a 10-11% relative increase over two years).

Font Awesome, a popular icon-font CDN, has seen a slow decline. In 2022 it was used by about 4.4% of sites, in 2024 by 4.0%, and in 2025 it’s down to roughly 3-4% of sites. This slide is likely caused by many sites moving to alternative icon solutions (inline SVGs, icon sprite sheets, or self-hosted icon sets) while other developers avoid loading a large icon font for performance reasons. Even so, Font Awesome’s continued use on about 1 in 25 websites is significant—few other single font families (icon or text) come close to that reach.

Every other service is essentially negligible at the web scale. Legacy services like Monotype’s Fonts.com, Extensis, Cloud.Typography, and Type Network each account for well under 1% of sites. Many of these have declined to the point of near-invisibility in our crawl data. In practical terms, if a website is using a hosted font service in 2025, it’s almost certainly Google Fonts. There’s a small chance it’s Adobe Fonts or Font Awesome, and very little chance that it’s something else.

Font hosting combinations

Because most sites don't rely on just one source for web typography, it's useful to look at how they mix and match delivery. In practice the web has converged on a few predictable combinations—self-hosting only, Google only, or a hybrid of the two.

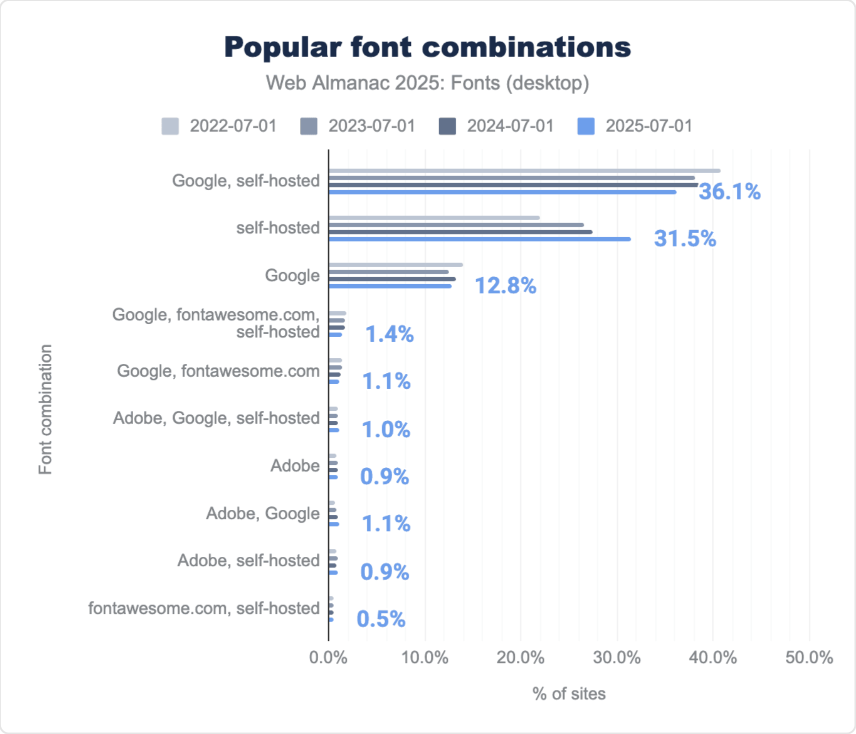


Figure 1.3. Popular font hosting combinations on desktop sites.

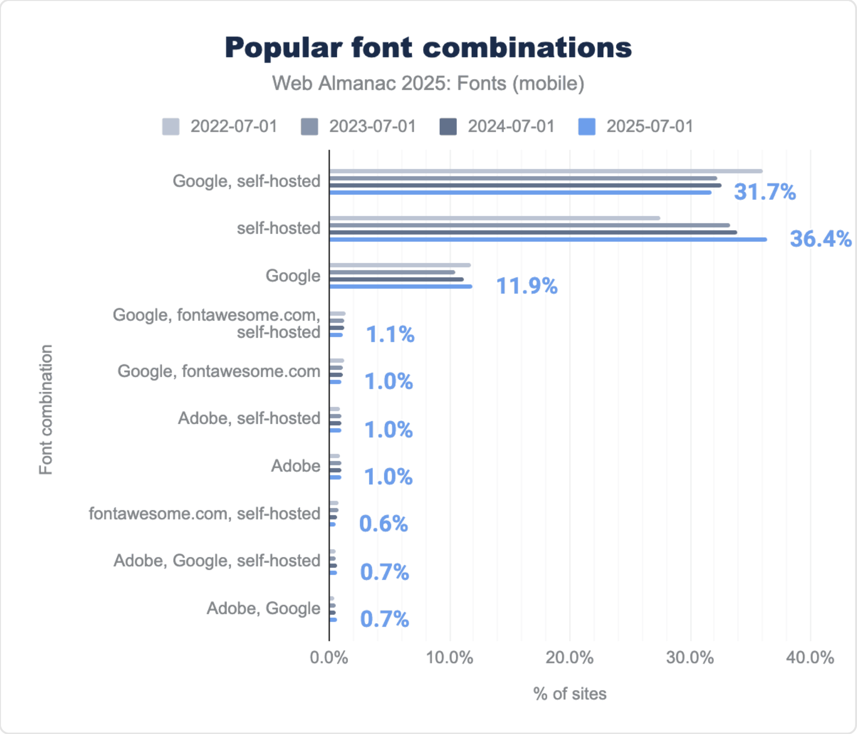


Figure 1.4. Popular font hosting combinations on mobile sites.

1. **Google Fonts together with self-hosted fonts**: used by ~36% of desktop sites and ~32% of mobile sites. This hybrid approach is very common: for example, a site might self-host some brand or icon fonts while also pulling in a Google-hosted font for body text (or vice versa).

2. **Self-hosted fonts only**: used by ~32% of desktop sites and ~36% of mobile sites (approximately 34% of sites overall). This means all custom fonts on the page are served from the site's own domain. As noted, this share has grown from about 30% last year to about one-third of sites this year.

3. **Google Fonts only**: used by ~13% of desktop sites and ~12% of mobile sites. These sites exclusively rely on Google's CDN for their font needs (often via the standard `<link>` to Google's stylesheet).

Together, these three configurations account for roughly four out of five websites. Every other combination (such as using Google + Adobe together, or Adobe + self-hosting, etc.) becomes rare very quickly in the crawl data. In fact, beyond the top few patterns, the share of any given combination falls below 1%, and by the time you get to the 8th or 10th most common configuration, their usage is almost imperceptible (around 0.05% of sites).

Beyond these commonplace service combinations, year-over-year the balance is tilting toward self-hosting. In 2024, about 30% of sites were exclusively self-hosted, but now it's roughly 34%. Correspondingly, the "mixed" approach (self-hosting + a service) has shrunk a bit (from ~38% to ~36% of sites on desktop, for example). In other words, fewer sites are hedging by doing both—more have decided to host everything themselves. This is likely due to performance and privacy motivations. As last year's chapter noted, modern browser changes like cache partitioning have removed some of the old performance advantage of using Google's CDN.

Font file hosting

Another way to look at font hosting is to ask: which specific font files are being requested the most, and are they coming from self-hosting or from services? This gives insight into the most popular font families on the web and how they're being delivered.

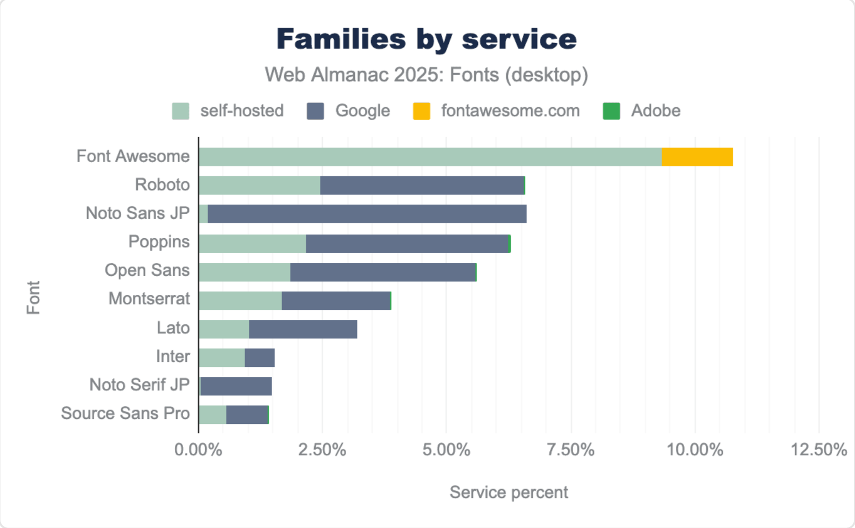


Figure 1.5. Families by service (desktop).

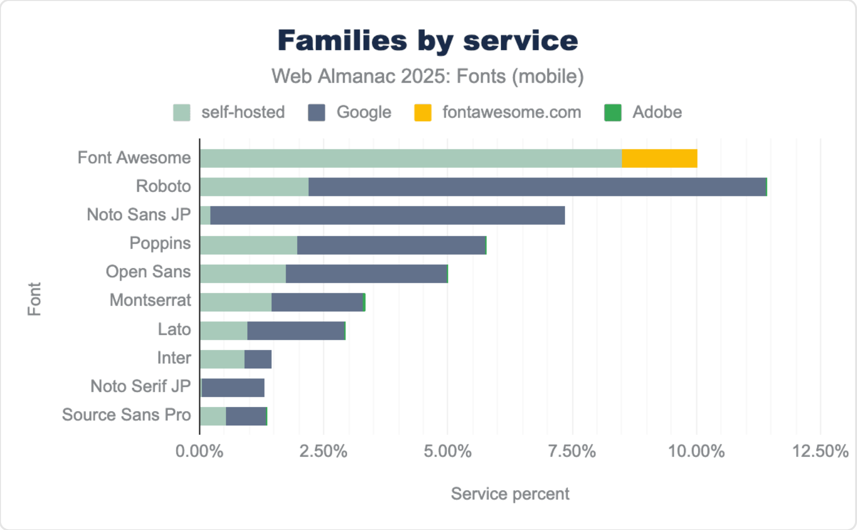


Figure 1.6. Families by service (mobile).

On the self-hosted side, traffic is heavily focused on icon font files and common library assets. A locally served copy of Font Awesome (the TTF/WOFF files of the icon set) accounts for about 8.5% of desktop font requests and 9.3% of mobile font requests in the 2025 crawl. That makes Font Awesome one of the most frequently fetched font resources on the web, even though it's an icon set rather than a text face. Notably, most of Font Awesome's usage is via self-hosting—only a much smaller fraction is fetched directly from the `fontawesome.com` CDN. Other popular utility fonts show a similar pattern: for example, IcoMoon (another icon font generator output) accounts for about 1.0% of desktop and 1.1% of mobile font requests, and ETmodules (often seen in WordPress themes) about 0.5% desktop and 0.7% mobile. These too are almost always self-hosted fonts residing on the site using them. In practice, this segment of "self-hosting" on the web is about bundling these common icon/font assets with your own site's files, not just self-hosting bespoke text fonts.

On the Google Fonts side, font requests have a classic long-tail distribution: a few extremely popular fonts at the head, and many more with smaller usage. On desktop, Roboto is the single most requested Google-hosted font, making up about 9.2% of all Google font requests. Right after it is Noto Sans JP (Japanese), at about 7.1% of Google's requests. This immediately shows how global Google's reach is—one of the top two fonts is a Latin/English UI face, and the other is a Japanese font. Below those, the list looks exactly like one might expect in 2025: Poppins (~3.8% of Google font requests on desktop), Open Sans (~3.3% via Google on desktop), Lato (~2.0%), and Montserrat (~1.9%). On mobile, the order shifts slightly: Noto Sans JP is actually first at about 6.4%, with Roboto at 4.1%, but Poppins (~4.1%), Open Sans (~3.7%), Lato (~2%), and Montserrat (~2.2%) are all in the same top group. In short, Google Fonts' highest-volume fonts include both widely-used Latin families and important non-Latin fonts (especially for CJK scripts).

It's interesting to note which of these fonts tend to be self-hosted versus served from Google. For example, Noto Sans JP (a large Japanese font) is almost entirely served by Google's CDN in our data. The self-hosted share of Noto Sans JP was only about 0.2% of requests, meaning nearly everyone who uses that font chooses to let Google handle the delivery (likely because it's a very large file, and Google's infrastructure is optimized for it). Noto Serif JP is similar: it appears around 1.3-1.4% of requests (mostly Google-hosted). In contrast, many popular Latin-script families have sizable self-hosted footprints in addition to their Google-served usage. We see Roboto being self-hosted too (about 2.2% of font requests on desktop are for self-hosted Roboto files, on top of the 9.2% from Google). Similarly, Poppins is ~2.0% self-hosted, Open Sans ~1.7% self-hosted, Montserrat ~1.4%, Lato ~1.0%, and Inter ~0.9% self-hosted (desktop figures). In each of these cases, the same font is also delivered by Google on many sites. Because the most popular Google Fonts (Roboto, Open Sans, Poppins, Montserrat, Lato, Inter) also appear at non-trivial rates in the self-hosted data, and because self-hosting is growing while Google's share is easing, it's likely that many projects begin with Google Fonts and later bring the same families in-house for performance or privacy reasons. (NB: Our data can show the overlap and directional shift, but not the per-site migration path.)

Adobe Fonts, being a different kind of service (one that requires a kit and serves a wide variety of distinctive commercial fonts), has a different profile. No single Adobe-served font dominates in share. The top Adobe font family in 2025 is Proxima Nova, which accounts for roughly 0.8-0.9% of font requests on both desktop and mobile. Below that, Adobe's traffic is spread across a long list of families. These are mostly popular branding and editorial faces like Futura PT, Brandon Grotesque, Freight Sans, Acumin, Sofia Pro, Aktiv Grotesk, and Museo, each contributing between a few hundredths of a percent up to half a percent of requests. This aligns with how Adobe Fonts is positioned in general: a wide variety of very fine fonts, each used by relatively few sites, rather than one font being found on millions of sites. It's essentially the opposite of Google's pattern, in which a handful of fonts account for a huge portion of usage.

Performance

Typographic performance on the web in 2025 is about shipping web fonts in the smallest, cleanest way possible. This section looks at the two performance levers that matter most: the formats (where the web has basically consolidated on WOFF2, with WOFF hanging on as fallback) and the actual transfer sizes those formats produce (where the median is perfectly reasonable but the long tail can really lag). If you wish to read these results as a web font hygiene checklist: be sure to use WOFF2, check your MIME types, subset large fonts as appropriate, and pay special attention if you self-host, because that's where most of the inefficiencies still reside.

File formats and optimization

When it comes to font file formats on the web, the modern WOFF2 format continues to lead the way. WOFF2 (Web Open Font Format 2) offers superior compression, resulting in smaller file sizes for faster transfer, and it's been supported by all major browsers for years now. In 2025, WOFF2 accounts for about 65% of font file requests on both desktop and mobile pages. This is an overwhelming majority of modern font formats, and represents the continued consolidation around WOFF2 as the web's standard font format. In other words, about two-thirds of all web fonts in this year's data are in WOFF2 format. This consolidation around WOFF2 is a positive trend for performance, as the better compression leads to faster font loads and less bandwidth usage.

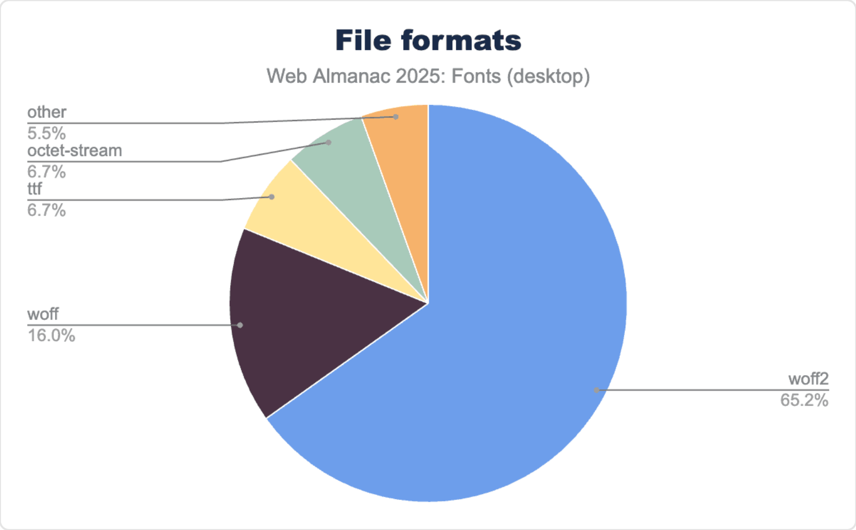


Figure 1.7. File formats for fonts on desktop sites.

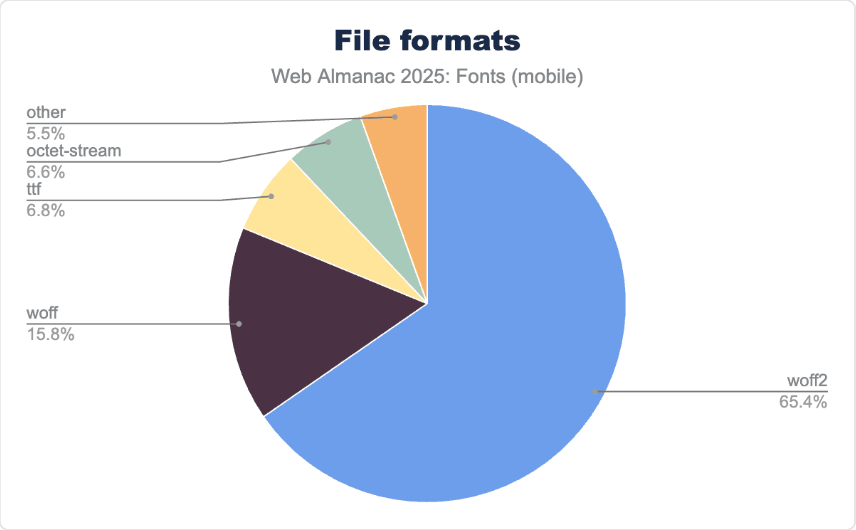


Figure 1.8. File formats for fonts on mobile sites.

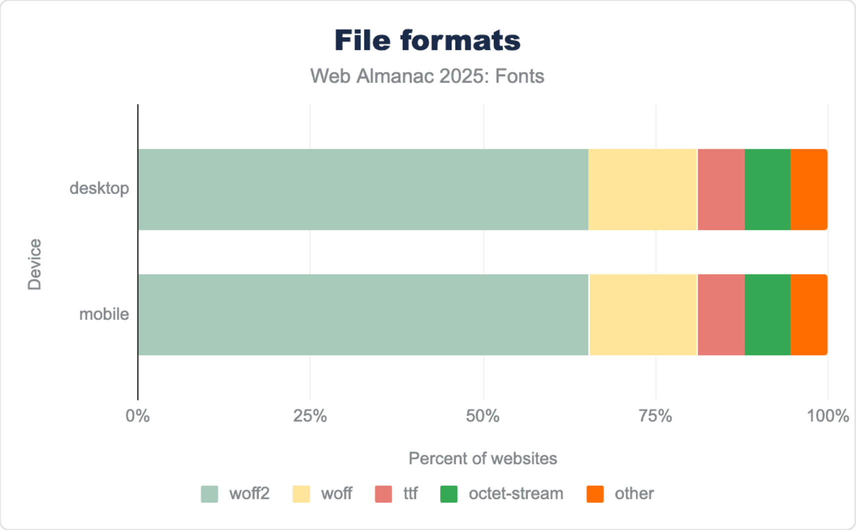


Figure 1.9. File formats for fonts by device.

Right behind WOFF2 is the older WOFF 1.0 format, which remains a significant fallback. In 2025, WOFF 1.0 comprises about 16% of font requests on both desktop and mobile. Combined with WOFF2's 65% share, the two WOFF formats together account for about 81% of all font requests. This reflects an ongoing consolidation around WOFF2 as the primary format, with WOFF 1.0 hanging on mainly for legacy and compatibility cases. More sites and font providers are phasing out WOFF in favor of WOFF2, since nearly all use cases are covered by WOFF2's broad browser support now. WOFF was a great improvement back in the early 2010s when it was introduced, but today it's largely a legacy fallback—used mainly for older browsers or by sites that haven't updated their font files in a long time.

Beyond WOFF and WOFF2, only minor formats remain, mostly as edge cases. As we note below, the arrival of Incremental Font Transfer technology next year may be an especially valuable improvement for large fonts and begin to cut into WOFF2's share. Still, a notable percentage of fonts are still served as raw TrueType (`.ttf`) files. In 2025 about 6.7% of font requests are TTF, often due to CDNs or self-hosting setups that serve the original font file without repackaging it as WOFF. Similarly, around 6.6% of font requests are delivered with the generic `application/octet-stream` MIME type. Note that `octet-stream` isn't a distinct font format—it usually indicates a misconfiguration where the server isn't specifying the proper font MIME (e.g., a WOFF2 file might be served with `Content-Type: application/octet-stream` instead of the correct `font/woff2`). In 2025, this misconfigured MIME issue unfortunately persists—roughly 6-7% of font files are still being sent with an incorrect MIME type. This is largely attributable to a few major hosts (in last year's analysis, cdnjs and Wix's CDN were notable culprits) that have configuration issues. It's a small but noteworthy slice, as using the correct MIME type (like `font/woff2` or `font/woff`) ensures better content handling and debugging.

Crucially, if we combine the modern formats, WOFF2 + WOFF together account for over four-fifths of the web font ecosystem. Other formats like OpenType/CFF (`.otf`) or SVG fonts have effectively vanished from the public web (despite being very much alive on the desktop), and EOT (the old IE-specific format) is practically extinct. Back in 2020, the Web Almanac already noted that SVG and EOT were nearly gone, and that trend has only continued. It's now rare to encounter a non-WOFF font file in a typical site crawl. The overwhelming advice remains: use WOFF2 for your web fonts, and drop the older formats unless you have a very specific need. This year's crawl data strongly validates this advice as we see virtually everyone converging on WOFF2, with WOFF as a trailing fallback, while everything else is negligible.

File sizes and performance

From a performance standpoint, using modern formats is important because font files themselves have been getting larger on average. Last year's analysis found that the median web font file size increased considerably from prior years, implying that sites are using more complex fonts (e.g., fonts with more glyphs to support multiple languages, or variable fonts that bundle many styles). Our 2025 data shows that this trend toward larger font files has continued, making optimization even more critical.

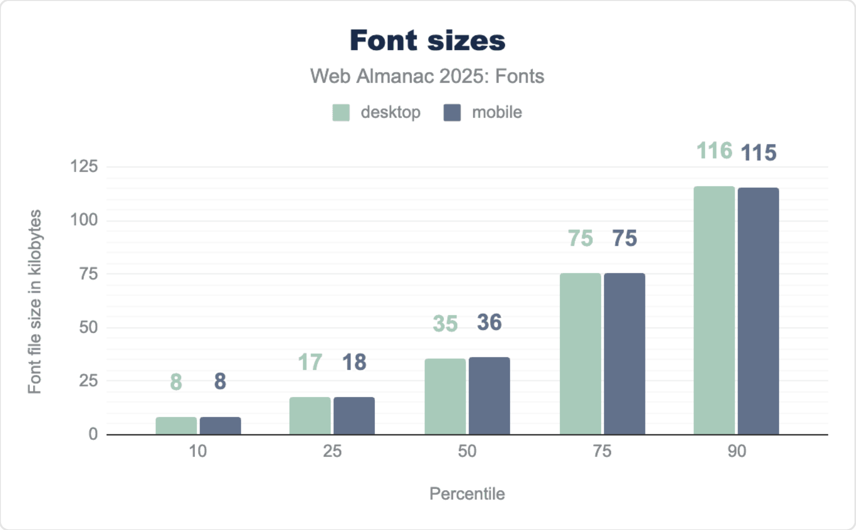


Figure 1.10. Font file sizes.

Looking at the distribution of font transfer sizes on the web:

- In 2024, the median font file size was around 36-39 KB (gzip-compressed), up from the low-30s KB a couple of years before. Now in 2025, the median remains in the mid-to-upper-30s KB range (mid-30s on desktop, slightly higher on mobile), indicating that typical font files haven't grown much further in the past year, they're holding at that larger size.

- At the 75th percentile, 2024 font files were about 76-77 KB. In 2025, this is roughly similar (mid-70s KB). So three-quarters of web fonts are under 80 KB, which is still manageable especially with WOFF2 compression.

- However, at the 90th percentile, 2024 saw font files around 112-116 KB. The 2025 crawl shows a similar 90th percentile in the 115-116 KB range. So, the top 10% of fonts are pretty large (well above 100 KB). These tend to be fonts for complex scripts (CJK fonts with thousands of glyphs), or full-featured variable fonts with many axes, or simply un-subsetted files.

- The tail end of the distribution is very heavy: at the 99th percentile, desktop font files in 2024 were around 776 KB (and 572 KB on mobile). In 2025, the largest 1% of fonts we see are still in the hundreds of kilobytes. These are likely massive CJK fonts, icon fonts with embedded graphics, or fonts accidentally served with debug tables. It's a reminder that while most fonts are reasonably sized, a few outliers can be extremely heavy.

The takeaway is that most websites serve moderately sized font files, but we have a non-trivial number of very large font files out there. This makes it important to use efficient formats and techniques. WOFF2 helps a lot by squeezing down file sizes, but if a font is half a megabyte uncompressed, even WOFF2 can only do so much. Techniques like subsetting (i.e., including only the needed glyphs) are essential for those very large fonts, especially for CJK or icon fonts.

While it's encouraging that the prevalence of WOFF2 will help mitigate the performance impact of larger font files, we also found that sites which self-host fonts have more room to optimize on average. For example, in 2024 only about 58% of self-hosted font requests were WOFF2 (versus about 78-81% overall). Self-hosted setups had a larger share of WOFF (around 18-19%) and even some raw TTF (about 6%). This implies that some developers self-hosting their fonts might be using older files or not compressing to WOFF2, whereas big providers like Google will always serve WOFF2 if the browser supports it.

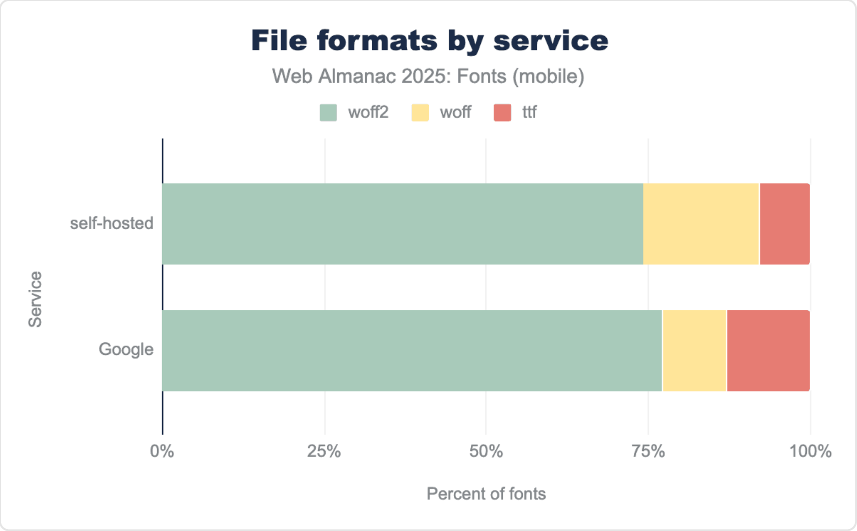


Figure 1.11. File formats by service (mobile).

The 2025 data shows improvement—more self-hosters are moving to WOFF2—but there's still a gap. It's a reminder that if you self-host, you need to be diligent: ensure you have WOFF2 versions of your fonts and configure your server to serve them with the correct `Content-Type`. A significant portion of self-hosted font files in 2024 were served with misconfigured MIME types (5-6% as mentioned), which is something that developers can fix with a simple server tweak. The good news is that WOFF2's dominance is growing across the board, just a bit more slowly in the self-hosted sphere than on Google's platform. In essence, self-hosted sites are catching up, but they still lag slightly in terms of using the optimal format.

CSS techniques and font loading

CSS techniques and font loading are where typography actually becomes real: pages don't just choose a font, they tell the browser how soon to connect to the font origin, which files to fetch first, and what to show while those files are in transit. In this section we look at those mechanisms—resource hints (`preconnect`, `preload`, and the fading `dns-prefetch`), the now-standard `font-display` patterns that split clearly between text and icon fonts, and the underlying outline formats.

Resource hints

Because web fonts can cause visual abnormalities through render-blocking or delayed text rendering, many sites use resource hints to optimize how fonts are loaded. The data shows a steady adoption of these hints, with some shifting as best practices evolve.

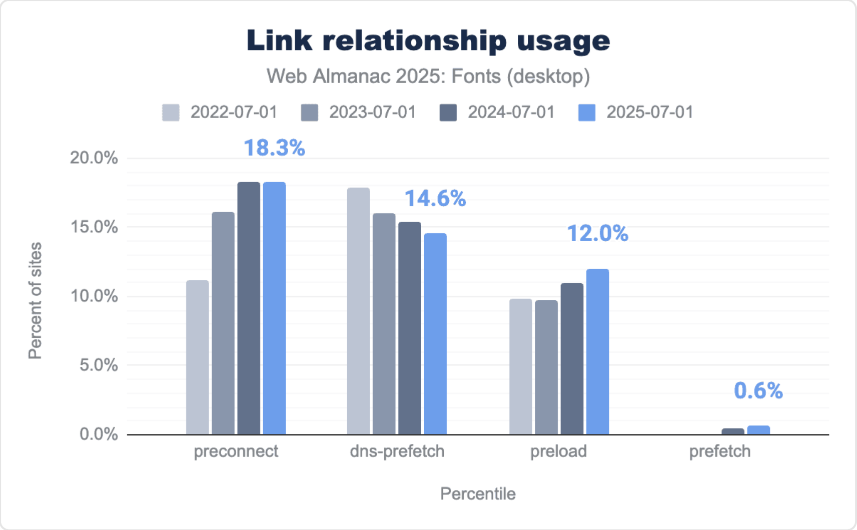


Figure 1.12. Usage of link relationship values for fonts on desktop sites.

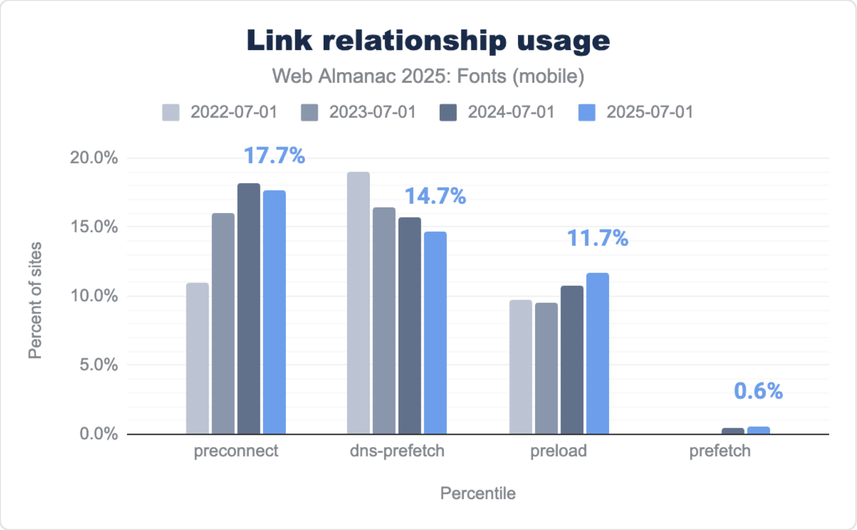


Figure 1.13. Usage of link relationship values for fonts on mobile sites.

The most popular hint is still `<link rel="preconnect">` (often used to preemptively open a connection to a font host). In the 2025 data, about 18.3% of desktop pages include a preconnect hint (virtually unchanged from 18.3% in 2024), and 17.7% of mobile pages do so. This plateau suggests that preconnect usage has leveled off after climbing in previous years. It's a common technique, present on nearly one in five pages, which makes sense now that many performance conscious sites add preconnect for domains like fonts.googleapis.com or their own CDN.

Meanwhile, `dns-prefetch` (a lighter hint that only resolves DNS) has been trending downward. In 2025, we found about 14.6% of desktop pages using `dns-prefetch` for fonts, down from ~17.9% in 2022. Mobile is similar at 14.7%. This decline makes sense: once developers discover `preconnect` (which includes DNS resolution as part of establishing the connection), the separate `dns-prefetch` becomes less necessary. Many sites that previously used `dns-prefetch` have upgraded to using `preconnect`, which provides a more substantial optimization by also doing the TLS handshake early.

The preload hint has seen a quiet but significant rise. In 2025, we found about 12.0% of desktop pages using `<link rel="preload" as="font">` for at least one font, up from about 11.0% in 2024. Mobile shows a similar increase (around 11.7% from last year). This indicates that more sites are explicitly preloading their critical web font files, rather than relying on the browser to discover them. Preloading a font can cut down on load delay because the browser starts fetching the font as soon as it sees the preload hint, instead of waiting for the CSS and then the font declaration. This technique is especially popular in performance-focused setups and is encouraged when font files are large or if you want to avoid FOIT (flash of invisible text).

Finally, `prefetch` remains a niche tool. It's used on only about 0.6% of pages in 2025 (up slightly from ~0.5% last year). `Prefetch` is generally used to load resources that might be needed in the near future (like another route or page), so seeing `prefetch` under 1% aligns with it being a rather specialized hint for fonts. The small uptick could be from single-page application frameworks or aggressive optimizers that prefetch assets for subsequent pages.

In summary, `preconnect` and `preload` are the dominant font-loading hints in use for 2025, with `preconnect` appearing on roughly one-fifth of pages and `preload` on about one-eighth. This matches the decline of `dns-prefetch` as it gets replaced by `preconnect`, whereas `prefetch` is rare but slowly growing. This reflects the industry's move toward stronger, more explicit loading signals: developers increasingly favor immediate actions (`preconnect`, `preload`) over speculative ones (`dns-prefetch`, `prefetch`).

Font display

The CSS `font-display` descriptor has effectively become standard practice in web font usage. Whereas a few years ago it was a "nice optimization" to consider, in 2025 most pages that use web fonts now also specify a font-display policy to control how text renders while fonts load. The data shows a strong preference for fast text rendering, with a clear split between text fonts and icon fonts.

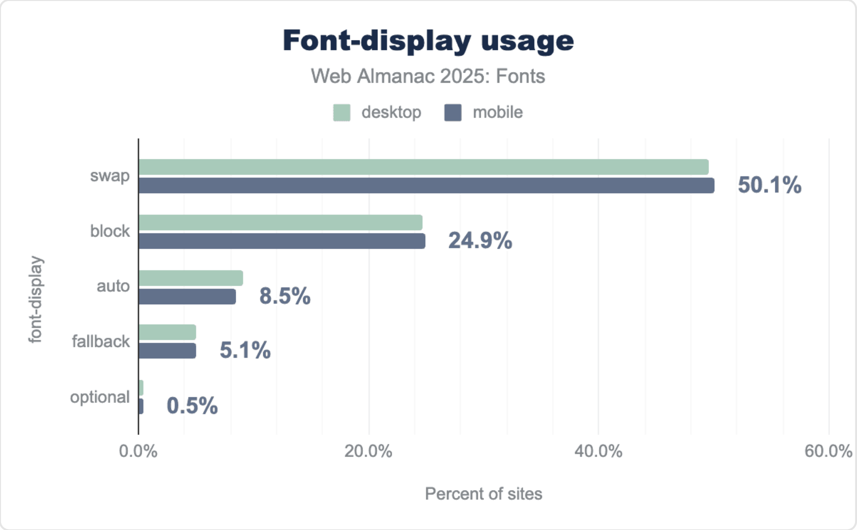


Figure 1.14. Usage of font-display values.

The most common choice is `font-display: swap`. It appears on about 49.6% of desktop pages and 50.1% of mobile pages in our 2025 crawl. In other words, about half of all pages using web fonts now specify `font-display: swap` for at least one font. This value ensures text is displayed immediately using a fallback font and then swapped to the web font when it finishes loading. The popularity of `swap` aligns with web best practices (it's recommended by Google Fonts and many experts). Essentially, it's now the default mindset for text content: developers prefer a quick first paint of text, even if it's in a fallback font for a moment.

Next, `font-display: block` is the second most common value, on about 24.7% of desktop pages and 24.9% of mobile pages—roughly half the prevalence of `swap`. By definition, `block` hides the glyphs briefly while the font loads, which could look risky if we assumed it was being used mainly for body text. But the data shows that about 70% of real-world `font-display: block` usage comes from icon font styles (Font Awesome, theme icon sets, etc.), where authors intentionally prefer a short delay to missing icons. In those cases, `block` is doing exactly what it's supposed to do: prevent broken UI rather than optimize for first text paint. So, virtually all the `block` usage is not from body text fonts, but from UI iconography that needs to either render fully or not at all. This pattern was a "surprise" finding in the 2024 chapter, and the 2025 data confirms it: the prevalence of `block` is not because many developers prefer invisible text for content, but because icon kits still ship with `block` as the default to prevent missing icons.

Beyond `swap` and `block`, the other values are much less common. `font-display: auto` (which defers to the browser's default behavior) is used on 9.1% of desktop and 8.5% of mobile pages. This share is shrinking as more people explicitly set a value. We also see `font-`

`display: fallback` (which allows a very brief invisible period and then always swaps) on about 5.1% of pages. This value is sometimes favored by developers who want text to swap quickly but still give a small window for critical fonts to load. Finally, `font-display: optional` remains a rarity, with only about 0.4-0.5% of pages using it. The `optional` value is the most aggressive for performance (often it will not swap at all if the font isn't loaded immediately), and it's typically used only by performance-sensitive sites or highly specific typographic setups.

Looking at individual font families and their font-display behavior reinforces these conclusions. The most commonly loaded text fonts (e.g., Roboto, Open Sans, Montserrat, Poppins, Inter, Lato) are overwhelmingly served with `font-display: swap` in their `@font-face` rules. For example, Roboto with `swap` shows up on about 12% of pages, Open Sans with `swap` on about 8%, etc. Conversely, the fonts we see with `font-display: block` are predominantly icon kits. Font Awesome's CSS (which by default uses `block`) was observed on about 17% of pages, while other icon fonts like ETmodules, IcoMoon, and Bootstrap Icons also contribute with `block` usage in the low-single-digit percentages. A small segment of text-oriented sites use `font-display: fallback` (for instance, some sites using Inter or certain serif fonts deliberately set `fallback` to balance flash of unstyled vs. flash of faux-bold text). But hardly any sites use `block` for regular text fonts.

The modern best practice is clearly reflected in the numbers: about half the web uses `swap` to ensure quick text rendering, and the quarter that uses `block` is mostly doing so for non-text glyphs. The remaining quarter either has not specified a `font-display` (thus using browser default) or is trying specialized strategies (`fallback` / `optional`). Compared to 2024, these proportions are essentially unchanged: last year, `swap` was used around 50% and `block` about 25% of the time. This stability means the guidance from 2024 still holds: if you see `font-display: block` in a site, it's likely an icon font. For almost all textual web content, developers today prefer `swap` for better UX.

Font outline formats

Another technical aspect of typography is the outline format (TrueType vs PostScript outlines) inside the font file. Most web fonts are either in TrueType (glyf table) format or CFF (Compact Font Format) outlines, regardless of the container. The data from 2022-2025 shows the web has overwhelmingly standardized on TrueType outlines.

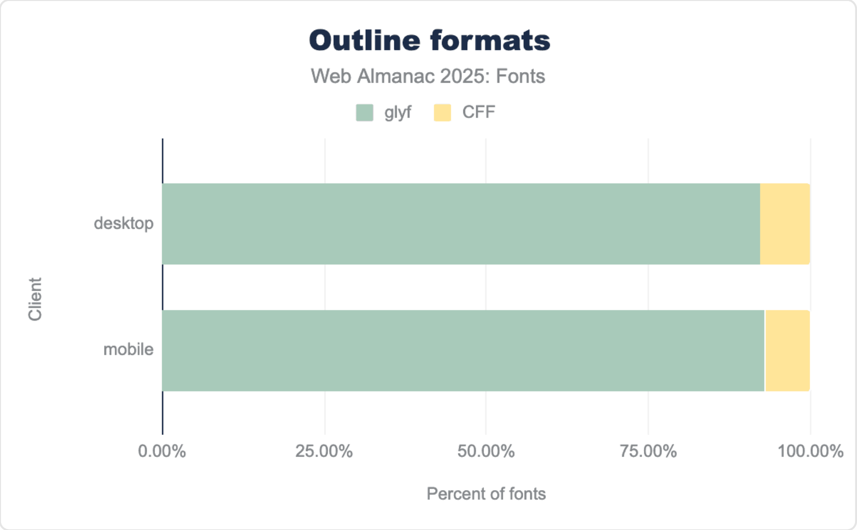


Figure 1.15. Outline formats.

This year, about 92-93% of fonts loaded on the web were using TrueType outlines (the traditional quadratic Bézier “glyf” table). This is up slightly from about 90% in 2022. In contrast, PostScript/CFF outlines (the cubic Bézier outlines often found in `.otf` fonts) have declined to around 7% of web fonts (down from ~9-10% a few years ago). Modern web font services and foundries often provide TTF/WOFF2 even for fonts originally designed as PostScript outlines, because TrueType can have performance or size advantages for certain use cases (especially when not hinting with bytecode).

Everything else is a tiny fraction. CFF2 (the newer variant used for variable fonts with PostScript outlines) is almost nonexistent in the wild, showing up in 0.01-0.02% of fonts. SVG glyphs (fonts with SVG tables, often color fonts) are also in the very low fractions of decimals. Essentially, outside the specialized case of color/emoji fonts, web developers don’t need to worry much about these—they’ll virtually always be dealing with TrueType-based fonts, with a minority of CFF outlines still around, especially from older packages.

Hyphenation and justification

Proper text layout involves breaking lines in the right places for readability. Browsers by default use a greedy line-breaking algorithm which can lead to uneven spacing or “rivers” of whitespace in justified text. Instead, we can use hyphenation (to break words) and the newer CSS properties for improved line breaking (`text-wrap: balance` or `pretty`). Let’s see how these are being used.

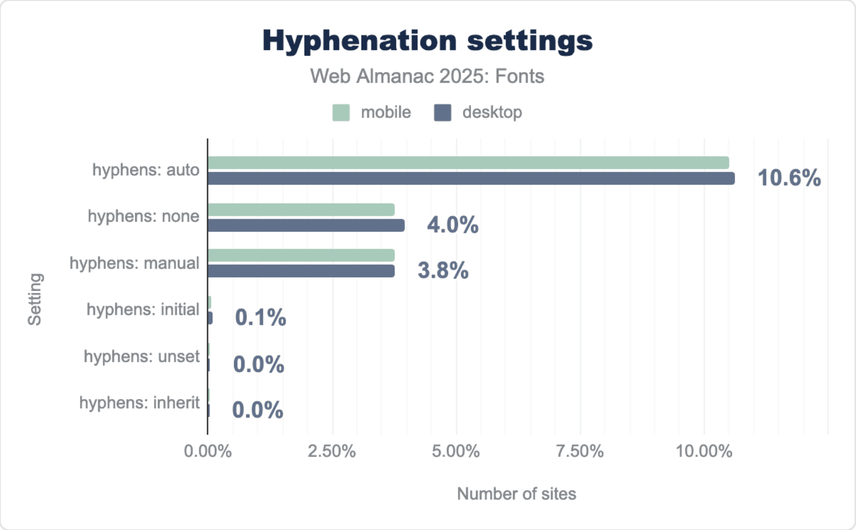


Figure 1.16. Hyphenation settings.

First, let's talk about hyphenation (CSS `hyphens` property). By default, browsers only hyphenate text if certain conditions are met (like `lang` is specified and the words are long, etc.), and some (like Chrome) might not hyphenate at all unless told. The `hyphens: auto` property allows automatic hyphenation when needed. In 2025, only about 10.5% of pages globally use `hyphens: auto` in their CSS. That's up a bit from previous years, but still quite low. This suggests that roughly one in ten sites explicitly enables hyphenation (usually to improve justified text or narrow column text).

On the other hand, about 3.8-4.0% of pages have `hyphens: none` or `hyphens: manual` set (often by resets or frameworks). This is usually done to ensure no automatic hyphenation (for example, some designers prefer to avoid hyphens in titles or certain components, or older CSS resets set `hyphens: none` to avoid any unexpected breaks).

Combining these, roughly 14-15% of pages mention the `hyphens` property at all. The vast majority (85%) do not touch it, effectively leaving it to the browser default (which for many languages means no hyphenation). So, hyphenation is far from universal, likely because it has to be used carefully (with correct language tagging and expectation of varying browser support), and not all content benefits from hyphenation (it's mostly a concern for justified text or narrow columns).

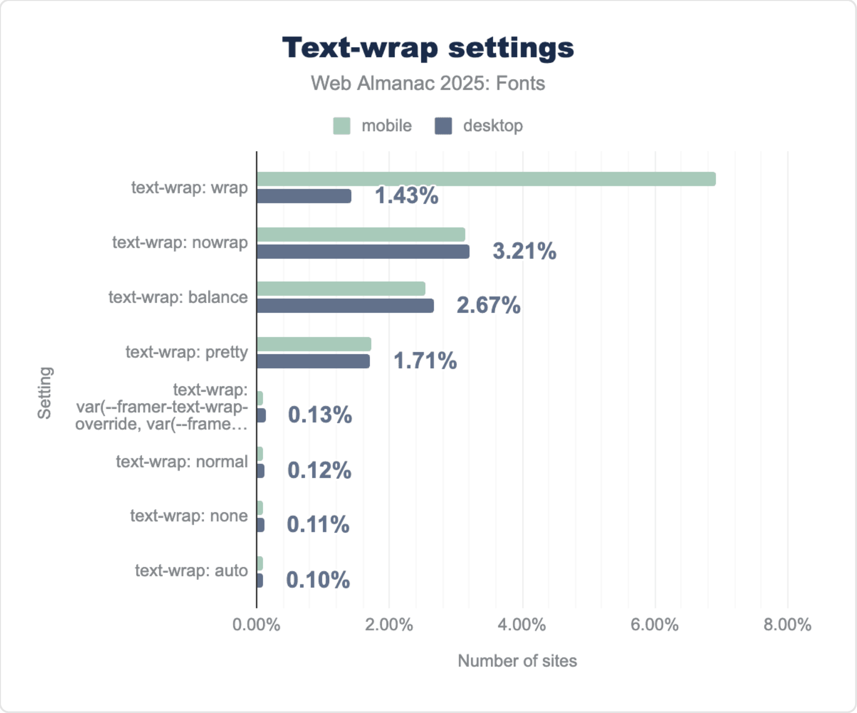


Figure 1.17. Text-wrap settings.

Text wrapping and line breaking (CSS `text-wrap` property): CSS Text Module Level 4 introduced `text-wrap: balance` and `text-wrap: pretty` to help browsers distribute text more evenly. These are fairly new (only recently supported in some browsers like Safari and Chromium-based ones). Their adoption, while small, is telling of an emerging best practice for multi-line headings and such.

On desktop sites, about 2.7% of pages use `text-wrap: balance`. On mobile, about 2.5% do. This property attempts to make lines roughly equal length (great for multi-line titles). It's impressive to already see it on about 1 in 40 pages given how new it is. This likely reflects quick adoption by frameworks or large sites for their hero sections.

The `text-wrap: pretty` property (which allows a bit of looseness in spacing to avoid very short words and other funny placements) is on about 1.7% of pages (both desktop and mobile). This is slightly lower than the `balance` property. `Pretty` is more for paragraphs, to avoid harsh breaks, and might be used selectively where full justification or certain line-breaking issues occur.

Interestingly, on mobile, `text-wrap: wrap` (which is the default normal behavior) appears explicitly on about 6.9% of pages. This could be due to frameworks restating it, or toggling from `balance` back to `wrap` for smaller screens. And `text-wrap: nowrap` (to prevent wrapping within an element) is used on about 3.1% of mobile pages and 3.2% of desktop pages, often to keep things like buttons or logos on one line.

The relatively higher usage of `nowrap` and explicit `wrap` on mobile indicates that controlling line breaks is particularly important on small screens (e.g., preventing awkward wraps in tight UI elements).

The hyphenation and text-wrap features together paint this picture: A minority of sites are really taking fine-grained control of text rendering (these likely include news sites, carefully designed editorial sites, or app-like interfaces). Those sites use hyphenation and balanced wrapping to improve text blocks. The majority of sites still rely on the default (which is non-hyphenated, greedy line breaks).

For justification specifically: If a site sets `text-align: justify` (which some do, especially for article content), ideally they should also use `hyphens: auto` and perhaps `text-wrap: pretty` to avoid large gaps. We see that not many do so yet, which means a lot of justified text on the web probably has suboptimal spacing. The tools exist (hyphenation, `balance`, `pretty`) but adoption is lagging. Part of that is browser support (these are somewhat cutting-edge properties), and part might be lack of awareness.

Lastly, properties like `hyphenate-character` (to specify a custom hyphen glyph) or `hyphenate-limit-chars` (to avoid hyphenating very short words) seem to be essentially unused—we saw negligible usage of those properties, which are very fine tweaks that few authors go into. Most just accept the browser defaults.

So, hyphenation is slowly growing but still used by only about 1 in 10 sites, and new line-wrapping options are emerging—a few percent of sites are already using balanced line wraps for nicer titles. We expect these numbers to rise as responsive design and multi-line layout techniques push developers to seek better control.

Families and foundries

This section looks at the supply side of web typography. First we trace which fonts and foundries actually get declared in CSS, then we look at the role of system and generic stacks, and finally we follow the metadata back to foundries, designers, and licenses to see what's most prevalent for users across the web.

Popular font families

Which font families are web designers actually using? By examining the CSS of pages, we can see which font-family names are most frequently declared. The results for 2025 show a mix of ubiquitous icon fonts, widely-used web fonts, and system font fallbacks.

The single most common font family seen in CSS is still Font Awesome (a leading icon font). It's declared on about 8.5-9.3% of pages in our dataset. This high prevalence is because many sites include Font Awesome's CSS (even if they use only a few icons from it), and some popular frameworks bundle it by default. Font Awesome being at the top underscores that developers frequently rely on icon fonts for UI icons, although this might change as inline SVG icons continue to rise.

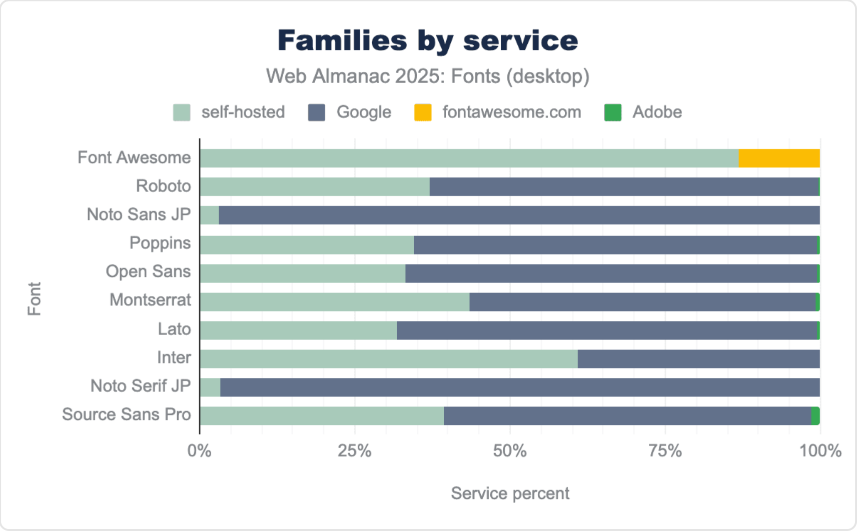


Figure 1.18. Font families by service on desktop sites.

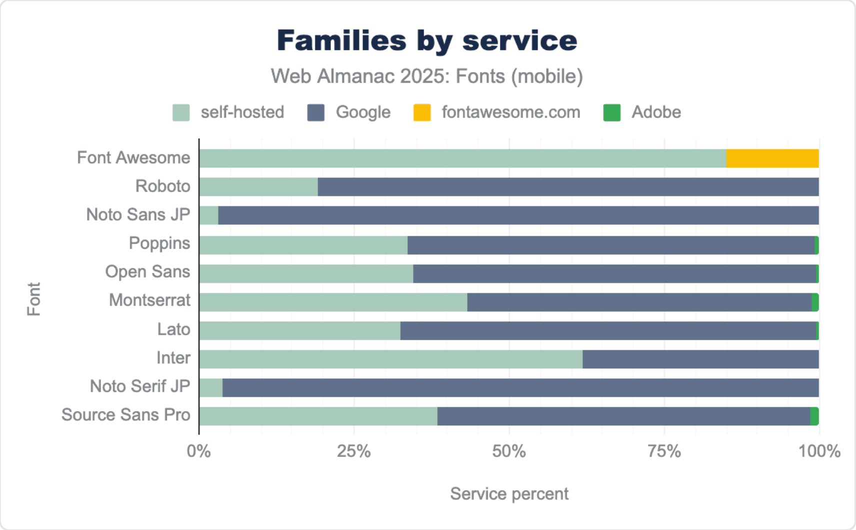


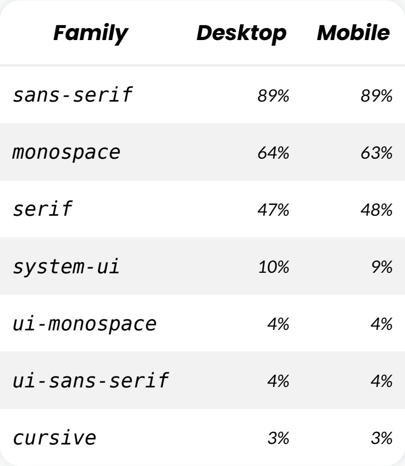
Figure 1.19. Font families by service on mobile sites.

Right behind are the usual text workhorses from Google Fonts and similar libraries. Roboto is declared on about 10-10.7% of pages, making it the second most popular family overall. Poppins (5.8-6.0% of pages) and Open Sans (5.0-5.6%) follow. These Google Fonts staples are top choices for sans-serif body and UI text on the web. Next is Montserrat (around 3.3-3.9% of pages), growing in popularity for headings and branding in recent years. One notable climber is Inter, which is now on roughly 1.4-1.5% of pages. Inter is a highly regarded open-source font for UI design, and its rise in usage reflects adoption by many modern frameworks and designers. We can also see other familiar names like Oswald, Nunito, and Raleway on a few percent of pages.

The main takeaway is that icon fonts and a few versatile sans-serifs dominate the CSS landscape. Font Awesome's prominence in CSS (even if not all sites using it render an icon to the end user) shows how frameworks can drive adoption. Among text fonts, Google's open-source offerings (Roboto, Open Sans, Montserrat, Poppins, Lato, etc.) continue to lead by a large margin—no surprise, as they're freely available, widely known, and cover many languages. We also see evidence that new open-source fonts (like Inter) can still break into the mainstream when they fulfill a need (in Inter's case, optimized for screens and very complete in features).

Generic and system fonts

Despite the prevalence of custom web fonts, system fonts and generic family names remain a bedrock of web design, especially for performance and native look-and-feel. Many sites specify system font stacks or fallbacks for body text or UI elements. The data from 2025 highlights how common these are:



Family	Desktop	Mobile
`sans-serif`	89%	89%
`monospace`	64%	63%
`serif`	47%	48%
`system-ui`	10%	9%
`ui-monospace`	4%	4%
`ui-sans-serif`	4%	4%
`cursive`	3%	3%

Figure 1.20. Most common system families

The generic `sans-serif` is by far the most common family name in CSS (as a fallback at the end of font-family lists). It's included on about 89% of desktop and mobile pages. Essentially, nearly every site has a line in its CSS that ends with `sans-serif` (or starts with it in a system font stack), to ensure there is always a default font. The generic `monospace` is also extremely common (seen on ~64% of pages) because developers often set monospaced fonts for code snippets or preformatted text. The generic `serif` appears on roughly 48% of pages, likely as a fallback for headings or when a serif design is desired.

What's more interesting is the rise of platform-specific generics like `system-ui`. The `system-ui` value (which maps to the operating system's UI font, like San Francisco on macOS/iOS, Segoe UI on Windows, Roboto on Android, etc.) is now used on about 9.7% of pages (desktop) and 9.3% (mobile). This is a significant uptick—these values barely registered a few years ago. Adoption is largely driven by modern CSS frameworks and design systems that aim for a "native app" aesthetic or which defer to the user's default UI font for performance reasons. Similarly, the newer generic `ui-monospace` (the system's default monospaced font) is used on ~4% of pages, and `ui-sans-serif` on ~3.5%. These numbers indicate that a non-trivial subset of sites (around one in ten) are explicitly choosing system fonts for their text, often for speed and consistency with native apps. This matches trends with sites like GitHub and Medium using system fonts to avoid loading any web font for body text.

In summary, system fonts have typically been the unsung heroes on many websites. Developers often use a hybrid approach, system fonts (or at least a safe generic fallback like `sans-serif`) for most UI text, and then load a few custom fonts for branding or headers. The generic `sans-serif`, `serif`, and `monospace` are ubiquitous as fallbacks, and the `system-ui` family and related ones are now more prominent in many CSS frameworks.

Font foundries

If we look at the `@font-face` metadata to see which foundries or vendors are credited with creating the fonts, we find that the majority is served by a small number of foundries—plus a large chunk of uncredited fonts—with a long tail of others in the low percentiles.

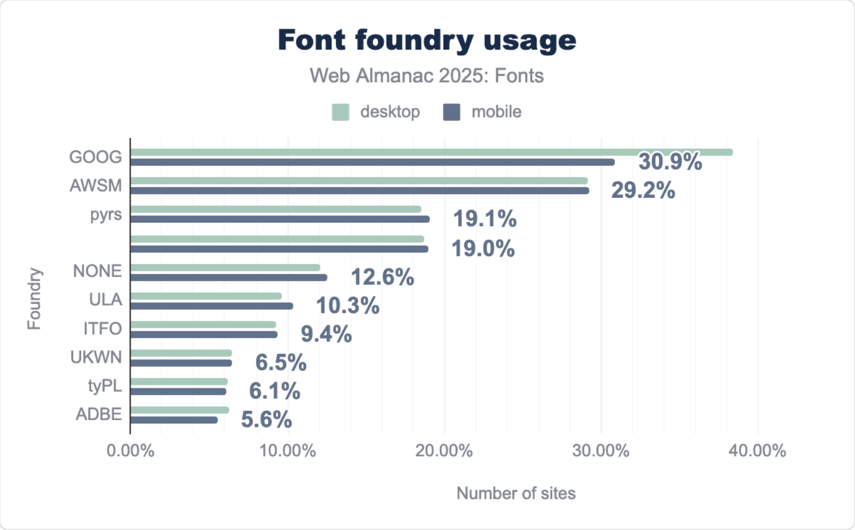


Figure 1.21. Font foundry usage.

By page coverage, the top “foundry” labels on the web are:

- **Google**, which is listed as the foundry for about 34% of websites using web fonts. This makes Google (via Google Fonts) the largest single source of fonts by far. This is largely due to Google’s funding of open-source designs by independent designers.

- **Font Awesome (AWSM)**, which is credited on roughly 29% of websites. This is the second largest share (trailing Google by ~5 percentage points) and entirely due to the ubiquity of the Font Awesome icon font. It’s a bit unique since Font Awesome is an icon set (the foundry “Dave Gandy” could also be cited, but often the font metadata uses “Font Awesome” or “AWSM” as foundry).

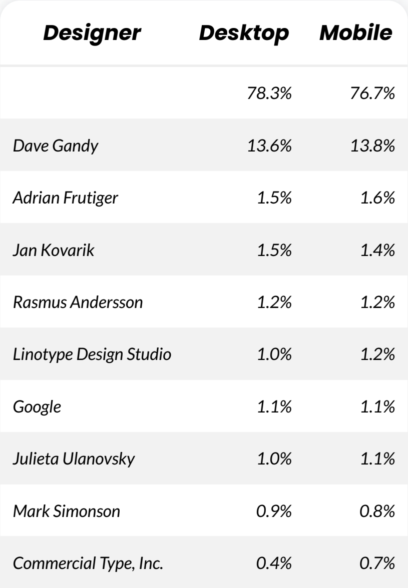
Together, Google and Font Awesome-affiliated fonts appear on about two-thirds of all pages, which is enormous. After those two, the next biggest category is actually unattributed or custom fonts. Around 19% of pages have fonts with no foundry specified (“UNSPEC”) and another 12% list “none” or a generic value for foundry. In total, well over a quarter of fonts lack clear foundry info in their metadata. This chunk likely represents custom self-hosted fonts, smaller foundries, or fonts where the naming wasn’t standardized.

Among identifiable foundries beyond Google and Font Awesome, a few stand out: “pyrs” (Fontlab) is listed on ~17% of pages (this is a label often associated with open-source icon sets or certain CJK fonts, and might need normalization from multiple case variations). ULA (for Julia Ulanovsky, the designer of Montserrat) is around 10%. Indian Type Foundry is around 9%. ADBE (Adobe) is around 6%. These numbers are on the same order of magnitude as last year, with some slight reordering due to how fonts like Noto are attributed. For example, Noto fonts were a collaboration between Google and others, so some of these collaborative fonts now show up under Adobe’s foundry code, slightly boosting Adobe’s share compared to last year’s method of counting.

In short, attributions for fonts on the web now fall to Google, Font Awesome, a few dozen publishers of popular fonts, plus a big anonymous middle. About 30% or so of fonts have unclear or no foundry info. The rest is split among a “who’s who” of active web font foundries: Indian Type Foundry (which has released many popular fonts via Google), independent designers grouped under foundry tags like “pyrs”, and a smattering of commercial foundries like Adobe, Monotype, etc. This distribution underscores the influence of Google Fonts as a purveyor of open-source fonts and the widespread use of Font Awesome, while also reminding us that a large portion of web fonts come from the broader open-source community or custom kits where attribution isn’t strong.

Font designers

Many font files include the name of the type designer(s) in the metadata. By tallying those, we can see whose work is most prevalent on the web (keeping in mind that many fonts don’t list a designer at all in the metadata).



Designer	Desktop	Mobile
	78.3%	76.7%
Dave Gandy	13.6%	13.8%
Adrian Frutiger	1.5%	1.6%
Jan Kovarik	1.5%	1.4%
Rasmus Andersson	1.2%	1.2%
Linotype Design Studio	1.0%	1.2%
Google	1.1%	1.1%
Julieta Ulanovsky	1.0%	1.1%
Mark Simonson	0.9%	0.8%
Commercial Type, Inc.	0.4%	0.7%

Figure 1.22. Most popular font designers

The data shows that most web fonts do not specify a designer. Approximately 77-78% of pages use at least one font that has no designer listed in its metadata. This could be due to omission or because the font is attributed to a foundry or project rather than individuals. That huge blank category means we should take the “designer rankings” with a grain of salt—it’s a partial view.

That said, one name dominates the credited share. Dave Gandy is the creator of Font Awesome, and his name appears on about 13.7% of pages (basically every page that self-hosts Font Awesome and has the designer field populated). This makes him by far the most visible type designer on the web (in metadata terms) because of the prevalence of his icon font. It’s an interesting quirk: it’s not that web developers are consciously choosing “a Dave Gandy typeface” for aesthetic reasons, but rather that one highly successful toolkit has put his name on millions of sites.

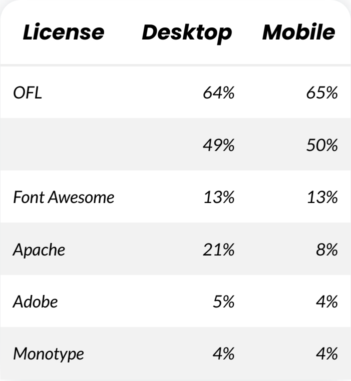
Beyond that, there’s a cluster of well-known type designers whose names each show up on around 1% of pages (give or take a few tenths). These include classics like Adrian Frutiger (1.5%) thanks to Univers and the eponymous Frutiger typeface, Rasmus Andersson (1.2%, the designer of Inter), the generic label “Google” (1.1%, used on some Google fonts when a specific designer isn’t credited), Linotype’s design studio (1.1%), Julieta Ulanovsky (1.0%, designer of Montserrat), and Mark Simonson (0.9%, designer of Proxima Nova among others).

These numbers reflect the web’s most popular fonts: Montserrat carries Ulanovsky’s name, Inter carries Andersson’s, Proxima Nova carries Simonson’s (though Proxima via Adobe might not list him, depending on the kit metadata), and so on. We also see names like Łukasz Dziedzic (designer of Lato, most likely showing up under multiple spellings) in the longer tail, and other contributors to Google font projects and major foundries. Each of these shares, however, is below 1% individually.

The key point is that designer attribution on the web is very sparse and highly skewed. One icon font author overshadows the stats due to a utility tool's popularity. The rest of the visible designer attributions reads like a mix of famous 20th-century type designers (Frutiger, etc.), creators of the most popular open-source fonts (Andersson for Inter, Ulanovsky for Montserrat), and large foundry teams (Linotype, Monotype, etc.). It's a testament to how fonts are distributed on the web: often the platform (Google, etc.) or foundry is front and center, and individual creators aren't always named unless they're attached to a big project that retains their name.

Font licenses

Font files often include a license URL or identifier in their metadata, which we can use to gauge what licenses are common on the web. The caveat here is that about half of fonts don't clearly specify a license in a parseable way. With that in mind, the data suggests that open source licenses completely dominate web fonts.



License	Desktop	Mobile
OFL	64%	65%
	49%	50%
Font Awesome	13%	13%
Apache	21%	8%
Adobe	5%	4%
Monotype	4%	4%

Figure 1.23. Most common font licences

By far the most common license is the SIL Open Font License (OFL). Around 64% of websites in our sample use at least one font under the OFL. This isn't surprising, as the OFL is the license used by the majority of Google Fonts and other open-source font projects. It essentially allows free use, modification, and redistribution of the font as long as the name is changed upon modification. The prevalence of OFL fonts means that nearly two-thirds of sites are using open-source fonts.

The next big “license” category is actually no license specified. On roughly 50% of pages, at least one font had a blank or unrecognized license field. This could mean the font is a custom one with no metadata, or a commercial font where the license isn’t embedded (as often happens), or even some data entry as “unknown”. It’s a reminder that a lot of font use on the web isn’t easily traceable by license in the file—but likely many in this bucket are proprietary fonts that just don’t declare their license in the `@font-face` (since web designers might not always fill that out).

The Font Awesome license (a proprietary/open license hybrid) is present on about 13% of pages, aligning with Font Awesome’s usage numbers. Adobe’s font license (for fonts via Adobe Fonts) shows up on about 4-5% of pages, which also matches Adobe’s usage share. Similarly, Monotype’s typical EULA covers about 4% (though many Monotype fonts on the web might appear as “no license” if self-hosted).

Aside from those, the Apache License (another open-source license used by some older Google Fonts and other projects) is the second most common explicitly listed license after OFL, reinforcing that open licenses are the norm. Interestingly, there’s a desktop/mobile split: it’s higher on desktop crawls (~21%) and lower on mobile (~8%), possibly due to certain fonts or frameworks being more common on desktop sites.

Beyond these, it’s a long tail of specific foundry and vendor licenses: we see markers for Commercial Type, Typotheque, Hoefler&Co (Typography.com), Dalton Maag, and Naver (for some Noto Sans CJK distributions) on less than 1% of pages each. These indicate specific use of commercial font libraries on certain sites, but for the most part, the web runs on open font licenses.

Global language support

The web is increasingly global, and so the web’s typography must support the world’s many writing systems. While Latin script is still the most commonly supported writing system in web fonts, other scripts are steadily rising thanks to the efforts in recent years to create and distribute fonts for them.

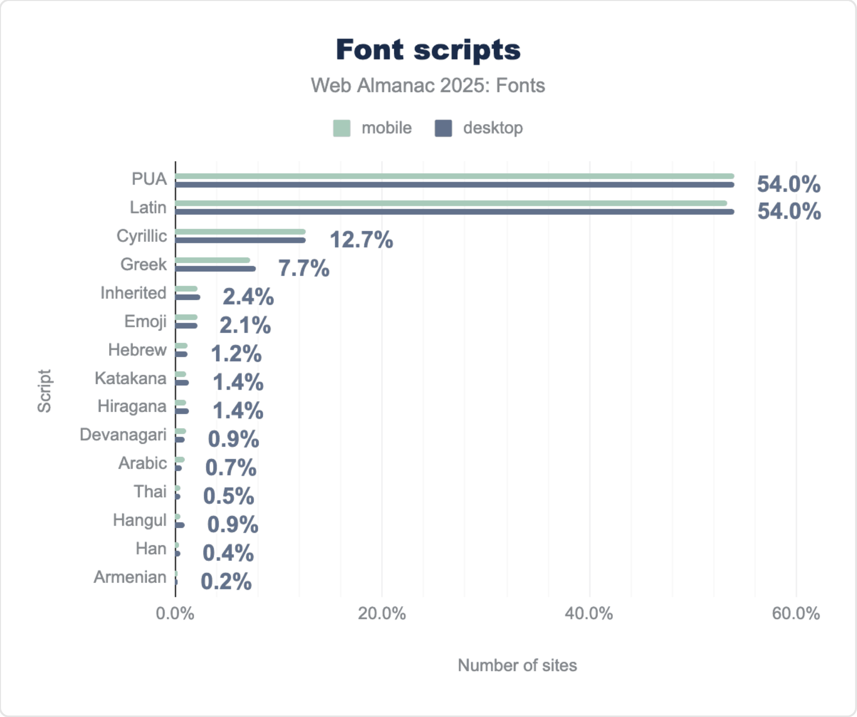


Figure 1.24. Font script usage.

In 2025, just over half of all web fonts include Latin characters (around 54% of fonts in our dataset). Since Latin is included in most multi-script fonts, this essentially means any font intended for European or American audiences is counted there. Alongside Latin, about 54% of fonts also include the Unicode Private Use Area (PUA), which is where icon fonts and custom symbols reside. The prevalence of PUA indicates (once again) how many fonts on the web are actually icon sets or have embedded icons (Font Awesome, theme icon fonts, etc., all map to PUA).

After Latin, the next most common script is Cyrillic, supported by roughly 12-13% of web fonts. Cyrillic's share has grown a bit (it was about 7% in 2022 and 10% in 2024), reflecting more fonts covering languages like Russian, Serbian, and Ukrainian. Greek coverage is present in about 7-8% of fonts, up from a few years ago as well. These extensions often come together because many extended character sets have Latin, Latin-extended, Cyrillic, and Greek in one file.

The presence of CJK (Chinese, Japanese, Korean) scripts is visible but more modest than their populations might suggest, mainly due to file size concerns. Japanese writing systems show up via Katakana (around 1.3-1.4% of fonts include Katakana characters) and Hiragana (similarly about 1.3-1.4%). Additionally, Han ideographs (the characters used in Chinese and Japanese writing) appear in about 0.4% of fonts. The Han percentage is small because full Chinese/Japanese font files are huge and relatively few sites serve them fully. Often, sites will rely on system fonts for those or use subsets (the Katakana/Hiragana presence suggests some Japanese fonts or subsets are being used). Korean Hangul syllabary is included in about 0.9-1.0% of fonts. While these numbers seem small, they represent the fact that some open-source CJK fonts (like Noto Sans JP, Noto Sans KR, etc.) are being used—a significant change from a decade ago, when hardly any websites served CJK web fonts due to size.

Other scripts with notable presence: Hebrew appears in about 1.2% of fonts (there are a number of Hebrew web fonts now, both via Google and self-hosted). Arabic script is supported by roughly 0.7% of fonts on desktop and 0.9% on mobile—a slight rise, possibly as mobile-heavy regions use more web fonts. Devanagari (used for Hindi and other languages in India) is present in just under 1% of fonts. Thai characters are seen in about 0.5% of fonts. Armenian and Georgian are each in the 0.2-0.3% range. Other South and Southeast Asian scripts like Gujarati, Gurmukhi, Tamil, Lao, Khmer each register below 0.1% of fonts.

So, while many scripts outside the Latin alphabet are used by under 2% of fonts, collectively they indicate broader adoption. The key trend is that the dominance of Latin-only fonts is slowly decreasing. More fonts are multi-script or targeted at other writing systems. A lot of this expansion in global language support is thanks to open-source font projects. For example, the significant Japanese and Korean usage can be attributed to open fonts like Noto Sans/Serif JP, Nanum Gothic, and Noto Sans KR, which have made high-quality CJK fonts freely available. In 2024, we noted that all of the top 10 web font families in Korean were open-source, and that pattern continues, showing that when designers have access to free, well-crafted fonts for their language, they will use these fonts. Similarly, the top web fonts for Japanese were open-source (Noto, etc.), carrying a lot of weight across neighboring languages due to shared Han characters.

One caveat is Chinese (particularly Simplified Chinese): it still remains rare to see Chinese-specific web fonts in use (the data above shows very low Han usage outside of Japanese subsets). Chinese font files are extremely large if you include thousands of characters, so most sites still avoid self-hosting them. They either use system fonts or serve a very limited subset for maybe a logo or heading. So, while we see growth in Japanese and Korean web fonts, Chinese has not seen the same uptake, largely for performance reasons. As a preview of developments to come, the development of Incremental Font Transfer (IFT) promises to shift the performance dynamics that have hindered Chinese web fonts in the past. IFT will enable users to fetch only the characters needed to display a specific page, dramatically reducing file sizes for languages with large character sets. If standardized and widely adopted, IFT could make full-featured Chinese web fonts much more practical by cutting the effective cost of those large character sets. The same mechanism would also benefit large, multi-axis variable fonts more broadly, since IFT can avoid transferring unused glyphs and unused variation data, making rich variable typography cheaper to deploy at scale.

In summary, the web font landscape is gradually becoming more multilingual. Latin is still on roughly half of fonts (and nearly all sites need Latin, even if just for numbers or basic symbols). But the share of fonts supporting other scripts is expanding year by year. It's a positive sign of increased localization and the fruits of efforts like Google's Noto project and others that have provided fonts for under-served languages. For a developer, this means that if you need, say, a Devanagari or Arabic font for your site, there's a decent chance an open web font exists and is being used by your peers. It also means that font file sizes and performance require attention from experimental developments like IFT, as multi-script fonts can be huge.

Advanced formats and features

As web fonts have expanded to cover more scripts and larger character sets, the next question is what we can actually do with these fonts. In this final section, we look beyond basic coverage to the advanced machinery inside modern font files—OpenType features, variable fonts, and color fonts—and how often developers are using those capabilities across the web.

OpenType features

OpenType features enable advanced visual styles like ligatures, alternate characters, old-style figures, and other details that signalled the hallmark of typographic craft in the print era. We can observe the adoption of these features through two approaches: whether fonts contain OpenType features, and whether sites are using those features via CSS.

First, looking at the fonts themselves: OpenType layout features are now the norm in web fonts. In the 2022 data, under half of fonts had an OpenType layout table (GSUB/GPOS). By 2024, it was a small majority (around 54% on mobile). In 2025, roughly 61-62% of distinct font files on the web include at least one OpenType feature table (like substitutions or positioning). This level of usage represents a clear majority of fonts, and it's growing each year. It reflects that newer font files (especially open-source ones) are being built with full features included, whereas older, simpler fonts (or icon fonts) might not have them.

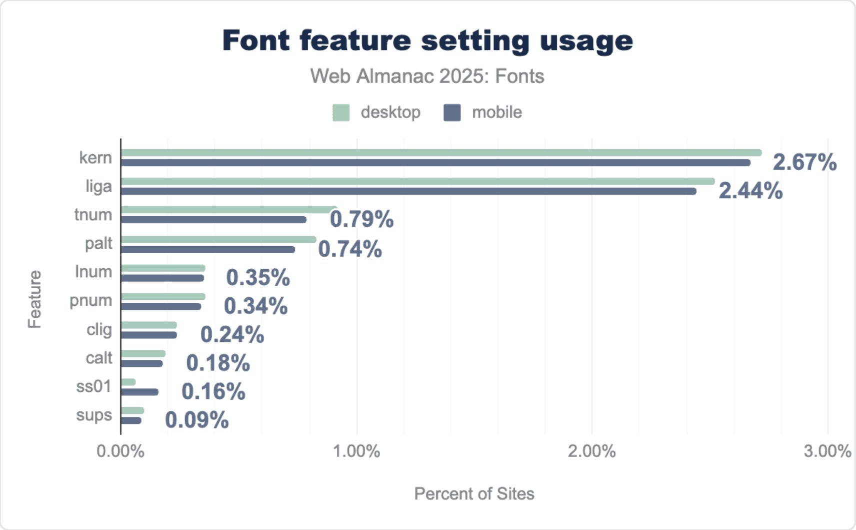


Figure 1.25. Usage of CSS font-feature-settings by feature.

If we weigh OpenType feature adoption by usage (font requests), the share is even higher. By requests, well over 99% of font fetches are for fonts with OpenType features. This number suggests that the fonts without OpenType features tend to be niche or very low-usage (or icon fonts that count as many files but are a smaller share of traffic). Essentially, almost every font that users actually see on the web is capable of features like ligatures or kerning. Even if the site isn't activating these features, the font file supports them.

So, many developers can assume OpenType features are available in the fonts they use (especially for text fonts). Only a minority (less than 4 out of 10 fonts, and shrinking) lack these tables, mostly older web-safe-ish fonts or specialized cases.

Now, which OpenType features are common in fonts? Among the fonts that have OpenType features, the most common ones are:

- **Kerning (`kern`)**: about 44-46% of fonts include a kerning feature. This was only ~35-38% a few years ago, so its presence has increased significantly. It means many fonts rely on GPOS kerning (or have a kern table) to adjust spacing between specific pairs of letters. It's a sign of quality and typographic refinement becoming standard.

- **Standard Ligatures (`liga`)**: Roughly 43% of fonts include this feature, which replaces sequences of characters (like “fi”) with a single ligature glyph. This huge jump from about 10% in 2022 now indicates that most new fonts are supporting ligatures (even if it’s just the basic ones).

- **Localized forms (`locl`)**: About 33% of fonts have this, which allows different glyphs for different languages (like Polish kreska accents or Turkish dotted i). This being one-third means many multi-language fonts have built-in local optimizations.

- **Numerical features**: Around 33% support fractions (frac), and about 25% support numerators (numr) and denominators (dnom) (for building arbitrary fractions). Tabular figures (tnum) and proportional figures (pnum) are each in roughly 20% of fonts. This is great to see—it means if you pick a random modern font, there’s a decent chance it has the fancy number alignments and fraction-building ability, which is useful for tables, financial data, etc.

- **Mark positioning (`mark` and `mkmk`)**: About 17% have basic mark positioning, and 14% have mark-to-mark. These are critical for Arabic, Indic scripts, and any accent stacking, such as Vietnamese characters. The fact that mark positioning is found in about 1/6 of fonts suggests the impact of complex scripts since it’s less common for Latin-only fonts to need a mark table. On the other hand, Google’s efforts to increase Pan-African support within many Latin fonts may also be boosting the presence of mark tables.

- **Stylistic sets (`ss01` , `ss02` , etc.) and alternates**: These show up in the data at healthy but lower levels. ~12% of fonts have an ss01, and smaller percentages have ss02, ss03, etc., or a salt (stylistic alternate) feature. This corresponds to fonts that offer optional alternate designs for certain characters (like a different “a” or “g”).

- **Caps and small caps**: Features like smcp (small caps) or c2sc (caps to small caps) are present in some fonts (a few percent), reflecting more advanced typographic families.

- **Script-specific features**: For instance, rlig (required ligatures, used in Arabic) and various Indic features (like abvs, blws for above-base, below-base substitutions) each appear in around 1% or less of fonts. That matches roughly the share of fonts for those scripts, which we discussed above.

Overall, this means web fonts are getting more sophisticated. Over half the fonts seen in this year's web crawl have at least one OpenType trick up their sleeve, and many have a dozen or more. Common things like ligatures and kerning are now expected, not just special perks.

Now, are developers using these features in CSS? Many browser engines apply core features like `liga` and `kern` by default (unless turned off), so some features are used without the author doing anything. But authors can explicitly control features via CSS `font-feature-settings` or higher-level `font-variant` properties.

The usage of low-level `font-feature-settings` is relatively small in absolute terms. Only about 2-3% of pages manually set specific OpenType features through this property. The most commonly toggled features in CSS are `kern` and `liga`—likely because some CSS resets or frameworks ensure kerning is on or ligatures are on/off deliberately. In 2025, `font-feature-settings: kern` appears on roughly 2.7% of desktop pages, and `liga` on about 2.5%. After that, there's a drop: `tnum` (tabular numbers) on ~0.8-0.9%, `palt` (proportional alternate width, used for some CJK spacing tweaks) on ~0.7-0.8%, and then things like `lnum` (lining nums) or `pnum` on ~0.4%. Stylistic sets (like `ss01`) or discretionary ligatures appear on only ~0.1% or less of pages via direct settings.

However, if we count pages that have any `font-feature-settings` usage (including those that might just set `normal` or include it via a library), that number is higher. Around 11-12% of pages have at least one `font-feature-settings` declaration. This suggests that many sites include some boilerplate or framework that touches font features (for example, setting `font-feature-settings: "liga" 0` to turn off ligatures in certain cases, or some older kits enabling certain features).

In contrast, the more human-friendly `font-variant` CSS properties are used on about 5-6% of pages. The most common is `font-variant-numeric`, often used to request tabular numbers or old-style figures. For instance, `font-variant-numeric: tabular-nums` is seen on 3.2-3.3% of pages, and we see small usage of properties like `font-variant-caps: small-caps` (~1.5% of pages). A not insignificant number of pages (around 0.5%) use `font-variant-ligatures: none` to deliberately disable ligatures (commonly as part of a CSS reset or for monospaced code where ligatures are typically unwanted).

The trend here is that developers are gradually adopting OpenType controls, but not universally. Most rely on browser defaults (which usually handle the basics). A subset—roughly one in ten pages—includes some advanced tweaking, either via their framework or by hand. The most frequent intentional uses are aligning numbers (tabular vs proportional), explicitly enabling/disabling ligatures or kerning, and occasionally using small-caps. The fact that direct `font-feature-settings` usage is about double the `font-variant` usage might indicate that frameworks output the low-level syntax, or that for certain less common features (like stylistic sets) developers have no choice but to use `font-feature-settings` since a higher-level property doesn't exist.

For day-to-day guidance: it's good news that if a font has features like ligatures, they're likely already working by default in browsers. And if you need special features (say, tabular numbers), a small but growing number of sites demonstrate how to use them via CSS. We expect that as more designers become aware of the fine typography controls available, these numbers will continue to rise.

Variable fonts

Variable fonts (which allow multiple styles such as weights and widths in one file) have moved from experimental to fairly mainstream in the past few years. The 2025 data shows continued growth in adoption.

How popular are variable fonts? This year, 39.4% of desktop websites and 41.3% of mobile websites used at least one variable font on their pages. In other words, now about 4 in 10 sites are using variable fonts. This is up from roughly 33% on desktop and 34% on mobile in 2024. The growth (~6-7 percentage points in a year) is steady and significant. Notably, mobile has consistently led desktop by a small margin in variable font adoption—possibly because mobile performance benefits more from combining font files, or because many modern mobile-first sites or PWAs have embraced variable fonts for their flexibility.

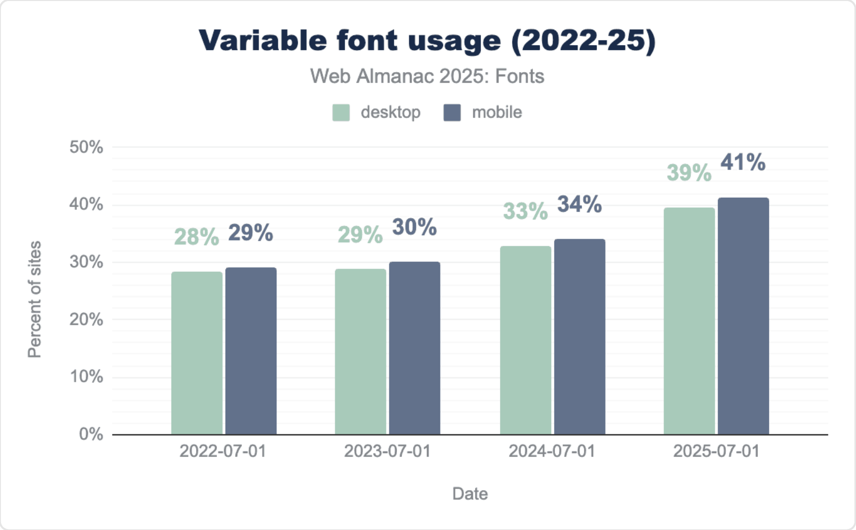


Figure 1.26. Usage of variable fonts by device.

However, the usage distribution of specific variable fonts is very top-heavy. Just a few font families account for a very large share. The single most used variable font on the web remains Noto Sans JP (the Japanese sans-serif from Google's Noto family). In 2024, Noto Sans JP was found on about 27% of all sites using variable fonts, and in 2025 it now appears on roughly 25-26% of pages using variable fonts. That makes Noto Sans JP the top variable font by reach—a testament to how many sites rely on this font in East Asia and globally for CJK content. After that, the next three biggest variable fonts are Roboto, Open Sans, and Montserrat. Combined with Noto Sans JP, these four families account for almost 60% of all variable font requests on both desktop and mobile. Specifically, Roboto is used on about 15% of variable font sites, Open Sans on 11%, and Montserrat on 7-8%. These are all available through Google Fonts in variable form, which explains their prominence (as Google started serving these as variable fonts by default).

Beyond the top four, we see a cluster of other notable variable fonts: Noto Sans KR (~5% of variable font sites, reflecting Korean usage), Noto Serif JP (~4%), Noto Sans TC (~2%), which are all part of the same Noto family for CJK scripts. Then popular Latin variable fonts like Inter (~3.3%), Raleway, Oswald, Playfair Display, Rubik, Roboto Slab, Google Sans, Nunito, each on 2-3% of sites that use variable fonts. This long tail shows that while many variable fonts exist, the ones being used at scale are mostly those that solve a big problem (like covering multiple scripts or multiple styles efficiently) and are easy to obtain (mostly open-source via Google).

It's also interesting to observe how these variable fonts are being used. The common axes that are set in CSS on variable fonts provide insight: The weight axis (`wght`) is the most used by far—weight is explicitly adjusted on about 57-61% of pages using variable fonts. This indicates that most people use variable fonts basically to have multiple weights in one file (e.g., making text bolder or lighter at different breakpoints). The next most common axes, however, come from a specific variable font system: `FILL` and `opsz` (optical size) each appear on roughly 35-39% of variable font pages, while `GRAD` follows at about 27%. These correspond to the Material Symbols icon font (which has axes for glyph fill, grade, and optical size)—a widely used variable icon font by Google. So, a large portion of variable font usage is actually due to icons with multiple stylistic axes, not for text. Width (`wdth`) and slant (`slnt`) are the other main axes in use: slant is found on about 19% of desktop (24% of mobile) variable font pages, and width on about 17% of both. This suggests a smaller but solid fraction of sites are tweaking width or using intermediate slants (perhaps as replacements for true italics or responsive adjustments). All other axes (like custom design axes or many of the more granular axes in something like Roboto Flex) show up at under 5% or even 1% of usage—possibly even as noise from demos or very specific projects.

So in practice, variable fonts on the web are mostly being used for weight adjustments, and in some cases width or slant. The high showing of `opsz` , `FILL` , and `GRAD` seems to be an artifact of one popular icon font's usage rather than a broad trend. The promise of variable fonts (fine-tuning typography with continuous variation) is supported by high levels of technical integration, but still in an early phase of creative exploration for most designers—many are using variable fonts as convenient multi-weight files, not yet as fully dynamic resources for typographic expression.

And then, variable font animation (using CSS transitions or keyframes on font variation properties) is extremely niche. Only about 0.3% of all pages in 2025 include an animation or transition involving variable font axes (either via the low-level `font-variation-settings` or via `font-weight` or other shorthands on a variable font). Even among just the pages that use variable fonts, that's under 1% (around 0.7-0.8%) that do any kind of animation with them. The cases where it does happen are usually decorative, like a hero title expanding or contracting, or an icon button with a hover effect that changes weight. It's not common presumably because it can cause layout shifts or just because not many have ventured into that design space yet.

Color fonts

Color fonts (a new technology offering fonts with multicolored glyphs, often for emoji or icons) have been an intriguing technology, but their adoption on the web remains extremely limited. The 2025 data shows a slight increase from an almost nonexistent base, but they're still far from mainstream.

0.06%

Figure 1.27. Percent of mobile pages using color fonts.

Only about 0.05-0.06% of websites observed in our crawl were using a color font. That's on the order of 1 in 2,000 pages. This has grown from roughly 0.01% a few years ago—still a 5x increase, but tiny overall. In raw terms, we found color fonts on roughly 6,000-7,000 desktop pages and a similar number of mobile pages. So, while the usage has "multiplied," it still resides in the territory of rounding errors compared to other figures in this report.

Why so low? A few reasons: Until recently, cross-browser support for the newer color font formats (COLR/CPAL v1) was incomplete. Authoring color fonts or finding use cases beyond emoji has been uncommon. And many potential uses (icons, symbols) have been served just as SVG or images instead.

Looking at the few sites that do use color fonts, what are they used for? The data shows that a vast majority of pages with color fonts are using them for emoji or specialized iconography. In fact, the most common color font by far is Noto Color Emoji (the Google/Android emoji font, which has multiple formats including COLR and CBDT bitmaps). Noto Color Emoji appears on about 73% of desktop pages that use a color font (and a majority of mobile color-font pages too). Essentially, most sites that use a color font are using it to render emoji with a consistent look, rather than relying on the OS emoji. After that, a substantial chunk is due to two Japanese color SVG fonts (used as decorative text in certain sites): those account for a respective 7% and 5% each of the color-font-using pages. Everything else is a long tail: Twemoji (Twitter's emoji) on a few percent, some icon fonts with color layers, and a handful of decorative multi-color text fonts.

In terms of color font technology, there are a few formats: SVG-in-OpenType, COLR/CPAL (v0 and v1), and older bitmap fonts (CBDT/CBLC for Google's bitmaps, sbix for Apple). The usage breakdown by font requests in 2025 is roughly 58-60% SVG, 25% COLR v0, 16% COLR v1, and a small couple percent for CBDT/sbix (bitmap). SVG being the highest is interesting—it's actually because of those two Japanese fonts which are SVG-based and used on some high-traffic pages. COLR v0 (the first version of the layered vector glyph format) is used by many current color fonts (including some emoji sets and icon fonts). COLR v1, which allows gradients and is more compact for emoji, is new but already accounts for ~16%. This likely comes from updated emoji fonts like Noto Emoji COLR v1. The bitmap formats (CBDT for Google's color emoji, sbix for Apple's) are almost negligible now on the web—they were more for native apps.

Another notable phenomenon: Some color fonts include multiple formats for compatibility. For example, Noto Color Emoji might have both CBDT and COLR tables. This means counting by "families" can double count that font in multiple format categories. But from a usage perspective, COLR v1 is gaining traction with increased browser support (it's much more efficient than SVG or bitmaps for emoji).

Color palettes within color fonts (where a font provides different preset color schemes) are not widely used. Over 95% of color fonts have either zero or one palette defined. That is, they have a fixed set of colors, or else rely on the default. Only a tiny fraction have multiple palettes (such as an emoji font that could switch skin tones or themes via the palette mechanism). And those color fonts that do have multiple palettes, often just have two or three at most. Most color fonts today are not aiming to offer thematic color switching—they just hard-code the colors needed. This is as it should be: the place for thematic switching is in CSS as font palette overrides, not in the font.

The number of color layers is also small for most color fonts used today. Many icon fonts might use just 2 colors (e.g., foreground and background). Larger numbers of layers appear only in full emoji sets (which can have hundreds of layers to construct all emoji characters).

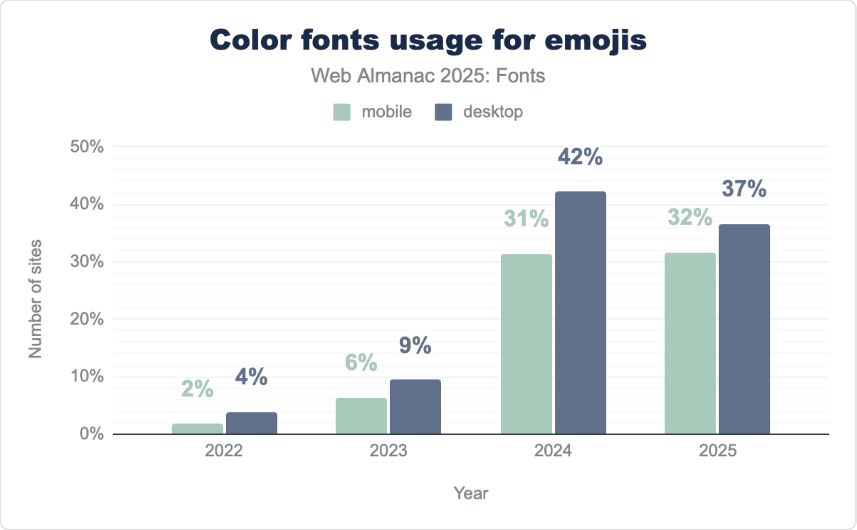


Figure 1.28. Color fonts usage for emojis.

So, we can see that by volume, about 32-37% of color font requests are for emoji (this percentage went up in 2024 and slightly down this year as other uses grew). That trend means the majority (~63-68%) of color font requests are now for non-emoji uses (like those decorative CJK fonts, or multi-color icons). However, by page count, emoji fonts (Noto Color Emoji in particular) are still the principal use of color fonts. While some large platforms or sites may include Noto Color Emoji as a fallback, contributing to many pages, other color fonts might be used on a few pages but fetched many times (like decorative fonts on a site with many page views).

In 2025, color fonts remain a novelty on the web. They're growing, but from a tiny base. Most developers either don't need them, or else avoid them due to past support issues. The ones who do use color fonts are primarily ensuring consistent emoji rendering or doing something very custom (like a multi-color logotype or icon). As browser support stabilizes (COLR v1 is now supported in Chrome, Edge, and Firefox), we might see more usage—for example, some people might start embedding emoji fonts to have colored emojis that match across OSes. But for now, in terms of advanced formats, color fonts are hundreds of times less common than variable fonts on the web.

Conclusion

The state of web fonts in 2025 depicts a mature, nearly ubiquitous technology that's continuing to see incremental improvements rather than major changes. Over the last decade, web fonts have gone from a fringe feature to an essential part of the web: almost nine out of ten sites use them, and that proportion grows slowly each year toward total saturation.

Several key trends observed in previous years have solidified:

- **Self-hosting versus services:** Web font delivery is shifting slowly from services toward self-hosting. Google Fonts still delivers fonts for over half of all websites, but its share is gently declining as more developers choose to host fonts themselves for performance or privacy. About one-third of sites now self-host all their fonts, up from about one-fifth a few years ago. Another large share of sites use a hybrid of Google's CDN plus self-hosting. This suggests a balance where developers are not entirely reliant on third-party infrastructure and can optimize caching and loading to their needs.

- **Format consolidation:** The web has standardized on WOFF2 as the primary font format. About 65% of font files are WOFF2, and combined with WOFF, that's about 81%. Old formats like raw TTF, EOT, or SVG fonts are almost completely phased out. This consolidation is great for performance (WOFF2 has superior compression) and simplifies what developers need to support. The data shows that most sites have followed in this direction, though a few self-hosted environments still lag in offering WOFF2 or sending correct MIME types.

- **Font sizes and performance:** The typical web font is in the 30-40 KB range (compressed) and sites often use multiple fonts, so fonts still carry a slight weight cost. The median size went up a bit recently, reflecting richer fonts (e.g., variable fonts or ones with more glyphs). A significant portion of fonts are larger than 100 KB, especially for Asian scripts or when not subsetted. This underlines the importance of subsetting and careful loading strategies. The fact that most of these fonts are now WOFF2 helps mitigate the cost, but developers should still be mindful: a heavy use of many large font files can hurt load speeds. The 2025 crawl data suggests most sites are moderate in their font usage, but a small percentage go overboard and skew the “tail” of the distribution dramatically.

- **Global scripts:** Web typography has become more globally focused, with greater support for languages beyond the Latin script. We see a meaningful presence of Cyrillic, Greek, Arabic, Hebrew, Devanagari, CJK, and others in web font usage every year. This is largely thanks to collaborative projects and open-source efforts producing high-quality fonts for these scripts. It’s a major shift from the early 2010s, when web fonts were almost exclusively developed for the Latin alphabet. Now, developers can reasonably expect to find web fonts for a variety of writing systems. The data especially highlights Japanese and Korean adoption of web fonts, which continues to grow due to font families like Noto. Chinese remains a more difficult problem due to file size, but perhaps techniques like Incremental Font Transfer (IFT)—which is a way to load only needed portions of fonts on the fly—might change that in the future (this is on the horizon and could be a breakthrough for huge fonts, though still experimental).

- **Variable fonts:** Once a cutting-edge concept, variable fonts are now mainstream. Around 40% of websites use them, often transparently via Google Fonts serving a variable file instead of multiple static files. They haven’t revolutionized design yet in terms of fancy animations or continuous variability in content, but they have delivered practical benefits: simplifying font requests and sometimes reducing total bytes when multiple styles are needed. Many popular fonts are now variable by default (Roboto, etc.), and developers use them mostly like they’ve used multiple weights before—albeit with the bonus of more fine-grained control if desired.

- **Color fonts:** Despite growing support for color fonts (COLR/CPAL, SVG glyphs), they are still rare on the web. Aside from the use of emoji, few sites have taken advantage of multicolor glyphs. It remains an area to watch, but as of 2025, color fonts have yet to find a foothold in general web use.

- **CSS for fendering:** More sites are using the available CSS to optimize font loading and rendering. `font-display` is now specified on the majority of sites using fonts, reflecting a conscious choice about FOIT/FOUT. The dominant use of `swap` shows a shared priority for quick text appearance, with only icon fonts intentionally delaying for integrity. Resource hints like `preconnect` and `preload` are not universal, but a healthy minority employ these techniques, indicating performance-aware development practices on the rise. And while still niche, the adoption of properties like `text-wrap: balance` and `hyphens: auto` shows that even finer typographic tweaks are gradually gaining traction for improving text layout. In essence, web developers are more actively managing how fonts load and how text flows, rather than leaving everything to default. This is a sign of the ecosystem maturing, where front-end developers are treating typography with the same seriousness as other performance and UX aspects.

Overall, web fonts in 2025 tell a story of convergence and refinement. We see convergence on formats (WOFF2 everywhere), on delivery patterns (self-hosting on the rise, though services still important), on widely used families (a handful of web fonts are practically core now), and on best practices (`font-display: swap` , preloading critical fonts, etc.). Refinements include better global language support, open-type feature usage, and new CSS capabilities to smooth out the rough edges of text display. So, the web font landscape is largely stable, but not stagnant. No single change grabs the headline, but collectively these developments make the web more polished and accessible.

What might be the next major development in fonts? As noted, Incremental Font Transfer (IFT) is a very promising new technology. It would allow browsers to download just the needed glyphs of a font for a specific set of text, rather than frontloading the entire file. If standardized and adopted, IFT could be revolutionary for huge fonts like CJK, emoji, or icon sets—effectively solving the “font size” problem. This technology is still in development, but next year’s Web Almanac may have data available to report on the adoption and impact of IFT.

And so, the 2025 chapter closes on a confident note: fonts are a fundamental building block of the web, used smartly by most developers, and the path ahead is about making them faster, more inclusive, more expressive, and easier to work with for everyone.

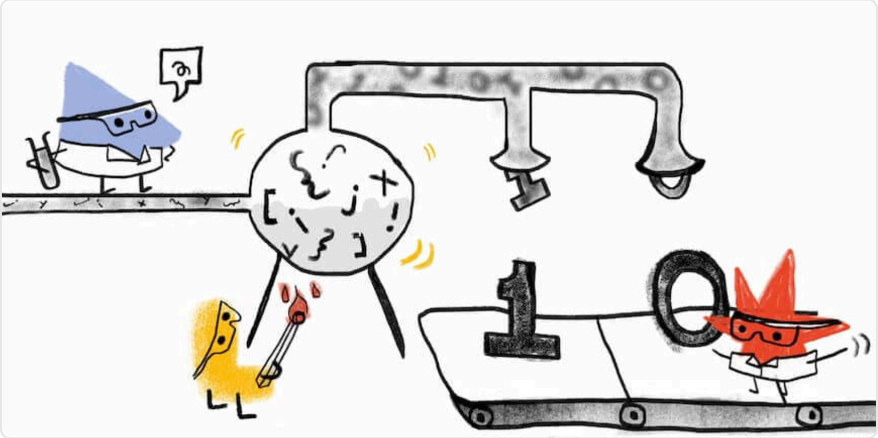
Author

Charles Berret

charlesberret https://charlesberret.net/

Charles Berret is a journalist, developer, and media scholar who studies the history and philosophy of information technologies. He lives in New York City and works at Enigma finding stories in data.

Part I Chapter 2
WebAssembly



Written by Nimesh Vadgama - VN
Reviewed by Nurullah Demir and Barry Pollard
Analyzed by Nimesh Vadgama - VN
Edited by Barry Pollard

Introduction

WebAssembly is no longer just a “web” technology; it has evolved into a high-performance, universal bytecode format. Officially a W3C standard since 2019[1], the ecosystem reached a major milestone with the release of Wasm Version 3.0 in December 2025[2], marking significant growth in both browser-based and standalone environments.

The mission

To provide a secure-by-default foundation for running code across any infrastructure—from cloud and edge computing to blockchain and IoT devices. Supported by industry leaders like Google, Microsoft, Fastly, and Intel, Wasm is redefining software portability.

[1] *https://www.w3.org/TR/2019/REC-wasm-core-1-20191205/*
[2] *https://webassembly.github.io/spec/core/*

The ecosystem at a glance

WebAssembly is managed largely by the Bytecode Alliance and CNCF, the ecosystem consists of:

- **Runtimes**: High-performance engines like *Wasmtime* (server-side) and *WasmEdge* (cloud-native/edge), alongside native integration in all major browsers.
- **WASI (WebAssembly System Interface)**: A standardized API (akin to POSIX) that allows modules to securely interact with filesystems and networks.
- **Component Model**: An emerging standard enabling different languages to interoperate within a single modular application.
- **Toolchains**: Mature compilers like *Emscripten* and cargo-component that bridge high-level source code to Wasm bytecode.

In essence, Wasm is moving beyond the browser to provide a secure, high-performance, and portable universal runtime for modern computing.

Methodology

We follow the same methodology from the 2021 Web Almanac[3], which is the first edition that included WebAssembly. WebAssembly is compiled into bytecode and distributed as binary format. The findings here infer the source language used in the modules. The accuracy of that analysis is covered in more detail in the respective sections.

Limitations

- **Data Acquisition & Retrieval** Our analysis relies on the almanac-wasm tool to fetch WASM binaries using parameters from the initial HTTP Archive crawl. However, retrieval depends on third-party server availability; 404/403 errors or resource updates since the initial crawl may occur. We assume the retrieved file is identical to the one present during the original recording.

- **Source Language Identification** To identify source languages, we utilize the

[3] *https://almanac.httparchive.org/en/2021/webassembly#methodology*

WebAssembly Binary Toolkit (`wasm2wat`) (see the also MDN guide[4]) and Rust-based parsers to analyze imports, exports, and custom sections for compiler metadata and language fingerprints.

- **Limitations**: This is not a definitive approach. "Stripped" binaries (using tools like *wasm-strip*) or obfuscated code remove the debug info and metadata required for identification.
- **Example**: Compilers like TinyGo generate unique structures and WASI imports that often diverge from standard *go_exec.js* patterns, resulting in these binaries being classified as "Unknown".

Feature detection constraints

- **No Runtime Check:** WebAssembly lacks a built-in instruction for modules to detect host features from within the sandbox; detection is handled by the host environment (for exmaple, JavaScript or Wasmtime).
- **Compile-time Dependency:** Advanced features like SIMD or Threads are enabled via compiler flags. If the host environment does not support these specific features at instantiation, the binary will fail to load.

WebAssembly usage

0.35%

Figure 2.1. Desktop sites using WebAssembly.

We see that 0.35% of desktop sites and 0.28% of mobile sites are using WebAssembly. This is approximately 43,000 sites in our dataset for both, but with a larger dataset for mobile the relative percentage is lower.

[4] https://developer.mozilla.org/docs/WebAssembly/Guides/Text_format_to_Wasm

Year-on-year trend

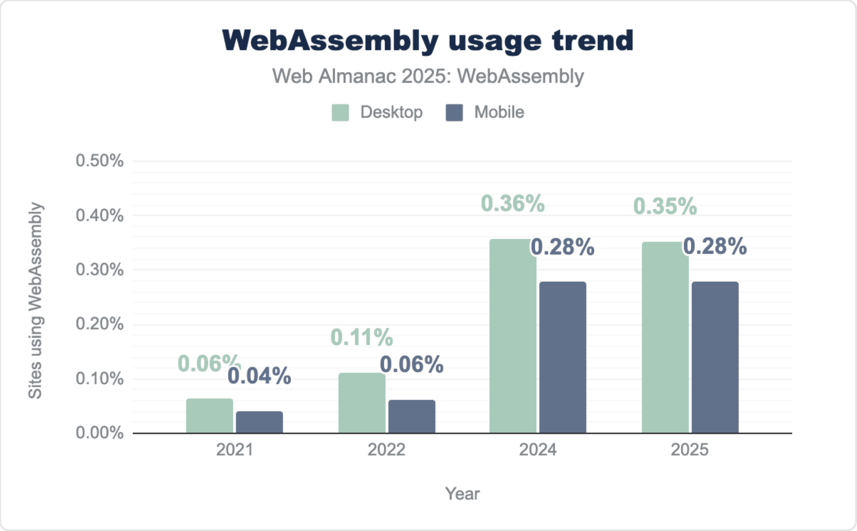


Figure 2.2. WebAssembly usage trend

Interestingly, looking over previous Web Almanac years, WebAssembly requests have increased drastically from ~0.05% in 2021 to 0.28%/0.35% this year. There was also a slight dip from 2024.

WebAssembly by rank

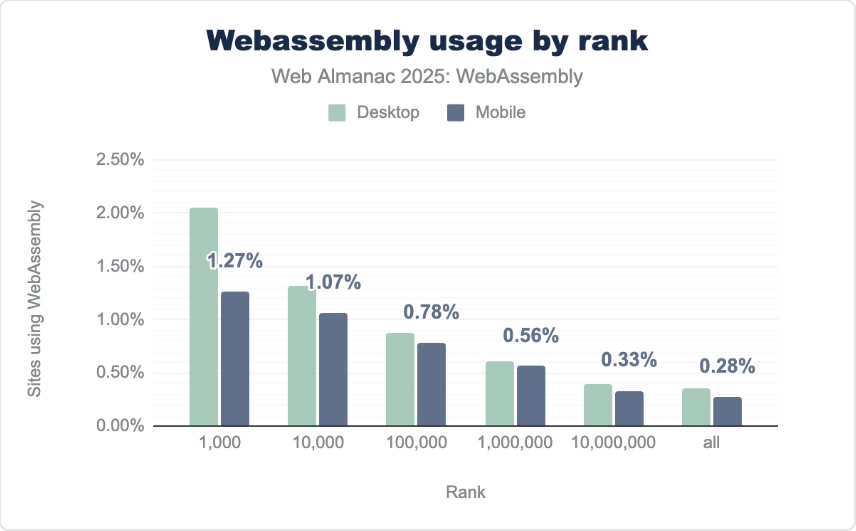


Figure 2.3. Page Ranking (Group) for Web Assembly 2025

We see a clear "Power Law" in WebAssembly adoption: more popular sites are more likely it is to use Wasm.

Key Insights from the Page Ranking Data:

- **Top-Tier Dominance**: Wasm is most prevalent among the top 1,000 sites mobile. These are typically "heavy" applications like Figma, Adobe, or Google Meet that require high performance for complex tasks.

- **The "Long Tail" Effect**: While the percentage of adoption drops significantly as you move down there ranks, there is still usage across all ranks.

- **Platform Parity**: There is a very close correlation between desktop and mobile usage across all ranking groups. This confirms that Wasm is being used as a cross-platform solution rather than being restricted to desktop-only environments.

High-ranking sites are more likely to be complex web apps that need Wasm for performance, while lower-ranking sites (like simple blogs or small business pages) have not yet found a massive need for it—though they often use it "silently" via third-party libraries.

WebAssembly requests

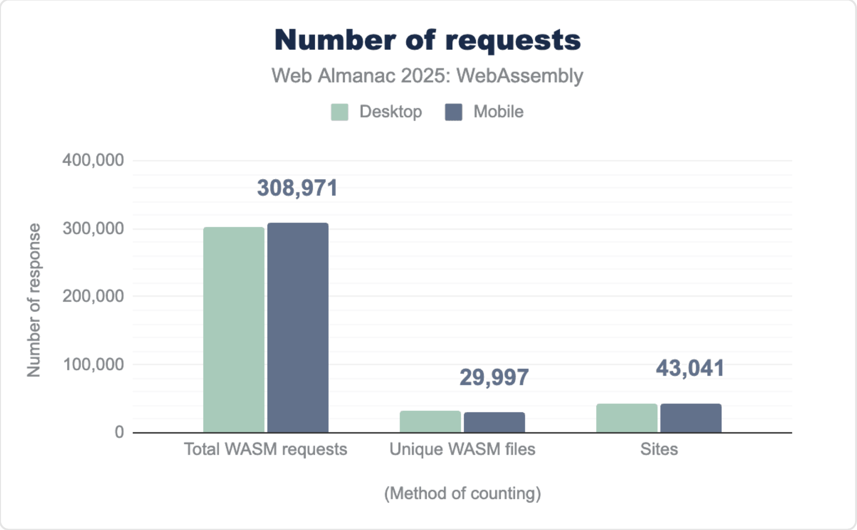


Figure 2.4. Number of WASM requests

There are over 300,000 wasm requests in our dataset, which come down to 32,197 unique wasm file requests on desktop and 29,997 on mobile. The large number of requests shows many sites are requesting multiple (hundreds in come cases!) WASM files.

MIME type

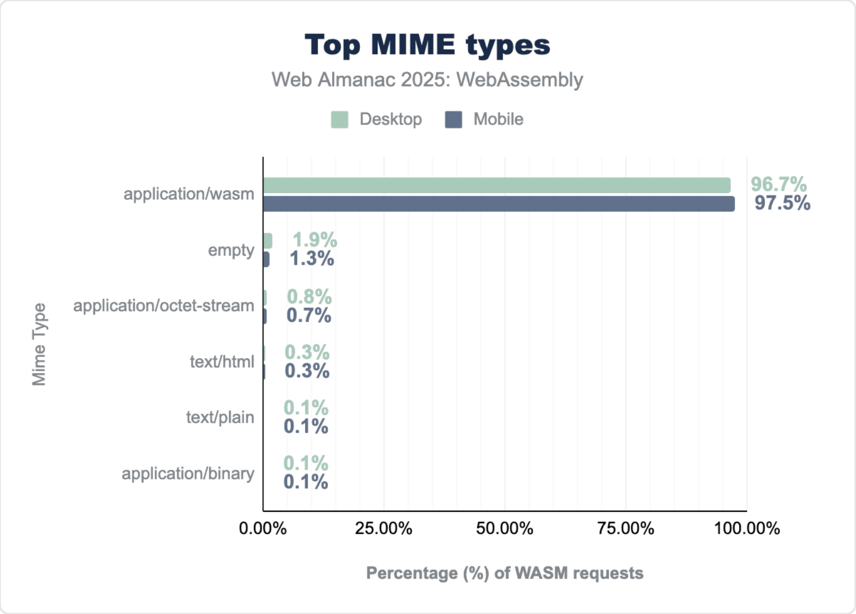


Figure 2.5. Top MIME types

We observed that the MIME type `application/wasm` is used in 293,470 requests on desktop clients and 301,127 requests on mobile clients.

Some requests lacked a `Content-Type` header, and some had incorrect MIME types, such as `text/html` or `text/plain`. These account for 3.2% and 2.4% of requests, respectively. Compared to 2022, these percentages have dropped significantly, indicating increased awareness in setting the correct MIME type for WASM applications.

Module size

The smallest WebAssembly modules are likely used for specific functions, such as "Micro-Utility" like *Base64 Encoder/Decoder* or a *CRC32 Checksum* utility—These are typically used for performance-critical calculations or polyfills where JavaScript might be too slow or lack the specific precision needed.

Larger modules are probably full applications compiled to WebAssembly. These are massive modules where an entire desktop-grade codebase (often millions of lines of C++ or Rust) is compiled to run in the browser. For example, Adobe Photoshop Web, AutoCAD, and Google

Earth. Large modules handle complex image rendering, layer management, and 3D engine calculations directly on the client side browser.

Small modules of WebAssembly often found using AssemblyScript or Rust and Large modules of WebAssembly often found using languages like C++ or C# (.net). You can get more details on the Wasm Language Usage section.

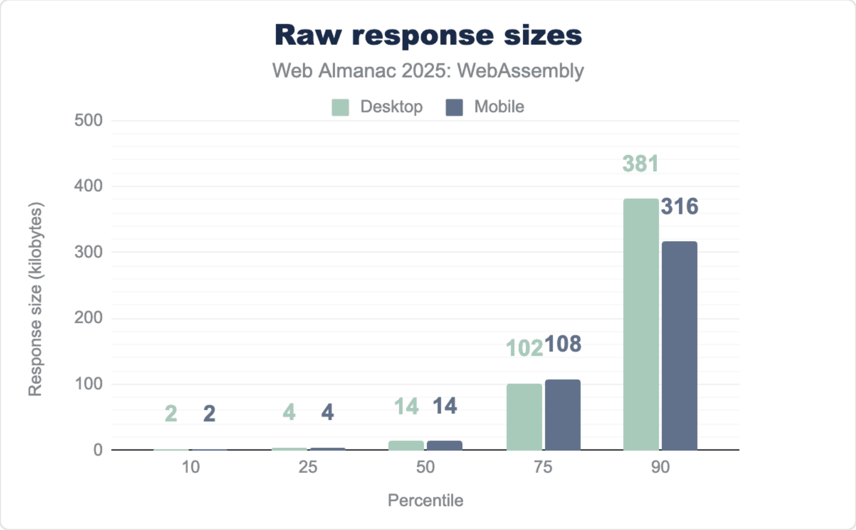


Figure 2.6. Raw response sizes.

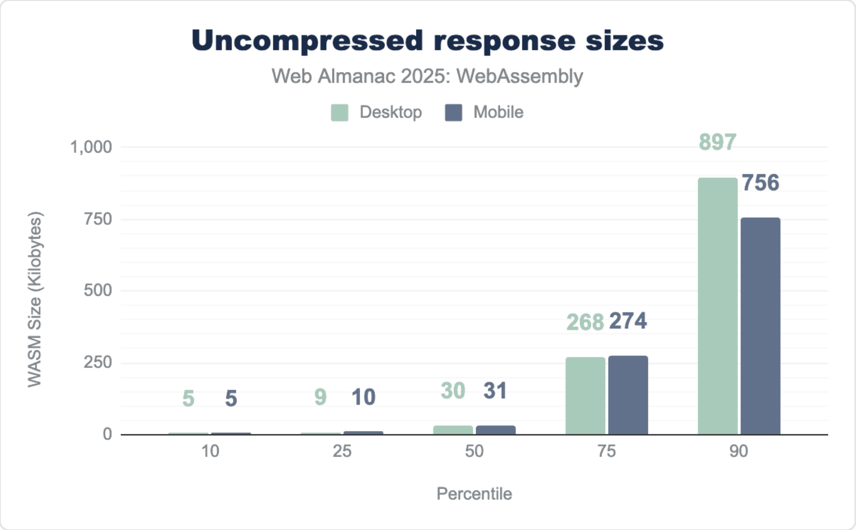


Figure 2.7. Uncompressed response sizes.

These WebAssembly modules differ considerably in size, with the smallest being just a few kilobytes, and the largest one is 228.102 MB in desktop's client and 166.415 MB for mobile client.

228 MB

Figure 2.8. Largest WebAssembly file detected.

WebAssembly libraries

Our crawl identified a modest number of modules, it is possible to analyze and learn about the most popular libraries in requests for wasm.

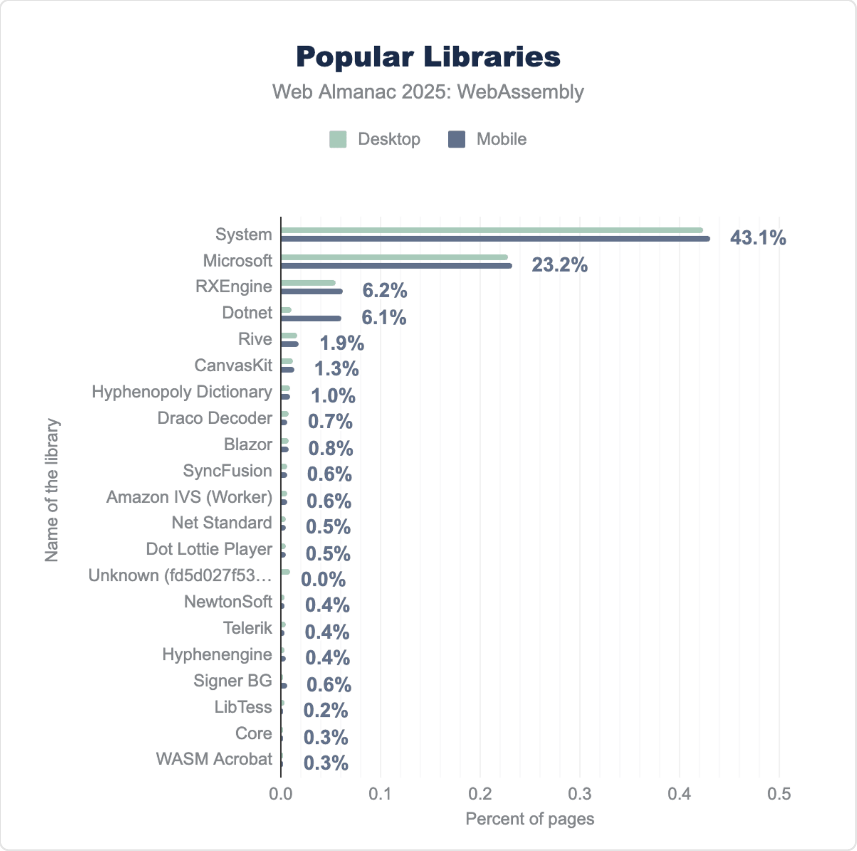


Figure 2.9. Popular WebAssembly libraries.

Let’s look a bit more into the top three libraries:

- **Library : System (43.1%)** is used for fundamental “glue” code and It often includes “system-level” bindings (like those from WASI or Emscripten) that allow a Wasm module to communicate with the host environment (the browser) to handle tasks like memory management or basic I/O. system utilities.

- **Library : Microsoft (23.2%)** represents the massive footprint of the Microsoft ecosystem on the web, primarily driven by Blazor WebAssembly. Blazor allows developers to build interactive web UIs using C# and .NET instead of JavaScript.

The high percentage reflects many enterprise and business applications that have been ported to the web using Microsoft's specialized Wasm runtime for the .NET framework.

- **Library : RXEngine (6.2%)** is a more specialized entry, often associated with high-performance execution engines used for specific industries like gaming or advanced data processing. While more niche than the top two, its 6.2% share indicates it is a popular choice for developers who need a pre-built, optimized engine to handle computationally intensive tasks (such as real-time analytics or complex UI interactions) without building the entire infrastructure from scratch.

WebAssembly languages

WebAssembly can use various languages and using toolchains It can be compiled in binary format to server browser and desktop applications. It can carry much of the information in the source (programming language, application structure, variable names).

Each WebAssembly has import and or export components, Most WebAssembly toolchains create a small amount of JavaScript code, for the purposes of 'binding', making it easier to integrate components into JavaScript applications. These bindings often have recognisable function names which are present in the components exports or imports, giving a reliable mechanism for identifying the language that was used to author the component.

If WASM has not used obfuscation and or other techniques that are stripped down while building deliverable then we can use the rust libraries and WebAssembly Binary Toolkit (WABT) to understand the source programming language.

We enhanced the wasm-stats project and created tool `almanac-wasm` that helps to download wasm file from the 3rd party server with preferred request parameters for example user-agent, compression method etc, validates the downloaded wasm file and with rust library and WABT (wasm2wat), It finds the author's language and populates wasm statistics along with language.

For example, wasm-bindgen[5] is a suite of tools that helps to generate high level code Rust-compiled WebAssembly (Wasm) component and JavaScript with name as "wbindgen" so If We import the component from WebAssembly and find "wbindgen" then there is clearly indication that component in WebAssembly was written in Rust language.

[5] *https://crates.io/crates/wasm-bindgen*

Like wise, We have researched and found various language indicators inside WebAssembly's different components with the tool.

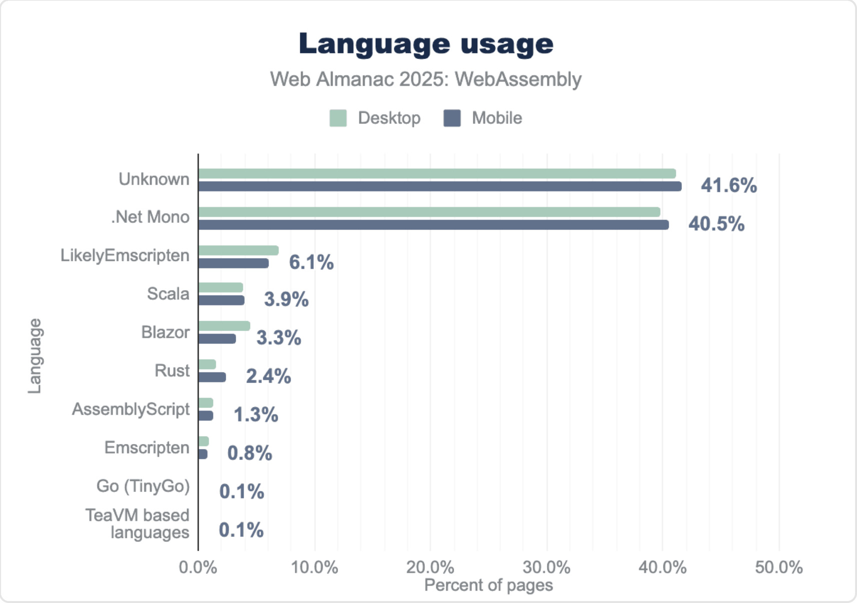


Figure 2.10. WebAssembly language usage.

We have found the .Net / Mono eco system based languages (including Blazor) reserves the first position for the language used in WebAssembly for desktop clients with 40.5% and for mobile clients with 39.8%

However 41.1% clients in desktop and 41.6% clients in mobile have language "Unknown" that means We could not find the author's (source) language because of missing language indicators or WebAssembly is stripped with the obfuscation or other techniques; These techniques are used to reduce the size, enable privacy/security features or to optimize the performance of the WebAssembly by the modern compilers.

WebAssembly features

The initial release of WebAssembly was considered an Minimal Viable Product (MVP). In common with other web standards, it is continually evolving under the governance of the World Wide Web Consortium (W3C). This year saw the announcement of the WebAssembly version 2.0, adding a number of new features. The version 3.0 shows the true vision and its potential for WebAssembly. You can explore the various additional features on features page[6].

Single Instruction, Multiple Data (SIMD)[7] instructions are a special class of instructions that exploit data parallelism in applications by simultaneously performing the same operation on multiple data elements. Compute intensive applications like audio/video codecs, image processors, are all examples of applications that take advantage of SIMD instructions to accelerate performance. Most modern architectures support some variants of SIMD instructions.

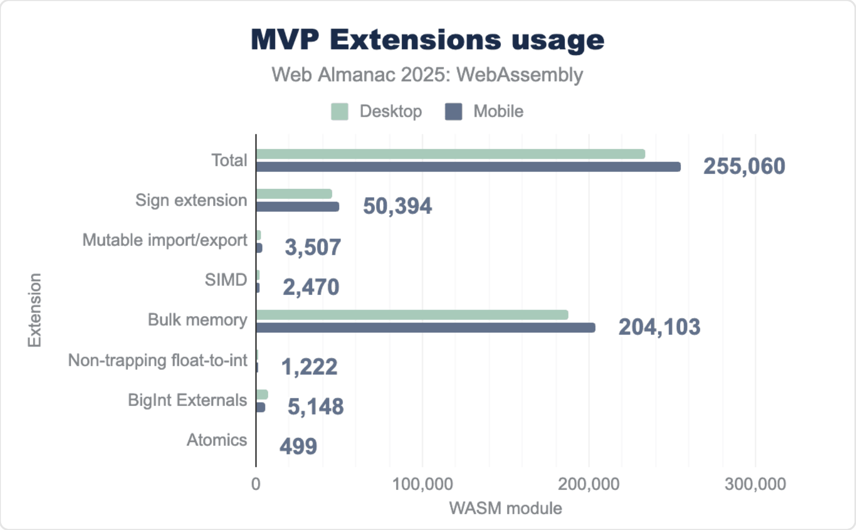


Figure 2.11. Post-MVP extensions usage.

SIMD is a new feature and isn't yet available in all browsers with WebAssembly support. In the year 2021, We had found SIMD extension usage in 20 Wasm modules on desktop and 21 Wasm modules on mobile clients, This feature usage is now increased with 2265 Wasm modules on desktop and 2,470 on mobile clients.

Other exciting features include Sign-extension-ops[8] and Non-Trapping float to int[9].

With respect to the total extension usage in year 2021, It is observed that total extension usage in 2025 drastically ~61 times more on desktop clients and ~80 times more on mobile based clients. To make the usage of very complex tasks, the WebAssembly, We have marked the Bulk Memory stats are increased to 8936 times higher on desktop and 25,512 times higher on mobile clients with respect to year 2021 stats.

[6] *https://webassembly.org/features/*
[7] *https://v8.dev/features/simd*
[8] *https://github.com/WebAssembly/sign-extension-ops/tree/master*
[9] *https://github.com/WebAssembly/nontrapping-float-to-int-conversions*

Conclusions

There is a significant increase in the number of webpages using this technology for serverless, containerization, machine learning components and plug-n-play types of applications. The future of WebAssembly could be as a niche web technology, but as an entirely mainstream runtime on a wide range of other platforms. WebAssembly runtime (multi-language, lightweight, secure) are making it a popular choice for a wider range of non-browser applications for agnostic platforms.

Despite being a niche technology, WebAssembly is already adding value to the web. There are a number of web applications that benefit greatly from this technology. However, web applications are often not visible to the ‘crawl’ which forms the basis of this study.

Author

Nimesh Vadgama - VN

nimeshgit ops-ml-architect https://ops-ml-architect.blogspot.com/

Nimesh provides digital transformation and automation solutions, with a focus on AI and ML analytics, operations, and business processes.

Part I Chapter 3

Third Parties

Written by Muhammad Jazlan and Muhammad Abu Bakar Aziz
Reviewed by Barry Pollard
Analyzed by Muhammad Jazlan
Edited by Barry Pollard

Introduction

Third parties are ubiquitous on the web. Website developers rely on them to implement key features such as advertising, analytics, social media integration, payment processing, and content delivery. This modular approach enables efficient and rapid deployment of rich functionality. However, it introduces potential privacy, security, and performance concerns. New this year, we analyze how user consent choices are propagated among third parties on the web, including the consent frameworks used and the third parties that receive these signals.

In this chapter, we conduct an empirical analysis of third-party usage patterns on the web. We examine:

- **Prevalence:** How many websites use third parties and in what proportions
- **Resource types:** The forms third parties take (images, JavaScript, fonts, etc.)

- **Functional categories:** Ad networks, analytics, CDNs, video providers, tag managers, and others
- **Integration methods:** How third parties are loaded directly or indirectly on pages
- **Consent infrastructure:** Which third parties transfer consent signals and how those transmissions happen in practice

Definitions

First, we establish some definitions and terminology that are used throughout our analysis.

Sites and pages

In this chapter, like previous years, we use the term site to depict the registerable part of a given domain which is often referred to as *extended Top Level Domain plus one* (eTLD+1). For example, given the URL `https://www.bar.com/` the eTLD+1 is `bar.com` and for the URL `https://foo.co.uk` the eTLD+1 is `foo.co.uk`. By page (or web page), we mean a unique URL or, more specifically, the document (for example HTML or JavaScript) located at the particular URL.

What is a third party?

We stick to the definition of a third party used in previous editions of the Web Almanac to allow for comparison with earlier versions.

A *third party* is an entity different from the site owner (also known as the first party). It involves the aspects of the site not directly implemented and served by the site owner. More precisely, third-party content is loaded from a different site rather than the one originally visited by the user. Assume that the user visits `example.com` (the first party) and `example.com` includes silly cat images from `awesome-cats.edu` (for example using an `<img>` tag). In that scenario, `awesome-cats.edu` is the third party, as it was not originally visited by the user. However, if the user directly visits `awesome-cats.edu`, `awesome-cats.edu` is the first party.

For our analysis, only third parties originating from a domain whose resources can be found on at least five unique pages in the HTTP Archive dataset were included.

When third-party content is directly served from a first party domain, it is counted as first party content. For example, self-hosted analytics scripts, CSS, or fonts are counted as first party content. Similarly, first-party content served from a third-party domain is counted as third-party content. Some third parties serve content from different subdomains. However, regardless of the number of subdomains, they are counted as a single third party.

Further, it is becoming increasingly common for third parties to be masqueraded as a first party. Two key techniques enable this:

- **CNAME cloaking** involves using a CNAME record to make a third party's content appear to come from the first party domain. We consider CNAME-cloaked services as first parties in this analysis.

- **Server-side tracking** is an emerging trend where the site owner embeds the tracker as a first party and routes all requests through the first party domain, making the tracker appear as a first party. For example, a website `www.example.com` may embed server-side Google Tag Manager with Google Analytics and cloak the subdomain `sst.example.com` to send requests to a Google Tag Manager container. In this way, requests to third parties originate from the tag manager's server rather than the user's browser.

In our analysis, we treat such cases as first party interactions because the third party communication occurs server-to-server and is not directly observable in the client-side HTTP Archive data. As a result, our measurements represent a lower bound on the actual prevalence of third parties on the web.

Categories

As previously indicated, third parties can be used for various use cases—for example, to include videos, to serve ads, or to include content from social media sites. Similar to the previous year, to categorize the observed third parties in our dataset, we rely on the Third-Party Web[10] repository from Patrick Hulce[11]. The repository breaks down third parties along the following categories:

- **Ad:** These scripts are part of advertising networks, either serving or measuring.

- **Analytics:** These scripts measure or track users and their actions. There's a wide range of impact here, depending on what's being tracked.

[10] *https://github.com/patrickhulce/third-party-web/#third-parties-by-category*
[11] *https://x.com/patrickhulce*

- **CDN:** These are a mixture of publicly hosted open source libraries (for example jQuery) served over different public CDNs and private CDN usage.
- **Content:** These scripts are from content providers or publishing-specific affiliate tracking.
- **Customer Success:** These scripts are from customer support/marketing providers that offer chat and contact solutions. These scripts are generally heavier in weight.
- **Hosting:** These scripts are from web hosting platforms (WordPress, Wix, Squarespace, etc.).
- **Marketing:** These scripts are from marketing tools that add popups/newsletters/etc.
- **Social:** These scripts enable social features.
- **Tag Manager:** These scripts tend to load many other scripts and initiate many tasks.
- **Utility:** These scripts are developer utilities (API clients, site monitoring, fraud detection, etc.).
- **Video:** These scripts enable video player and streaming functionality.
- **Consent provider:** These scripts allow sites to manage the user consent (e.g. for the General Data Protection Regulation[12] compliance). They are also known as the *Cookie Consent* popups and are usually loaded on the critical path.
- **Other:** These are miscellaneous scripts delivered via a shared origin with no precise category or attribution.

`Content-Type`

We use the `Content-Type` HTTP header to categorize third-party resources into different types, such as scripts, HTML content, JSON data, plain text, and images. This allows us to analyze the composition of third-party resources served across websites.

[12] *https://wikipedia.org/wiki/General_Data_Protection_Regulation*

Prevalence

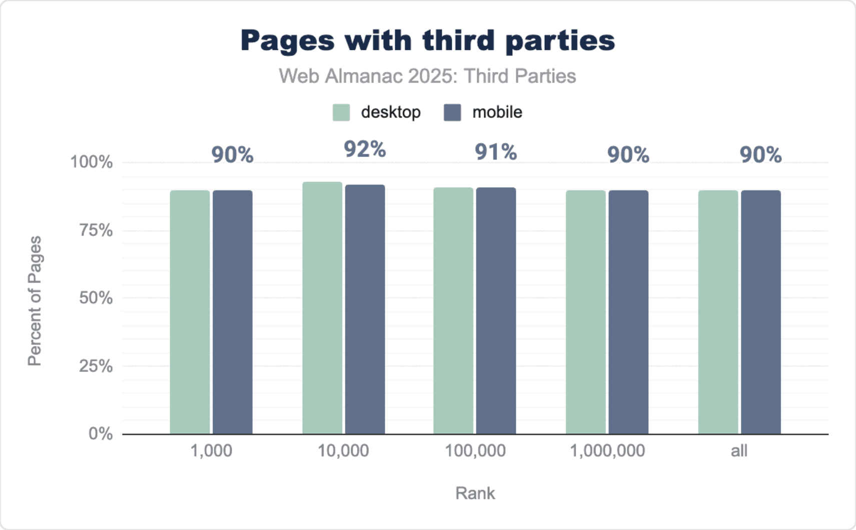


Figure 3.1. Percentage of pages that use one or more third parties.

Compared to the previous year[13], we observe a slight decrease in the percentage of pages that use one or more third parties across websites. However, despite this decrease, the percentage of pages with one or more third parties remains greater or equal to 90%.

[13] https://almanac.httparchive.org/en/2024/third-parties#prevalence

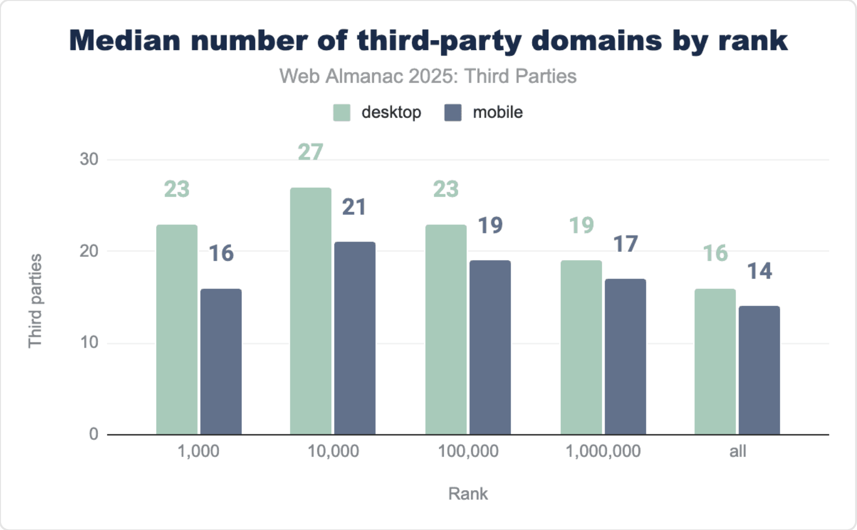


Figure 3.2. Distribution of the number of third parties by rank.

Compared to the previous year, we observe a significant decrease in the median number of third-party domains across all website ranks, with a particularly large decrease among low-ranked websites.

This decline may be due to several factors. First, third parties are increasingly obscured through `CNAME` cloaking and server-side tracking, which can reduce their visibility in client-side measurements. Second, HTTP Archive crawlers do not interact with web pages or scroll down the page, which may prevent some third parties from loading properly due to lazy loading. As a result, fewer third-party requests may be observed.

We also observe that desktop pages generally include more third parties than mobile pages.

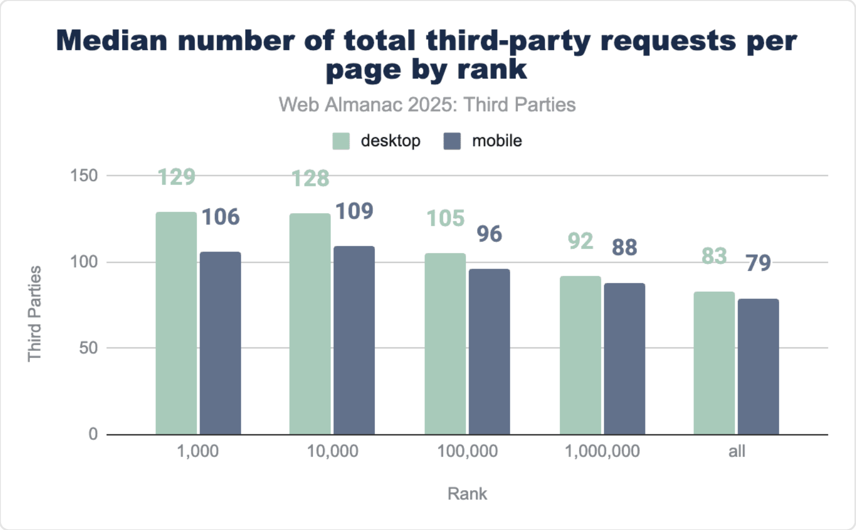


Figure 3.3. Distribution of the number of third party requests per page by rank.

Low-ranked websites load more third-party requests. The top 1,000 have a median of 129 requests on desktop and 106 on mobile, compared to 83 on desktop and 79 on mobile across all sites.

Year-over-year, third-party requests have increased across all ranks. The top 1,000 sites show an increase of 15 requests on desktop and 15 on mobile compared to 2024[14], while the broader dataset increased by five requests on desktop and five on mobile. This upward trend occurs despite the decrease in the number of unique third-party domains we observed earlier, suggesting that individual third parties are sending more requests per page.

[14] *https://almanac.httparchive.org/en/2024/third-parties#fig-3*

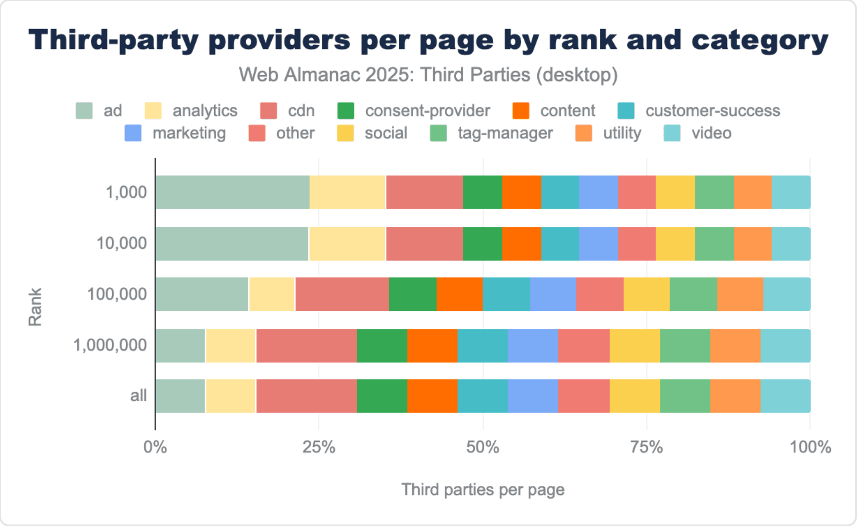


Figure 3.4. Distribution of the third-party request categories by rank.

The bar chart shows the median number of third-party providers per page by rank and category. In the previous edition, this analysis focused on the number of third-party domains per page by rank and category, whereas this year we measure the number of unique third-party providers, which results in lower counts overall. This year, the top categories are `ad`, `analytics`, and `cdn`.

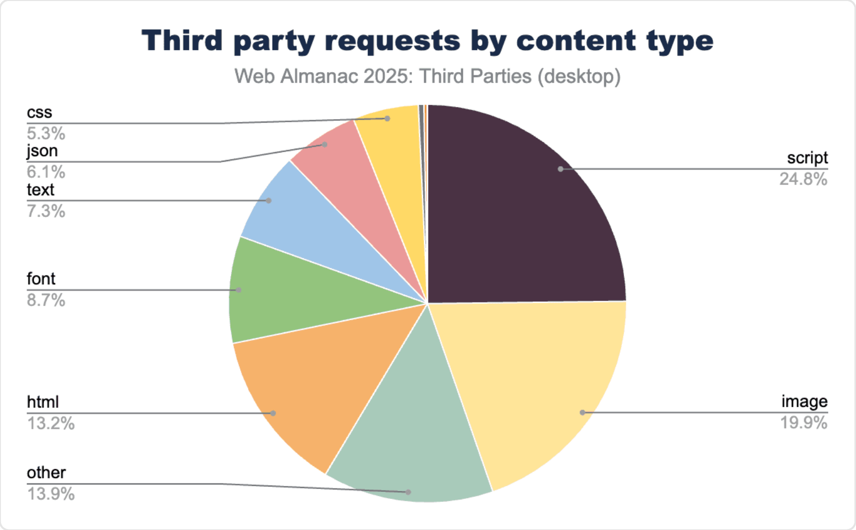


Figure 3.5. Distribution of the third-party request types by rank.

The chart shows that third-party requests are dominated by `script`, `image`, and the `other` category. Together, `script`, `image`, and `other` account for more than half of all third-party request content types. This pattern is consistent with the 2024 edition[15], which also identified `script`, `image`, and `other` as the top request types, indicating little change since last year.

[15] https://almanac.httparchive.org/en/2024/third-parties#fig-5

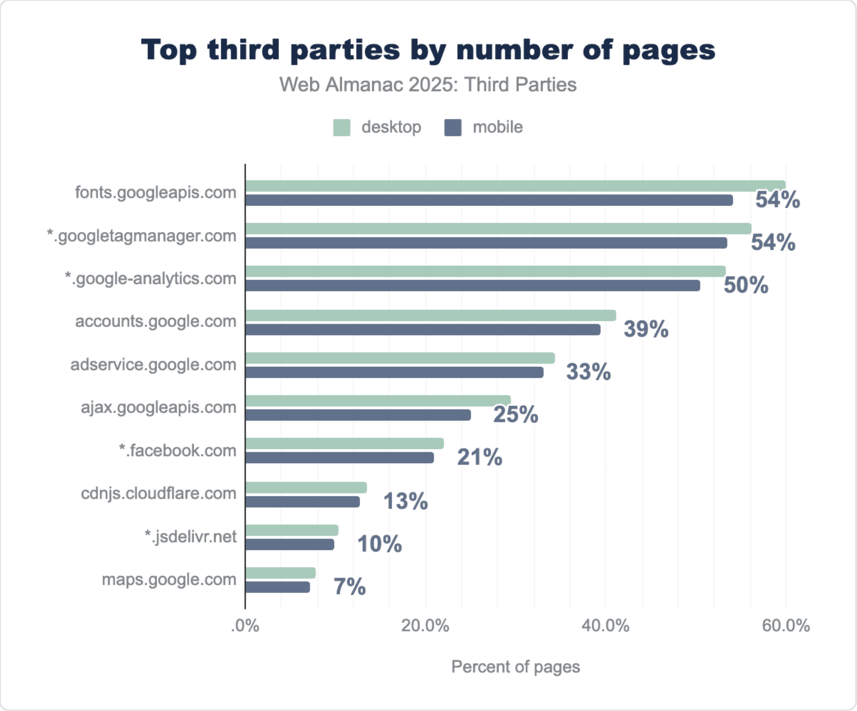


Figure 3.6. Top third parties by the number of pages.

The top 10 third-party domains are dominated by Google-owned services, including `fonts.googleapis.com`, `googletagmanager.com`, `google-analytics.com`, `accounts.google.com`, and `adservice.google.com`. Meta's `facebook.com` is the only non-Google domain in the top 10, appearing at rank 7 with 21% of pages.

Consent propagation among third parties

In this section, we examine how different third parties transmit user consent across the web. Previous research[16] has shown that third parties often rely on industry-standard frameworks to communicate consent information. In our analysis, we focus primarily on the IAB's three consent standards: the Transparency and Consent Framework (TCF)[17], the CCPA Framework[18], and the Global Privacy Protocol (GPP)[19].

[16] *https://petsymposium.org/popets/2024/popets-2024-0120.pdf*
[17] *https://iabeurope.eu/transparency-consent-framework/*
[18] *https://iabtechlab.com/standards/ccpa/*
[19] *https://iabtechlab.com/gpp/*

These frameworks define how consent information is encoded and shared between websites and third parties. We begin by identifying which consent standards are most prevalent among the third parties observed in our dataset. To determine which framework a third party uses, we rely on the presence of specific parameters in the request URLs. Details of the different standards are below:

- **TCF Standard**: We identify use of the TCF framework by checking whether a third-party request includes the `gdpr` or `gdpr_consent` parameters, as specified by the IAB TCF.

- **GPP Standard**: We identify use of the GPP framework by checking for the presence of the `gpp` and `gpp_sid` parameters.

- **USP Standard and non-USP Standard**: We identify use of the USP Standard by checking whether a request transmits a `us_privacy` parameter, as defined by the IAB CCPA Framework. We also identify use of the non-standard USP Standard by detecting consent strings transmitted via non-standard parameters identified in the prior work[20].

We analyze consent signal prevalence across website ranks, third-party categories, and the most frequently observed consent-receiving third parties.

[20] *https://petsymposium.org/popets/2024/popets-2024-0120.pdf*

Prevalence of consent signals across different ranks

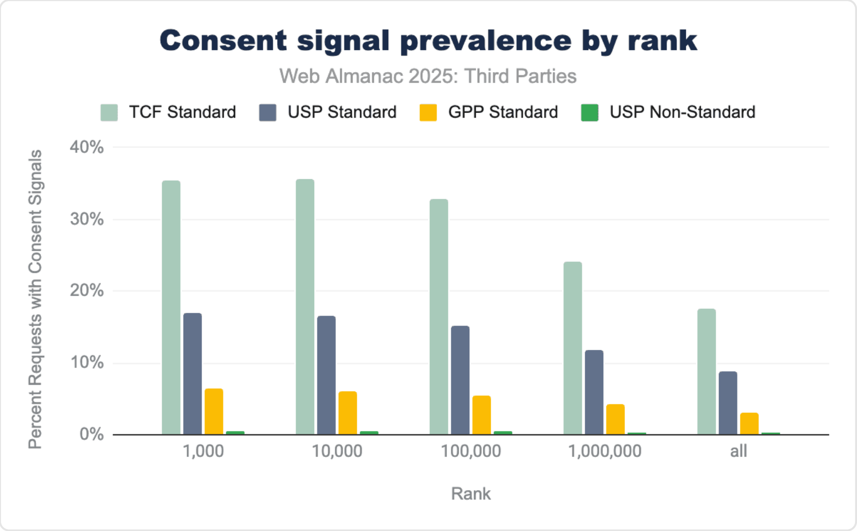


Figure 3.7. Consent signal prevalence by rank.

We find that TCF Standard is the dominant consent standard, particularly among low-ranked sites where it reaches 36% compared to 18% across all sites. This higher adoption aligns with stronger opt-in consent requirements under GDPR. The USP Standard is the second most prevalent, with adoption ranging from 9–17% across ranks. This reflects use of the IAB CCPA consent framework introduced in response to the CCPA. GPP adoption remains minimal at 3–6%, despite its goal to unify consent frameworks across jurisdictions.

Consent standard distribution across different categories

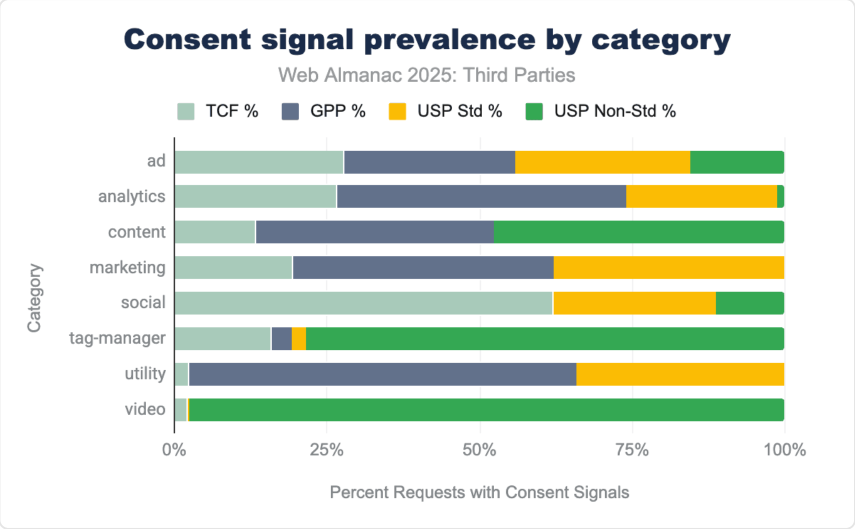


Figure 3.8. Consent signal prevalence by category.

We observe different consent standard preferences across different third-party categories. For example, Social services show the highest TCF adoption, while advertising vendors employ a more balanced mix of GPP, USP Standard, and smaller TCF shares. Furthermore, Analytics vendors predominantly adopt GPP.

Top third parties receiving consent

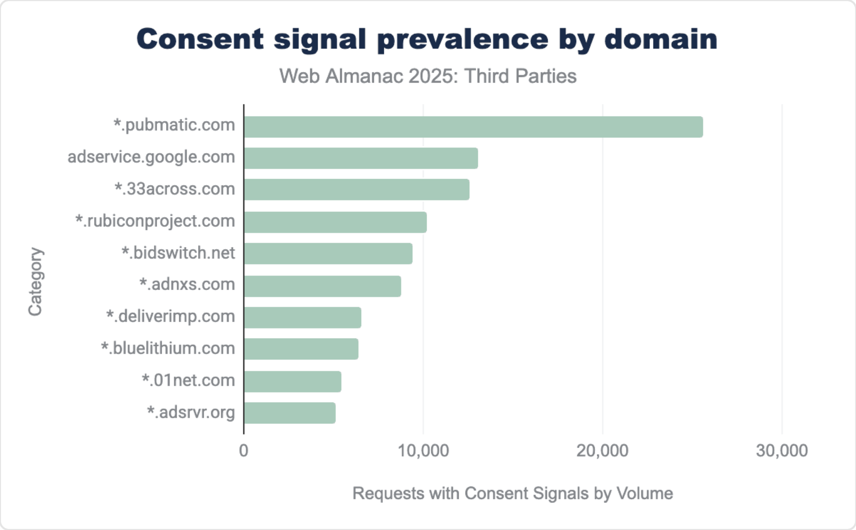


Figure 3.9. Consent signal prevalence by domain.

Among top-ranked websites, `pubmatic.com` receives the highest volume of consent signals, with `adservice.google.com` in second place. The majority of domains receiving the most consent signals are advertising and ad tech vendors—ad exchanges, DSPs, and ad servers. This makes intuitive sense, as in many jurisdictions third party advertising and analytics providers must obtain user consent before using user data for ads and other purposes.

Inclusion

Recall from our earlier example that `example.com` (a first party) can include an image from `awesome-cats.edu` (a third party via an `<img>` tag). This inclusion of an image would be considered direct inclusion. However, if the image was loaded by a third-party script on the site via the `XMLHttpRequest`, then the inclusion of the image would be considered indirect inclusion. The indirectly included third parties can further include additional third parties. For example, a third-party script that is directly included on the site may further include another third-party script. In this chapter, we do basic analysis of the depths of inclusion chains of the third parties.

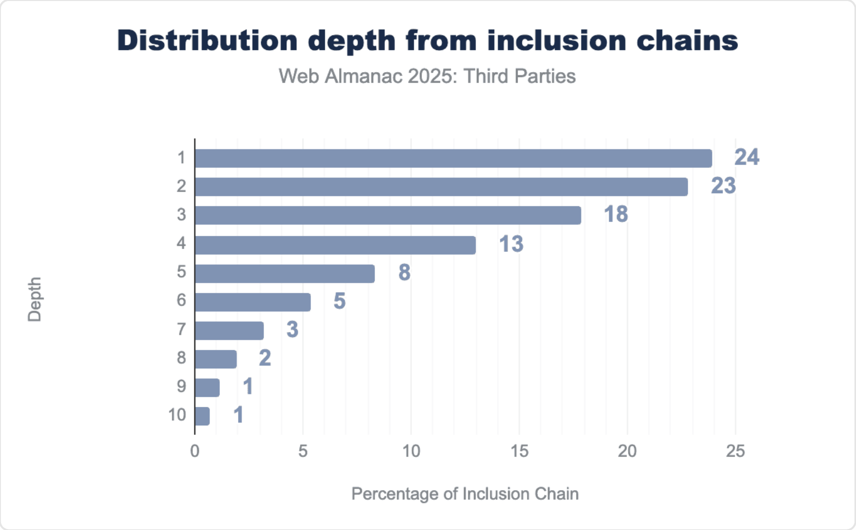


Figure 3.10. Median depth of third-party inclusion chains.

The median depth of the inclusion chain is 3 which means the majority of the third parties include at least another third party on a web page. The maximum depth of the inclusion chain is 2,285.

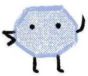
Conclusion

Our findings show the ubiquitous and increasingly concentrated nature of third parties on the web. More than nine-in-ten web pages include one or more third parties. While the median number of unique third-party domains has decreased compared to the previous year, we observe a significant increase in the total number of requests from third parties, suggesting individual vendors are sending more requests per page.

In terms of consent standards, TCF is the dominant consent standard across all website ranks. Among individual third parties, `pubmatic.com`, `adservice.google.com` and other ad tech domains receive the highest volume of consent signals.

Finally, the increasing use of obfuscation techniques such as CNAME cloaking and server-side tracking reduces visibility of third parties in client-side measurements, suggesting our findings represent a lower bound on actual prevalence.

Authors



Muhammad Jazlan

jazlan01

Muhammad Jazlan is a second year PhD student in Computer Science at University of California, Davis. His research focuses on the measurement, detection and mitigation of tracking on the web.

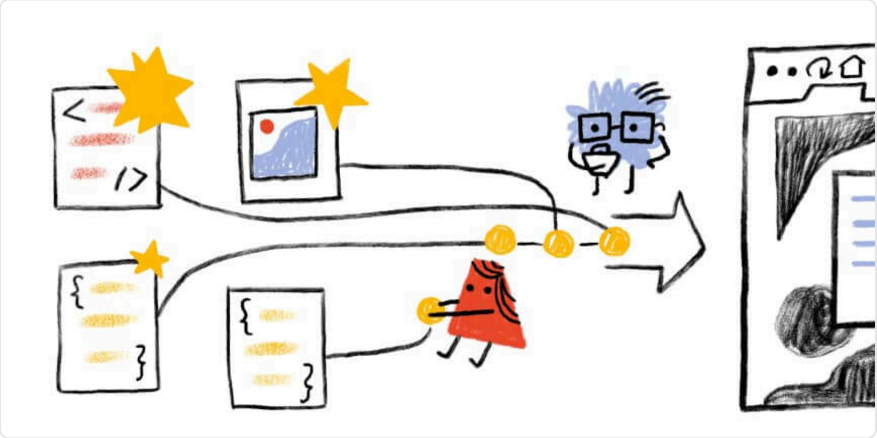
Muhammad Abu Bakar Aziz

abubakaraziz aziz313f

Muhammad Abu Bakar Aziz is a PhD candidate in Computer Science at Northeastern University in Boston. His research focuses on web privacy. In particular, he empirically measures how third parties and online advertisers comply with privacy laws such as the CCPA and GDPR.

Part I Chapter 4

Generative AI



Written by Christian Liebel, Yash Vekaria, and Jonathan Pagel
Reviewed by Thomas Steiner, Umar Iqbal, and Maxim Salnikov
Analyzed by Jonathan Pagel, Christian Liebel, and Thomas Steiner
Edited by Barry Pollard

Introduction

On November 30, 2022, OpenAI launched ChatGPT[21], a service that catapulted *Generative Artificial Intelligence* (Generative AI) from research labs into the daily lives of millions[22]. This launch fundamentally changed user expectations regarding how applications and the web should function. Furthermore, the accompanying *Application Programming Interface* (API) gave software developers a powerful tool to make their applications significantly smarter.

700,000,000

Figure 4.1. Weekly active users of ChatGPT.

[21] https://openai.com/index/chatgpt/
[22] https://openai.com/index/how-people-are-using-chatgpt/#:~:text=Given%20the%20sample%20size%20and%20700%20million%20weekly%20active%20users%20of%20ChatGPT

Generative AI is a specialized field that focuses on processing and generating human-understandable content, including text, source code, images, videos, speech, and music. *Large Language Models* (LLMs) represent a significant component of this field. Trained on vast amounts of textual data, LLMs interpret and generate natural language, expanding software architecture by enabling developers to process human language effectively for the first time. Recently, Generative AI features have been integrated into established applications, including Windows, Office, and Photoshop.

As Generative AI becomes increasingly widespread, this chapter examines its emerging trends on the web. Specifically, it focuses on the use of local Generative AI, enabled by "Built-in AI" and "Bring Your Own AI" approaches, the discoverability of Generative AI content via `llms.txt`, and the impact of Generative AI on content creation and source code (*AI fingerprints*).

Data sources

This chapter uses not only the dataset of HTTP Archive, but also npm[23] download statistics, and data provided by Chrome Platform Status[24] and other researchers.

Unless otherwise noted, we refer to the July 2025 crawl of the HTTP Archive, as described in Methodology. For detecting API adoption, we scanned websites for the presence of the specific API calls in code. Note that this only indicates their occurrence, but not necessarily actual usage during runtime. Usage data from Chrome Platform Status always refers to the percentage of Google Chrome page loads that use a certain API at least once, across all release channels and platforms.

Technology overview

In this section, we will explain the difference between cloud-based and local AI models, discuss the pros and cons of these approaches, and then examine local technologies in detail.

Cloud versus local

Most people use Generative AI through cloud services such as OpenAI[25], Microsoft Foundry[26], AWS Bedrock[27], Google Cloud AI[28], or the DeepSeek Platform[29]. Because these providers have

[23] *https://www.npmjs.com/*
[24] *https://chromestatus.com/*
[25] *https://openai.com/api/*
[26] *https://ai.azure.com/*
[27] *https://aws.amazon.com/de/bedrock/*
[28] *https://cloud.google.com/products/ai*
[29] *https://platform.deepseek.com/*

access to immense computational resources and storage capacity, they offer several key advantages:

- **High-quality responses**: The models are extremely capable and powerful.
- **Fast inference times**: Responses are generated quickly on powerful servers.
- **Minimal data transfer**: Only input and output data need to be transmitted, not the entire AI model.
- **Hardware independence**: It works regardless of the client's hardware and computing resources.

However, cloud-based models also have their drawbacks:

- **Connectivity**: They require a stable internet connection.
- **Reliability**: They are subject to network latency, server availability, and capacity limits.
- **Privacy**: Data must be transferred to the cloud service, which raises potential privacy concerns. It is often unclear if user data is being used to train future model iterations.
- **Cost**: They usually require a subscription or API usage fees, so website authors often have to pay for the inference.

Local AI technologies

The limitations of cloud-based systems can be addressed by migrating inference to the client via local AI technologies, referred to as Web AI[30]. Since models are downloaded to the user's system, their *weights* (internal model parameters) cannot be kept a secret. As a result, this approach is mostly used in combination with open-weight models, which are typically less powerful[31] than their commercial, cloud-based, closed-weight counterparts. According to Epoch AI[32], open-weight models lag behind state-of-the-art by around three months on average.

[30] *https://developer.chrome.com/blog/io24-web-ai-wrapup#:~:text=Web%20AI%20is%20a%20set%20of%20technologies%20and%20techniques%20to%20use%20machine%20learning%20(ML)%20models%2C%20client%2Dside%20*

[31] *https://www.vellum.ai/llm-leaderboard*

[32] *https://epoch.ai/data-insights/open-weights-vs-closed-weights-models*

3 months

Figure 4.2. The average lag of open-weight models behind state-of-the-art performance.

The Web Machine Learning[33] Community Group and Working Group of the World Wide Web Consortium[34] (W3C) are actively standardizing this shift to make AI a “first-class web citizen.” This effort follows two primary architectural directions: *Bring Your Own AI* and *Built-in AI*.

Bring Your Own AI

In the Bring Your Own AI (BYOAI) approach, the developer is responsible for shipping the model to the user. The web application downloads a specific model binary and executes it using low-level APIs on local hardware. This allows running highly specialized models, tailored to the use case, but also requires significant bandwidth.

There are three processing units[35] that can be used to run AI inference locally:

- *Central Processing Unit* (CPU): Responds quickly, ideal for low-latency AI workloads.
- *Graphics Processing Unit* (GPU): High throughput, ideal for AI-accelerated digital content creation.
- *Neural Processing Unit* (NPU) or *Tensor Processing Unit* (TPU): Low power, ideal for sustained AI workloads and AI offload for battery life.

The three key APIs that facilitate local AI inference are WebAssembly (CPU), WebGPU (GPU), and WebNN (CPU, GPU, and NPU).

It is essential to note that the use of WebAssembly and WebGPU does not confirm AI activity; these are general-purpose APIs frequently utilized for tasks such as complex calculations, 3D visualizations, or gaming.

WebAssembly

WebAssembly[36] acts as the bytecode for the web. Code written in various programming languages, including C++ and Rust, can be compiled into WebAssembly. It enables developers to write optimized, high-performance code that is executed by the browser’s scripting engine

[33] *https://webmachinelearning.github.io/*
[34] *https://www.w3.org/*
[35] *https://www.w3.org/2024/01/webevolve-series-events/media/slides/hu-ningxin.pdf#page=4*
[36] *https://developer.mozilla.org/docs/WebAssembly*

on the user's CPU.

WebAssembly has broad browser support, being implemented in all relevant browser engines since 2017.

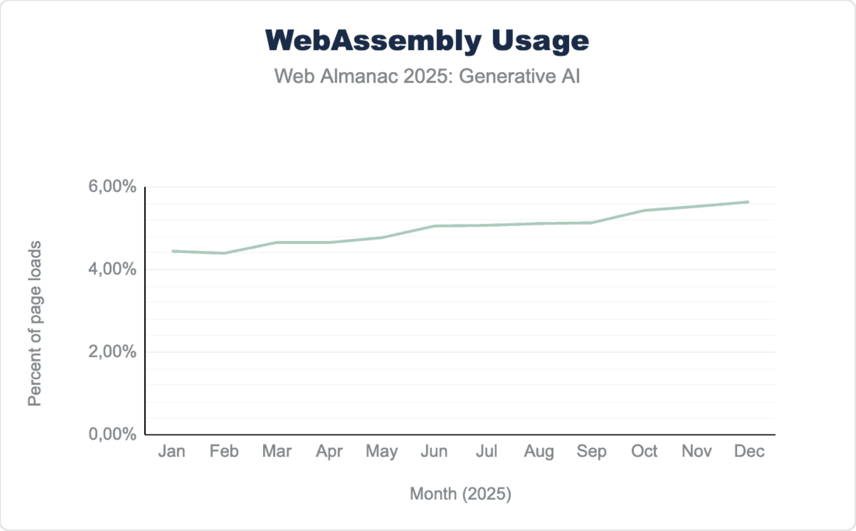


Figure 4.3. WebAssembly usage in 2025 according to Chrome Platform Status data.

According to Chrome Platform Status data[37], the usage of WebAssembly experienced 27% linear growth from being active on 4.44% of page loads in January 2025, and 5.64% in December 2025. In January 2024, WebAssembly was only active during 3.37% of page loads.

WebGPU

WebGPU[38] is the modern successor to WebGL[39], designed to expose the capabilities of modern GPUs to the web. Unlike WebGL, which was strictly for graphics, WebGPU provides support for *compute shaders*, allowing for general-purpose computing on graphics cards. This allows developers to perform massive parallel calculations, as required by AI models, directly on the user's graphics card.

WebGPU has become the standard foundation for running AI workloads in the browser. With the release of Firefox 141 in November 2025, WebGPU became available in all relevant browser engines[40] (Chromium, Gecko, and WebKit).

[37] *https://chromestatus.com/metrics/feature/timeline/popularity/2237*
[38] *https://developer.mozilla.org/docs/Web/API/WebGPU_API*
[39] *https://developer.mozilla.org/docs/Web/API/WebGL_API*
[40] *https://web.dev/blog/webgpu-supported-major-browsers*

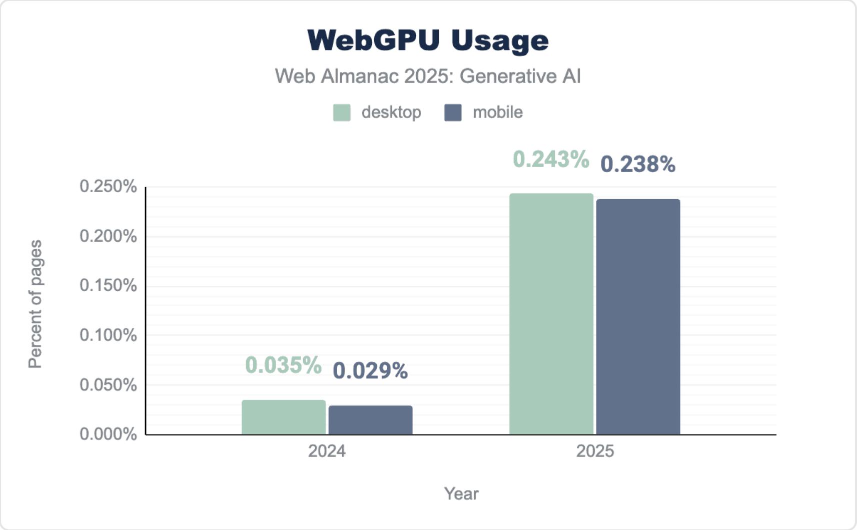


Figure 4.4. WebGPU usage in 2025.

The July 2025 crawl of HTTP Archive data shows that the API is used on 0.243% of all desktop sites and 0.238% of mobile sites. While still small overall, this represents a significant increase of 591% (up from 0.035%) on desktop and 709% (up from 0.029%) on mobile compared to the July 2024 crawl. Chrome Platform Status data[41] suggests an exponential growth in activations per page load, increasing by 147% over the course of 2025.

WebNN

The Web Neural Network API[42] (WebNN) is a dedicated API specifically for machine learning. Specified by the WebML Working Group, this API is on the W3C Recommendation track—the formal process for becoming a web standard.

WebNN serves as a hardware-agnostic abstraction layer, allowing the browser to route operations to the most efficient hardware available on the device. In contrast to WebAssembly (CPU-only) and WebGPU (GPU-only), WebNN can perform computations on the CPU, GPU, and NPU. It can achieve near-native execution speeds.

[41] *https://chromestatus.com/metrics/feature/timeline/popularity/3888*
[42] *https://webnn.io/en*

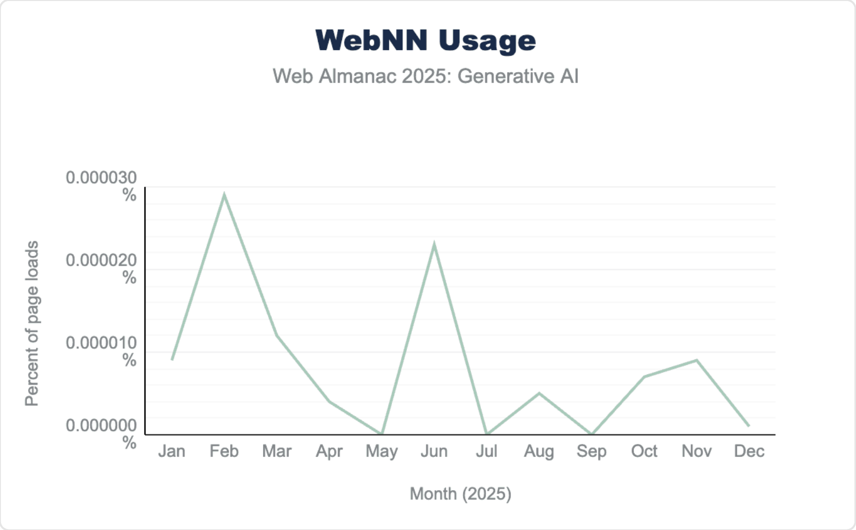


Figure 4.5. WebNN usage in 2025 according to Chrome Platform Status data.

WebNN is in active development and is currently implemented behind a flag in Chromium-based browsers[43] on ChromeOS, Linux, macOS, Windows, and Android. In November 2025, Firefox joined as the second engine formally supporting the API[44]. Given that WebNN is still an experimental API, the usage count is currently very low, with high fluctuation and a maximum activation rate of 0.000029% in February 2025 according to Chrome Platform Status data[45].

Runtimes: ONNX Runtime and Tensorflow.js

ONNX Runtime[46] (developed by Microsoft) and Tensorflow.js[47] (developed by Google) are two of the leading runtimes for executing AI models directly in the browser. These runtimes abstract away the complexities of low-level technologies like WebAssembly, WebGPU, and WebNN.

TensorFlow.js is tightly integrated with the TensorFlow ecosystem and supports both training and inference, while ONNX Runtime focuses on cross-platform inference using the ONNX standard, enabling models from multiple frameworks to run client-side.

[43] *https://webnn.io/en/api-reference/browser-compatibility/api*
[44] *https://github.com/mozilla/standards-positions/issues/1215#issuecomment-3520278819*
[45] *https://chromestatus.com/metrics/feature/timeline/popularity/5023*
[46] *https://onnxruntime.ai/docs/tutorials/web/*
[47] *https://www.tensorflow.org/js*

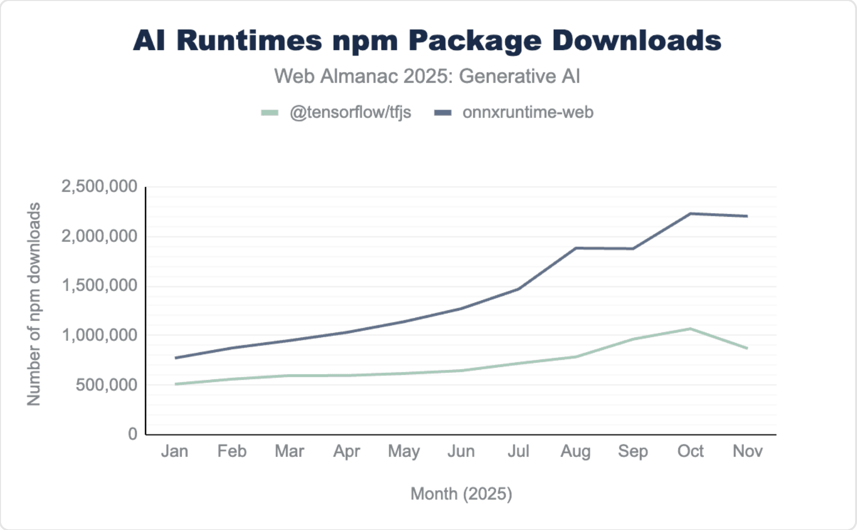


Figure 4.6. npm package downloads of `@tensorflow/tfjs` and `onnxruntime-web`.

From January to November 2025, npm package downloads of ONNX Runtime[48] increased by 185%, while TensorFlow.js's downloads[49] increased by 70%, demonstrating strong and growing developer interest in browser-based AI.

Libraries: WebLLM and Transformers.js

WebLLM[50], developed by the MLC AI[51] research team, is a high-performance in-browser inference engine specialized for LLMs. It allows running various open-weight LLMs, including Llama[52], Phi[53], Gemma[54], or Mistral[55], directly in the browser. Currently, WebLLM uses WebGPU for inference.

[48] *https://npm-stat.com/charts.html?package=onnxruntime-web&from=2025-01-01&to=2025-11-30*
[49] *https://npm-stat.com/charts.html?package=%40tensorflow%2Ftfjs&from=2025-01-01&to=2025-11-30*
[50] *https://webllm.mlc.ai/*
[51] *https://mlc.ai/*
[52] *https://www.llama.com/*
[53] *https://azure.microsoft.com/en-us/products/phi*
[54] *https://deepmind.google/models/gemma/*
[55] *https://mistral.ai/*

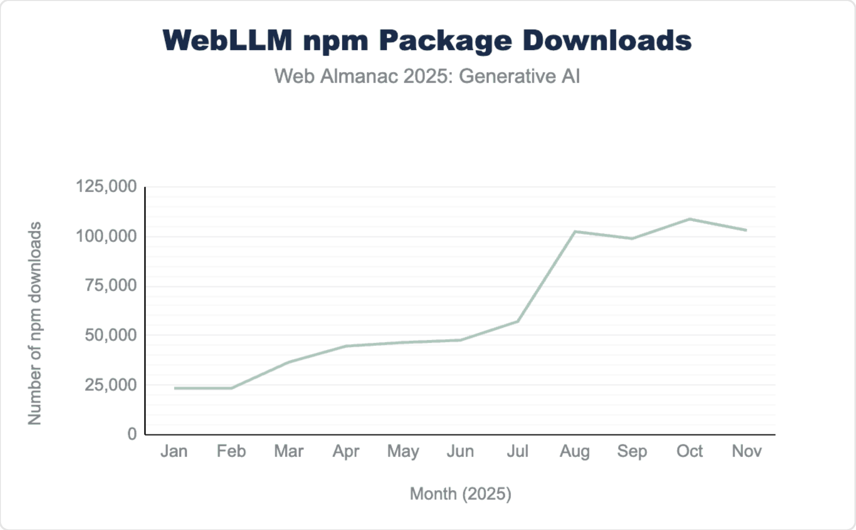


Figure 4.7. npm package downloads of `@mlc-ai/web-llm`.

WebLLM has quickly become one of the most prominent solutions for in-browser LLM inference. Between January and November 2025, WebLLM package downloads from npm[56] increased by 340%. In August alone, the downloads numbers nearly doubled. We were unable to attribute this surge to a specific event.

Transformers.js[57], developed by Hugging Face[58], functions as a comprehensive JavaScript library that mirrors the popular Python `transformers` API. Under the hood, it relies on ONNX Runtime to execute. It allows developers to run pre-trained models for various tasks through simple high-level pipelines, not just LLMs.

[56] *https://npm-stat.com/charts.html?package=%40mlc-ai%2Fweb-llm&from=2025-01-01&to=2025-11-30*
[57] *https://huggingface.co/docs/transformers.js/index*
[58] *https://huggingface.co/*

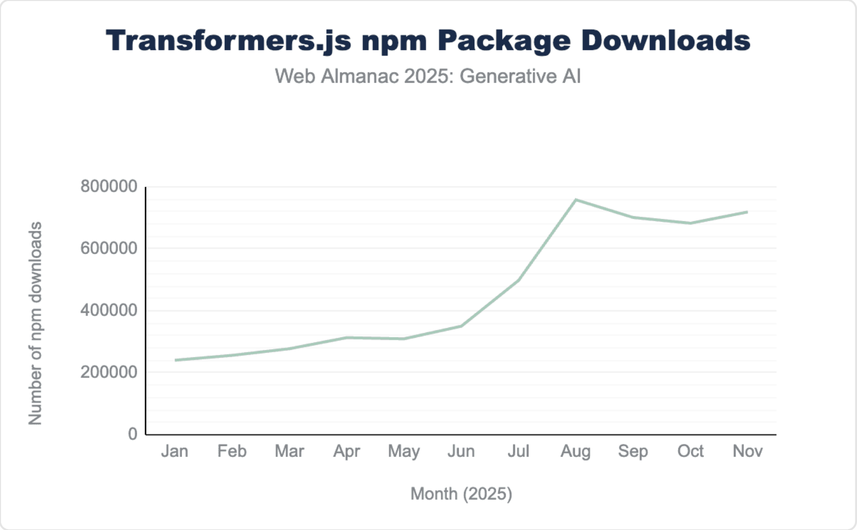


Figure 4.8. npm package downloads of `@huggingface/transformers.`

From January to November 2025, the npm package downloads[59] almost tripled, also with a notable surge in August 2025.

Built-in AI

Built-in AI[60] is an initiative by Google and Microsoft that aims to provide high-level AI APIs to web developers. Unlike BYOAI, this approach leverages models facilitated by the browser itself. While developers cannot specify an exact model, this method allows all websites to share the same model, meaning it only needs to be downloaded once.

The initiative consists of multiple APIs:

- **Prompt API**: Gives developers low-level access to a local LLM.
- Task-specific APIs:
  - **Writing Assistance APIs**: Summarizing, writing, and rewriting text.
  - **Proofreader API**: Finding and fixing mistakes in text.
  - **Language Detector** and **Translator APIs**: Detecting the language of text

[59] https://npm-stat.com/charts.html?package=%40huggingface%2Ftransformers&from=2025-01-01&to=2025-11-30
[60] https://developer.chrome.com/docs/ai/built-in

content and translating it into another language.

All APIs are specified within the WebML Community Group, meaning they are still incubating, and are not yet on the W3C Recommendation track. Some of the APIs are already generally available, while others need a browser flag to be enabled[61] or are in Origin Trial[62], requiring a registered token for activation—unlike WebGPU, which is now stable and widely available for shipping production features.

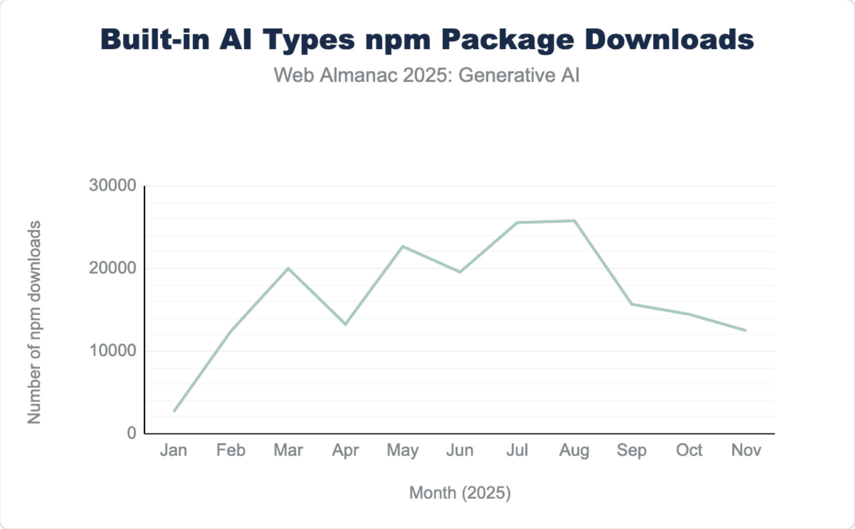


Figure 4.9. npm package downloads of `@types/dom-chromium-ai`.

The download numbers[63] of the `@types/dom-chromium-ai` package, containing TypeScript typings for Built-in AI, may reflect the experimental status of the APIs: Downloads peaked in August 2025 with 25,770 downloads, followed by a gradual decline. This trend may suggest that developers are hesitant to adopt experimental APIs. However, the download statistics for a typings package may not reflect the actual usage of the API.

Prompt API

The Prompt API introduces a standardized interface for accessing LLMs facilitated by the user's browser, such as Gemini Nano[64] in Chrome or Phi-4-mini[65] in Edge. By leveraging these one-time-download models, the API eliminates the bandwidth barriers and cold-start latency associated

[61] *https://developer.chrome.com/docs/ai/built-in-apis*
[62] *https://developer.chrome.com/docs/web-platform/origin-trials*
[63] *https://npm-stat.com/charts.html?package=%40types%2Fdom-chromium-ai&from=2025-01-01&to=2025-11-30*
[64] *https://store.google.com/intl/en/ideas/articles/gemini-nano-offline/*
[65] *https://learn.microsoft.com/en-us/microsoft-edge/web-platform/prompt-api*

with libraries that require downloading model weights.

However, as of December 2025, the technology remains in a transitional phase: it has fully shipped for browser extensions in Chrome 138[66], but web page access is still restricted to an Origin Trial. The API is not currently available on mobile devices.

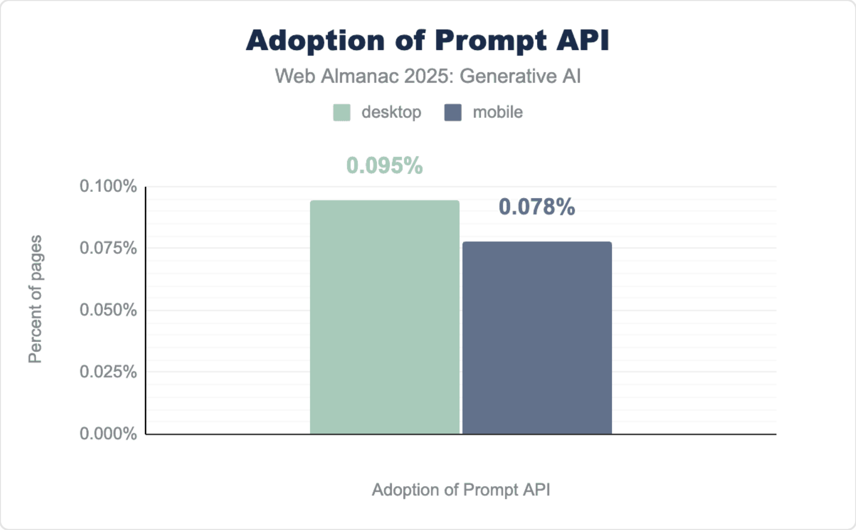


Figure 4.10. Prompt API adoption.

Consequently, adoption remains nascent; HTTP Archive data from July 2025 (the first measurement available) detected the API on just 0.095% of all desktop sites and 0.078% of all mobile sites, reflecting its current status as an experiment rather than a standard utility.

Most of the example sites we analyzed used the Prompt API through an external script, Google Publisher Tags[67]. This project enables authors to incorporate dynamic advertisements into their websites. The Google Publisher Tags script[68] uses the Prompt API to categorize the page's content into a list of the Interactive Advertising Bureau (IAB) Content Taxonomy 3.1 categories[69], and the Summarizer API (see next section) to generate a summary of the page's content, and sends both to the server. However, the code branch didn't seem to be active during our analysis.

[66] *https://developer.chrome.com/docs/ai/prompt-api*
[67] *https://developers.google.com/publisher-tag/guides/get-started*
[68] *https://securepubads.g.doubleclick.net/pagead/managed/js/gpt/m202512040101/pubads_impl.js*
[69] *https://github.com/InteractiveAdvertisingBureau/Taxonomies/blob/develop/Content%20Taxonomies/Content%20Taxonomy%203.1.tsv*

Writing Assistance APIs and Proofreader API

Next, we examine the task-specific APIs. The Writing Assistance APIs[70] abstract the complexities of prompt engineering; they utilize the same underlying embedded LLM but apply specialized system prompts to achieve distinct linguistic goals:

- **Writer API**: creates new content based on a prompt
- **Rewriter API**: rephrases input based on a prompt
- **Summarizer API**: produces a summary of text

Additionally, the Proofreader API[71] enables developers to proofread text and correct grammatical errors and spelling mistakes.

The Summarizer API shipped with Chrome 138. As of December 2025, the Writer, Rewriter, and Proofreader API are in Origin Trial. All APIs are not currently available on mobile devices.

In the July 2025 HTTP Archive crawl, only calls to `Writer.create()` were detected (on 0.127% desktop and 0.137% mobile sites). While this initially suggested usage of the Writer API, manual checks revealed that many sampled sites were actually using Protobuf.js's Writer[72], which shares the same API signature. As a result, we have decided to omit the chart for this metric.

Translator and Language Detector APIs

The final category of the Built-in AI APIs consists of the Translator and Language Detector APIs[73]. Unlike the Writing Assistance APIs, these APIs do not rely on LLMs, but instead utilize specialized, task-specific neural networks, placing them outside the strict definition of Generative AI.

The APIs shipped in Chrome 138[74] but are not currently available on mobile devices.

[70] *https://learn.microsoft.com/en-us/microsoft-edge/web-platform/writing-assistance-apis*
[71] *https://developer.chrome.com/docs/ai/proofreader-api*
[72] *https://github.com/protobufjs/protobuf.js/blob/827ff8e48253e9041f19ac81168aa046dbdfb041/src/writer.js#L142*
[73] *https://developer.mozilla.org/docs/Web/API/Translator_and_Language_Detector_APIs*
[74] *https://developer.chrome.com/release-notes/138#web_apis*

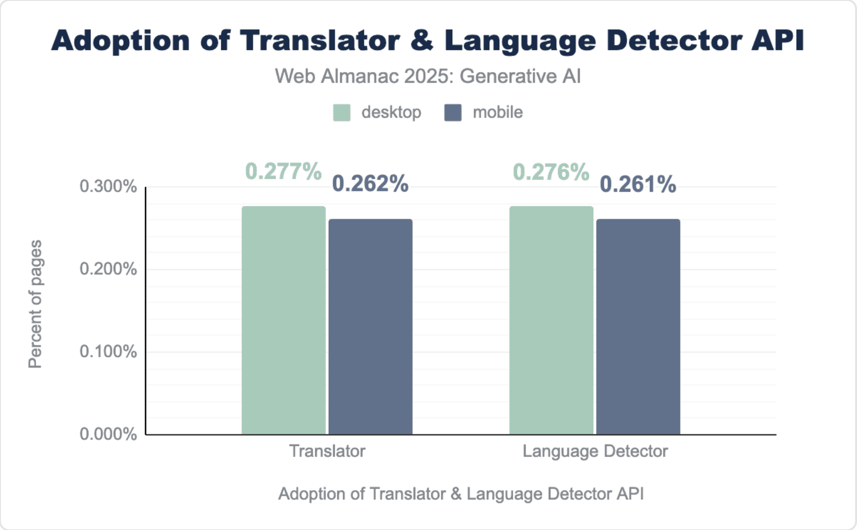


Figure 4.11. Translator and Language Detector API adoption.

They have achieved the widest adoption of the Built-in AI APIs. The July 2025 HTTP Archive crawl detected the Translator API on 0.277% of all desktop sites and 0.262% of all mobile sites, with the Language Detector API used on just 0.001% fewer sites.

Many of the sample sites we checked utilized the review tool Judge.me[75], which serves as an add-on for Shopify stores. Judge.me utilizes both the Language Detector and Translator APIs[76], which may be the reason for the tight coupling of usage: The Language Detector API was present on nearly the same absolute number of sites, trailing the Translator API by only approximately 100 sites.

Browser-specific runtimes: Firefox AI Runtime

As an alternative to the Built-in AI APIs proposed by the Chromium side, Firefox experiments with the Firefox AI Runtime[77], a local inference runtime based on Transformers.js and the ONNX Runtime, which runs natively. However, the runtime is not yet accessible from the public web. It can only be used by extensions and other privileged uses, such as the built-in translation feature[78] of Firefox.

[75] *http://Judge.me*
[76] *https://judge.me/help/en/articles/11379816-translating-reviews-in-the-review-widget*
[77] *https://firefox-source-docs.mozilla.org/toolkit/components/ml/index.html*
[78] *https://www.firefox.com/en-US/features/translate/*

Discoverability

In this section, we look at the dynamics of web discoverability with increasing adoption of Generative AI on the web. We primarily focus on two important approaches that influence how AI platforms and services discover content: the `robots.txt` and `llms.txt` files.

`robots.txt`

The `robots.txt` file, located at the root of a domain (for example, `http://example.com/robots.txt`), allows website owners to declare crawl directives for automated bots. Historically, web crawling was primarily performed by search engines and archives. However, in the age of Generative AI, websites can also be crawled by AI agents, or by model providers collecting internet-scale data to train their LLMs. Consequently, websites increasingly rely on `robots.txt` files to manage access for these crawlers.

An example directive targeting the user agent `GPTBot`, used by OpenAI for model training[79], is shown below:

```
User-Agent: GPTBot
Disallow: /
```

It is important to note that adherence to `robots.txt` is voluntary and does not strictly enforce access control.

[79] *https://platform.openai.com/docs/bots*

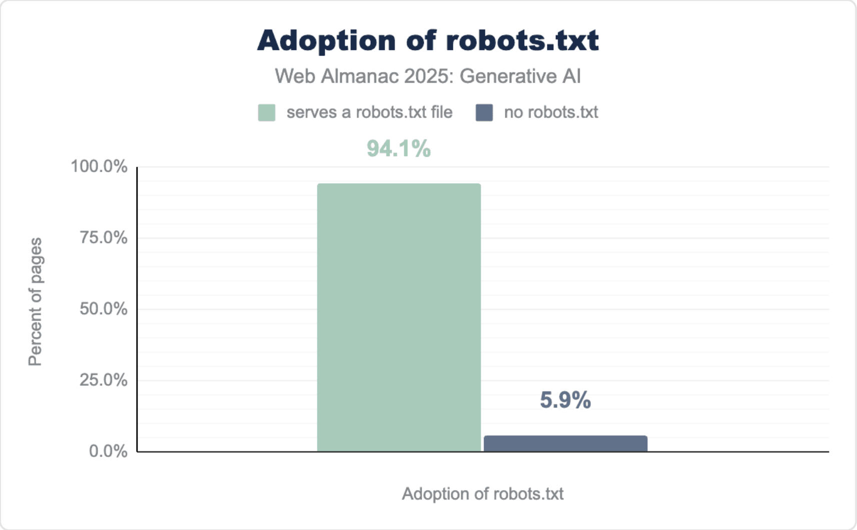


Figure 4.12. `robots.txt` *adoption.*

Usage of `robots.txt` remains highly prevalent: approximately 94.1% of the ~12.9 million sites analyzed include a `robots.txt` file with at least one directive.

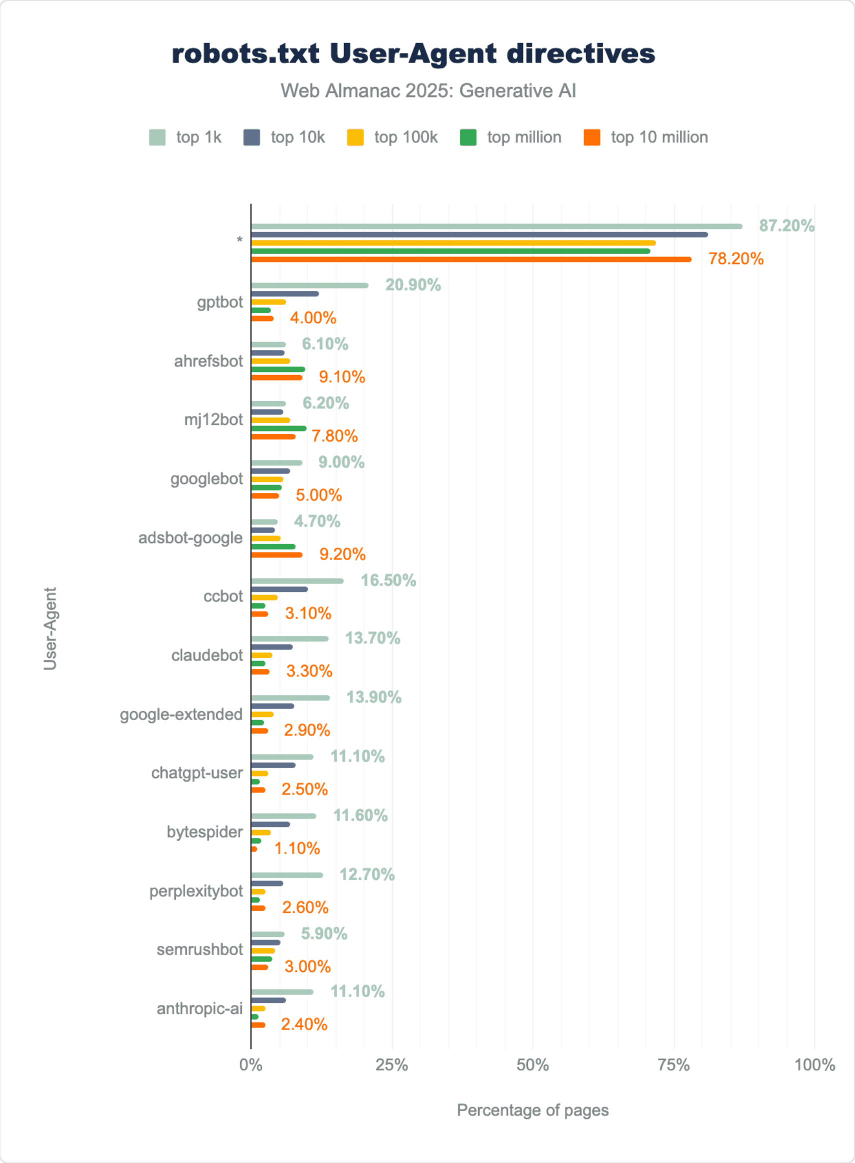


Figure 4.13. Top `robots.txt` directives by website ranks.

The figure above shows the top directives observed across all websites, grouped by their rank. The wildcard directive `*` is the most commonly used one, present in 97.4% of `robots.txt` files. It allows site authors to set global rules for all bots. The second most used user agent is `gptbot` (note, our analysis lower-cased the user-agent names), which saw adoption rise from

2.6% in 2024[80] to around 4.5% in 2025[81].

Other frequently targeted AI bots include those from Anthropic[82] (`claudebot` , `anthropic-ai` , `claude-web`), Google (`google-extended`), OpenAI (`chatgpt-user`), and Perplexity[83] (`perplexitybot`).

Directives targeting AI bots are significantly more common on popular websites, with nearly all of them disallowing access. This is likely because popular websites can restrict AI crawlers without negatively impacting their organic traffic. Overall, we observed a clear increase in the adoption of AI-specific directives within `robots.txt` files.

`llms.txt`

The `llms.txt` file is an emerging proposal designed to help LLMs effectively utilize a website at inference time. Similar to `robots.txt` , it is hosted at the root of a website's domain (for example, `https://example.com/llms.txt`). The key distinction lies in their function: while `robots.txt` instructs bots as to what they are allowed or disallowed to crawl during training or inference, `llms.txt` provides a structured, LLM-friendly overview of the website's content. This allows LLMs to efficiently navigate through a website in real-time when answering user queries.

[80] *https://almanac.httparchive.org/en/2024/seo#ai-crawlers*
[81] *https://almanac.httparchive.org/en/2025/seo#ai-crawlers-named-in-robotstxt*
[82] *https://www.anthropic.com/*
[83] *https://www.perplexity.ai/*

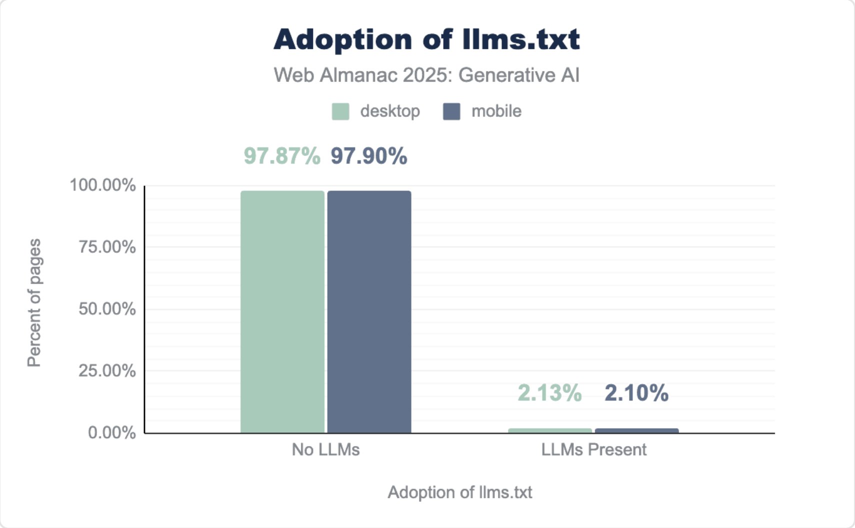


Figure 4.14. `llms.txt` *adoption.*

As `llms.txt` is relatively new, we observed very limited usage on the web. Across desktop pages, only 2.13% exhibit valid `llms.txt` entries, while 97.87% show no evidence of this file. Mobile pages showed a similar pattern, with 2.1% of pages containing valid entries and 97.9% lacking them.

The overwhelming majority of sites therefore do not yet articulate explicit AI access preferences via `llms.txt`. Where present, the file indicates that early adopters are likely experimenting with AI-driven discoverability, facilitating agentic use cases, or exploring *Generative Engine Optimization* (GEO). In contrast to Search Engine Optimization (SEO), GEO focuses on making content easily ingestible by LLMs or AI-enabled search tools like Perplexity or Google AI Mode[84]. For more information see the SEO chapter.

AI fingerprints

Services such as OpenAI ChatGPT[85], Google Gemini[86], and Microsoft Copilot[87] are already widely used. Increasingly, content is being written with the assistance of AI. This section analyzes the impact of Generative AI on web content and source code.

[84] *https://search.google/ways-to-search/ai-mode/*
[85] *https://chatgpt.com/*
[86] *https://gemini.google.com/*
[87] *https://www.copilot.com/*

AI colors

Models can tend to overrepresent certain words, which then serve as an AI fingerprint. Researchers at Florida State University compared research papers published on PubMed between 2020 and 2024. Their analysis[88] revealed a staggering 6,697% increase in the use of the word "delves." The other conjugations of the verb "delve" also increased significantly. While those kinds of indicators should never be used to classify individual cases as AI-generated, they are still useful for understanding the trend of adoption. In this part of the chapter, we will delve into ;) common patterns used by LLMs to generate websites.

6,697%

Figure 4.15. Usage increase of the word "delves" in research papers published on PubMed.

The most recognizable indicator of AI-generated web design is the pervasive use of purple and gradients, often referred to as "the AI Purple Problem[89]."

Adam Wathan[90], creator of the widely adopted CSS framework Tailwind[91], noted in a recent interview[92] that this trend likely stems from the timing of major AI training crawls. In the early 2020s, the Tailwind team used deep purples extensively in their documentation and examples to reflect contemporary design trends. Consequently, LLMs trained on that data seem to have "locked in" an aesthetic that has since been replaced by more modern trends, such as the current preference for black designs.

To quantify the "AI Purple" aesthetic, we analyzed the CSS (responsible for website styling) of all root pages in our dataset from the years following the release of ChatGPT. We tracked the presence of the well-known `indigo-500` color (`#6366f1`), the use of CSS gradients, and other colors associated with AI-generated websites.

[88] *https://aclanthology.org/2025.coling-main.426.pdf*
[89] *https://ai-engineering-trend.medium.com/the-mystery-behind-ais-purple-problem-revealed-0234afdb292e*
[90] *https://x.com/adamwathan*
[91] *https://tailwindcss.com/*
[92] *https://youtu.be/AG_791Y-vs4?t=86*

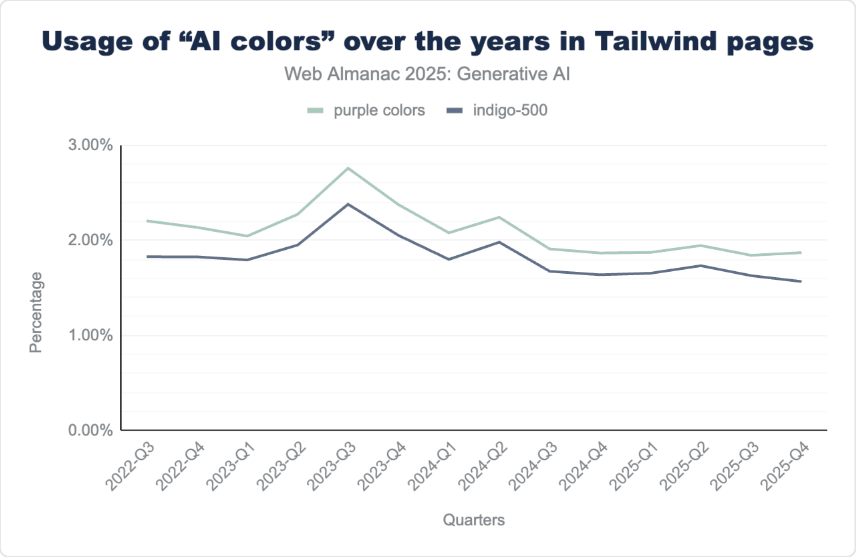


Figure 4.16. Usage of "AI colors" over the years in Tailwind pages.

Because Tailwind is the preferred framework for LLMs like Claude, Gemini, and the OpenAI models, we specifically examined the percentage of Tailwind-based websites using these indicators. Tailwind web pages were identified using a Wappalyzer fork[93] from the HTTP Archive. Surprisingly, there was no noticeable surge in websites using either `indigo-500` or the other two commonly mentioned AI colors (`#8b5cf6` violet and `#a855f7` purple).

[93] *https://github.com/HTTPArchive/wappalyzer*

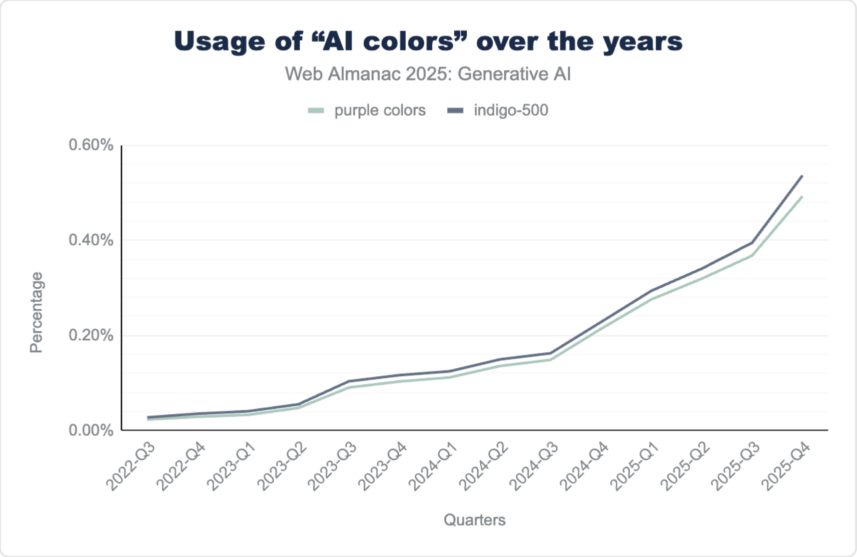


Figure 4.17. Usage of "AI colors" over the years.

However, due to the explosive overall growth of the Tailwind framework, these specific colors saw a significant rise when measured as a percentage of the entire web.

Our analysis relied solely on CSS variables, as parsing the entire source code would exceed practical limits. As a result, our figures represent a conservative estimate. Sites using hardcoded hex values (for example, `#6366f1`) or older preprocessors were invisible to our queries. Additionally, even a 1% adjustment to a color's lightness would cause it to evade our detection.

Despite the frequent online discourse regarding "AI slop"[94] and purple-tinted designs, our analysis of the HTTP Archive dataset suggested that this trend may be less a result of AI "choosing" purple and more a reflection of the general rise in Tailwind's popularity.

We also tested for surges in gradients, shadows, and specific fonts, but found no statistically significant increases.

Vibe coding platforms

One of the reasons for the large number of newly created websites is the emergence of platforms that make building them easier. After the spread of Content Management System

[94] *https://www.wikipedia.org/wiki/AI_slop*

(CMS) tools in the previous decade, users can now move beyond those guardrails and use tools like v0[95], Replit[96], or Lovable[97] to generate websites through conversational AI. These platforms allow users to publish pages using their own domains or a platform-provided subdomain (for example, `mywebsite.tool.com`).

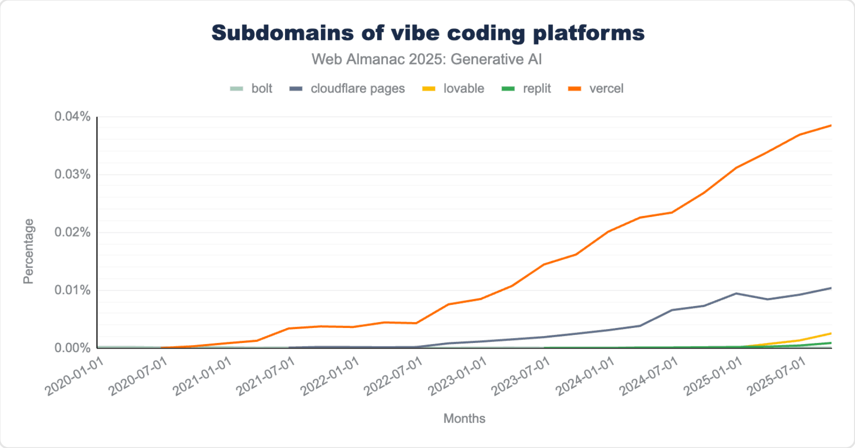


Figure 4.18. Subdomains of vibe coding platforms.

The chart above illustrates the relative growth of vibe coding platform subdomains within the HTTP Archive dataset. While Vercel (`*.vercel.app`) remains the dominant provider, it has offered subdomain hosting long before the release of its vibe coding tool v0 (`*.v0.dev`) in late 2023. The data showed no immediate surge following the launch of v0. Lovable (`*.lovable.app` and `*.lovable.dev`) grew from only 10 subdomains at the beginning of 2025 to 315 by October 2025. Please note that this only includes pages available in the dataset. According to Lovable, 100,000 new projects are built on Lovable every single day[98].

[95] https://v0.app/
[96] https://replit.com/
[97] https://lovable.dev/
[98] https://lovable.dev/blog/one-year-of-lovable

The `.ai` domain

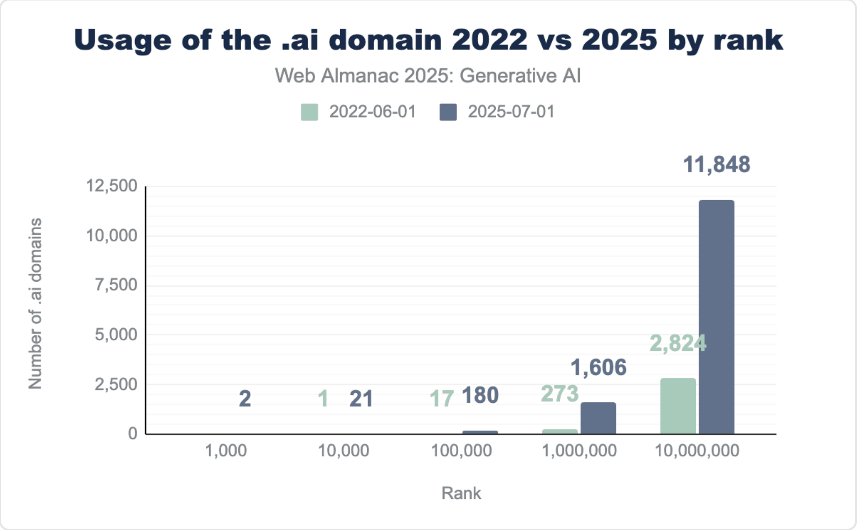


Figure 4.19. Usage of the .ai domain 2022 vs 2025 by rank.

While both AI and the `.ai` domain existed long before ChatGPT, the new possibilities that emerged in its wake led to a wave of AI-native businesses. Many of them adopted the `.ai` top-level domain, the country code for Anguilla[99]. This trend drove a substantial rise in `.ai` domain registrations across all ranks, as shown in the figure above, which compares the pre-ChatGPT landscape in 2022 with usage in 2025.

The only two `.ai` domains that made it into the top 1,000 most visited websites in the Chrome dataset were https://character.ai and an adult site.

Conclusion

In 2025, Generative AI transitioned from a cloud-only technology to a fundamental browser component. BYOAI dominated immediate adoption, driven by the growth of foundational APIs, such as WebAssembly and WebGPU, as well as libraries like WebLLM and Transformers.js. At the same time, Built-in AI began laying the groundwork for a standardized AI layer directly within the browser. A tension has formed between more restrictive `robots.txt` files and the welcoming structure of `llms.txt`. The rise of vibe coding and `.ai` domains proves that AI isn't just reshaping how apps function, but also how they are built.

[99] *https://www.iana.org/domains/root/db/ai.html*

Looking beyond 2025, the next evolutionary leap is already visible: Agentic AI. We are moving away from the era of interactive chatbots toward autonomous agents that are capable of making decisions, executing multi-step workflows, and solving complex tasks without continuous user intervention. This shift is giving rise to a new class of “agentic browsers” such as Perplexity Comet[100], ChatGPT Atlas[101], or Opera Neon[102].

To connect these agents to the vast resources of the web, new protocols such as WebMCP[103] (Web Model Context Protocol) are emerging. Just as HTTP standardizes the transmission of resources, WebMCP aims to standardize the way agents perceive and manipulate web interfaces, ensuring that the AI-native web is not only readable but also actionable.

Authors

Christian Liebel

@christianliebel @https://mastodon.cloud/@christianliebel christianliebel

christianliebel https://christianliebel.com

Christian Liebel, M.Sc., is an elected member of the W3C Technical Architecture Group[104] (TAG), a participant in the W3C Web Machine Learning[105] (WebML) Community and Working Groups, and a Google Developer Expert (GDE) for Web AI.

Yash Vekaria

@vekariayash Yash-Vekaria https://www.yashvekaria.com/

Yash Vekaria is a PhD candidate in Computer Science at University of California, Davis[106]. He carries out web-based large-scale internet measurements to study and improve the dynamics of web. Specifically, his research is focused at studying and bringing transparency to online tracking practices and user privacy issues.

[100] https://www.perplexity.ai/comet
[101] https://chatgpt.com/atlas/
[102] https://www.operaneon.com/
[103] https://github.com/webmachinelearning/webmcp
[104] https://w3ctag.org/
[105] https://webmachinelearning.github.io/
[106] https://www.ucdavis.edu/

Jonathan Pagel

jcmpagel jonathan-pagel https://jonathanpagel.com

Jonathan Pagel studied e-commerce in his bachelor's degree and has since been interested in the field, particularly in the areas of speed optimization and accessibility for shops and websites. Currently, he is freelancing in this field and pursuing a Master's in AI and Society.

Part II Chapter 5

SEO

Written by Amaka Chulwuma, Chris Green, and Sophie Brannon
Reviewed by Jamie Indigo
Analyzed by Augustin Delport and Chris Green
Edited by Michael Lewittes, Montserrat Cano, and Sharon McClintic

Introduction

Search Engine Optimization (SEO) continues to play a central role in how information is discovered and understood online. It encompasses the technical, structural, and content practices that determine whether a website can be effectively crawled, indexed, and surfaced in search results.

Strong SEO foundations not only support visibility in traditional search engines but are becoming increasingly important as AI systems begin to interpret and summarize web content in new ways.

This 2025 SEO chapter of the Web Almanac draws its data and insights from the HTTP Archive crawl, Lighthouse reports, Chrome User Experience (CrUX) reports, and custom metrics. Our goal is to document how the technical state of the web is evolving and to identify the factors that most influence organic visibility today.

While many SEO metrics have stabilized across the web, the context surrounding them is changing rapidly. The rise of AI crawlers, emergence of `llms.txt` and a growing emphasis on machine readability suggest that optimization is no longer only about being *found* by bots, but about being *understood* by them. How will these changing contexts in online search influence optimization moving forward and how do the latest data points already reflect AI's influence on SEO?

Crawlability & indexability

For web content to gain visibility in search results, it must first be crawled and indexed by search engine crawlers. Crawlability determines whether bots can find and access a page, while indexability defines whether that page is eligible to appear in search results. Together, these concepts form the foundational elements of search visibility. A page cannot rank or be served to users if it cannot first be found and understood by the bots. Similarly, content cannot be cited on AI platforms unless a site is indexable.

Google's documentation[107] clarifies that compliance with crawling rules depends not only on the presence of a `robots.txt` file but also on how correctly this file is structured. Search engines also apply practical limits and standardized caching behavior to ensure directives can be parsed efficiently and interpreted consistently.

At the same time, as Cloudflare explains[108], `robots.txt` functions more as a code of conduct than a command that will always be obeyed. While reputable bots respect these signals, others may ignore them entirely. This mix of cooperation and unpredictability defines the modern crawling environment and sets the stage for examining how sites actually manage crawler access.

`robots.txt`

Serving as the web's de facto "visitors' center" for crawlers, the `robots.txt` file is where bots learn which parts of a site are open or restricted. Since the IETF's standardization[109] of the Robots Exclusion Protocol (RFC 9309[110]) three years ago, its syntax, caching behavior, and error handling have been clearly defined—providing a stable framework for how crawlers interpret access rules.

Efforts to refine that framework are ongoing. In late 2024, the IETF introduced[111] a working

[107] *https://developers.google.com/search/docs/crawling-indexing/robots/robots_txt*
[108] *https://www.cloudflare.com/learning/bots/what-is-robots-txt/*
[109] *https://www.ietf.org/about/*
[110] *https://datatracker.ietf.org/doc/html/rfc9309*
[111] *https://garyillyes.github.io/ietf-rep-ext/draft-illyes-repext.html*

draft known as REPext, which builds on RFC 9309 by exploring page-level crawl controls through response headers and HTML `meta` tags, an approach that could make future implementations more granular and flexible.

For now, however, the `robots.txt` file remains the foundation of crawl management. Most websites now serve a valid file, with only a small minority omitting it entirely. Among those that do, site owners typically favor simple, universal directives rather than complex, bot-specific rules. The sections that follow examine how these preferences appear in the 2025 data.

`robots.txt` status codes

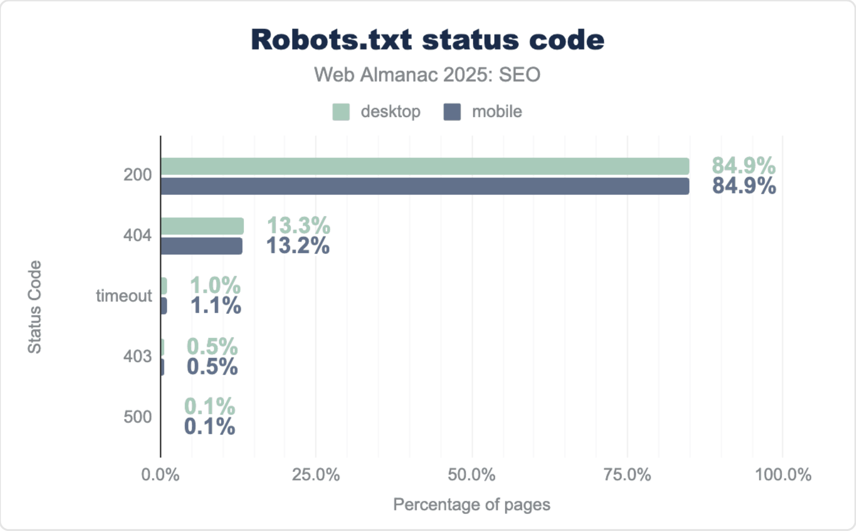


Figure 5.1. `robots.txt` status codes.

In 2025, 85% of requests for `robots.txt` files returned valid 200 status codes, up from 84% in 2024. The slight rise suggests ongoing trickle-down from the 2022 standardization and wider CMS defaults that serve valid `robots.txt` files. Importantly, however, a 200 response only confirms that a file exists. It does not guarantee that its directives are correct or beneficial to the site.

The mobile–desktop gap has effectively disappeared when it comes to valid (200) `robots.txt` status codes, with mobile now having just a 0.06% lead compared to desktop. This mirrors the industry's move away from separate "m-dot" sites toward responsive design with unified configurations.

The rate of 404 errors for `robots.txt` files declined to 13% from 14% in 2024. Fewer missing files imply that more sites are explicitly serving `robots.txt` files rather than leaving them absent, which would otherwise default to unrestricted crawling.

Timeouts are ~1.0%, 403 responses are ~0.5%, and 5xx are ~0.1%. Although uncommon, 5xx responses[112] on a `robots.txt` file can cause search engines to temporarily treat the site as blocked until a cached file or a later successful fetch is available.

`robots.txt` file size

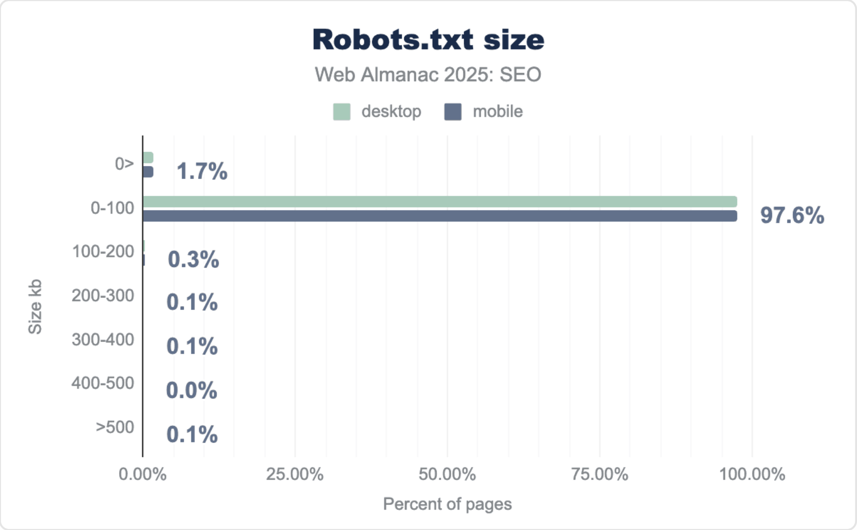


Figure 5.2. `robots.txt` size.

Nearly all `robots.txt` files stay well under size limits (Google enforces a 500 KB[113] parsing cutoff) and comply with standards such as not serving an empty file.

A small share of sites serve completely empty `robots.txt` files, now 1.8% on desktop and 1.7% on mobile, slightly up from 2024. While most major crawlers treat an empty file as permissive[114], the standard (RFC 9309[115]) doesn't define this behavior explicitly, leaving room for inconsistent handling by lesser-known bots. A safer approach is to either return a valid file or a 404 if no restrictions are intended.

[112] *https://developers.google.com/search/docs/crawling-indexing/robots/robots_txt#:~:text=Other%20errors-,A%20robots.,treated%20as%20a%20server%20error.*
[113] *https://developers.google.com/search/docs/crawling-indexing/robots/robots_txt#:~:text=Google%20enforces%20a%20robots.,the%20size%20of%20the%20robots.*
[114] *https://developers.google.com/search/docs/crawling-indexing/robots/robots_txt#:~:text=For%20the%20first%2012%20hours,checking%20for%20a%20new%20version).*
[115] *https://www.rfc-editor.org/rfc/rfc9309.html*

In 2025, 98% of files were under 100 KB, only slightly down from 2024. The minor change year over year points to a stable and mature implementation pattern.

Files between 100–200 KB accounted for 0.3% `robots.txt`, and 200–300 KB for just 0.1% of `robots.txt`, virtually unchanged from last year. Only 0.1% of sites exceeded the 500 KB parsing cutoff enforced by Google, confirming strong adherence to crawler limits and reinforcing that oversized `robots.txt` files are an edge case.

`robots.txt` file size rarely poses a barrier to crawlability. The more pressing issue continues to be empty or misconfigured files, which introduce uncertainty in how crawlers interpret site rules.

`robots.txt` user agent usage

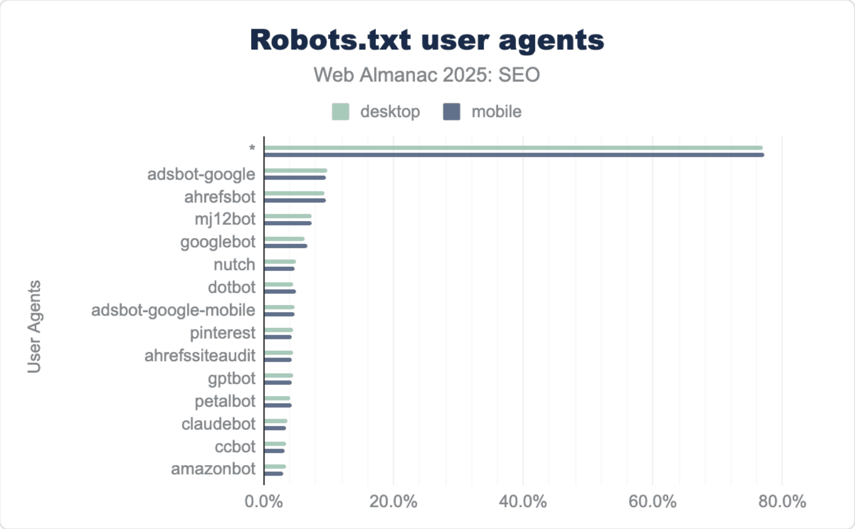


Figure 5.3. `robots.txt` user agents.

Note that for analysis we lowercased all the user-agent names identified in our crawl to ignore any differences between sites. While some crawlers may provide a preferred capitalization in their docs, they should, in theory, disregard case[116] when processing `robots.txt` files.

The catch-all user agent `*` is the most common approach to crawler directives found in `robots.txt` files today. In 2025, it appeared in 77% of files—up slightly from 2024, and from

[116] https://datatracker.ietf.org/doc/html/rfc9309#section-2.2.1:~:text=Crawlers%20MUST%20use%20case%2Dinsensitive%20matching%20to%20find%20the%20group%20that%20matches%20the%20product%20token%

the mid-70% range in 2022. The steady rise indicates that most site owners prefer to implement broad, universal rules rather than maintaining complex bot-specific instructions.

Where specific user agents *are* mentioned in `robots.txt` files, Google's advertising crawler (`adsbot-google`) and `ahrefsbots` take the lead again, as they did last year.

Targeting of `adsbot-google` in `robots.txt` directives rose to 9.8% (desktop) and 9.5% (mobile) in 2025, compared with 9.1% (desktop) and 8.9% (mobile) in 2024.

`ahrefsbot` also remained a leading named crawler, specified in 9.3% of desktop and 9.5% of mobile `robots.txt` files. Combined with `ahrefssiteaudit` (4.6% desktop / 4.3% mobile), this reflects the ongoing importance of controlling access by SEO tools, which can generate significant crawl activity.

Other named crawlers that appeared in notable volumes this year include:

- `mj12bot` (Majestic): 7.3% desktop / 7.3% mobile
- `googlebot`: 6.2% desktop / 6.7% mobile
- `nutch`: 5.0% desktop / 4.8% mobile

`bingbot` rarely named in `robots.txt`

`bingbot` ranks 22nd among named user agents in `robots.txt` files and appears in less than 3% of `robots.txt`. When a bot appears in `robots.txt` files, it means website managers care enough about that crawler to explicitly control its behavior, either allowing it, restricting it, or setting crawl rates. Low appearance rates suggest benign neglect. Despite Microsoft's massive investment in AI[117] and its integration of ChatGPT into Bing[118], the crawler itself hasn't become more prominent in `robots.txt` files. Even with its AI enhancements, Bing's web footprint and importance to site operators remain largely unchanged, a quiet contrast to the rapid rise of AI-focused crawlers like `gptbot` named in `robots.txt` files (more on that later).

[117] https://blogs.microsoft.com/on-the-issues/2025/01/03/the-golden-opportunity-for-american-ai/
[118] https://blogs.microsoft.com/blog/2023/02/07/reinventing-search-with-a-new-ai-powered-microsoft-bing-and-edge-your-copilot-for-the-web/

Slight growth in use of named user agents

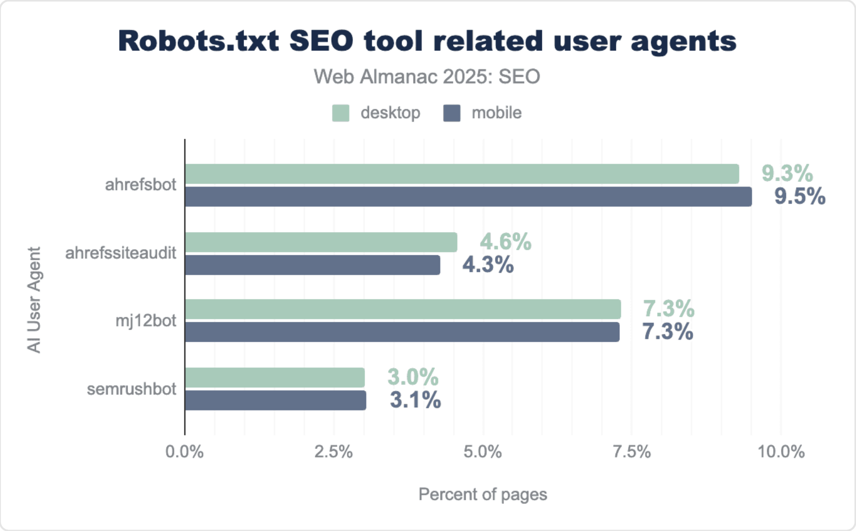


Figure 5.4. `robots.txt` *SEO tool related user agents.*

Overall, compared to 2024, a slightly larger share of `robots.txt` files in 2025 include directives for named crawlers rather than relying solely on the universal wildcard `*`. For instance, MJ12Bot rose from 6.6% (mobile) last year to 7.3% (mobile) in 2025, Googlebot rose from 6.4% (mobile) to 6.7% (mobile), and Nutch from 4.3% (mobile) to 4.81% (mobile) this year. These modest gains point to gradual refinement—more site owners are setting tailored crawl rules where it matters, without moving away from the simplicity of universal controls.

The continued dominance of `*` alongside rising mentions of specific bots in `robots.txt` suggests a pragmatic balance. Universal directives remain the norm, but targeted rules are added where business concerns justify them. Not all crawlers interpret `*` consistently. Google's AdsBot ignores it, and Applebot falls back to Googlebot rules before applying `*` making explicit targeting necessary in certain cases.

AI crawlers named in `robots.txt`

The rise of AI crawlers has transformed `robots.txt` from a traditional search optimization tool into a broader mechanism for managing content permissions. In 2025, directives targeting AI-related bots showed substantial growth from 2024 levels.

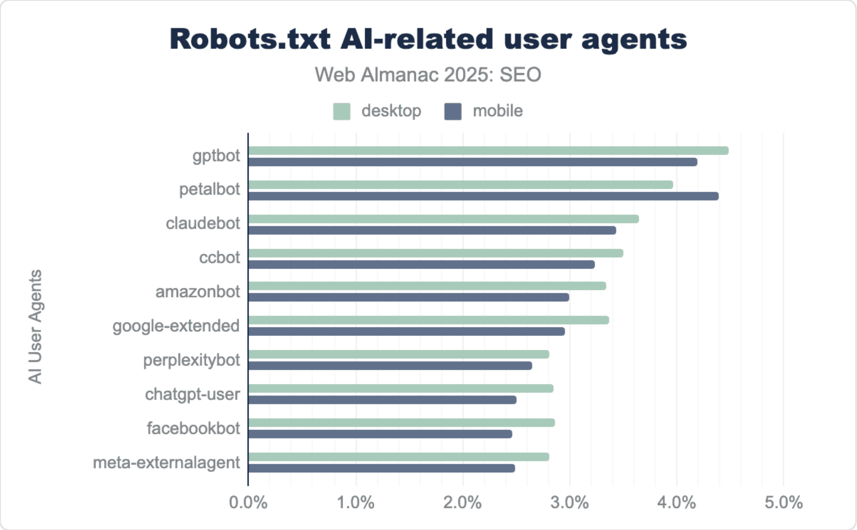


Figure 5.5. `robots.txt` AI-related user agents.

The year-over-year comparison reveals accelerating adoption:

- `gptbot` : 4.5% (desktop), 4.2% (mobile) is up from 2.9% (desktop) and 2.7% (mobile) in 2024, representing a ~55% increase.
- `ccbot` : 3.5% (desktop), 3.2% (mobile) is up from 2.7% (desktop) and 2.4% (mobile) in 2024.
- `petalbot` : 4.0% (desktop), 4.4% (mobile) this year; was not separately tracked in 2024.
- `claudebot` : 3.6% (desktop), 3.4% (mobile) is up from 1.9% (desktop) and 1.6% (mobile) in 2024, nearly doubling.

Notably, 2024's data included broader categories like `anthropic-ai` (2.0% desktop and 1.7% mobile last year) and `chatgpt-user` (2.0% desktop and 1.7% mobile last year), while our 2025 data shows more specific bot targeting.

Other entrants in the AI crawler space this year include `amazonbot` (3.3% desktop, 3.0% mobile), `google-extended` (3.4% desktop, 3.0% mobile), `perplexitybot` (2.8% desktop, 2.7% mobile), `chatgpt-user` (2.8% desktop, 2.5% mobile), `facebookbot` (2.9% desktop, 2.5% mobile), and `meta-externalagent` (2.8% desktop, 2.5% mobile).

While overall user agent targeting has grown only gradually year over year, the adoption of AI crawlers has been far more abrupt. `gptbot`'s near-doubling since 2024, alongside measurable adoption for `petalbot`, `claudebot`, and others, represents one of the fastest expansions of `robots.txt` directives for named user agents in recent memory, moving from a marginal presence in 2023 to multi-percent adoption rates by 2025.

This shift introduces new complexity for site owners. Instead of only asking, "Should this page be indexed for search?" they must now also ask, "Should this content be used to train AI models?" These are distinct considerations with different business and ethical implications.

`robots.txt` has become a dual-purpose control point, balancing visibility in search with protection against large-scale data harvesting. This trend is likely to intensify as AI models and crawlers proliferate.

`llms.txt`

The `llms.txt` file has been proposed as a new standard "to provide information to help LLMs use a website at inference time" (per llmstxt.org[119]). This text file contains a highly simplified version of the website's content in Markdown format, with a view toward making it easier for LLMs to ingest and subsequently use in generated responses.

This standard, has not been adopted widely and has become a point of controversy within the broader SEO industry. Google has often stated that they do not use `llms.txt`, and no Google service currently does[120]. Anthropic, however, has taken a lead on `llms.txt`, raising optimism that the format may evolve into a reliable mechanism for managing and optimizing content utilization during model inference.

`llms.txt`, adoption rate

Regardless of whether using `llms.txt` proves to be a valid approach for SEO or AI optimization, the introduction of a new file format and proposed standard like this could influence how websites are built and optimized moving forward.

As part of the 2025 Almanac, we have introduced monitors to assess the level of `llms.txt` adoption across the web.

[119] *http://llmstxt.org*
[120] *https://www.seroundtable.com/google-ai-llms-txt-39607.html*

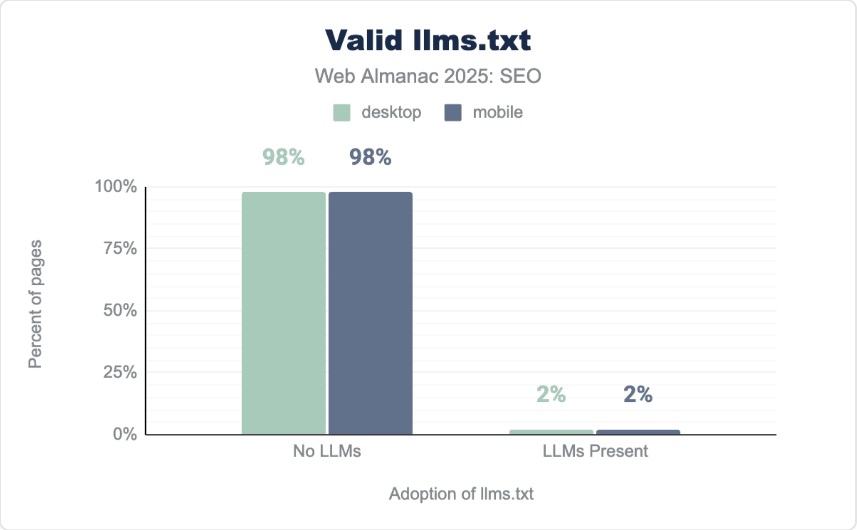


Figure 5.6. Valid `llms.txt`

We can see that the adoption rate is relatively low at 2%—but these figures are arguably still notable given how new the `llms.txt` format is and that a total of 324,184 valid files were identified across all mobile sites analyzed.

Digging into the content of these `llms.txt` files, we see some clues about the adoption of this standard so far, and likely what is going to make this move in the future.

- 39.6% of `llms.txt` files are related to All in One SEO
- 3.6% of `llms.txt` files are related to Yoast SEO

This was gleaned by the comments these CMS extensions leave (by default) in the files they generated, but it shows that a significant number of website owners with an `llms.txt` file in place are having them generated by their CMS/add-on extensions. Therefore, we cannot be sure this is always a conscious act or endorsement of the `llms.txt` standard or an unintentional inclusion.

Since January 2025, interest in this emerging standard has grown[121] and hinges on just one or two key AI companies' recognition. The 2026 numbers will be interesting to see, nonetheless.

[121] *https://trends.google.com/trends/explore?q=llms.txt*

Robots directives

A robots directive[122] provides page-level control over how a specific page is indexed and displayed in search results. While they have a similar function to `robots.txt` files, the two serve different purposes:

- Robots *directives* influence **indexing and serving**
- Whereas `robots.txt` governs **crawling**

For a directive to be applied, the crawler must be able to access the page. If a page is blocked by robots.txt, its directives may never be seen or obeyed.

Robots directives implementation

There are two main ways to implement robots directives:

1. Using a `<meta name=robots>` tag (placed within the `<head>` section of a webpage)
2. Using an `X-Robots-Tag` HTTP header

The method you choose depends on your specific use case, as well as the means and methods at your disposal.

`meta` robots tags are primarily for HTML pages and are widely supported by most major CMSs either natively or through add-ons, and other common, well-supported frameworks.

`X-Robots-Tags` have a major advantage in that they can be implemented for other non-HTMLs file types, such as PDFs or documents. However, setting them is often not as easy or simple for CMS users, so they may not always be an option.

[122] *https://developers.google.com/search/docs/crawling-indexing/robots-meta-tag*

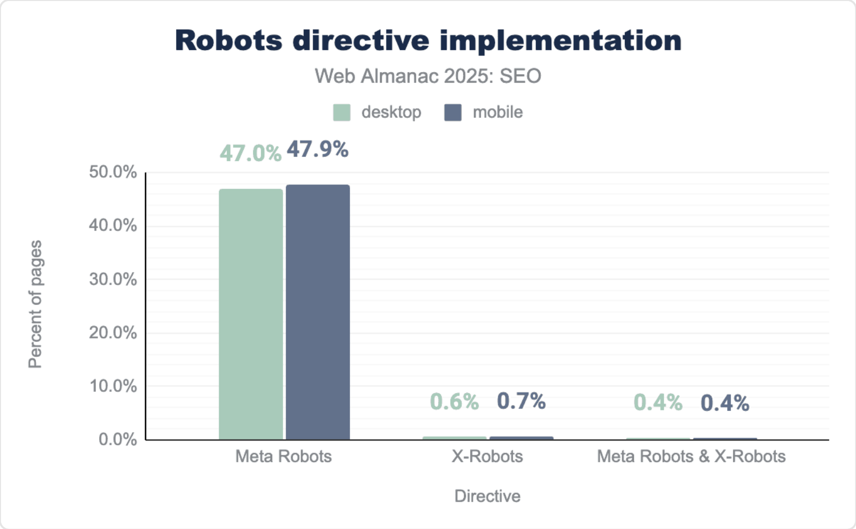


Figure 5.7. Robots directive implementation

By a large margin, `meta` robots is the most widely used method for implementing robots directives, appearing on 47.0% of desktop pages and 47.9% of mobile pages. X-Robots come in a distant second at 0.6% (desktop) and 0.7% (mobile).

The number of pages implementing `meta` robots increased from 45.5% (desktop) and 46.2% (mobile) recorded last year, showing growth year-over-year (YoY). In contrast, `X-Robots-Tag` implementation has stayed roughly the same YoY. Inner pages are 50% likely to have a robots directive, compared to home pages which are 46%.

Some pages (0.4% on both desktop and mobile) have both `meta` robots and `X-Robot-Tag` implemented at the same time. This figure of 0.4% is stable from last year but this is not a widely recommended practice, as it increases the likelihood of generating conflicting signals between the two methods of implementation.

Robots directives rules

The method of implementation is only part of the picture when it comes to robots directives; the directive rules[123] determine how the page or document should be handled.

Rules can be added to either `meta` robots or `X-Robots-Tag` as comma-separated values.

For our study of directive rules, we relied on the rendered HTML.

[123] https://developers.google.com/search/docs/crawling-indexing/robots-meta-tag#directives

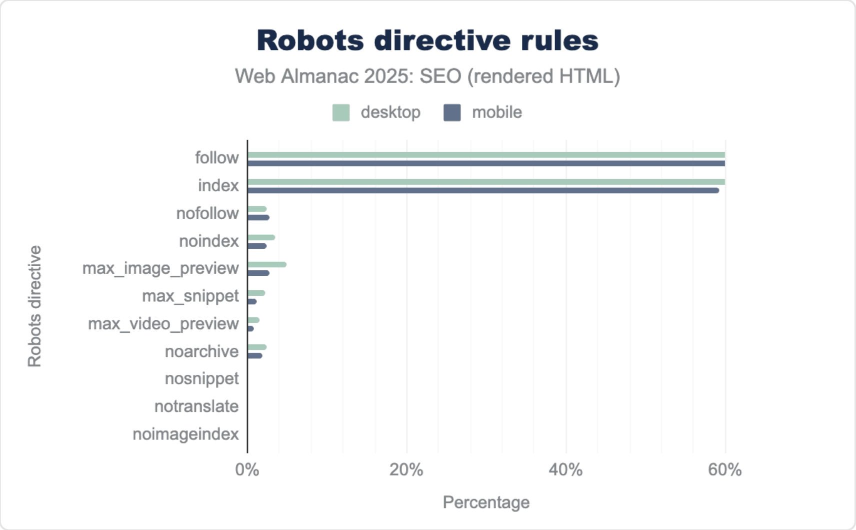


Figure 5.8. Robots directive rules

`follow` and `index`, the two most-used rules, are considered the default values (and are ignored by Google) in the absence of other rules like `noindex` and `nofollow`, which have the opposite purpose.

Their inclusion means that robots should index the page and follow the links from it. Their mobile usage at 60.5% and 59.3% for `follow` and `index`, respectively, implies these two tags are likely to be found together and are generally complementary.

A possible cause for this high number of a technically unnecessary combination of `meta` robots rules is Yoast SEO, which applies `index,follow` by default. Yoast has an approximate 16% adoption (desktop and mobile) when looking at home page use of SEO tools/plugins and, of all identified SEO tools[124], it is used nearly 70% of the time.

`nofollow` and `noindex`, the next two most-used `meta` robots rules, are used at a considerably lower frequency, showing up on 2.8% of desktop pages and 2.4% of mobile pages.

[124] https://www.wappalyzer.com/technologies/seo/

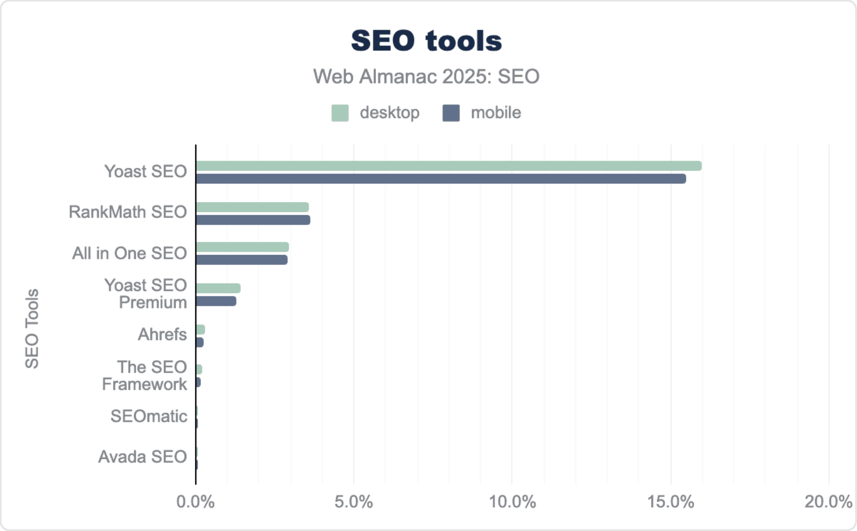


Figure 5.9. Most common robots directive implementation

Another element of `meta` robots directives is the `name` attribute, which enables us to specify whether these directives address a specific user-agent of a robot/crawler. We can, for example, tailor behavior if some crawlers require specific rules and others do not.

The most-named crawlers within this attribute are consistent with 2024: `bingbot`, `msnbot`, `googlebot`, and `google-news` appear most frequently, alongside the default `robots`.

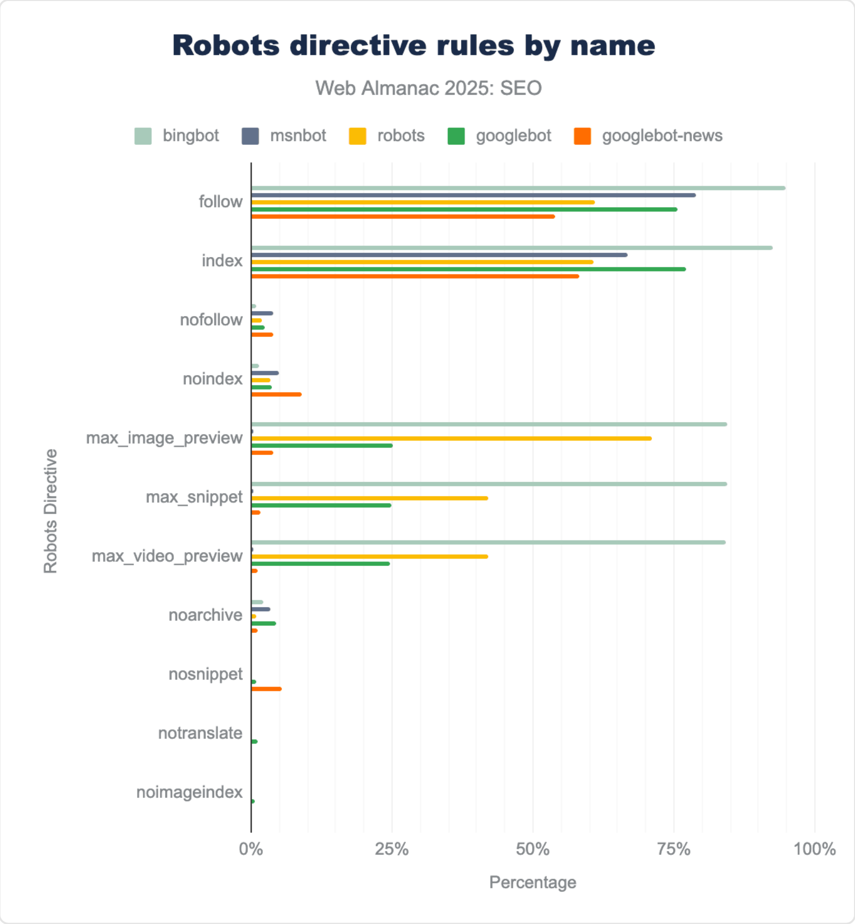


Figure 5.10. Robots directive rules by name

`msnbot` is a legacy crawler that has been replaced by `bingbot`. Its continued presence in robots directives suggests a delay in updating or removing outdated crawler names. Additional evidence of this lag can be seen in the fact that newer robots rules—such as `max_image_preview`, `max_snippet`, and `max_video_preview`—are commonly applied to `googlebot` and `bingbot`, but not to `msnbot`.

`indexifembedded` tag

A now well-established Meta robots rule, `indexifembedded` is a highly specific tag that enables us to specify when we want iframe content to be treated as being part of the page which it is embedded on. This needs to be paired with `noindex` in order to work, and is a rule which only Google currently supports.

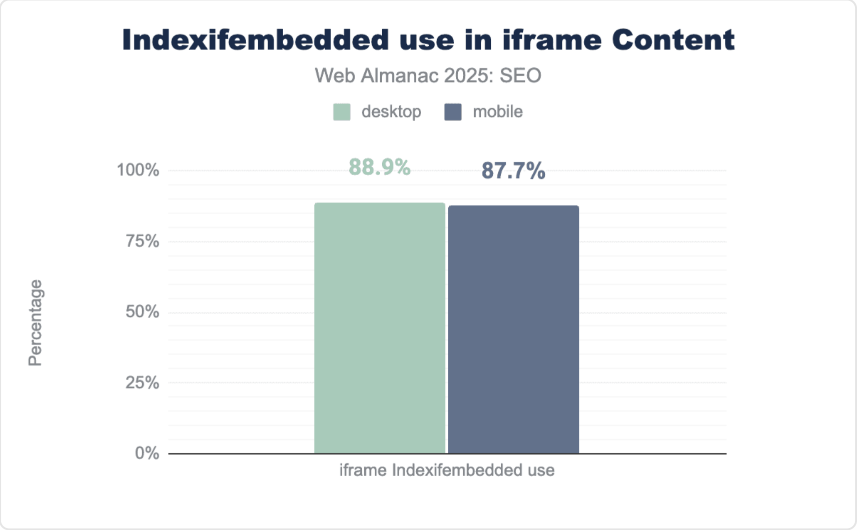


Figure 5.11. `indexifembedded` usage in `iframe` content

Within iframe content, the `indexifembedded` rule is found almost 90% of the time (88.9% on desktop, 87.7% on mobile). What is interesting is that after seeing raises in use from 2022 to 2024, 2025 saw a decline from 99.9% use.

Invalid `<head>` elements

Search engine crawlers follow HTML standards when parsing content. One issue they may encounter is when invalid HTML elements are found within the `<head>` of the page. This can cause the `<head>` to be treated as implicitly ending early, and all remaining `<head>` elements are then included within the `<body>` of the page.

Negative impacts to SEO are greatest here when important meta data, such as `title`, `canonical` tags, hreflang, Meta robots directives, etc., are located below the invalid `head` element (as their inclusion within the `body` renders them ineffective).

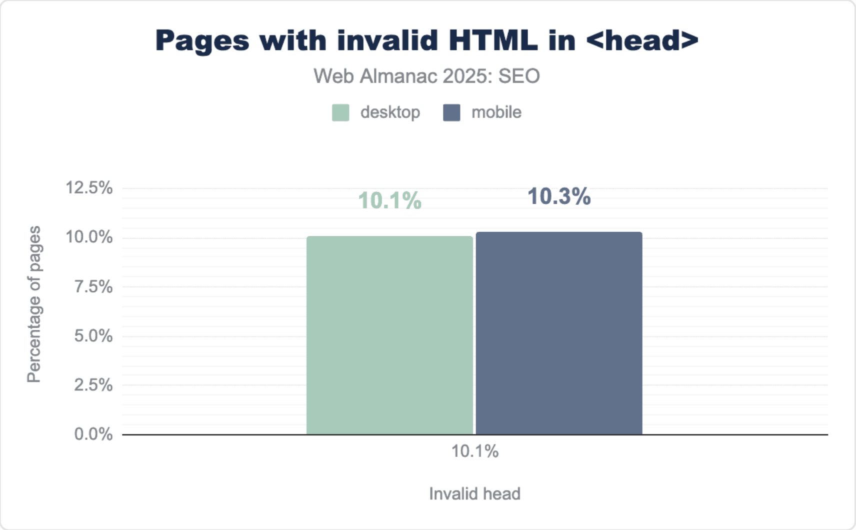


Figure 5.12. Pages with invalid HTML in `<head>`

Invalid head elements found this year are continuing the downward trend we saw in 2024's data; that is, we are once again seeing fewer invalid elements in pages' `<head>` sections year over year.

In 2025, we are seeing 10.1% invalid `<head>` elements on desktop pages and 10.3% invalid elements on mobile pages. Compared to 2024's data (10.6% on desktop and 10.9% on mobile), this represents drops of 4.7% (desktop) and 5.2% (mobile).

The definition of "invalid" `<head>` elements includes anything included in the page's `<head>` that is not based on W3C standards. There are eight valid elements that may be used in the `<head>`, according to Google Search documentation[125]. These are:

- `<title>`
- `<meta>`
- `<link>`
- `<script>`
- `<style>`
- `<base>`

[125] https://developers.google.com/search/docs/crawling-indexing/valid-page-metadata#use-valid-elements-in-the-head-element

- `<noscript>`
- `<template>`

The invalid elements which are most prevalent in 2025 are the same as they were in 2024's data: `<img>`, `<div>`, and `<a>` tags.

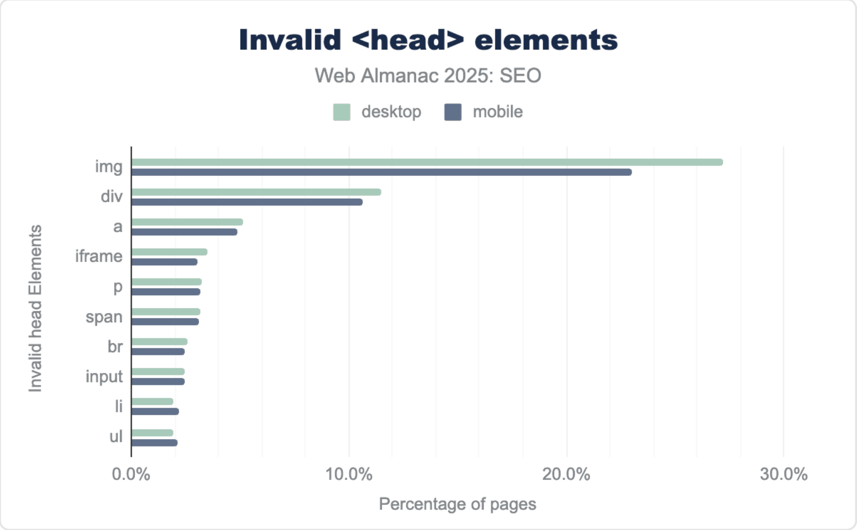


Figure 5.13. Invalid `head` elements

For invalid `<img>` tags, 2024 saw a sharp increase from 2022, whereas 2025's results are more stable (27.2% on desktop, 23% mobile this year versus 29% on desktop and 22% on mobile last year). Specifically, desktop usage has dropped, while mobile increased, albeit, by a small proportion.

`<div>` tags have no noticeable change at 11.5% on desktop and 10.6% on mobile. `<a>` tags similarly have seen no significant change from last year at 5.2% (desktop) and 4.9% (mobile).

Canonicalization

`canonical` tags tell search engines which version of a page is the main one when there are duplicates or near-duplicates. Without them, crawlers may split ranking signals, waste crawl budget, and show the wrong URL in search results. With them, all authority and indexing preferences point to a single, definitive page.

`canonical` tags are not directives (like `meta` robots); they are "strong hints" which provide a supporting signal that minimizes ranking issues through their correct implementation.

We have seen another increase in `canonical` tag usage, up from 65% on both mobile and desktop last year, to 68% (desktop) and 67% (mobile) in 2025.

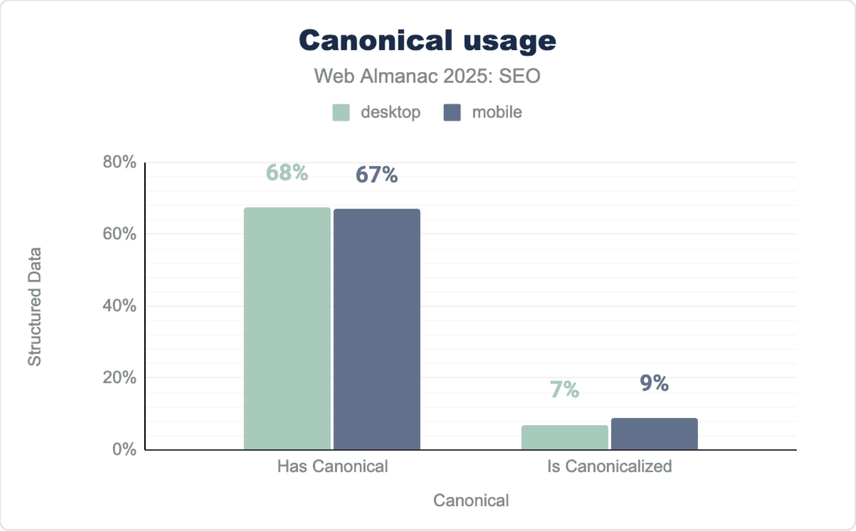


Figure 5.14. `canonical` usage

A page is "canonicalized" when its canonical URL points to a different location. Canonicalized pages have also seen an increase this year, up from 6% (desktop) and 8% (mobile) in 2024 to 7% (desktop) and 9% (mobile) in 2025.

`canonical` implementation

`Canonical` tags, similar to Meta robots, can be implemented as part of the HTTP response or within the `head` of the page. The motivations behind each implementation method may be dependent on your platform and requirements, with associated pros and cons.

`Canonicals` within the `head` of the page (`rel="canonical"`) are the most frequently utilized (as they are standardized by most CMS) and signal the canonical URL for the page.

HTTP canonicals can do the same and have the added advantage of being able to be used with other file types, such as PDFs. They are, however, not as widely implemented and require more technical understanding.

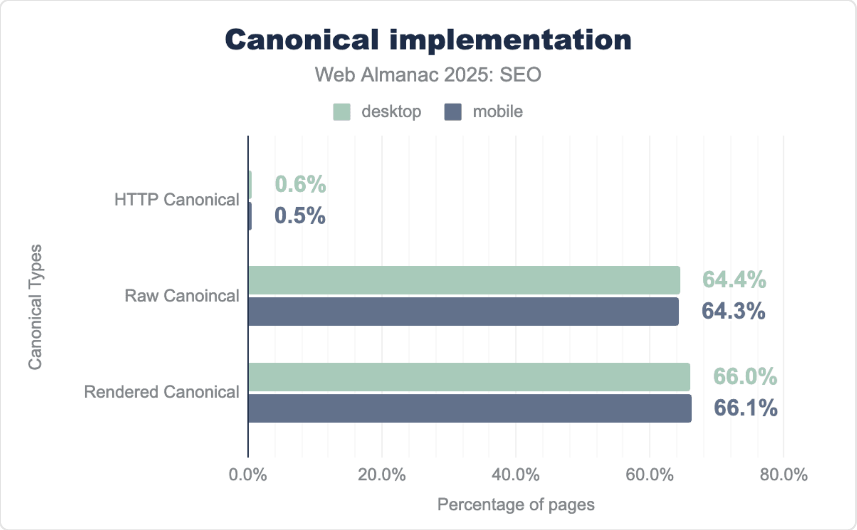


Figure 5.15. `canonical` implementation

HTTP canonical implementation is still very low in comparison to `rel="canonical"` tags, sitting at 0.6% (desktop) and 0.5% (mobile) in 2025. These data represent only small drops against 2024's numbers, which came in at 0.7% (desktop) and 0.6% (mobile).

Raw versus rendered canonical tags

`rel=canonical` tags, placed within the `head` of a page, can appear in the raw HTML, the rendered HTML, or both, depending on how the site is built. The most reliable approach, however, is to include a `canonical` tag in the raw HTML to ensure it remains unchanged after rendering.

For JavaScript-driven pages, the canonical tag is often inserted or updated in the browser once the page finishes rendering. The raw/rendered canonical results show us that only in around 2% of cases is there a canonical missing in the raw HTML but present in the rendered DOM. This indicates that most sites opt to include a canonical in the raw HTML, and it is not then removed at page render.

In 2025's data, we see that the raw HTML canonical numbers at 64.4% (desktop) and 64.27% (mobile) are on par with 2024's data. The rendered canonicals, however, have seen an increase in 2025 to 66% (desktop) 66.1% (mobile), compared to 64% (desktop) and 65% (mobile) last year.

Canonical conflicts

While rendering the canonical tag within the `head` using javascript can work, it introduces more room for error, so a static, server-side canonical tag is generally the safer option. Where the canonical tags mismatch, it undermines their usage and can cause unexpected results. Canonical mismatches saw a 0.1% drop from 2024's numbers, decreasing 0.8% (on desktop and mobile) to 0.7% this year.

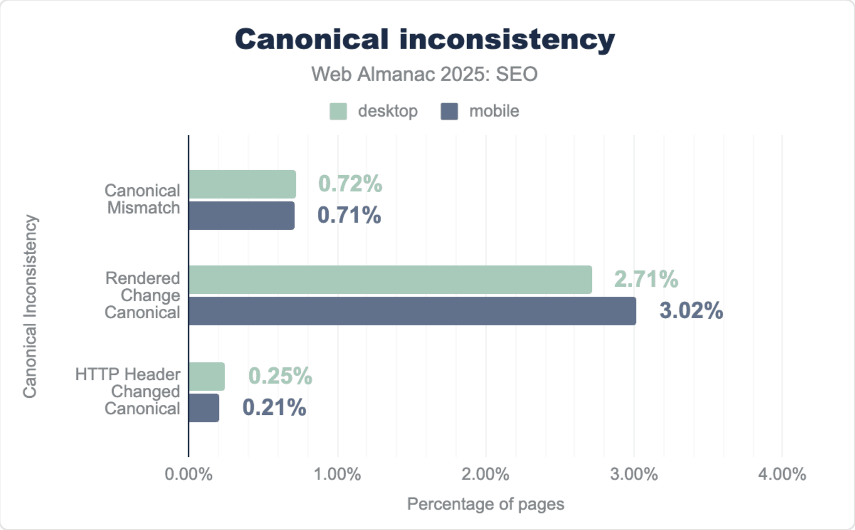


Figure 5.16. `canonical` *inconsistency*

Page experience

Page experience captures how users actually feel when interacting with a web page beyond what the content says or how well it ranks. Originally introduced by Google as part of its ranking systems in 2020, these signals measure usability factors such as page speed, mobile responsiveness, visual stability, and security. While Google has since folded the standalone page experience system into its core ranking framework[126], the underlying metrics continue to guide both SEO and UX best practices.

In 2025, the data reflects a web that's technically mature but still uneven in experience quality. HTTPS adoption is nearly universal, mobile-friendliness is now an expectation rather than a differentiator, and Core Web Vitals have become the industry standard for quantifying real-

[126] *https://developers.google.com/search/blog/2020/11/timing-for-page-experience*

world performance. Yet, even with these advances, friction points, ranging from slow rendering to inconsistent interactivity, still persist, reminding site owners that user experience is as much about consistency as compliance.

HTTPS

HTTPS was introduced to secure data exchanges between users and websites, protecting information from interception or tampering. It later became a formal ranking signal[127] in Google's algorithm.

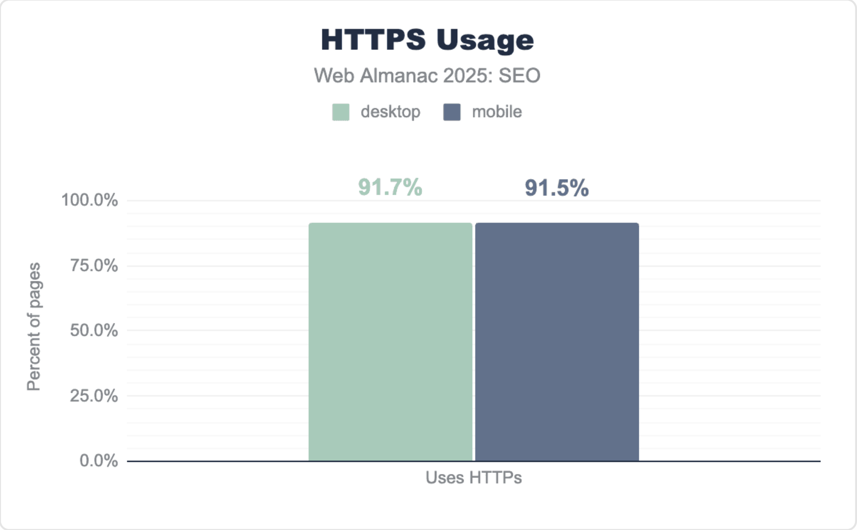


Figure 5.17. HTTPS usage

In 2025, HTTPS usage reached 91.7% of desktop pages and 91.5% of mobile pages, up slightly from around 89% across devices in 2024. Growth has slowed as HTTPS has effectively become the default for most of the web. The small remaining gap represents older or low-maintenance sites that have yet to migrate. As a result, HTTPS now functions less as a differentiator and more as a baseline expectation for both users and search engines.

Still, full universal encryption isn't trivial. Google's transparency reports[128] note persistent obstacles. Some countries and organisations block or degrade HTTPS traffic, and organizations with limited technical capacity may deprioritize it. Even Google's products face challenges, such as managing certificates for user-owned domains (e.g. in Blogger), as it can be complex when

[127] *https://developers.google.com/search/blog/2014/08/https-as-ranking-signal*
[128] *https://transparencyreport.google.com/https/overview*

those domains don't natively support HTTPS. These constraints help explain why a small share of sites still lag behind.

Mobile-friendliness

Google completed its migration to mobile-first indexing in 2023, meaning the mobile version of websites now determines search rankings. The following data examines how widely sites have adopted mobile-friendly design practices in response.

`viewport` meta tag

The `viewport` meta tag defines how a page scales and displays across different screen sizes, making it a key element of responsive web design. Introduced with the rise of mobile browsing, it ensures that content adapts smoothly to various devices without forcing users to zoom or scroll horizontally.

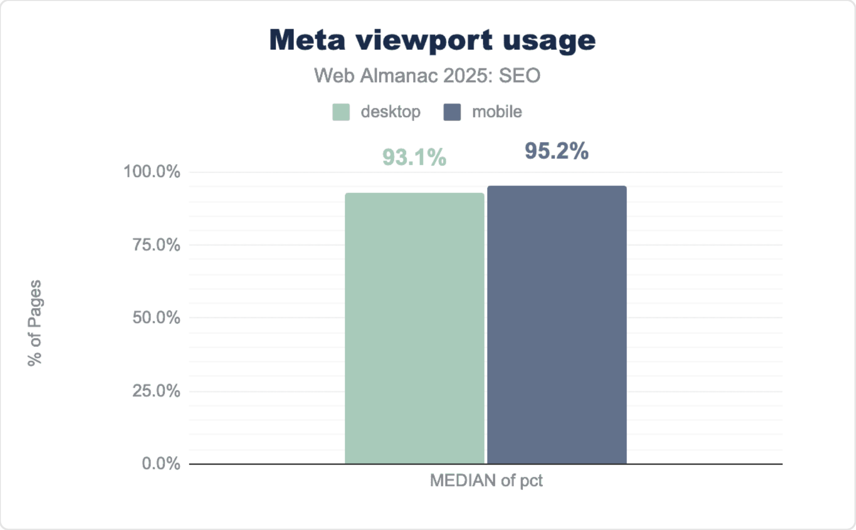


Figure 5.18. `meta viewport` usage

Use of the `viewport` meta tag is almost universal, appearing on 93.1% of desktop pages and 95.2% of mobile pages. This shows that responsive design principles are widely used across modern websites. The narrow desktop–mobile difference means developers are using consistent responsive design approaches across all devices rather than maintaining separate mobile and desktop codebases, signaling some maturity in mobile-friendly design practices.

`vary` header usage

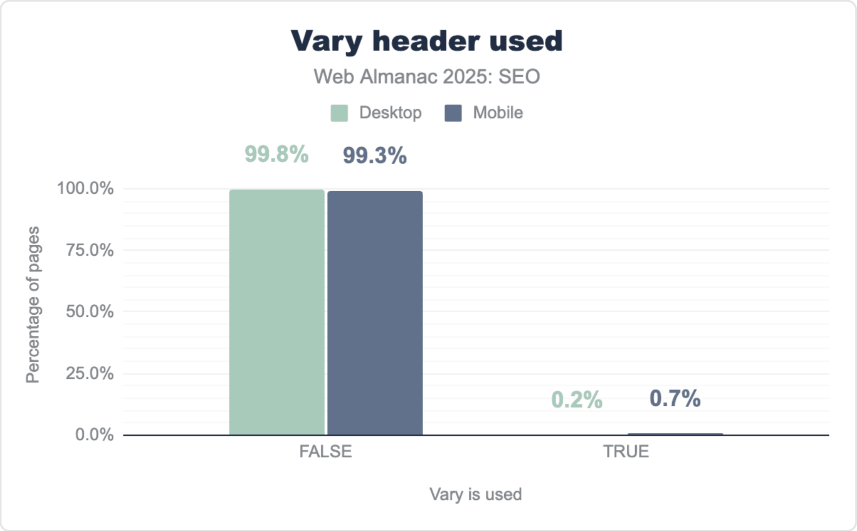


Figure 5.19. `Vary` *header user*

Once a standard tool for dynamic serving, the `Vary: User-Agent header` is now nearly extinct. In 2025, only about 1% of pages include it, with nearly all sites opting for unified rendering. This shift aligns with the widespread adoption of the `viewport` meta tag (over 95%) and Google's move to mobile-first indexing, which reduced the need to maintain separate mobile and desktop versions of a site. The web has largely consolidated around responsive, single-version design.

Legible font sizes

Legibility is one of the most fundamental aspects of mobile experience. Users should be able to read content comfortably without zooming or straining, and accessibility standards reinforce that. According to the Web Content Accessibility Guidelines (WCAG[129]), body text should be at least 16px (1rem), with the ability to scale up to 200% without breaking layout or functionality.

Lighthouse's legible font size audit[130] uses a similar benchmark, flagging pages where less than 60% of visible text exceeds `12px`. In 2025, 92% of mobile pages passed this test, mirroring 2024's performance and indicating that most sites now meet basic readability standards across both home pages and inner pages.

[129] https://www.w3.org/WAI/WCAG21/Understanding/resize-text.html#
[130] https://developer.chrome.com/docs/lighthouse/seo/font-size

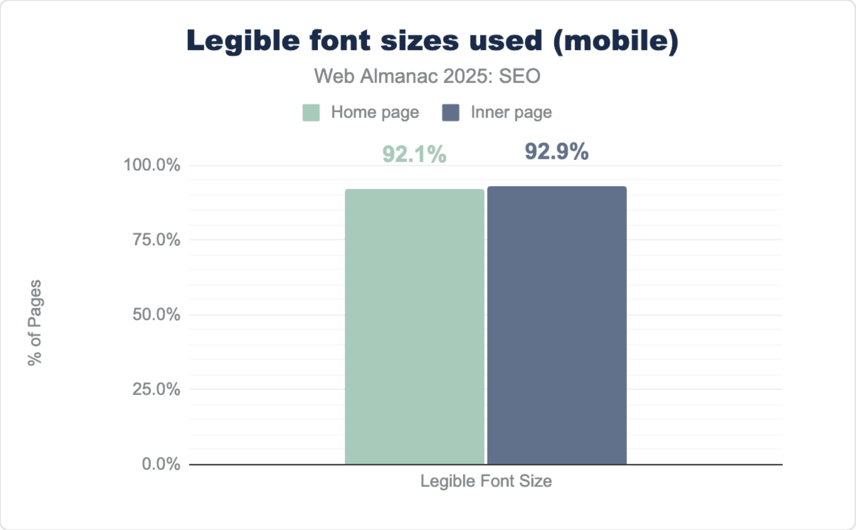


Figure 5.20. Legible font size used (mobile)

Core Web Vitals

Core Web Vitals measure how real users experience a page, focusing on loading (Largest Contentful Paint, LCP), interactivity (now measured by Interaction to Next Paint, INP), and visual stability (Cumulative Layout Shift, CLS).

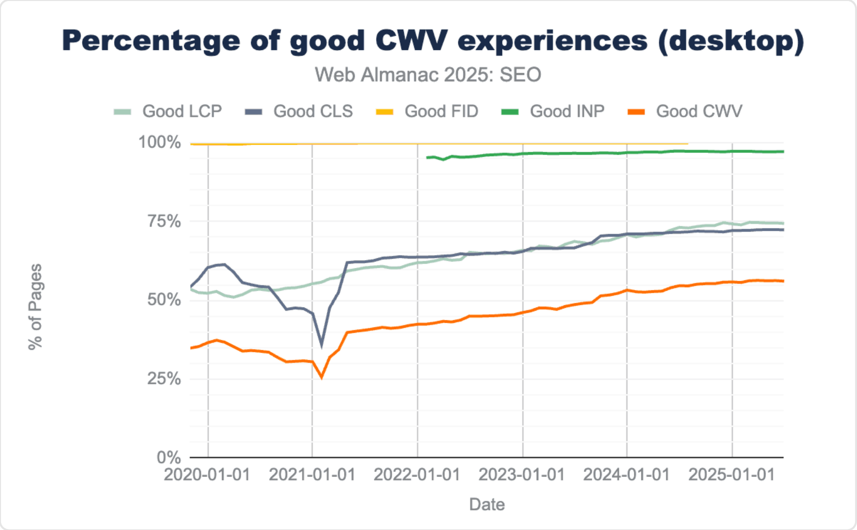


Figure 5.21. % of good Core Web Vitals experiences (desktop)

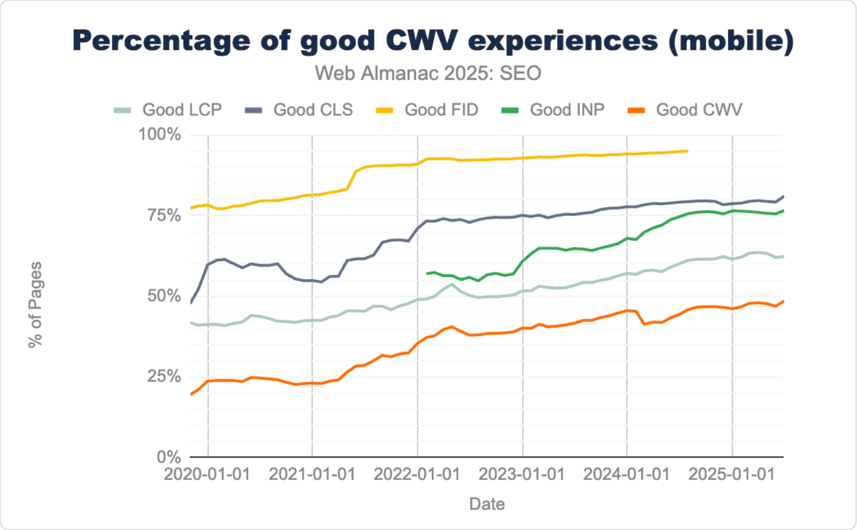


Figure 5.22. Percentage of good Core Web Vitals experiences (mobile)

In 2025, performance across Core Web Vitals has largely stabilized after several years of steady year-on-year improvement. Desktop continues to lead in Core Web Vitals performance, with around 60% of pages delivering a "good" overall Core Web Vitals experience, compared to just over 50% on mobile. The difference reflects desktop's inherent advantages, such as faster hardware, stronger connections, and fewer layout constraints.

Across both platforms, CLS remains the strongest metric, with around 75% of pages passing. However, LCP continues to lag near 55–60% on mobile, showing that loading speed remains the most persistent Core Web Vitals challenge for websites today. As research shows[131], this affects user satisfaction, and can also impact online conversions, particularly in an AI-driven landscape.

The visible dip from March 2024 particularly on mobile reflects the transition toward Google's March 2024 replacement of First Input Delay (FID) with Interaction to Next Paint (INP)[132]. INP began surfacing in performance tools ahead of the rollout, introducing stricter thresholds that exposed responsiveness issues FID often overlooked. This gradual shift raised the bar for what's considered a "good" user experience and explains the shift when the change took effect.

Image `loading` property usage

The `loading` attribute tells the browser when to fetch and render an image. it is a built-in performance hint that helps balance speed and resource use. Two main values shape how it behaves:

- `lazy` : delays loading images until they're close to appearing on screen, reducing initial page weight and improving perceived performance.
- `eager` : fetches images as soon as the page loads, ensuring instant visibility for above-the-fold visuals like hero banners or logos.

This attribute allows developers to fine-tune loading behavior for different contexts, prioritizing key visuals while deferring less critical ones to improve load times and bandwidth efficiency.

[131] https://www.speedcurve.com/blog/psychology-site-speed/
[132] https://searchengineland.com/google-replace-fid-inp-core-web-vitals-414546#:~:text=Interaction%20to%20Next%20Paint%20\(INP,nearly%20all\)%20interactions%20were%20below.&text=On%20the%20left%2C%20long%20tasks,rankings%2

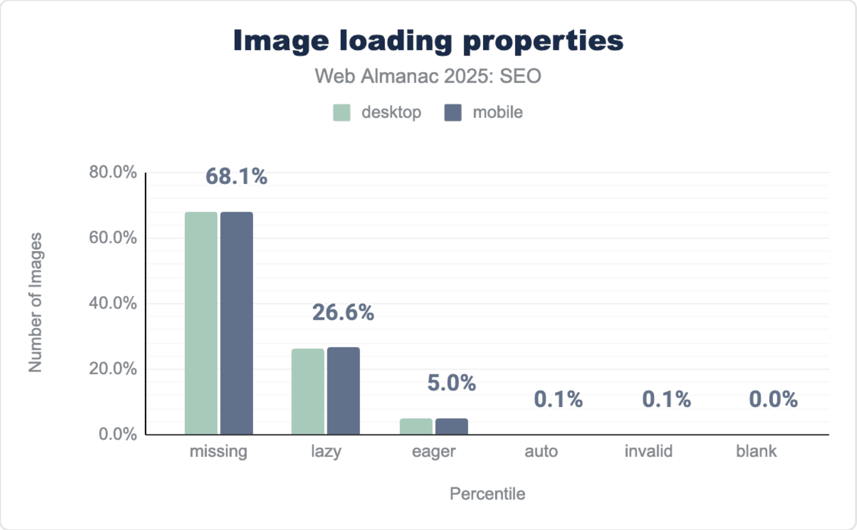


Figure 5.23. Image `loading` properties

In 2025, around 68% of images (across both desktop and mobile) have no image loading attribute set, leaving loading behavior to browser defaults. Meanwhile, around 26% use lazy loading, and just 4-5% explicitly use eager loading.

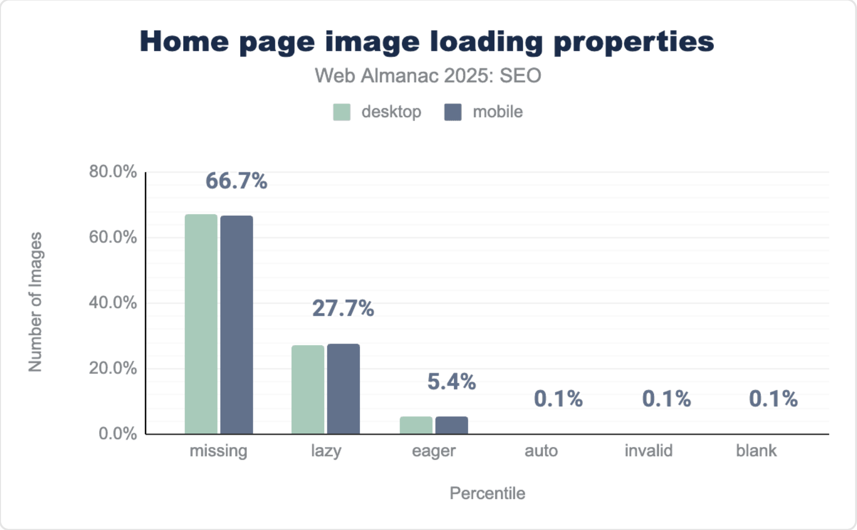


Figure 5.24. Home page Image `loading` properties

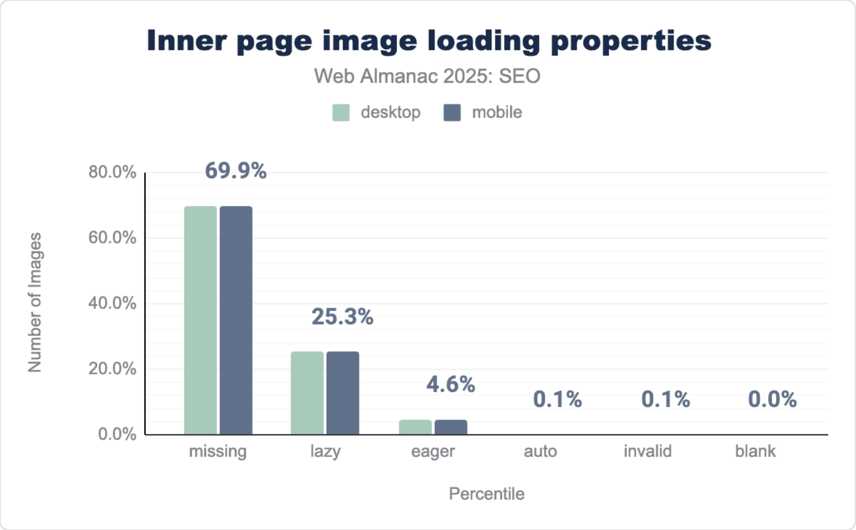


Figure 5.25. Inner page Image `loading` properties

The pattern is consistent across page types:

- Home pages: 27.3% (desktop) and 27.7% (mobile) lazy loaded
- Inner pages: 25.6% (desktop) and 25.3% (mobile) lazy loaded

The dominance of missing image loading attributes suggests lazy loading adoption remains concentrated among performance-focused sites, while many others (likely legacy or CMS-driven sites) haven't implemented explicit loading strategies. Given near-universal browser support for `loading="lazy"` and responsive design, the majority of images could benefit from explicit lazy loading to improve perceived performance and reduce bandwidth usage.

`lazy` loading versus `eager` loading iframes

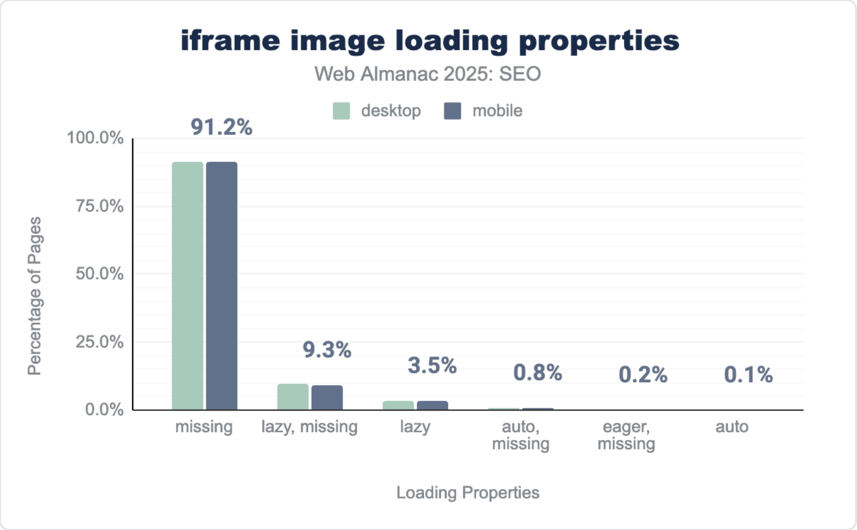


Figure 5.26. `iframe` image loading properties

Simply adding `loading="lazy"` to an iframe delays loading until it is near the viewport, saving around 500KB for a YouTube embed[133] or 200KB for a Facebook widget[134]. Yet, iframe loading attributes remain significantly underutilized in 2025, with 91.5% of desktop pages and 91.2% of mobile pages lacking any declared iframe loading properties—down only slightly from 92.8% (desktop) and 92.6% (mobile) in 2024.

Explicit iframe lazy loading attributes appear on roughly 13% of pages (3.5% standalone, plus 9.3% mixed with missing declarations), often inconsistently applied alongside non-declared iframes. As in previous years, this pattern likely reflects the dominance of embedded third-party elements such as ads, videos, and analytics dashboards, which publishers have limited control over. Explicit `eager` declarations remain virtually nonexistent (0.2% for both desktop and mobile), as they simply restate browser defaults. Despite the opportunity for substantial performance gains, iframe lazy loading adoption continues to lag far behind image lazy loading.

The inconsistency largely reflects how third-party embeds—such as ads, video players, and analytics dashboards—dominate iframe usage. Because these elements are controlled externally, publishers have limited ability to define loading behavior or add attributes directly. In many cases, third-party scripts inject iframes dynamically after load, which means crawlers and audits don't detect their `loading` attributes even when they exist. As a result, some of the

[133] https://web.dev/articles/iframe-lazy-loading
[134] https://web.dev/learn/performance/lazy-load-images-and-iframe-elements

apparent “missing” attributes may represent visibility gaps in measurement rather than true omissions.

This also ties to developer prioritization: teams typically focus on optimizing the performance of first-party assets (images, scripts, Core Web Vitals) before addressing iframe behavior, which is harder to control and yields smaller incremental gains.

On-Page

Page structure still matters in the current AI-driven landscape. Titles, meta descriptions, and headings communicate meaning and intent, while media attributes reinforce clarity and accessibility. The 2025 data shows overall stability in how these elements are used, with only minor shifts reflecting incremental adjustments rather than major changes.

Metadata

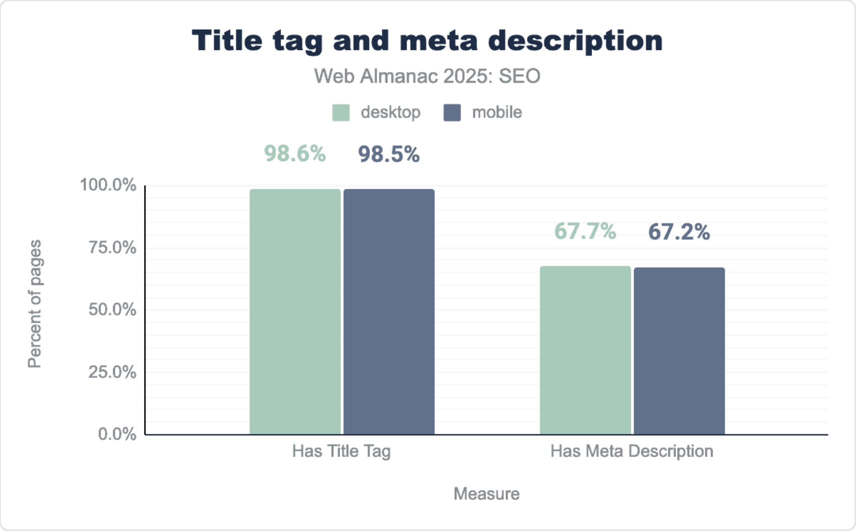


Figure 5.27. `title` tag and meta description

`title` tag adoption remains nearly universal, climbing slightly in 2025 to 98.62% of desktop pages and 98.54% of mobile pages, up from 2024’s 98.0% desktop and 98.2% mobile rates. This steadiness reflects the element’s enduring role in SEO strategy despite widespread title rewrites in search results. Publishers seem to still view title tags as foundational regardless of display control.

Meta descriptions, by contrast, have followed a different trajectory. After falling from appearing on 71% of pages in 2022 to 66.7% (desktop) and 66.4% (mobile) in 2024, they rebounded modestly in 2025 to 67.7% (desktop) and 67.2% (mobile). While adoption has not fully recovered to earlier 2022 levels, the uptick this year suggests publishers continue to see value in meta descriptions for click-through optimization and cross-platform consistency, even though Google may rewrite them.

`title` element length

`title` tags are one of the clearest signals of a page's topic for both users and algorithms, but their effectiveness depends partly on length. Too short, and they miss context; too long, and they risk truncation in search results. The 2025 data shows that publishers continue to cluster around stable midpoints, with little change from 2024.

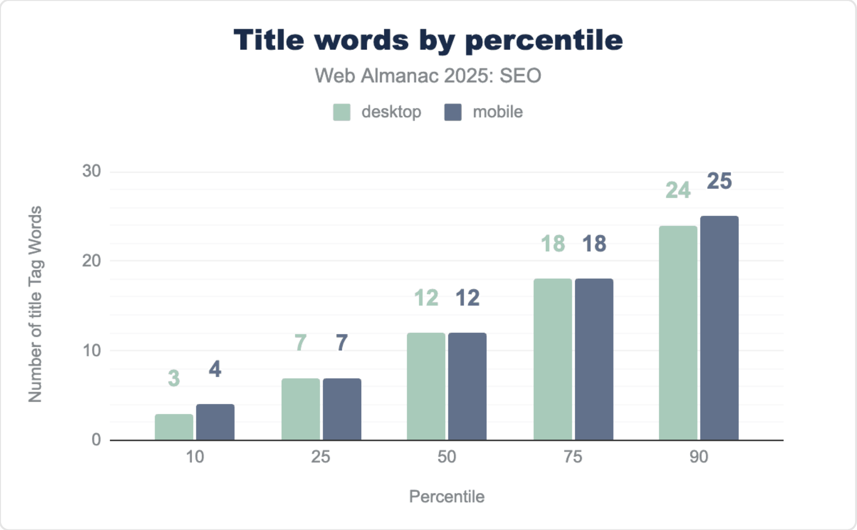


Figure 5.28. Title words by percentile

The median title tag contains 12 words across both desktop and mobile, unchanged from last year. At the 75th percentile, titles rise to 18 words, and at the 90th percentile, they extend to 24 words on desktop and 25 on mobile.

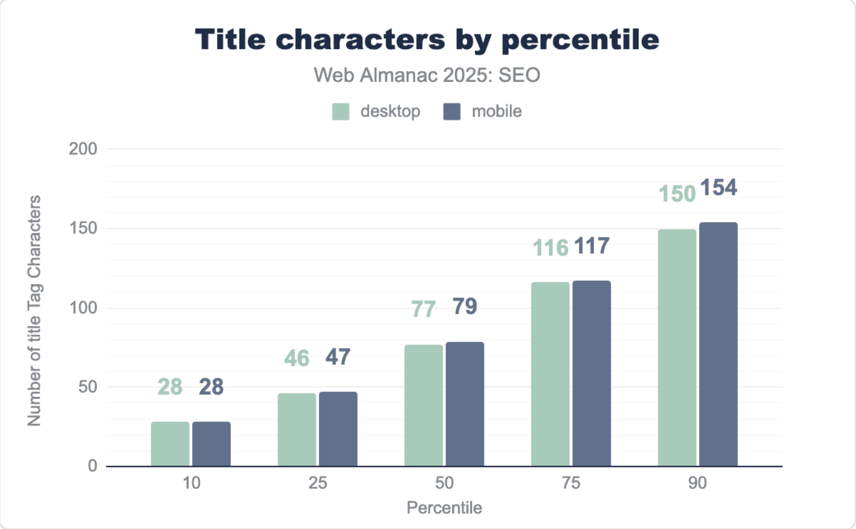


Figure 5.29. Title characters by percentile

The median title lengths sit at 77 characters on desktop and 79 on mobile, nearly identical to 2024. At the 75th percentile, lengths climb to 116–117, while the 90th percentile reaches 150–154 characters.

These numbers sit well above the often-cited 50–60 character “safe range” for titles, suggesting publishers are prioritizing keyword coverage and context even if display is truncated.

The persistence of this pattern from 2024 into 2025 may also reflect Google’s rewriting practices. Since titles are frequently modified in the SERPs, publishers face less incentive to fine-tune them to pixel-perfect display limits.

Meta description length

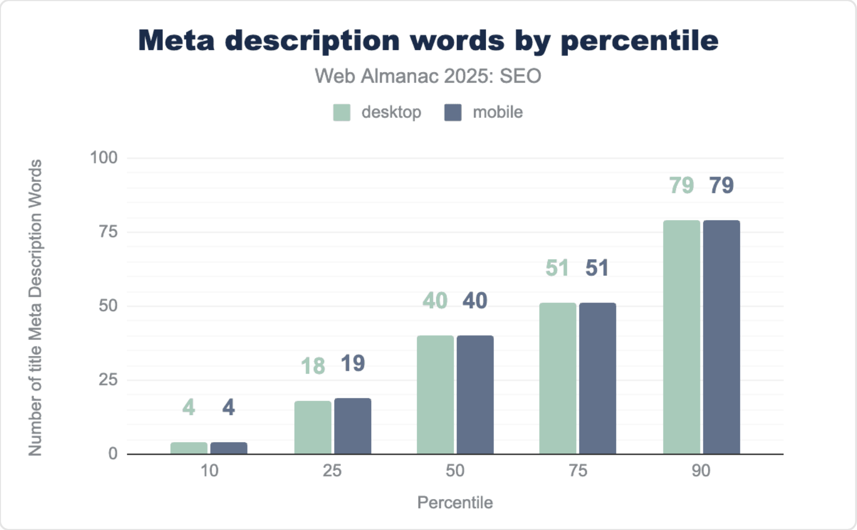


Figure 5.30. Meta description words by percentile

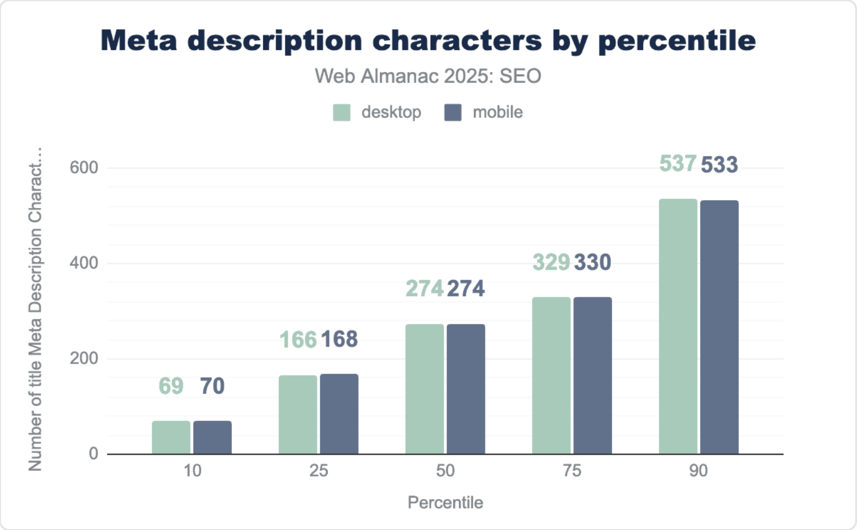


Figure 5.31. Meta description characters by percentile

Distribution patterns have largely stabilized after the sharp growth between 2022 and 2024. This year's data shows:

- **Median (50th percentile):** 40 words and 274 characters for both desktop and mobile, unchanged from 2024 but more than double the 2022 median of 19 words and ~135 characters.
- **10th percentile:** 4 words and ~69–70 characters, the same as last year.
- **25th percentile:** 18–19 words and 166–168 characters.
- **75th percentile:** 51 words and 329–330 characters.
- **90th percentile:** 79 words and 533–537 characters.

This distribution confirms that publishers have settled into a comfortable range: long enough to provide substance but generally capped below ~80 words or ~540 characters. Even with Google frequently rewriting snippets, maintaining well-structured descriptions gives site owners a measure of influence over how their content is presented elsewhere.

In practice, snippet display is constrained by pixel width, (roughly 920–980px[135] on desktop and 680px on mobile) so truncation varies by character width.

If this pattern holds, well-structured meta descriptions become more than traditional SERP signals: they help determine whether a page is even surfaced for AI-driven summarization or citation.

Heading tags

Well-structured headings do more than format text; they map the logical flow of a page, guiding both readers and crawlers through its main ideas. The 2025 data shows stability across all major tags, with only marginal shifts from last year.

[135] *https://ux.stackexchange.com/questions/125770/does-the-980px-screen-width-rule-still-stand*

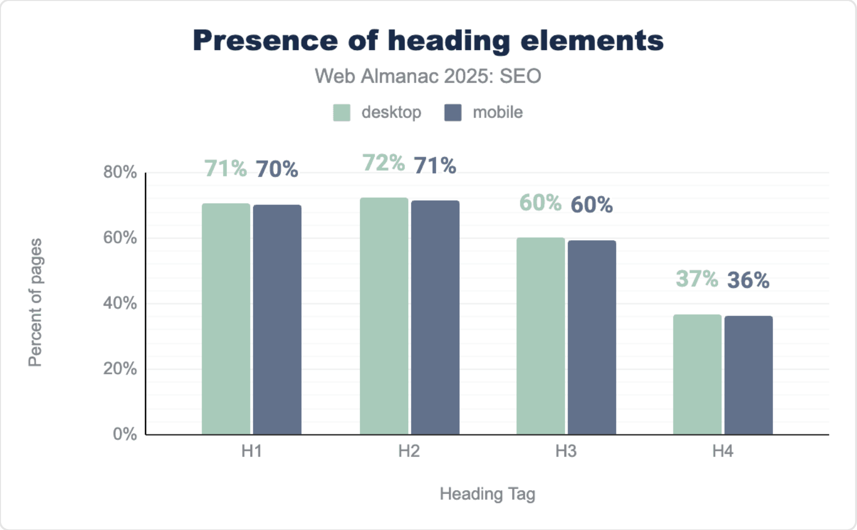


Figure 5.32. Presence of heading elements

- `<h1>` : Present on 71% of desktop pages and 70% of mobile pages. Non-empty usage is slightly lower at 66% for both.
- `<h2>` : The most widely adopted, at 72% of desktop pages and 71% of mobile, with nearly all non-empty.
- `<h3>` : Used by 60% of pages, with 58% non-empty, unchanged from 2024.
- `<h4>` : Present on 37% of desktop pages and 36% of mobile, with 35–36% containing meaningful content.

Adoption of heading tags has clearly leveled off. Earlier years saw `<h1>` growth and `<h3>` / `<h4>` declines, but 2025 confirms a steady pattern.

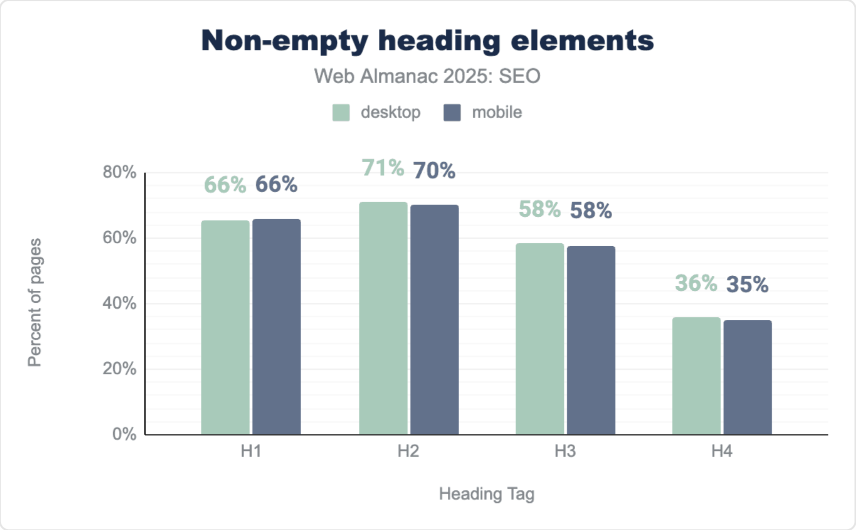


Figure 5.33. Non-empty heading elements

The non-empty data is especially telling. The gap between “present” and “non-empty” is now just a few percentage points, compared to wider gaps in 2022 (around 6% of desktop `<h1>`s were empty). This suggests cleaner, more semantically structured markup, with empty headings now the exception rather than the norm.

Beyond accessibility and SEO, this refinement in heading structure also aligns with a wider trend: publishers optimizing on-page clarity for AI-driven platforms. As AI overviews, summarizers, and citation systems increasingly rely on clean semantic hierarchies to extract and attribute information, well-structured headings have become part of ensuring a page’s ideas are accurately represented—and credited—in AI outputs.

Still, progress has limits. `<h1>`s appear on 70% of pages, but only 66% are non-empty. That 4% gap points to a ceiling effect, a natural upper limit where adoption is unlikely to reach 100%. Some designs, like single-page apps or minimalist home pages, deliberately skip meaningful `<h1>`s. These cases reflect design choices rather than errors.

Images

The 2025 data on image usage shows broad consistency with prior years, but also underscores how image use scales dramatically toward the upper end of sites.

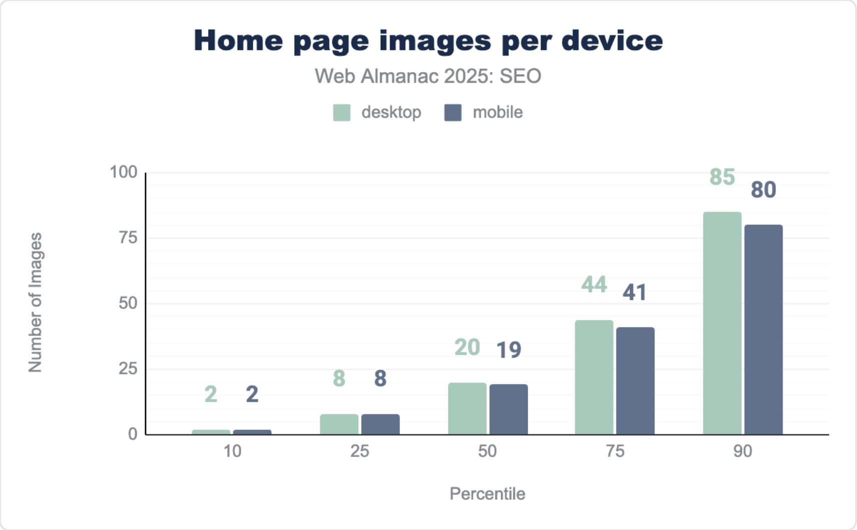


Figure 5.34. Home page images per device

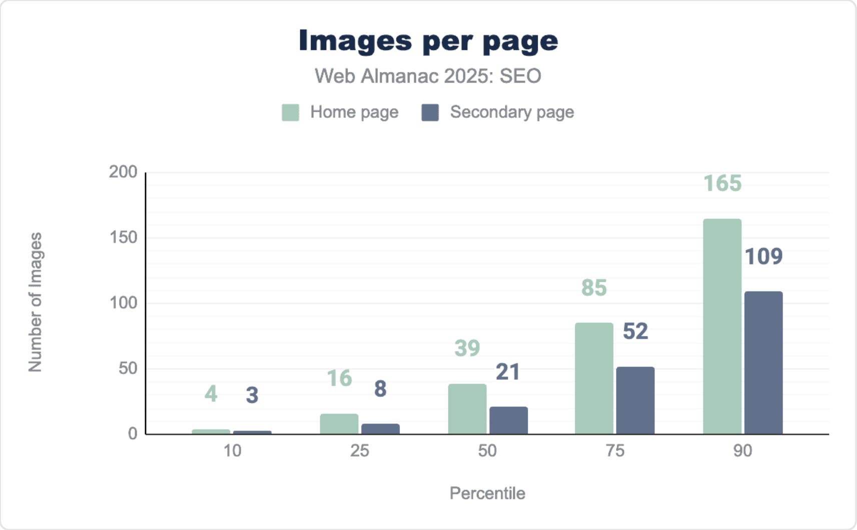


Figure 5.35. Images per page

- At the median (50th percentile), home pages include 20 images on desktop and 19 on mobile.
- At the 75th percentile, image counts rise to 44 images on desktop and 41 on mobile.

- At the 90th percentile, image use jumps even further to 85 on desktop and 80 on mobile.

By comparison, the 25th percentile sits at just 8 images, showing the wide variation in home page design. Yet across the distribution, desktop and mobile numbers remain closely aligned, suggesting responsive design practices continue to keep image use consistent across devices.

`alt` attributes

The `alt` attribute is central to accessibility and also supports image indexing in search. This year sees steady improvement in adoption, with fewer missing attributes overall, but also a slight rise in blank values worth noting.

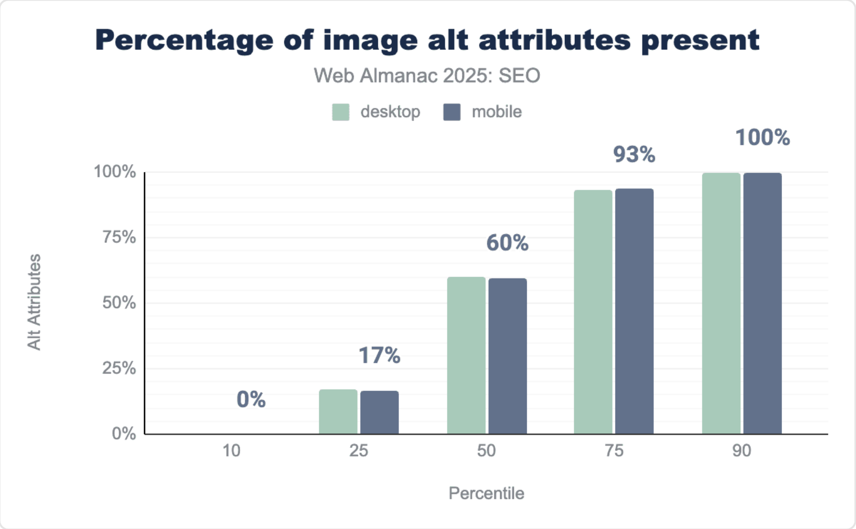


Figure 5.36. Percentage of image alt attributes present

At the median, 60% of images on mobile pages included `alt` attributes, up from 58% in 2024. By the 90th percentile, adoption reaches 100%, showing that a significant share of publishers consistently apply alt text to all images.

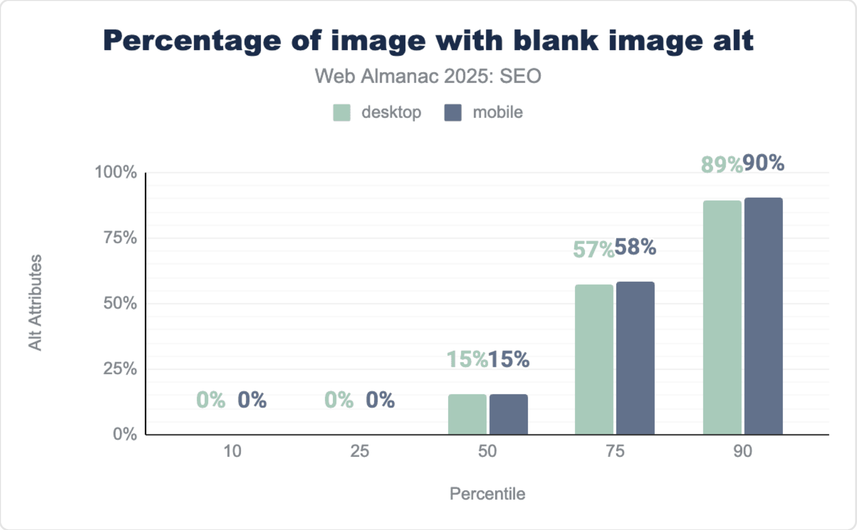


Figure 5.37. Percentage of image with blank image alt

At the median, 15% of images on mobile pages carried blank `alt` values, up just one point from 14% in 2024. At the 75th percentile, 58% were blank, and at the 90th percentile, 90% were blank—both also one point higher than last year. The increase is small, but it flags the concern that not all adoption translates into meaningful descriptions.

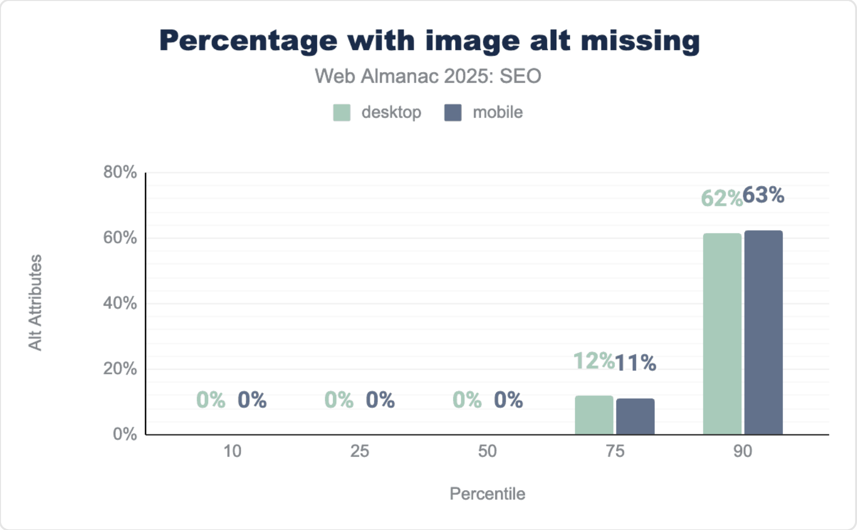


Figure 5.38. Percentage of image with alt image missing

The median page again shows no missing `alt`s, continuing the positive trend from 2022 when 12–13% of pages lacked them entirely.

The broader shift appears to be from "missing" to "blank". Images are more likely to have an attribute in place, but not always with descriptive content. While blanks are preferable to omissions, they offer limited value for screen readers and search engines.

Although the increase in blanks is small and steady rather than dramatic, it still underscores the gap between technical compliance and meaningful accessibility. With home page image counts ranging from as few as 8 to more than 80 in 2025, the importance of thoughtful alt text only grows. The web is clearly moving in the right direction, but the challenge is ensuring that progress in coverage also delivers progress in accessibility and meaning.

Video

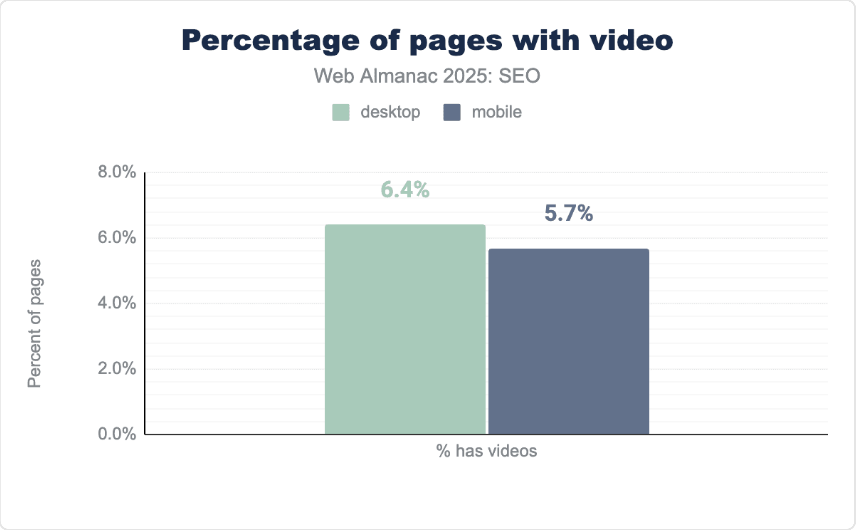


Figure 5.39. Percentage of pages with video

Video adoption on the web continues to grow, but only gradually. In 2025, 6.40% of desktop pages and 5.68% of mobile pages included a video element, up slightly from 5.87% (desktop) and 5.13% (mobile) in 2024. Desktop pages still feature video more often than mobile, repeating the same gap as last year.

The modest growth might reflect video's higher production and implementation costs compared to images. But the long-term trajectory remains upward. As SEO becomes more holistic and branding emerges as one of the strongest signals across the web, video is likely to play a larger role in reinforcing authority and cross-channel visibility.

The bigger challenge is optimization. While videos are appearing more often, only 0.9% of pages in 2024 used VideoObject markup, leaving most video content invisible to rich results. Until structured data adoption rises, much of video's SEO potential will remain untapped.

Links

Links are still a crucial signal signal of importance. However, the days of more links simply meaning higher rankings are largely over. Today link signals (internal and external) are balanced alongside content quality, relevance, and user experience.

Our 2025 data shows that link practices have matured, with stable patterns in descriptive anchors, internal navigation, and external references, alongside selective use of attributes like rel to qualify relationships.

Non-descriptive links

In 2025, most sites continue to provide descriptive anchor text rather than vague phrases like "Click here" or "Read more." This is reflected in Lighthouse test pass rates, which remain high across both home pages and inner pages.

- 2024: Home pages passed at 84% (desktop) and 91% (mobile), while inner pages were stronger at 86% (desktop) and 92% (mobile).
- 2025: Home pages are nearly unchanged at 84.64% (desktop) and 86.64% (mobile), but inner pages improved to 92.42% (desktop) and 93.45% (mobile).

The gap between home pages and inner pages persists, with inner pages consistently ~6–9 points higher. This likely reflects how home pages rely more on generic navigation and promotional CTAs, while inner pages use contextual, content-driven links that are naturally more descriptive. Device parity remains close, indicating that responsive design practices are effectively maintaining accessibility standards across devices.

Overall, descriptive link practices appear mature, with only marginal year-over-year changes. The remaining opportunity lies in improving home page link clarity, where vague CTAs are still most common.

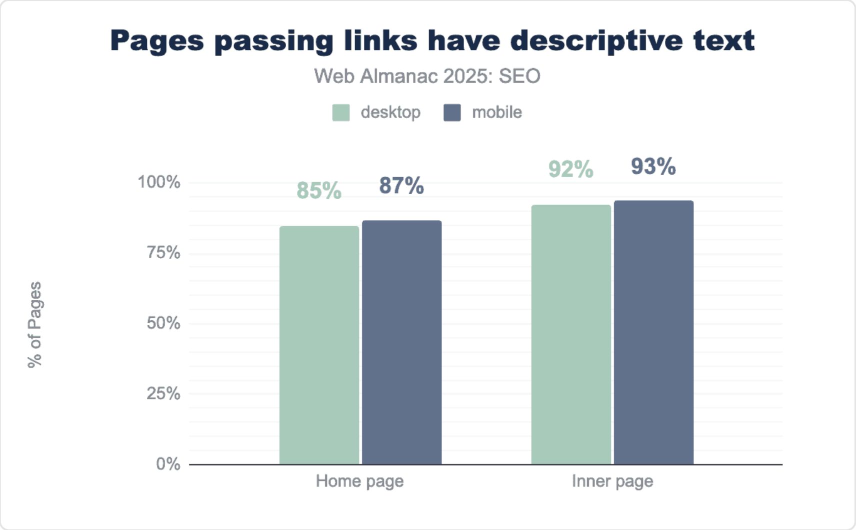


Figure 5.40. Pages passing links have descriptive text

Outgoing links

Outgoing links in 2025 see stable patterns for both internal and external linking. Internal links dominate; at the median, desktop pages contained 43 links to the same site and mobile pages 39, while at the 90th percentile those numbers jumped to 174 and 161.

This steep climb toward the high end might be a reflection of how larger or more complex sites (often news outlets, e-commerce platforms, or enterprise portals) surface extensive navigation through mega-menus and dense footers. Desktop pages consistently show a few more links than mobile, since developers streamline menus on smaller screens for ease of use, but they still keep the key pathways intact.

External linking remains far lighter. The median page included just six outbound links across both desktop and mobile, and even at the 90th percentile the numbers only reached 25 and 24. This continues the flat trend noted in 2024, which itself mirrored 2022.

Internal links have grown steadily over the years, while external links remain conservative, likely reflecting publisher focus on keeping users within their own ecosystems. Together, the figures suggest a mature equilibrium. Internal linking grows to support navigation and crawlability, while external linking holds steady at modest levels.

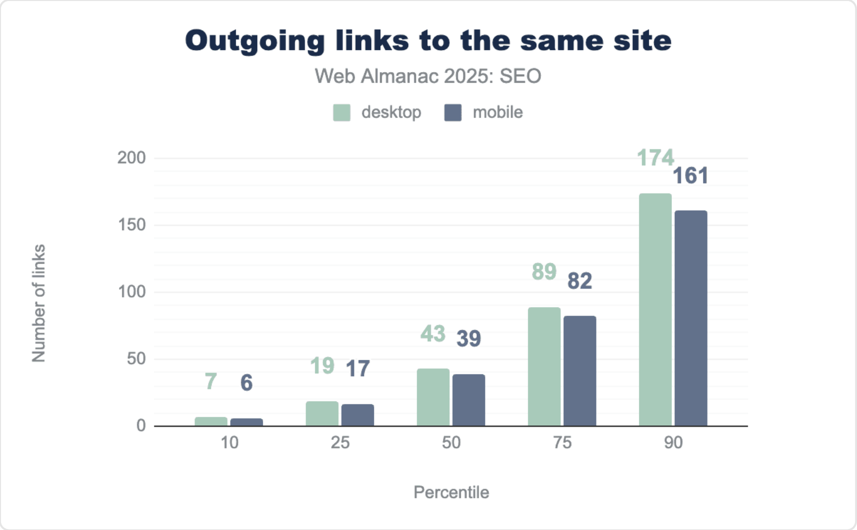


Figure 5.41. Outgoing links to the same site

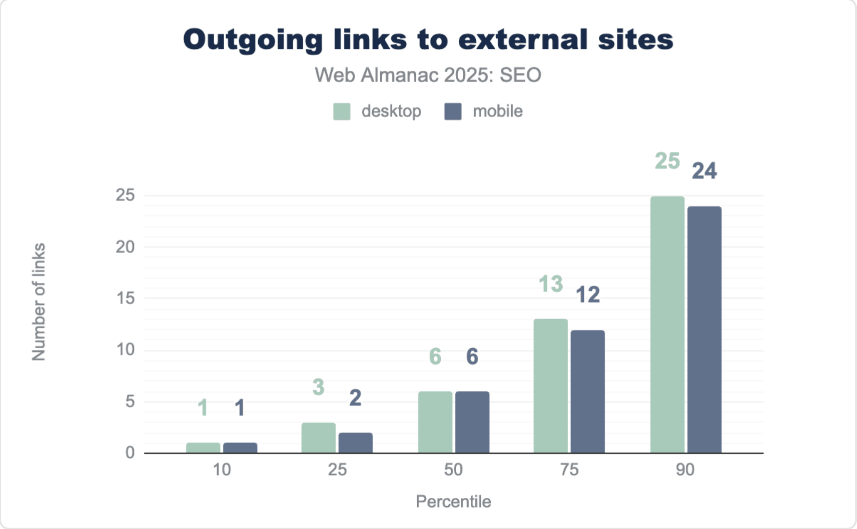


Figure 5.42. Outgoing links to external sites

Anchor `rel` attribute use

Although the `rel` attribute was originally intended to qualify links for search engines, its most common uses today are security-focused. In 2025, noopener appears on 40.9% of pages and noreferrer on 25%. Although neither affects positioning, both help increase site health and protect users by preventing tab hijacking and hiding referral data, which explains their widespread adoption.

For search-related qualifiers, `nofollow` continues to dominate at 32.3% of pages, nearly identical to 2024 levels. By contrast, the more specific `sponsored` and `ugc` attributes remain stuck at 0.5% each. Despite Google's efforts to promote finer-grained classifications, adoption has not moved, suggesting webmasters still see little value in going beyond the broad " `nofollow` versus normal link" distinction.

Taken together, the data shows that usage patterns for rel attributes have largely stabilized, with little year-over-year movement. Security practices lead usage, SEO qualifiers remain steady, and experimental attributes show no momentum. Even outdated attributes like dofollow and follow linger on fewer than 0.5% of pages, a reminder that old SEO myths still echo in small corners of the web without a practical effect.

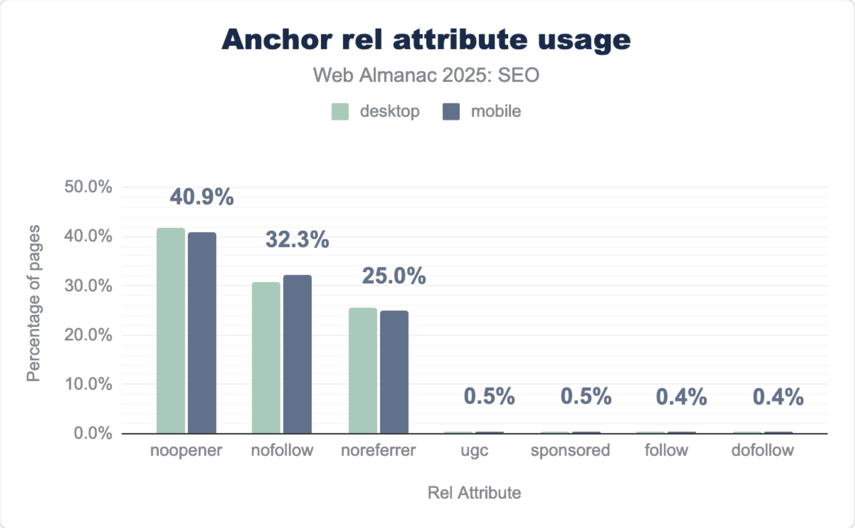


Figure 5.43. Anchor `rel` attribute usage

Word count

Word count has long been a point of discussion in SEO, often treated as a proxy for content quality or thoroughness. Despite this, there is little consensus on what constitutes an appropriate length for a page. In this section, we examine the distribution of word counts across raw and rendered pages.

Rendered word count

Rendered word count refers to the number of visible words on a page after JavaScript has been executed.

Home pages' rendered word count

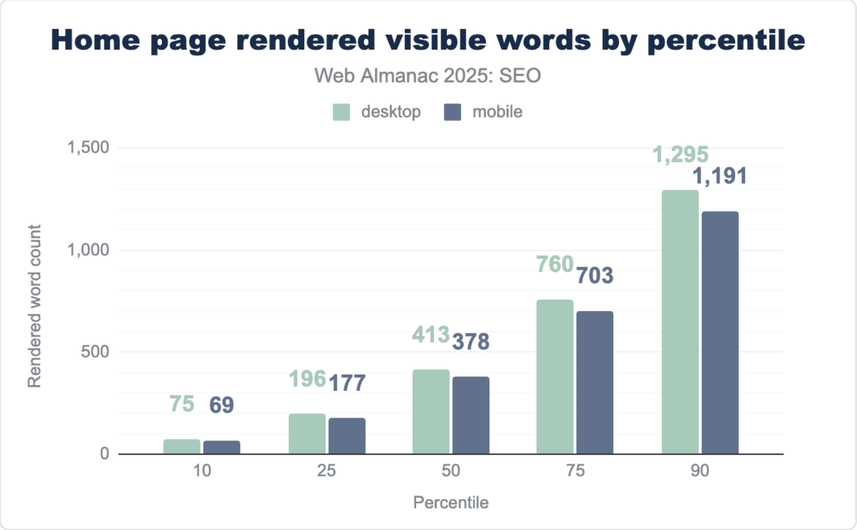


Figure 5.44. Home page rendered visible words by percentile

The median mobile home page in 2025 contained 378 words, up slightly from 2024 where the median was 364 words. The median rendered word count on desktop home pages was also up slightly year over year, with 2025 showing 413 words (up from 400 words in 2024). Both mobile and desktop rendered word counts have seen a decline since 2022, which had a median rendered word count of 421 words on desktop and 366 words on mobile. The parity between desktop and mobile is marginally closer in 2025 compared to last year, with the gap between mobile and desktop home page word counts narrowing to just 35 words, down by 1 from 2024's report where the desktop-mobile parity was 36 words. This indicates some consistency year over year.

Inner pages' rendered word count

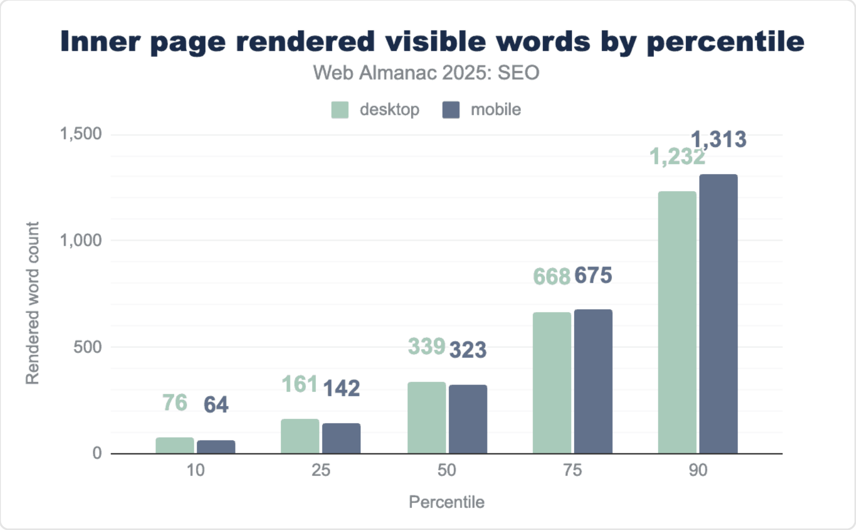


Figure 5.45. Inner page rendered visible words by percentile

Inner pages tend to contain fewer words compared to home pages. The median mobile inner page in 2025 had 339 (desktop) words, while mobile had 323 words (This is up from 2024 when the median desktop word count was 333 and the mobile word count was 317.)

Percentiles here represent the position of a page's word count relative to the entire dataset of inner pages—for example, the 90th percentile reflects pages with more words than 90% of all tested pages. Interestingly, up until the 50th percentile, the general trend is that desktop inner pages generally have more words than mobile inner pages, but as the percentiles get higher, the gap narrows. In the 90th percentile, there is a significant increase in mobile inner page word count compared to desktop. This trend is not replicated with home page content, where desktop always has higher word counts.

Raw word count

The raw word count refers to the total number of words included in the original HTML delivered by the server, prior to any JavaScript execution or subsequent alterations to the DOM or CSSOM.

Home pages' raw word count

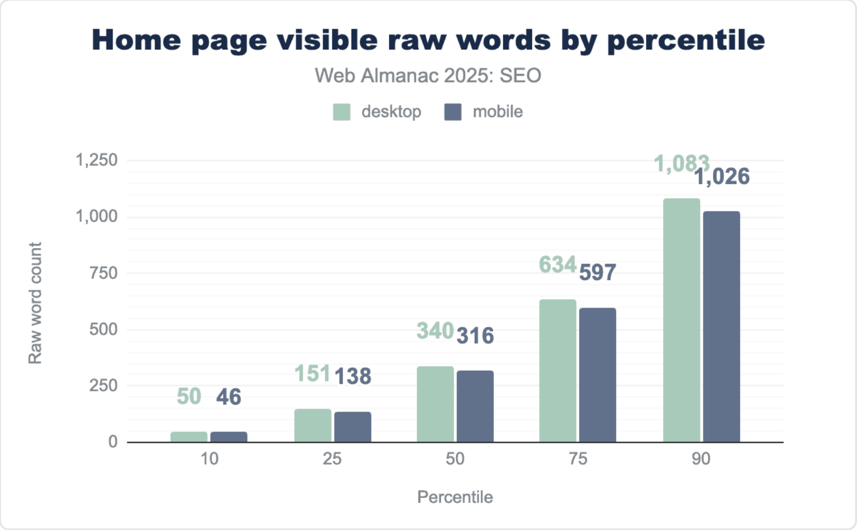


Figure 5.46. Home page visible raw words by percentile

The median page's raw word count in 2025 is 340 words for desktop user agents and 316 words for mobile—up a small amount from 2024 when the median page's raw word count was 330 words for desktop and 311 words for mobile.

Inner pages' raw word count

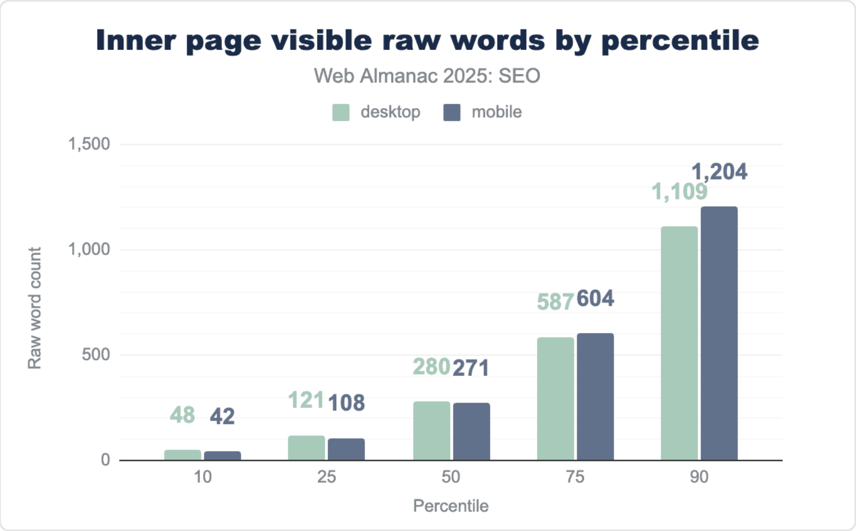


Figure 5.47. Inner page visible raw words by percentile

Unlike the raw word count on home pages where mobile did not contain more words than desktop in any percentile, the inner pages' visible word count showed desktop pages having fewer words than mobile pages at the 75th percentile and above.

Rendered versus raw word counts

When the rendered and raw word counts are compared on the home page, the discrepancy is small, with the median showing a difference of 18% on desktop versus 16% on mobile. This has seen growth on mobile, which was just 14% in 2024.

Across all percentiles, home pages served to desktop user agents have a higher percentage of words visible after rendering versus mobile. This is likely because of common layouts on mobile like accordions where content is hidden, even if it is rendering in the DOM.

The highest difference across all percentiles between mobile and desktop is just 2%, with some lower percentiles like the 10th percentile showing equal word counts.

Home pages' rendered versus raw word counts

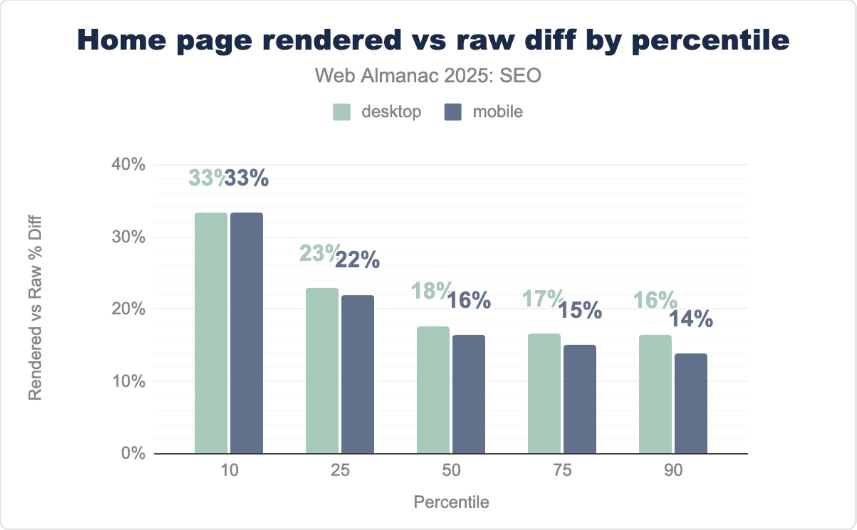


Figure 5.48. Home page rendered versus raw diff by percentile

In 2025, we saw a significant hike in the 10th percentile with 37% on desktop and 34% on mobile, compared to 28% on desktop and 22% on mobile in 2024.

Inner pages' rendered versus raw word counts

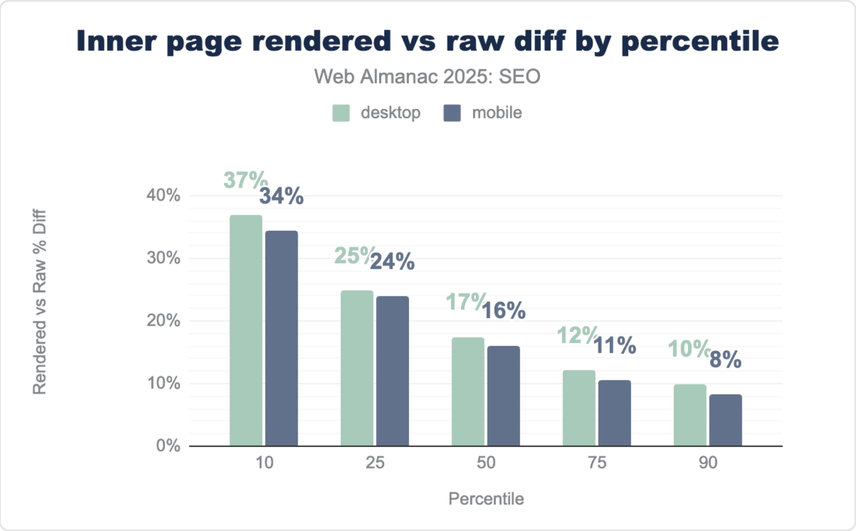


Figure 5.49. Inner page rendered versus raw diff by percentile

The gap between rendered and raw word counts on inner pages sees the biggest gap in the 10th percentile, likely due to reliance on content existing in the HTML but hidden on mobile breakpoints, such as within accordions and tabbed layouts. This closes the gap in the 25th, 50th, and 75th percentile with just a 1% difference and 2% in the 90th percentile.

The pattern seems to be the more words, the less they rely on client-side rendering, which was also the case in the 2024 chapter.

Structured Data

While not a direct ranking factor, structured data remains a powerful addition to websites to help search engines to understand the context of the page, while also powering rich results in search engine results pages (SERPs) .

Structured data[136] is increasingly being used by large language models to understand what a page represents, not just what it says. Microsoft has confirmed that Bing uses schema.org[137] markup to help its models (including Bing Chat and Copilot) distinguish between expert articles,

[136] *https://insightland.org/blog/structured-data-ai-search/*
[137] *http://schema.org*

products, reviews, and FAQs. Google's AI Overviews[138] behavior suggests a similar reliance, and crawlers such as GPTBot are capable of parsing schema embedded directly in HTML in some circumstances.

Home pages' structured data

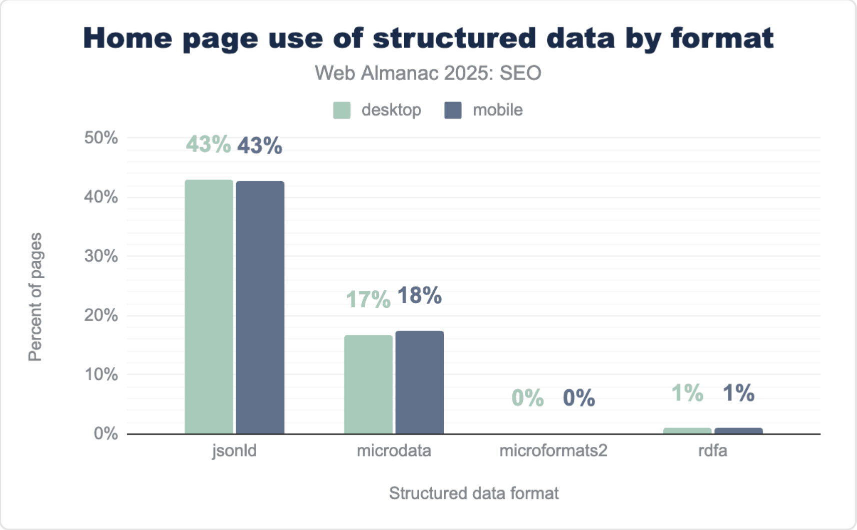


Figure 5.50. Home page use of structured data by format

Overall usage of structured data grew to 50% on both desktop and mobile in 2025, compared to 48% of desktop home pages and 49% of mobile home pages in 2024.

The majority of sites provide structured data in raw HTML, while just 2% of mobile and desktop crawls have structured data added via JavaScript (which has not changed compared to 2024).

While Google can process JavaScript-injected schema, providing it in the initial HTML avoids potential delays, crawling issues, and poor user experience.

JSON-LD remains by far the most popular format for home pages and accounts for 43% of both desktop and mobile home pages crawled compared to 41% of desktop and 40% of mobile home pages in 2024.

Google typically recommends using JSON-LD structured data[139] as it is the easiest form to be implemented and maintained, although Google can still crawl and understand other formats

[138] https://www.semrush.com/blog/how-can-schema-markup-specifically-enhance-llm-visibility/
[139] https://developers.google.com/search/docs/appearance/structured-data/intro-structured-data

like Microdata and RDFa. We have seen a slight drop in microdata usage, with both desktop and mobile reducing by 1% compared to 2024.

Inner pages' structured data

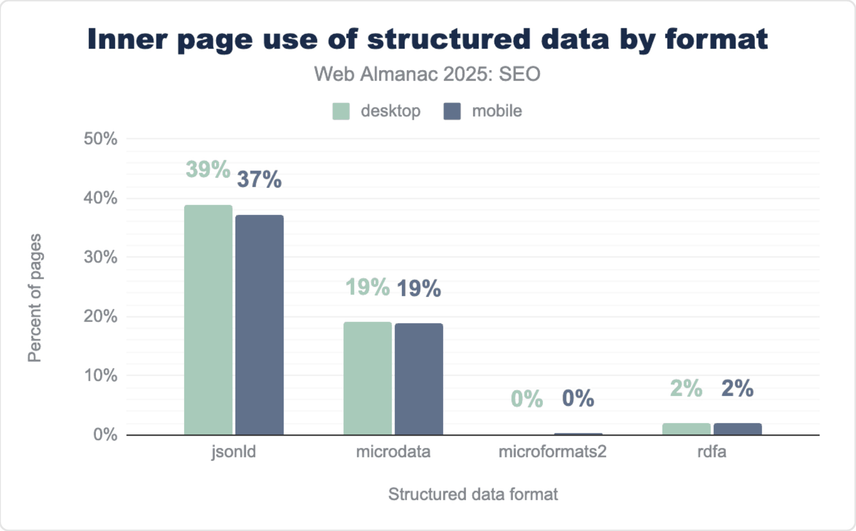


Figure 5.51. Inner page use of structured data by format

For inner pages, we see a slight difference, with microdata at 19% on mobile and desktop, and JSON-LD at 39% on desktop compared to 37% on mobile.

This may be because home pages usually receive the most SEO attention and are more likely to have JSON-LD implemented directly, often manually or through a global site template like Organization or Website schema. Because home pages are often prioritized for rich results and brand knowledge panels, JSON-LD adoption rates are typically higher and more consistent across devices.

In contrast, inner pages like product or article pages often rely on dynamic or CMS-generated markup that can vary between desktop and mobile templates, leading to some slight discrepancies.

Most popular structured data types

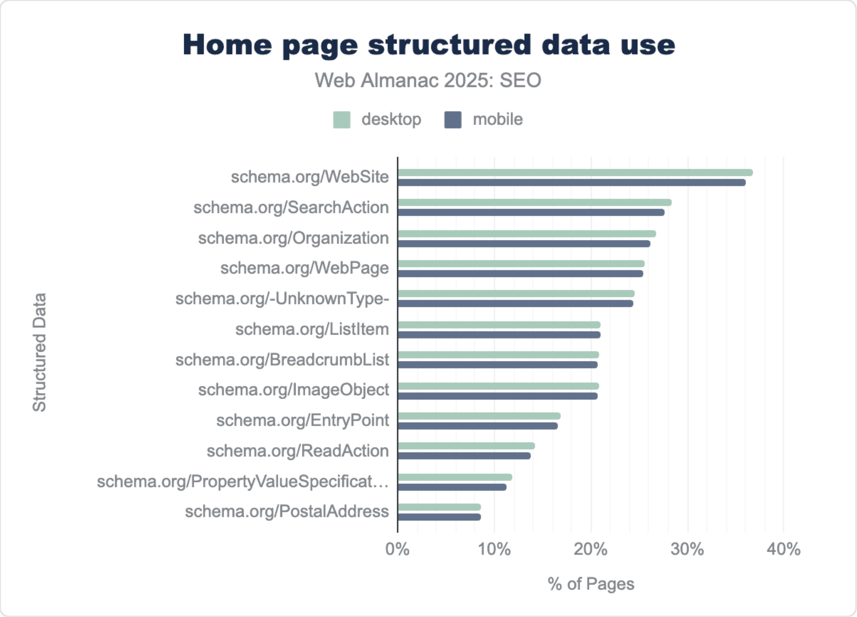


Figure 5.52. Home page structured data use

There hasn't been a major shift in the top 2 most popular structured data types, with WebSite and SearchAction being the most popular in 2022, 2024, and 2025. These types power Google's sitelinks search box which, visually, was sunset in November 2024, but there was no need to remove the structured data that supported it.

We have seen a slight increase in popularity for Organization schema, which overtook WebPage schema (not to be confused with Web**Site** schema) for the first time in 2025's data.

WebSite schema, still the most popular structured data type of all, has seen some slight growth in 2025 for mobile at 36% usage compared to 35% in 2024 and 30% in 2022. Mobile structured data usage showed only minor differences year over year.

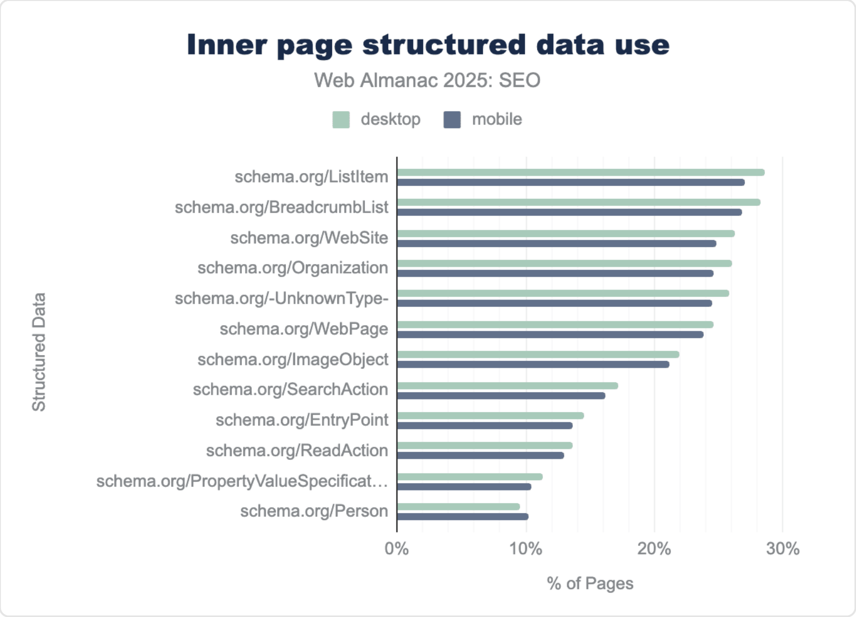


Figure 5.53. Inner page structured data use

On inner pages, ListItem structured data continues to be the most popular inner page schema type, although mobile usage has dropped from 30% in 2024 to 27% in 2025.

Interestingly, WebSite structured data is still the third most popular structured data for inner pages, despite it being a homepage-specific schema type (at least according to Google[140]). This may be due to many types of schema being templated at CMS-level, which may be duplicating it through to inner pages.

You can find out more about schema usage in our dedicated Structured Data chapter.

AMP

Accelerated Mobile Pages (AMP) were once promoted as a fast-track to performance and visibility, particularly in mobile search. But with Core Web Vitals now providing direct performance metrics for all pages, AMP's role has sharply diminished. The 2025 data shows adoption at consistently low levels across home and inner pages, with usage lingering below 1%. What remains reflects niche or legacy implementations rather than mainstream strategy.

[140] *https://developers.google.com/search/docs/appearance/structured-data/organization?_gl=1*jf94bx*_up*MQ..*_ga*OTY2MjgzMjIwLjE3NjIzMjkyODU.*_ga_SM8HXJ53K2*czE3NjIzMjkyODQkbzEkZzAkdDE3NjIzMjkzMDIkajQyJGwwJGgw#guidelines*

Home page AMP usage

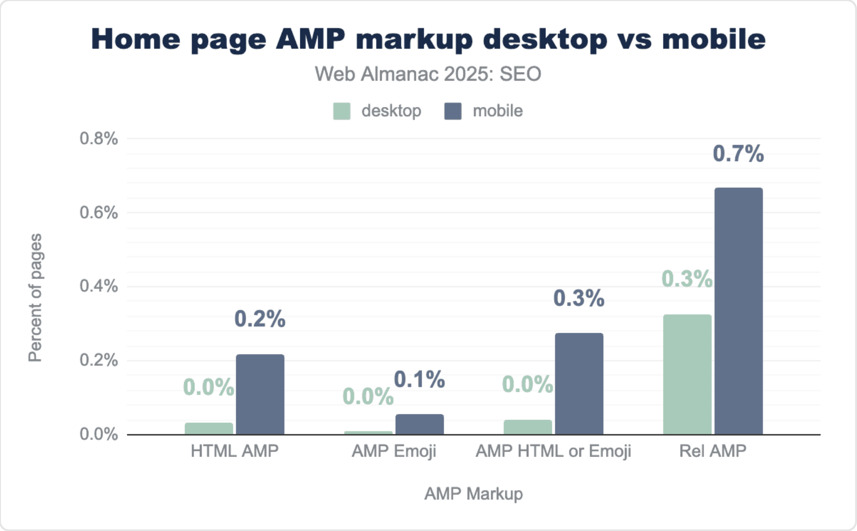


Figure 5.54. Home page AMP markup desktop versus mobile

Adoption of AMP markup stayed extremely low in 2025, with inner pages showing only marginal use across all device types. rel=amphtml is the most common signal, appearing on 0.8% of desktop pages and 0.9% of mobile pages, while HTML AMP is seen on just 0.1% of mobile pages and virtually none on desktop. Other variants like AMP Emoji or AMP HTML + Emoji are nearly absent, registering at or below 0.2%.

Compared to 2024, the overall picture is one of stagnation. The slight uptick noted in 2024 has not carried forward, and usage continues to hover well below 1%. The shift away from AMP after Google removed its Top Stories requirement and the rise of Core Web Vitals as a universal performance measure have effectively cemented AMP as a fringe technology.

Home pages versus inner pages' AMP usage

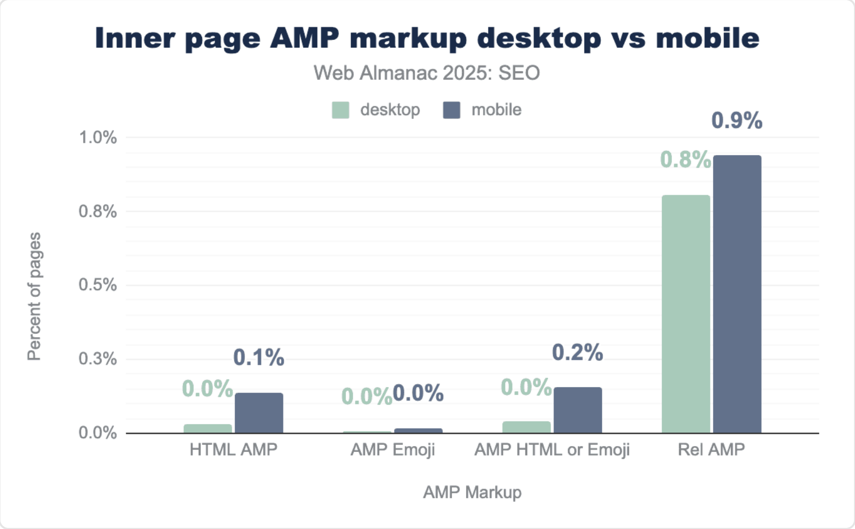


Figure 5.55. Inner page AMP markup desktop versus mobile

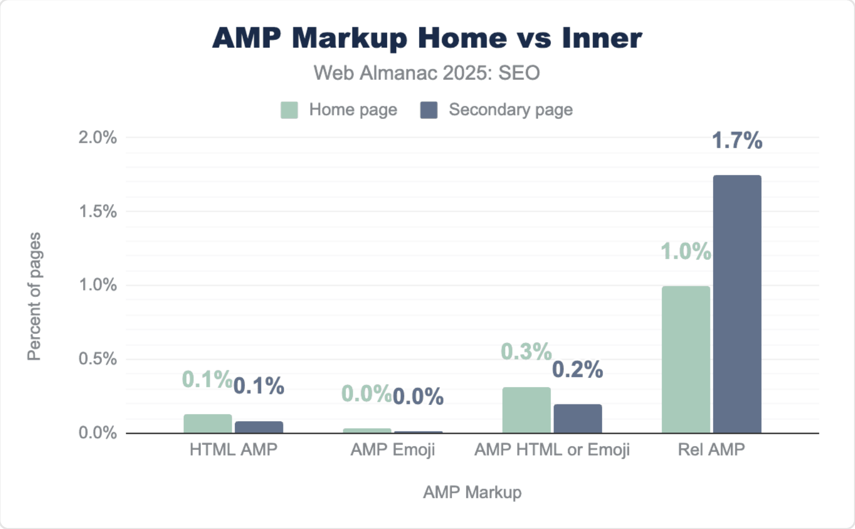


Figure 5.56. AMP Markup Home versus Inner

In 2024, home pages were slightly more likely to feature AMP than inner pages. By 2025, the opposite is true. `rel=amphtml` appears on 1.7% of inner pages compared with 1.0% of home pages. The same holds for HTML AMP, where adoption is now evenly split (0.1% each).

Internationalization

Being a critical feature for international and multilingual websites, the hreflang attribute helps search engines serve the right language or regional version of a page to users, a critical feature for international sites. In 2025, hreflang appears on roughly 20% of pages. Adoption is most often prioritized on home pages, where international targeting has the greatest impact, while inner pages see less consistent coverage. Growth has been steady but uneven, concentrated in a small set of widely used languages.

`hreflang` implementation

Around one in five pages now include hreflang markup; 20.3% raw (desktop) and 19.7% raw (mobile), with rendered values slightly higher at 20.8% and 20.2% respectively. This essentially doubles last year's rates, where only about 9–10% of pages carried hreflang tags.

HTTP hreflang is nearly gone, with just 0.2% of desktop pages and 0.1% of mobile pages using it. That's consistent with 2024's already tiny share and confirms HTTPS has become the default across almost all internationalized websites.

The close alignment between raw and rendered values (less than a 1% difference) suggests that most sites implementing hreflang are doing so in a technically correct way that survives rendering. This is an improvement over 2024, when gaps were slightly larger.

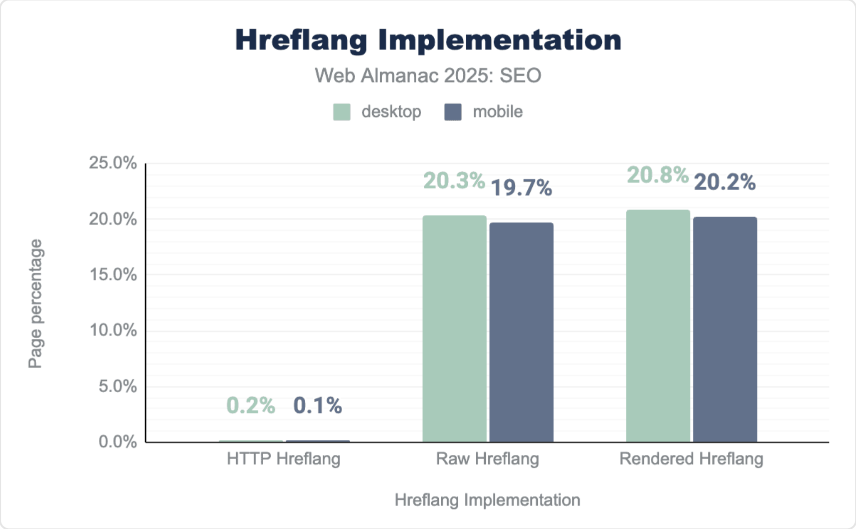


Figure 5.57. `hreflang` implementation

Home page `hreflang` usage

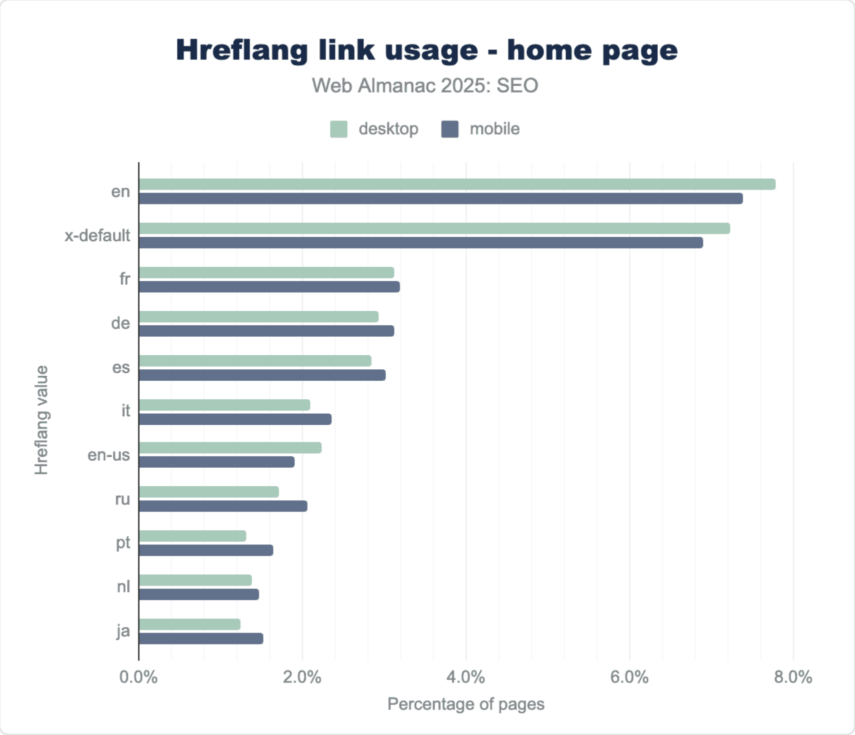


Figure 5.58. Hreflang link usage - home page

`hreflang` usage on home pages remains highly concentrated in a few values. English (`en`) continues to dominate, appearing on 7.4% of desktop home pages and 7.3% of mobile, followed closely by `x-default` at 7.2% (desktop) and 6.9% (mobile).

Secondary languages such as French (3.2% on mobile), German (3.1% mobile), and Spanish (3.0% mobile) round out the next tier, while others like Italian, Russian, Portuguese, Dutch, and Japanese remain below 2.5%.

This concentration underscores that while `hreflang` adoption has grown to cover 20% of pages overall, most of that usage is focused on a small set of widely targeted audiences.

Compared to 2024, when `en` appeared on 8% of both desktop and mobile home pages, the 2025 data shows a slight dip. However, adoption across secondary languages has stayed stable, keeping the overall distribution consistent year over year.

Inner page `hreflang` usage

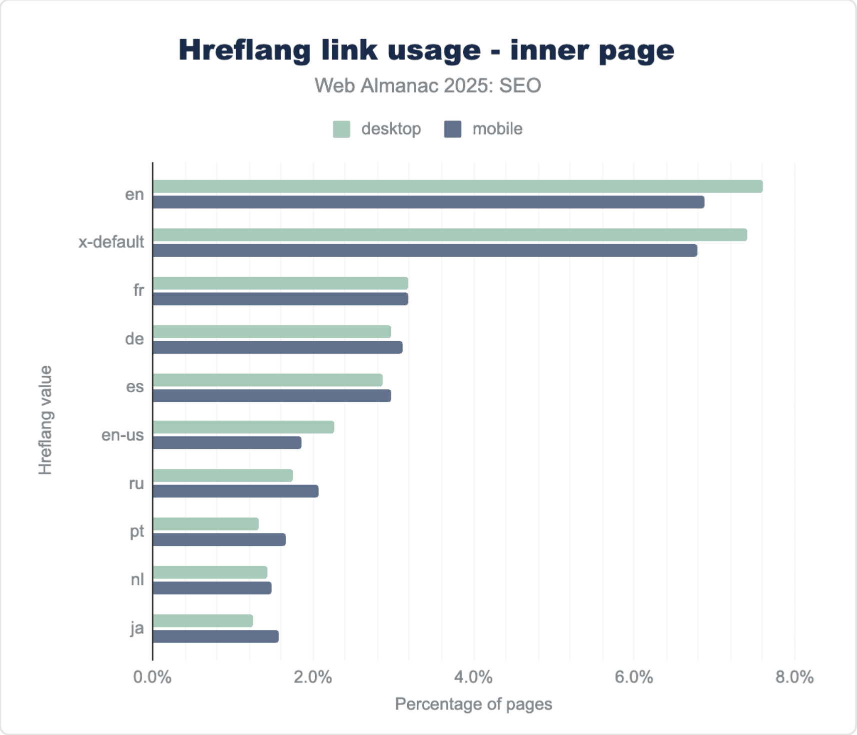


Figure 5.59. Hreflang link usage - inner page

On inner pages, hreflang adoption mirrors the home page pattern but at slightly lower levels. English (en) leads at 7.6% (on desktop pages) and 6.9% (mobile), followed closely by x-default at 7.4% (desktop) and 6.8% (mobile). Secondary languages such as French (3.2% of inner mobile pages), German (3.1% mobile), and Spanish (3.0% mobile) again form the next tier, while all other language values remain under 2.5%.

Compared to 2024, when desktop use of x-default and en was closer to 8%, the 2025 figures suggest a small decline. The distribution again confirms that hreflang is generally prioritized on home pages rather than inner pages. The tight clustering of French, German, and Spanish (3.0-3.2% on inner mobile pages) reinforces that major European markets drive most multilingual web adoption beyond English.

Conclusion

As AI search reshapes how content is discovered, the web's fundamentals matter more than ever. Reassuringly, the data suggests those foundations are holding firm.

Crawlability and indexability remain strong; most sites serve valid `robots.txt` files and `canonical` tags, and markup hygiene is improving year on year. The role of `robots.txt` itself is evolving, moving from a gatekeeper of access to a statement of intent about content use. Millions of files now explicitly address AI crawlers like `gptbot` and `claudebot`. Combined with solid `meta` robots adoption and stable canonical signals these technical pillars provide a reliable foundation for both human and machine-led search systems to work with.

Beyond these access signals, the web's middle layers—page experience, performance, and on-page structure—show encouraging stability. HTTPS adoption now exceeds 91%, Core Web Vitals have plateaued at historically high levels, and on-page metadata such as titles, descriptions, and headings are increasingly optimized for both search and AI-driven indexing. Structured data adoption has reached 50%, with most implementations embedded directly in HTML. All to ensure a faster, cleaner interpretation by crawlers and models alike.

`llms.txt` is the web's latest experiment in speaking directly to AI systems; site owners are being invited to think about how content is ranked, read, and reused by AI. Adoption is still low (around 2%) and mostly driven by CMS plugins rather than deliberate strategy, but its existence signals a shift in mindset. Whether `llms.txt` becomes a standard or a footnote is a core question to take into 2026.

Much of this progress stems from CMS platforms that now embed best practices by default, quietly raising the technical baseline across the web. The overall trajectory of technical SEO is encouraging: a web that's generally more coherent, crawlable, and semantically aware than it was a year ago.

For years, SEOs have known these fundamentals mattered (structured data, metadata, and clean architecture) but constant algorithm volatility often made them feel secondary to quick-win tactics. Now, as large language models increasingly rely on structured and well-labeled data to interpret and cite content, those same fundamentals are being revalidated. The basics were never obsolete and they aren't likely to become so any time soon.

Authors

Amaka Chulwuma

Amaka-maxi   amaka-chukwuma-jubilee

Amaka is an SEO and content strategist who has spent the last seven years shaping how brands show up online. She has worked with agencies in the UK, US, and Australia, including Whitecoat SEO and Switch Key Digital, where she builds content systems, technical SEO foundations, and search-led storytelling for clients in legal, health care, home services, and B2B. Her work reflects a mix of clarity, thoughtful strategy, and empathy. She brings those same qualities into her life outside work, especially when spending time with her daughter, who remains her favourite part of every day

Chris Green

@chrisgreenseo   @chris-green.net   chr156r33n   chrisgreenseo

https://www.torquepartnership.com/

Chris Green is a Technical Director at Torque Partnership[141] and a search veteran of 15+ years. He advises Fortune 500 companies on search strategy and the evolving relationship between brands, algorithms, and AI systems.

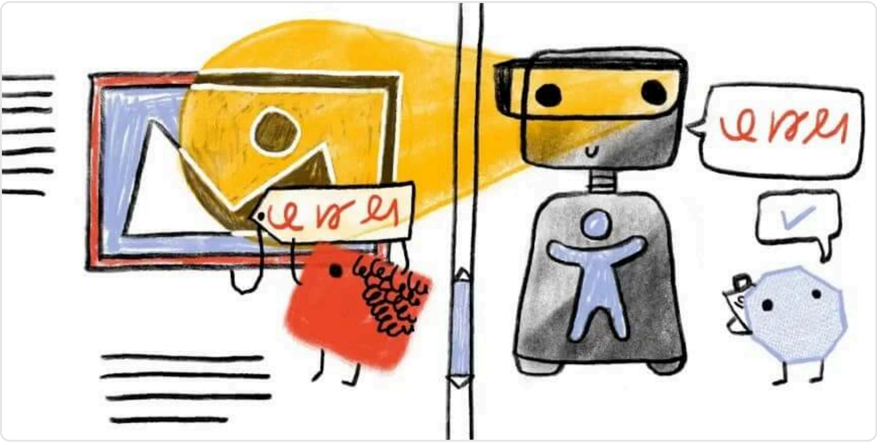
Sophie Brannon

@SophieBrannon   SophieBrannon

Sophie Brannon is the co-founder & director of StudioHawk US, based in Atlanta. With over a decade of SEO experience spanning agency, in-house, and consultancy roles, she has worked across a wide range of industries including eCommerce, finance, gaming, health, and SaaS. Sophie brings a strategic, data-driven approach to organic growth, helping brands scale sustainably through technical SEO, user experience testing, content and digital PR. She's passionate about making SEO accessible and mentoring the next generation of search marketer

[141] *https://www.torquepartnership.com/*

Part II Chapter 6
Accessibility



Written by Bogdan Lazar and Mike Gifford
Reviewed by Hidde de Vries, Aidan Tierney, and Scott Davis
Analyzed by Mike Gifford and Barry Pollard
Edited by Bogdan Lazar

Introduction

The web is changing fast. In 2025, web accessibility matters more than ever as mainstream technologies increasingly rely on inclusive features. For example, voice-activated assistants use screen reader technologies. Features originally designed for accessibility, such as video captions and haptic feedback, are now common.

Universal Design principles are fundamentally important for our work in modern web development. We're increasingly creating solutions that address diverse needs and improve experiences for all users. As Sir Tim Berners-Lee famously said, "The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect."

Recent global events and shifting legal requirements have pushed digital inclusion into focus. Microsoft's Inclusive Design Guidelines[142] show that accessibility helps more than just people

[142] *https://inclusive.microsoft.design/*

with permanent disabilities. The guidelines specifically mention temporary and situational limitations. For example, the ability to use a device with one hand can help individuals with injuries, parents with young children, as well as people carrying items.

In 2025, web accessibility laws have real teeth. The European Union's (EU) European Accessibility Act (EAA)[143] is a major step forward. It set a deadline of June 2025 for numerous websites and apps to conform to the EN 301 549[144] standard, which references the Web Content Accessibility Guidelines (WCAG)[145].

The United States updated its regulations as well. State and local government sites must now meet WCAG 2.1[146], as required by Title II of the Americans with Disabilities Act[147]. The 2024 data gives us a critical baseline to measure the tangible impact of these deadlines on the accessibility of websites globally.

Google Lighthouse powers our analysis using Deque's axe-core engine[148]. We benchmark 2025 findings against 2024 data and identify key trends. With broader adoption of WCAG 2.2, we examine the uptake of new Success Criteria and continued changes from deprecated rules such as `duplicate-id`.

Our approach is similar to that of the WebAim Million[149] but with differences in sites crawled and analysis tools used. The HTTP Archive crawls 17 million sites each month across home and secondary pages using Lighthouse and other tools. WebAim surveys the top million home pages[150] with WAVE[151].

Automated tests, including axe-core which is used by Lighthouse, can only partially check a subset of WCAG Success Criteria[152]. Alphagov from GOV.UK offers a comparison of popular automated audit tools[153] and they all detect less than 50% of accessibility errors. Many criteria lack automated tests altogether, and not all accessibility issues have matching criteria in WCAG.

But remember Goodhart's Law. When a metric becomes a target, it stops being a reliable metric. A perfect score doesn't guarantee full accessibility. You should treat Lighthouse accessibility scores as a starting point for evaluation rather than a final goal. Still, tracking these scores offers a valuable snapshot of the web's overall progress.

Our report focuses exclusively on HTML and doesn't include PDF or other office documents.

[143] *https://en.wikipedia.org/wiki/European_Accessibility_Act*
[144] *https://en.wikipedia.org/wiki/EN_301_549*
[145] *https://www.w3.org/WAI/standards-guidelines/wcag/*
[146] *https://www.w3.org/TR/WCAG21/*
[147] *https://www.ada.gov/resources/2024-03-08-web-rule/*
[148] *https://www.deque.com/axe/axe-core/*
[149] *https://webaim.org/projects/million/*
[150] *https://webaim.org/projects/million/#method*
[151] *https://wave.webaim.org/*
[152] *https://html5accessibility.com/stuff/2025/03/24/a-tools-errand/#:~:text=automation%20blues*
[153] *https://alphagov.github.io/accessibility-tool-audit/*

Compared to 2024, the median Lighthouse Accessibility score improved by 1%, reaching over 85% in 2025. Since the first Web Almanac in 2019, we've seen steady and incremental progress. Google Lighthouse assigns different weights[154] to axe-core issues, so organizations may prioritize fixes differently[155].

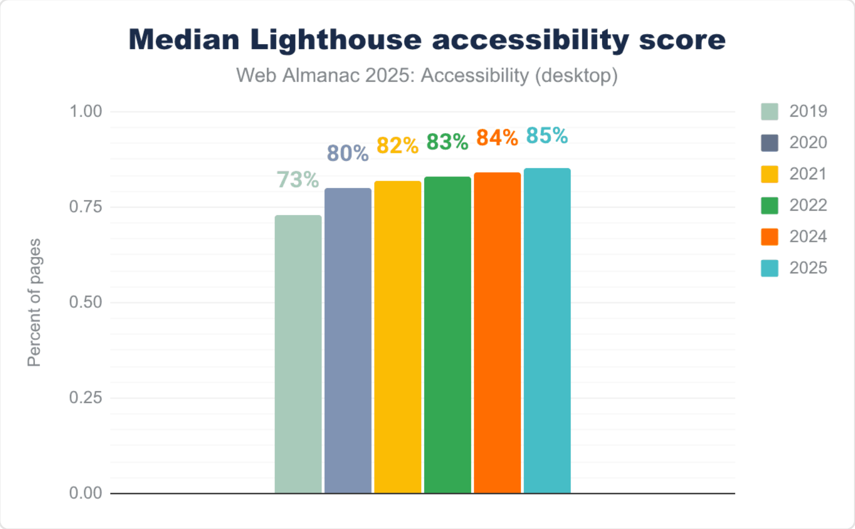


Figure 6.1. Lighthouse audit improvements year-over-year.

This year, we've seen the biggest advances in the following axe-core tests:

- ARIA input fields must have an accessible name[156]: +3% over 2024
- ARIA `meter` nodes must have an accessible name: +15% over 2024
- ARIA `progressbar` nodes must have an accessible name: +5% over 2024
- ARIA `tooltip` nodes must have an accessible name: +13% over 2024
- Avoid delayed refresh under 20 hours[157]: +1% over 2024
- `<object>` elements must have alternate text: +1% over 2024
- `<select>` element must have an accessible name: +5% over 2024

[154] *https://developer.chrome.com/docs/lighthouse/accessibility/scoring*
[155] *https://accessibility.civicactions.com/posts/prioritizing-accessibility-bugs-for-maximum-impact*
[156] *https://dequeuniversity.com/rules/axe/4.11/aria-input-field-name*
[157] *https://dequeuniversity.com/rules/axe/4.11/meta-refresh*

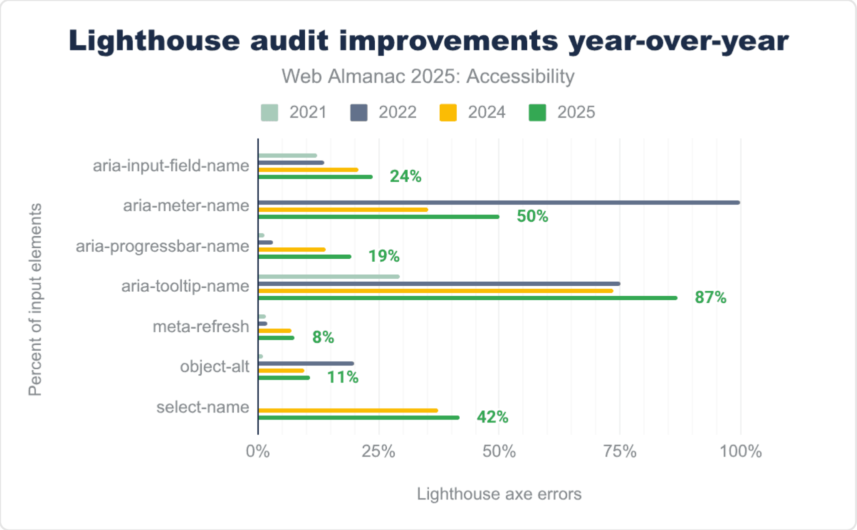


Figure 6.2. Most improved Lighthouse accessibility tests (axe).

This year, we're adding a new section on Artificial Intelligence (AI) to the Web Almanac. AI is changing how we build websites, write code, generate content, optimize performance, and interact with interfaces. It already plays a growing role in accessibility work, from generating image descriptions and captions to assistants that help teams find and fix issues.

At the same time, AI introduces risks and unanswered questions. There's no reliable way yet to see when AI has created or assisted in creating a website. Language models are trained on code and content that often contain accessibility problems. Automated descriptions or patterns can easily miss context, user intent, or nuance. Broader concerns about data use, environmental impact, and encoded bias directly affect who benefits from AI on the web and who is harmed or excluded.

The new AI chapter explores these tensions: how AI is already helping teams, where it falls short, and what standards, safeguards, and practices are needed. One principle runs through the analysis: AI should support human expertise and inclusive design, not replace them.

Throughout this chapter, you will find actionable links and practical solutions to help you improve accessibility on your own sites.

Ease of reading

Users need to easily read and understand web content. This goes beyond picking legible fonts.

It also covers using clear language, organizing pages logically, and following predictable design patterns.

While this report focuses on measurable technical metrics, qualitative factors, such as writing in plain language, matter just as much. WCAG 3.0 is exploring how to incorporate requirements for clear language, but it's still in the development phase.

Similar to plain language, numbers pose their own accessibility challenges. Some users struggle to interpret them, and automated tests can't reliably catch this as a barrier. To address this, review resources like Accessible Numbers[158] for practical advice on presenting numeric information clearly on the web.

Readability metrics exist for English content. The Flesch-Kincaid[159] readability score is one example. But the web is global. It spans many languages and diverse audiences. No standardized automated test covers all cases or languages.

Color contrast

The difference between foreground and background colors determines whether people can perceive web content. Insufficient color contrast remains a common barrier, especially for users with low vision or color vision deficiencies[160].

Color contrast is especially important for older users, people with temporary disabilities, like missing reading glasses, and anyone reading under bright sunlight or in challenging environments.

WCAG requires contrast ratios of at least 4.5:1 for standard text and 3:1 for large text to achieve AA conformance. AAA conformance demands 7:1 for normal text. WCAG contrast ratios[161] are an important baseline, but these guidelines don't address every form of color blindness or individual variation in perception.

Other documents, including the Accessible Perceptual Contrast Algorithm (APCA)[162], aim to offer a more perceptually accurate measurement of contrast.

Open source tools[163], like the newly released Contrast Report[164], make it easier than ever to find and fix color contrast issues. They even suggest modifications when colors fail to meet required ratios. For additional guidance, you can consult expert resources, such as Dennis Deacon's article on color contrast testing[165].

[158] *https://accessiblenumbers.com*
[159] *https://en.wikipedia.org/wiki/Flesch%E2%80%93Kincaid_readability_tests*
[160] *https://www.colourblindawareness.org/*
[161] *https://developer.mozilla.org/docs/Web/Accessibility/Guides/Understanding_WCAG/Perceivable/Color_contrast*
[162] *https://git.myndex.com/*
[163] *https://accessibility.civicactions.com/guide/tools#color*
[164] *https://contrast.report/*
[165] *https://www.dennisdeacon.com/web/accessibility/testing-methods-use-of-color/*

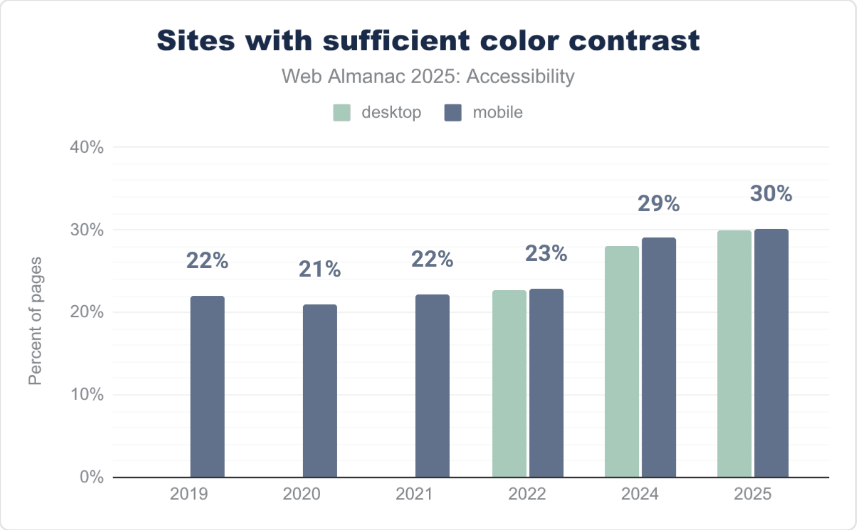


Figure 6.3. Sites with sufficient color contrast.

This year, text contrast pass rate improved by roughly 1% compared to 2024. But only 31% of mobile sites currently meet minimum color contrast requirements. Since mobile experiences depend heavily on clear visibility, this gap is a real problem for users accessing the web on their phones.

Browsers and operating systems increasingly support light, dark, and high-contrast modes. Users have more control now. Most sites still don't respond to these preferences though.

Zooming and scaling

Users must be able to resize content to suit their needs. Disabling zoom removes user control and is a direct violation of WCAG resizing requirements. This is more than a minor inconvenience. It may make a site completely unusable for people with low vision or those who rely on screen magnification for reading.

In 2025, this restrictive pattern still appears, often because developers want pixel-perfect layouts on mobile devices. Unfortunately, that comes at the cost of usability and accessibility.

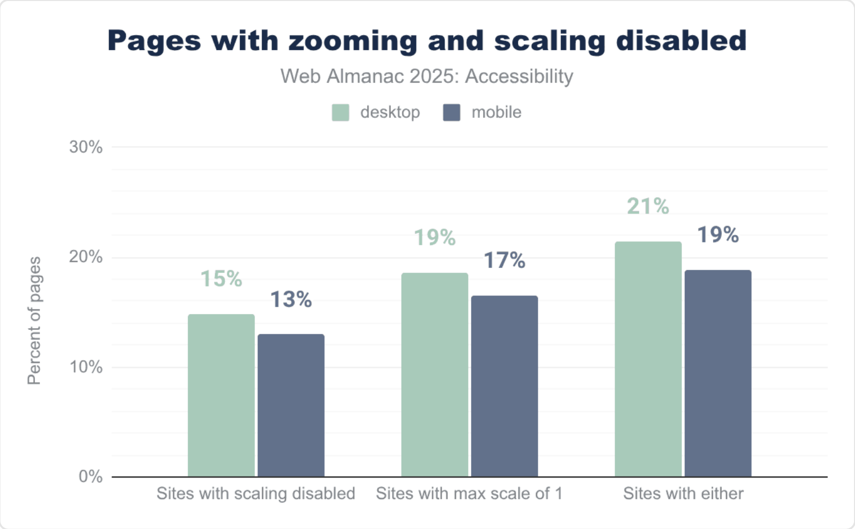


Figure 6.4. Pages with zooming and scaling disabled.

The number of sites that disable zooming or scaling continues to drop. In 2025, only 19% of mobile sites and 21% of desktop sites restrict scaling, either by using `user-scalable=no` or setting a restrictive maximum scale. That's a 1–2% improvement over 2024, showing slow but steady progress.

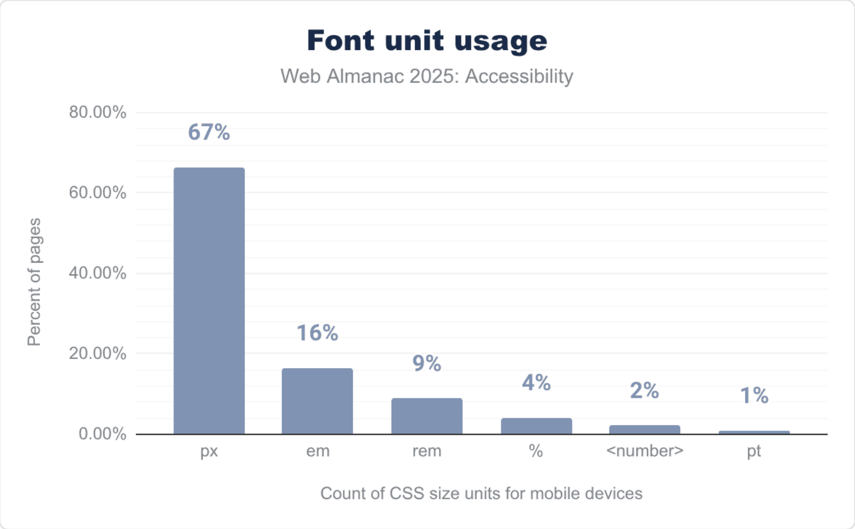


Figure 6.5. Font unit usage.

Font size units directly affect how text can respond to user preferences. Relative units, such as `em` and `rem`, let text to scale predictably with browser settings. In 2025, the use of `em` on mobile sites increased by 2%, improving user experiences for those who adjust font sizes to increase readability. Otherwise, font size unit usage stays largely the same as last year.

If you want to check whether your site is restricting zoom, examine its source code for the `<meta name="viewport">` tag. Avoid using values like `maximum-scale`, `minimum-scale`, `user-scalable=no`, or `user-scalable=0`, as these limit resizing. Instead, let users freely adjust content size. WCAG requires that text can resize up to 200%[166] without loss of content or functionality.

Language identification

Declaring a page's primary language with the `lang` attribute is essential. It lets screen readers select the correct pronunciation rules and enables browsers to provide more accurate automatic translations. Beyond the primary language, it's equally important to specify the language of sections that differ from the main language. This ensures that screen readers properly switch pronunciation for foreign words or phrases.

Despite being a straightforward Level A WCAG requirement[167], language declaration remains an area where many sites fall short. In 2025, roughly 86% of sites include a valid `lang` attribute, largely unchanged from 2024. This suggests steady adoption but also highlights room for improvement.

Correctly applying the `lang` attribute begins with including it on the `<html>` tag to specify the page's primary language. Pages often contain multiple languages. Use the `lang` attribute on individual elements or sections as needed. The W3C's documentation on specifying the language of parts[168] provides detailed guidance on this topic.

Missing or incorrect language declarations can cause translation errors.

For example, Chrome's automatic translation might misinterpret page content without a declared language, leading to confusing or inaccurate translations. Proper language tagging also supports styling for right-to-left languages and other language-specific behaviors.

[166] *https://www.w3.org/TR/UNDERSTANDING-WCAG20/visual-audio-contrast-scale.html#:~:text=Content%20satisfies%20the%20Success%20Criterion%20if%20it%20can,more%20extreme%2C%20adaptive%20layouts%20may%20introduce%20usability%20problems.*
[167] *https://www.w3.org/WAI/WCAG22/Understanding/language-of-page*
[168] *https://www.w3.org/WAI/WCAG21/Understanding/language-of-parts.html*

User preference

Modern CSS includes User Preference Media Queries[169] that let websites adapt to a user's operating system or browser settings. Users get a more comfortable, personalized experience. Websites can respond to preferences for motion, contrast, and color schemes.

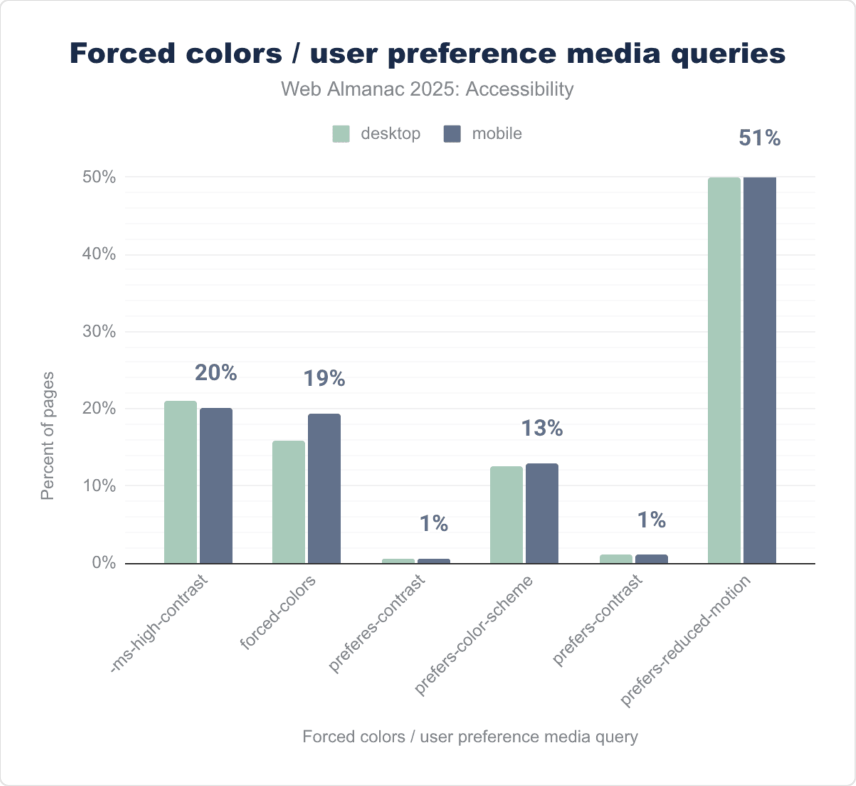


Figure 6.6. User preference media queries.

The most familiar queries, `prefers-reduced-motion` and `prefers-color-scheme`, remain widely supported by browsers. In 2025, adoption of these queries by websites shows little change. However, the use of `forced-colors`, which supports high-contrast modes for users with low vision, increased by 5% to 19%. Meanwhile, use of the outdated `-ms-high-contrast` media query has declined by 3% down to 20%. This reflects a gradual shift towards modern CSS standards.

[169] https://developer.mozilla.org/docs/Web/CSS/Reference/At-rules/@media/prefers-color-scheme

Continuing to incorporate these preferences advances accessibility and user satisfaction by respecting individual needs and system settings.

Broader implementation of personalization through CSS media queries hasn't seen significant growth despite these incremental gains. Encouraging further adoption helps ensure websites honor users' preferences, including reducing motion for vestibular disorder sensitivities and adapting display colors or contrast for visual comfort.

Navigation

Users navigate websites in different ways. Some use a mouse to scroll. Others use a keyboard, switch control device, or screen reader to navigate through headings. An effective navigation system must work for every user, regardless of their input device.

Wide-screen TVs and voice interfaces like Siri and Amazon Alexa create unique navigation challenges. Building good semantic structure into a site helps screen reader users navigate. It also helps users of many other types of technology.

Focus indication

Focus indication is essential for users who rely primarily on keyboard navigation and assistive devices to move through web content. It provides a visible cue that highlights which element is currently focused, so users understand where they are on the page.

Automated testing tools like Google Lighthouse can identify many basic requirements and flag obvious failures around focus indicators. But they're limited when it comes to complex interactions like keyboard traps, focus order, and whether focus moves logically to new content. Passing automated audits doesn't guarantee a site's keyboard accessibility or a good user experience for keyboard users.

Comprehensive manual testing is irreplaceable.

Tools like the open source Accessibility Insights for the Web[170] extension leverage Deque's axe-core and offer guided manual tests. The "Tab Stops" visualization feature helps testers see the path keyboard users take and identify potential issues effectively, like missing focus styles or unexpected focus traps.

Users of alternative navigation devices with limited motor abilities have unique needs related to focus visibility and sequence. Customizing assistive technology interfaces helps maximize

[170] *https://accessibilityinsights.io/docs/web/overview/*

control tailored to their abilities.

Focus testing best practices include:

- No focus traps where keyboard users get stuck[171]
- All interactive controls are keyboard focusable[172]
- Tab order is logical and intuitive[173]
- Focus is appropriately directed to new or dynamically loaded content[174]

The A11y Collective's report on understanding focus indicators[175] offers practical insights for implementing and testing visible focus outline styles.

Focus styles

WCAG mandates that all interactive content must have a clearly visible focus indicator[176]. This visual cue helps keyboard users identify which element is currently focused as they move through the page.

Without a prominent focus indicator, keyboard users and those relying on assistive technologies can easily become lost. Robust focus styles, like a high-contrast outline, are fundamental to accessible design. Many institutions, like GOV.UK[177], have established standards for focus indicators to ensure consistency and clarity.

Design annotations need to specify keyboard interactions, as Craig Abbott clearly laid out in the TetraLogical blog[178]. Shortly after this post, GitHub released their accessibility Annotation Toolkit[179], addressing the same problem.

[171] *https://developer.chrome.com/docs/lighthouse/accessibility/focus-traps*
[172] *https://developer.chrome.com/docs/lighthouse/accessibility/focusable-controls*
[173] *https://developer.chrome.com/docs/lighthouse/accessibility/logical-tab-order*
[174] *https://developer.chrome.com/docs/lighthouse/accessibility/managed-focus*
[175] *https://www.a11y-collective.com/blog/focus-indicator/*
[176] *https://www.w3.org/WAI/WCAG21/Understanding/focus-visible.html*
[177] *https://www.gov.uk/*
[178] *https://tetralogical.com/blog/2025/09/23/annotating-designs-using-common-language/*
[179] *https://github.com/github/annotation-toolkit*

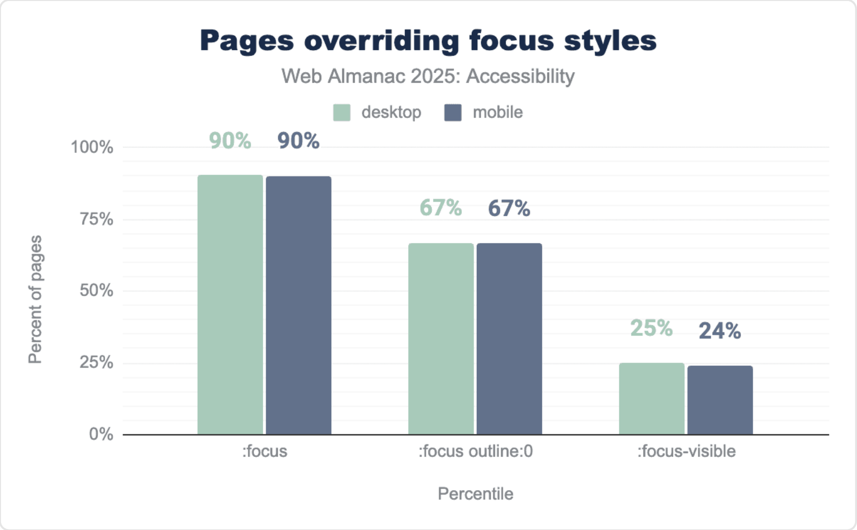


Figure 6.7. Pages overriding browser focus styles.

In 2025, 67% of sites explicitly removed default focus outlines, up 14% from 2024. This concerning trend may impair accessibility if not replaced with effective styles. On the positive side, adoption of the `:focus-visible` pseudo-class has grown. This means developers are starting to create context-aware focus indicators that are visible only when necessary.

`tabindex`

The `tabindex` attribute controls an element's participation in the keyboard focus order. It lets developers include, exclude, or reorder focusable elements. Correct use supports logical navigation and accessibility, and is a WCAG requirement[180]. Misuse, especially with positive values, can disrupt natural tab order and confuse users.

[180] *https://www.w3.org/WAI/WCAG21/Understanding/focus-order.html*

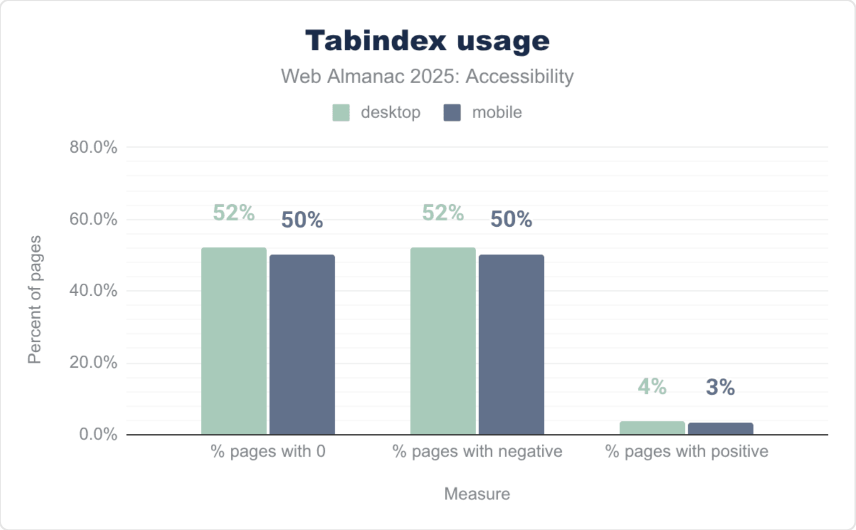


Figure 6.8. `tabindex` usage.

In 2025, `tabindex` usage has increased slightly. Just over 50% of sites used it, around 3-4% higher than 2024. Positive `tabindex` use remains stable, generally low, reflecting continued awareness that positive tabindex values should be avoided.

Landmarks

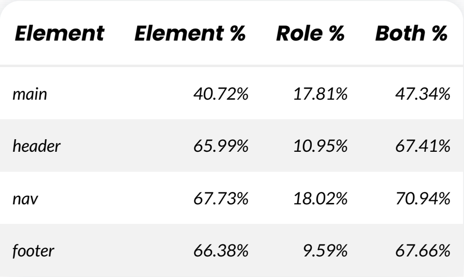
Landmarks structure a web page into distinct thematic regions, using native HTML elements such as `<header>`, `<nav>`, `<main>`, and `<footer>`. These elements create a clear, high-level page outline that help users of assistive technologies quickly understand the layout and jump directly to relevant sections.

A common accessibility anti pattern persists when developers add redundant ARIA attributes. For example, adding `role="navigation"` to a `<nav>` element. The `<nav>` element inherently carries the navigation role, so this duplication adds clutter to the code without benefit and may confuse assistive technology. Best practice is to favor native HTML5 elements[181] first before adding ARIA landmark roles. That's ARIA's primary guideline.

Accessibility experts like Eric Bailey have highlighted the pitfalls of overusing ARIA[182] in contexts where native semantic HTML is enough. Heydon Pickering's twelve principles of web accessibility[183] also emphasize the critical role semantic structure and landmarks play in

[181] https://www.w3.org/WAI/WCAG21/Techniques/html/H101
[182] https://www.smashingmagazine.com/2025/06/what-i-wish-someone-told-me-aria/
[183] https://heydonworks.com/article/pride-shame-and-accessibility/

accessible navigation.



Element	Element %	Role %	Both %
main	40.72%	17.81%	47.34%
header	65.99%	10.95%	67.41%
nav	67.73%	18.02%	70.94%
footer	66.38%	9.59%	67.66%

Figure 6.9. Landmark element and role usage (desktop).

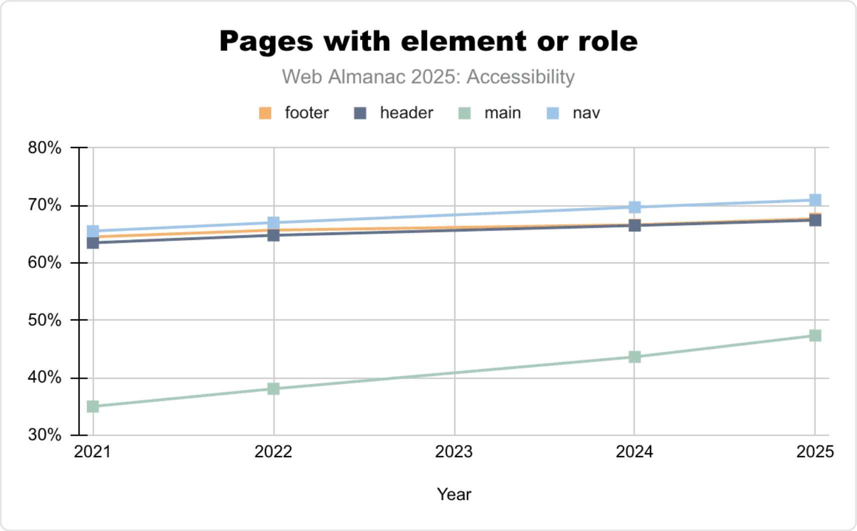


Figure 6.10. Yearly growth in pages with element role.

In 2025, the adoption of ARIA landmarks has increased slightly, led by the growing use of the native `<main>` element, now at 47%, up 3% from 2024. This progress reflects better compliance with semantic HTML and more robust page structure for users relying on assistive tools.

Screen reader users often navigate via "rotors" or landmark menus to jump between these page regions. Skip links pointing to landmarks improve usability by allowing immediate access to core content. They circumvent repeated navigation blocks or banners. We discuss Skip links in a later section.

Continued education on leveraging native HTML5 landmarks and minimizing redundant ARIA roles will further improve keyboard and assistive technology navigation experiences. The growth in semantic structure adoption supports accessibility goals and aligns web content with modern best practices.

Heading hierarchy

A coherent heading structure acts like a table of contents for a web page. It supports accessibility, SEO, and user comprehension. For screen reader users, navigating via headings is a key way to find information quickly. Search engines also rely on heading hierarchy to understand a page's organization and relevance.

Headings should communicate document structure, not just visual styling. Using heading tags such as `<h3>` or `<h4>` solely for their font size breaks the logical order. It makes navigation difficult for users of assistive technologies and violates the principle of separating structure from presentation. Instead, developers should style headings with CSS and use heading tags according to content hierarchy.

For a refresher on why semantics matter, review this article by Jono Alderson[184].

After a multi-year decline, heading hierarchy scores improved by almost 2% in 2025, indicating a renewed focus on proper heading structure.

59%

Figure 6.11. Mobile sites passing the Lighthouse audit for properly ordered heading.

Nevertheless, misusing headings for styling instead of structure remains common.

Skip links

Skip links are navigation aids that allow keyboard users and others using non-mouse input devices to bypass large, repetitive blocks of content, such as site navigation menus, and jump straight to the main page content. Typically, a "skip to main content" link is present as the first focusable element on the page for efficient navigation.

Bypassing blocks of content is a WCAG requirement[185], and basic implementations remain the norm.

[184] *https://www.jonoalderson.com/conjecture/why-semantic-html-still-matters/*
[185] *https://www.w3.org/WAI/WCAG21/Understanding/bypass-blocks.html*

Sophisticated tools, like PayPal's open source SkipTo[186], exist to generate dynamic menus of all major landmarks and headings on a page. This richer interaction benefits a wider range of users, enhancing overall navigability and usability. Eleanor Hecks wrote a compelling article on the importance of keyboard accessibility[187], as did TetraLogical[188].

Adoption of skip links has remained largely static from 2024 to 2025.

24%

Figure 6.12. Mobile and desktop pages likely featuring a skip link.

Approximately 24% of desktop and mobile pages include skip links detectable by common analysis methods. This figure might under-represent actual usage, as some skip links appear deeper in the page or target landmarks beyond navigation menus.

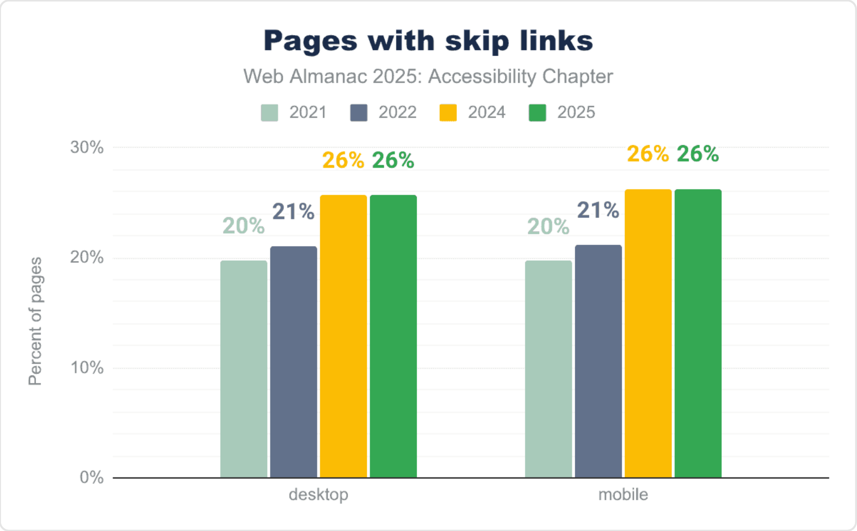


Figure 6.13. Yearly growth in pages with skip links.

Document titles

A descriptive page `<title>` is a basic necessity. It provides context for users navigating

[186] https://paypal.github.io/skipto/
[187] https://www.smashingmagazine.com/2025/04/what-mean-site-be-keyboard-navigable/
[188] https://tetralogical.com/blog/2025/05/09/foundations-keyboard-accessibility/

between browser tabs and windows and is often the first piece of information announced by a screen reader, helping users get oriented. WCAG also mandates that every page should have a meaningful title[189].

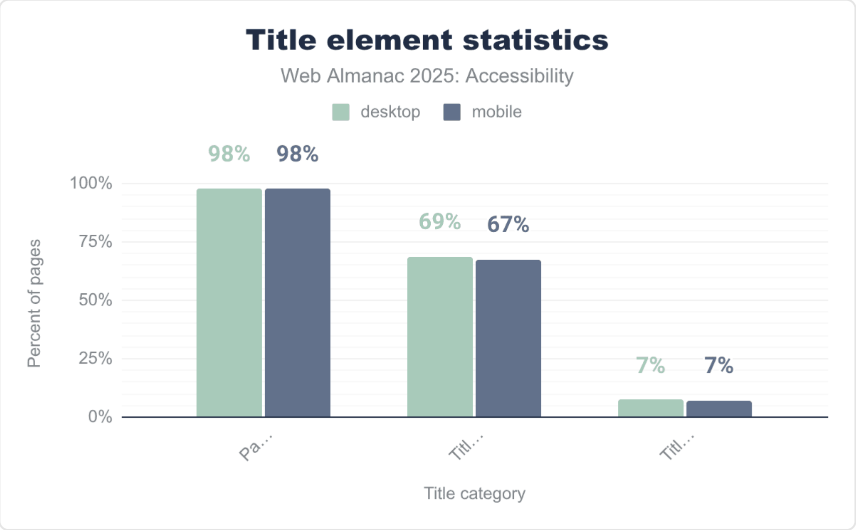


Figure 6.14. Title element statistics.

The 2025 data shows a slight improvement in the presence and descriptiveness of document titles compared to previous years. Approximately 98% of sites now include a `<title>` element, a 1% increase from 2024.

This is positive. But despite this high inclusion rate, many titles remain insufficiently descriptive. This impacts usability, especially for screen reader users who rely on clear titles for orientation.

There was a 2% decrease in mobile sites having titles with four or more words, which may indicate shorter or less specific titles on mobile pages. Including both a brief description of the page content and the website's name remains best practice for enhancing navigation and context.

Document titles remain a fundamental accessibility feature that benefits all users. They provide context when navigating browser tabs and windows. While near-universal in presence, improving title descriptiveness and consistency continues to be an important focus in 2025 and beyond.

[189] *https://www.w3.org/WAI/WCAG21/Understanding/page-titled*

Tables

HTML tables present data in a two-dimensional grid. Accessibility depends on structuring them with appropriate semantic elements. Using `<caption>` provides crucial context for screen reader users, while `<th>` elements define headers for rows and columns, helping users understand relationships within the data. Steve Faulkner's tool[190], released in 2025, can help developers quickly inspect the semantics of any HTML element.

The use of `<caption>` remained steady in 2025 compared to 2024, with only a small percentage of sites including captions. This low adoption is similar to prior years: roughly 1.6% of desktop sites include captions, which is an important, though often overlooked, accessibility feature.

Tables shouldn't be misused for layout purposes. CSS Flexbox and Grid handle layout. When tables are used purely for layout, the `role="presentation"` attribute removes their semantic meaning to avoid confusion with assistive technologies.

	desktop	mobile	desktop	mobile
Captioned tables	*5.6%*	*4.7%*	*1.7%*	*1.7%*
Presentational table	*4.9%*	*5.4%*	*3.6%*	*4.8%*

Figure 6.15. Table usage.

In 2025, 4.9% of mobile tables use this technique, up from 4% in 2024 and 1% in 2022. The emphasis remains on using semantic HTML elements correctly to make tables accessible.

Forms

Forms are how users interact with the web, from logging in to making a purchase to sharing information. Ensuring they're accessible is critical for users to complete tasks and participate fully online.

`<label>` element

The `<label>` element remains the standard, recommended way to provide accessible names for input fields. By programmatically associating descriptive text with a form control, typically through the `for` attribute pointing to the input's `id`, it ensures users of assistive technology

[190] *https://codepen.io/stevef/pen/ByoMebv*

clearly understand what information is required. Proper labels also improve usability by increasing the clickable area, since clicking the label sets focus to the input.

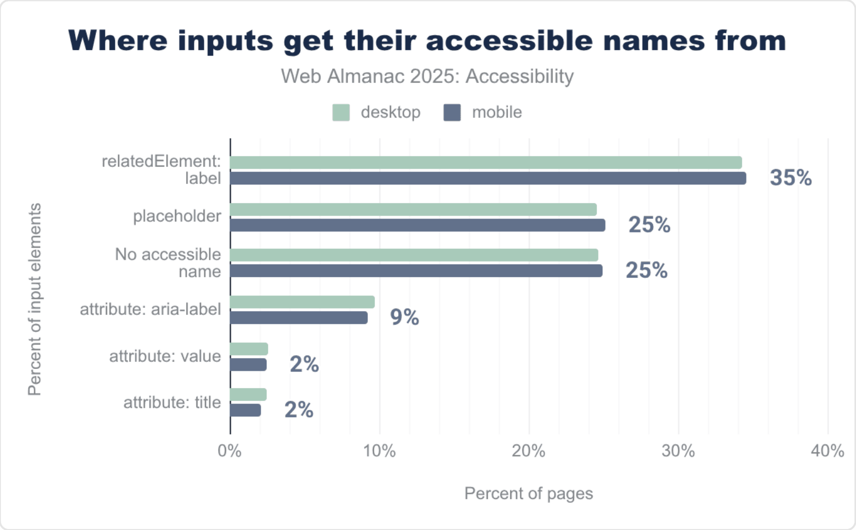


Figure 6.16. Where inputs get their accessible names from.

In 2025, about 35% of mobile inputs receive their accessible names from `<label>`, up from 32% in 2024. This is a positive trend.

We also saw a modest 2% reduction in inputs deriving accessible names only from placeholder text. Placeholder text is less reliable and not a substitute for labels. However, the proportion of inputs lacking accessible names altogether remained unchanged from last year, indicating ongoing accessibility gaps.

The 2025 data shows incremental improvement in `label` usage. It also underscores the need to continue expanding proper labeling practices to achieve full accessibility compliance and usability.

`placeholder` attribute

The placeholder attribute provides a hint or example of the expected input format inside a form field. But it should never replace the `<label>` element as the accessible name for that input. Placeholder text disappears as soon as the user starts typing, making it unavailable for reference.

Placeholder text also usually has poor default contrast, often failing WCAG color contrast requirements. Screen reader support for placeholders varies widely as well.

The recommended approach is to use visible, programmatically associated labels for inputs, with the placeholder serving only as a supplementary hint or example.

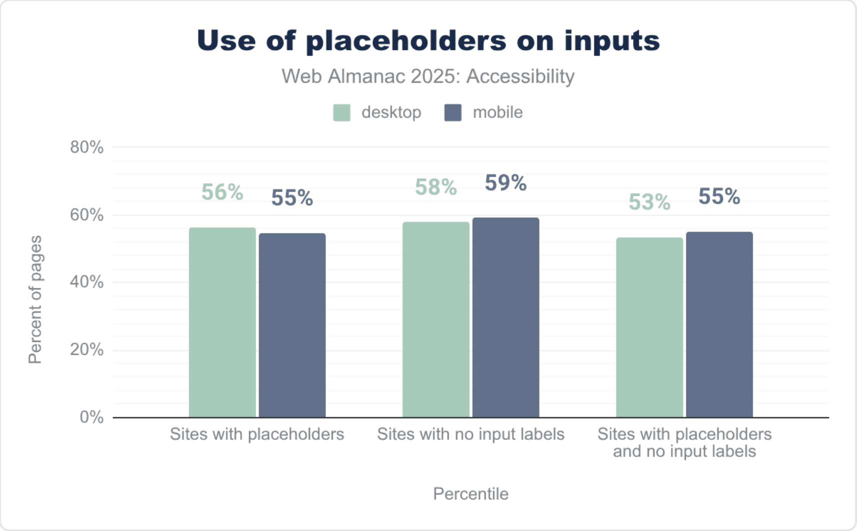


Figure 6.17. Use of placeholders on inputs.

In 2025, there was a 2% reduction in the use of placeholder text as the only accessible name for inputs. Despite this positive trend, the practice remains all too common. 53% of desktop and 55% of mobile inputs rely solely on placeholder text for accessible naming, which still poses significant accessibility barriers.

Requiring information

Communicating that a form field is mandatory is essential for usability and accessibility. While a visual indicator such as an asterisk (*) is common, it alone is insufficient because it lacks semantic information.

The HTML5 `required` attribute provides a native, machine-readable way to indicate that a user must fill in a field before submitting the form. This attribute works with many input types like text, email, password, date, checkbox, and radio. Browsers enforce validation and assistive technologies convey the required status to users.

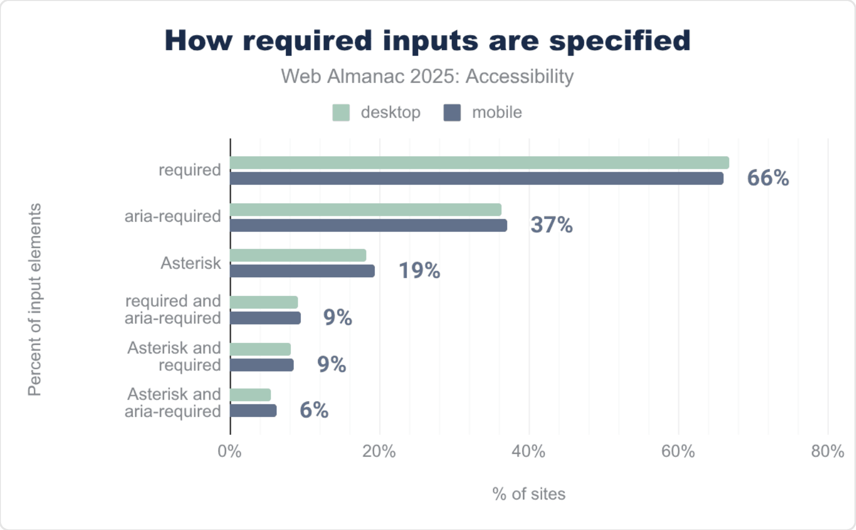


Figure 6.18. How required inputs are specified.

We are seeing a modest increase in the adoption of the required attribute, up 1% in 2025 to 66% for mobile. Use of `aria-required` has dropped 3% to 37% for mobile. This indicates a gradual shift towards more semantic usage of native HTML validation over ARIA, which is intended to supplement but not replace native semantics.

Progress in 2025 reflects slow but steady movement toward better semantic indication of required inputs, improving form accessibility and user experience.

Captchas

CAPTCHAs differentiate humans from automated bots, mitigating malicious activity. The acronym stands for "Completely Automated Public Turing Test to Tell Computers and Humans Apart." While CAPTCHAs serve a necessary security function, they frequently create significant accessibility barriers, particularly for people with visual, motor, or cognitive disabilities.

Traditional visual CAPTCHAs can be difficult or impossible for users with disabilities to solve. The W3C recommends exploring alternative verification methods that are more inclusive, such as:

- Audio CAPTCHAs that provide spoken challenges
- "Invisible" CAPTCHAs that work in the background without user input

- Incorporating multi-factor authentication methods or simpler verification flows.

In 2025, CAPTCHA use has remained roughly steady compared to previous years.

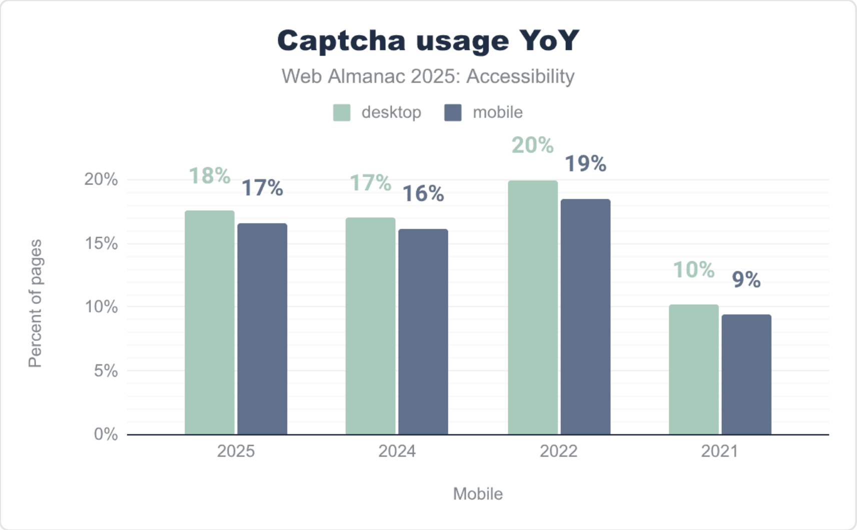


Figure 6.19. CAPTCHA usage year over year.

Notably, the Government of Luxembourg released a CAPTCHA scanner[191] tool. It assesses and monitors CAPTCHA implementations across government websites, aiming to improve accessibility compliance in the public sector.

Continued efforts to replace or supplement visual CAPTCHAs with more accessible options are essential. All users should be able to complete verification steps without undue difficulty or exclusion.

Media on the web

Accessible media on the web requires providing alternative formats to ensure content is usable by everyone. Users with visual impairments benefit from audio descriptions that convey important visual information. Users who are deaf or hard of hearing rely on captions or sign language interpretation to access audio content.

Audio descriptions and captions aren't enough. Transcripts are necessary for audio-only and

[191] *https://accessibilite.public.lu/en/news/2025-09-22-captchas.html*

video-only content, offering a complete textual alternative. For non-text content like images, provide appropriate alternative text. If they don't add meaningful information, mark them as decorative.

The principles and requirements for accessible media remain consistent between 2024 and 2025, emphasizing the ongoing importance of providing inclusive multimedia experiences to users with disabilities.

Images

The `alt` attribute provides a textual description of an image. It's essential for screen reader users to understand the visual content. In 2025, this attribute remains fundamental to image accessibility, with no significant change in error rates from previous years.

69%

Figure 6.20. Percentage of images passing the Lighthouse audit for images with `alt` text.

JPG and PNG files continue to dominate web images, but there is encouraging growth in the use of WEBP and SVG formats. SVG files offer rich semantics that benefit complex and interactive images.

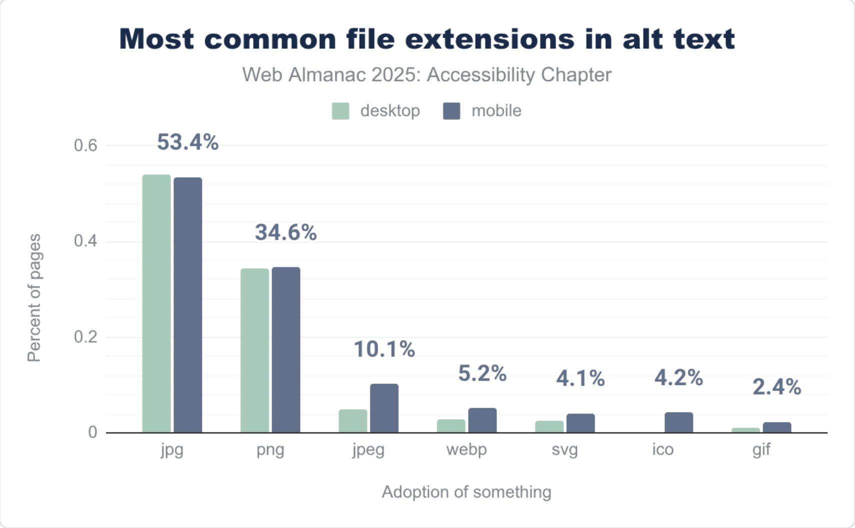


Figure 6.21. Most common file extensions in `alt` text.

However, we noticed one issue that continues to persist: approximately 8.5% of image alt texts end with common file extensions like `.jpg` or `.png`. This typically happens when automated authoring tools insert filenames as `alt` text. Unfortunately, this adds no value and doesn't help users relying on assistive technologies.

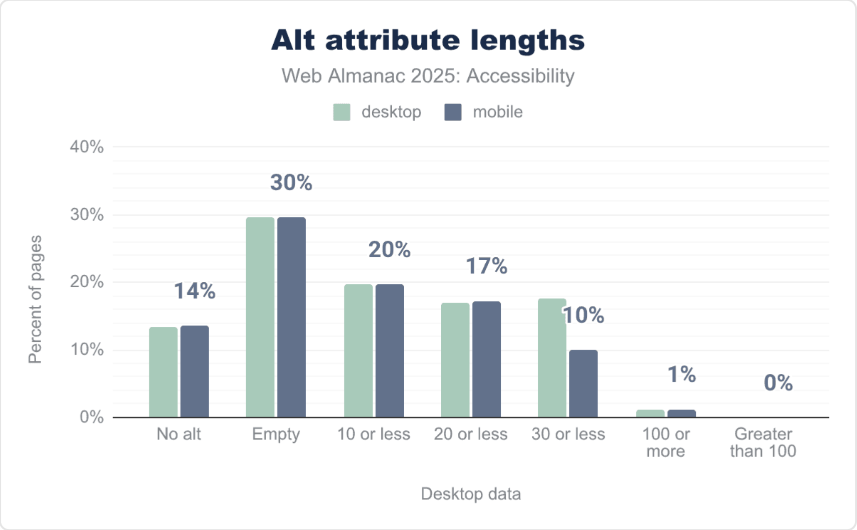


Figure 6.22. `alt` attribute lengths.

There is a positive trend toward alt texts between 20 and 30 characters in length, which tend to balance descriptiveness and brevity. But about 50% of images still have either empty alt attributes or text shorter than 10 characters. Empty alt text is appropriate only for purely decorative images. Most images however convey important information deserving meaningful descriptions and therefore will benefit from a meaningful alt text.

Best practices continue to emphasize providing concise yet descriptive alt text tailored to image context, avoiding filenames, and using semantic file formats like SVG when appropriate. Artificial Intelligence (AI) tools show promise too. Drupal's integration of AI-assisted alt text suggestions[192] helps authors create better alt attributes by providing editable examples. Brian Teeman wrote an interesting critique of the AI generation of Alt Text[193].

Images remain an area with opportunities for significant accessibility improvement despite steady progress.

[192] *https://www.drupal.org/project/ai_image_alt_text*
[193] *https://magazine.joomla.org/all-issues/june-2024/ai-generated-alt-text*

Audio and video

The HTML `<track>` element provides timed text tracks like captions, subtitles, and audio descriptions for media elements like `<video>` and `<audio>`. In 2025, this element is still underutilized. Despite its importance for users who are deaf, hard of hearing, or blind, adoption rates stay exceptionally low, at under 1%.

Many modern video platforms now commonly use HTTP Live Streaming (HLS)[194] instead of the native `<track>` element. This may contribute to the low usage statistics. HLS, which requires custom implementation, means that developers need to be more intentional about building in caption support themselves. This means there's more room for developers to forget about captions or do them poorly.

Captions are essential for deaf and hard of hearing users. They also benefit viewers in noisy environments or those with difficulty understanding spoken language. Audio descriptions enable users with visual impairments to gain context about visual content.

Compared to 2024, we've seen no significant growth in `<track>` usage for captions and subtitles, going from 0.1% to 0.4% on desktop and 0.1% to 0.2% on mobile. This tells us that the industry still has substantial room for improvement.

Assistive technology with ARIA

Accessible Rich Internet Applications (ARIA) is a set of HTML attributes to improve the accessibility of web content. ARIA is particularly valuable for complex or custom components that can't be made accessible with native HTML alone. ARIA enhances dynamic, interactive user interfaces, making sure people using screen readers or other assistive technologies can understand and interact with all page elements.

ARIA must be used with care.

Incorrect or excessive use can introduce new barriers, causing confusion for both users and accessibility tools. For example, ARIA attributes that don't match the intended functionality, roles added to inappropriate elements, or redundant ARIA can disrupt the user experience and increase accessibility errors.

Adrian Roselli's work highlights the limitations of certain ARIA properties[195], such as `aria-description`, and underscores the importance of understanding both the strengths and pitfalls of ARIA.

[194] *https://en.wikipedia.org/wiki/HTTP_Live_Streaming*
[195] *https://adrianroselli.com/2025/01/aria-description-does-not-translate.html*

The most important principle for ARIA is:

If you can use native HTML, you should.

Native elements like `<button>` , `<input>`, and `<nav>` come with built-in accessibility that ARIA can't fully reproduce. ARIA should only supplement native semantics where required, not replace them.

Recent guidance[196] by experts including Florian Schroiff as well as current best practices reinforce applying ARIA only for complex custom elements, and strictly following specifications to avoid accidental exclusion or miscommunication.

In 2025, ARIA continues to play a vital but occasionally problematic role in web accessibility.

ARIA roles

ARIA roles communicate an element's purpose or type to assistive technologies. In 2025, they continue to play a significant role in making web content accessible. While native HTML elements like `<button>` come with built-in semantics, ARIA provides the ability to assign roles to custom components that lack native equivalents, such as tabbed interfaces or `dialog` components.

[196] *https://www.a11y-collective.com/blog/aria-in-html/*

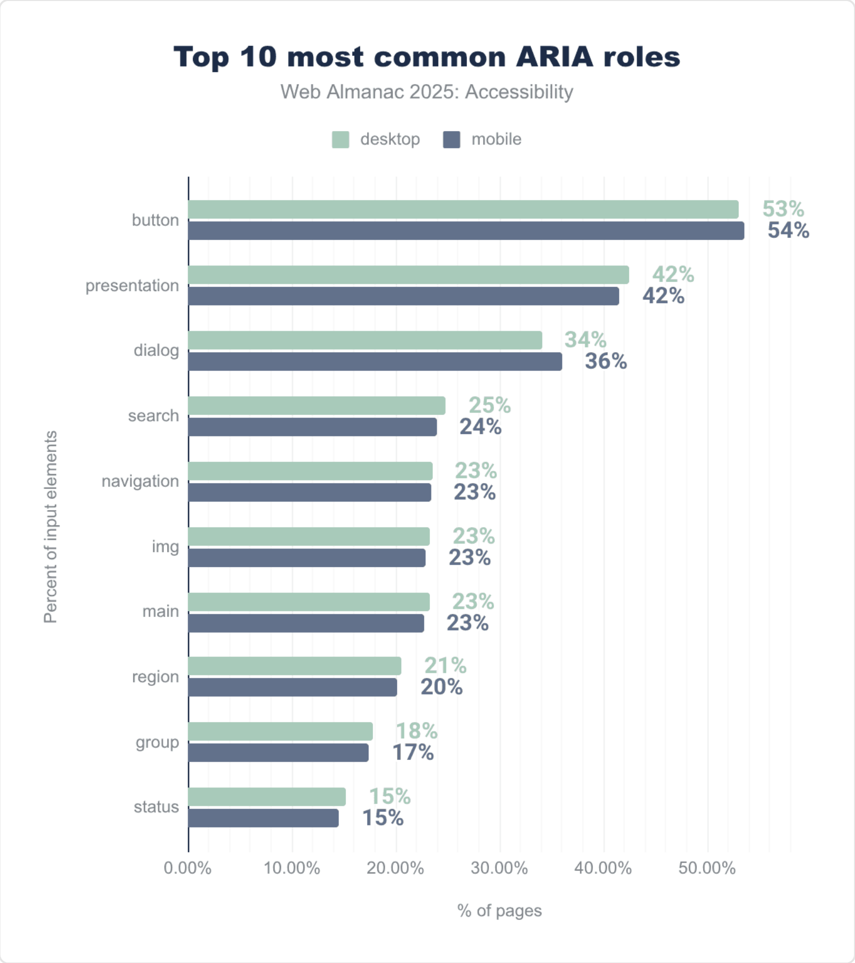


Figure 6.23. Top ten most common ARIA roles.

We've seen an approximately 4% increase in the use of the ARIA `button` role, reaching 53% on desktop and 54% on mobile sites in 2025. We've seen similar increases in the use of roles like `presentation` and `dialog`, whereas the use of the `search` role usage remains stable.

The increased use of the ARIA `button` role raises concerns. It often indicates that websites are applying roles like `button` to non-semantic elements such as `<div>` or `<span>`. Or they're redundantly assigning roles to native HTML elements like `<button>`.

Using the presentation role

Applying `role="presentation"` or `role="none"` instructs assistive technologies to treat the element as purely presentational. It removes its native semantics from the accessibility tree. While this can be useful for layout elements that convey no meaningful information, overuse or misuse can create significant accessibility barriers.

For example, applying `role="presentation"` to a `<ul>` element causes the entire list semantics, including those of child `<li>` elements, to be ignored. Screen reader users lose crucial contextual and structural information, like how many items are in a list.

While the `presentation` role can help remove misleading semantics when elements are used purely decoratively or for layout, it should be applied sparingly and with clear intent.

42%

Figure 6.24. Percentage of desktop sites and mobile sites have at least one `presentation` *role.*

In 2025, the use of `role="presentation"` increased by 2%, continuing a concerning trend.

Labeling elements with ARIA

Browsers maintain an accessibility tree that exposes information about page elements, such as their accessible names, roles, states, and descriptions. Assistive technologies rely on this to convey context to users. An element's accessible name is crucial and is usually derived from visible text content. However, ARIA attributes like `aria-label` and `aria-labelledby` can be used to explicitly set or override accessible names when native text is insufficient or unavailable.

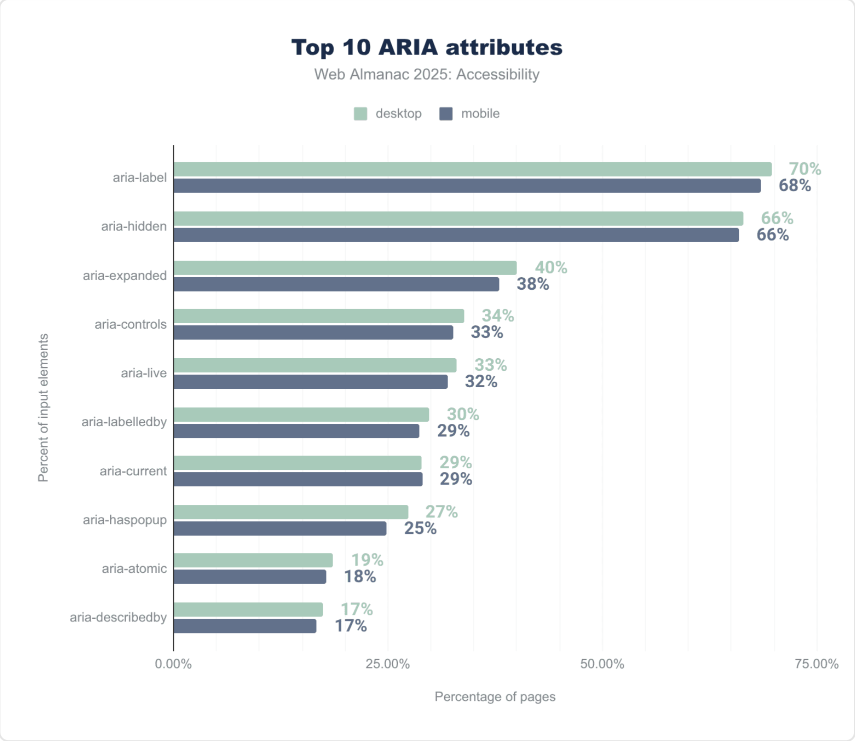


Figure 6.25. Top 10 ARIA attributes.

In 2025, the use of almost all top ARIA attributes increased. Desktop usage of `aria-label` rose by 5% and `aria-labelledby` by 3%. Use of `aria-describedby` on desktop decreased by 1%.

These changes suggest developers increasingly assign accessible names programmatically to more elements. This can be helpful but also problematic if not carefully implemented.

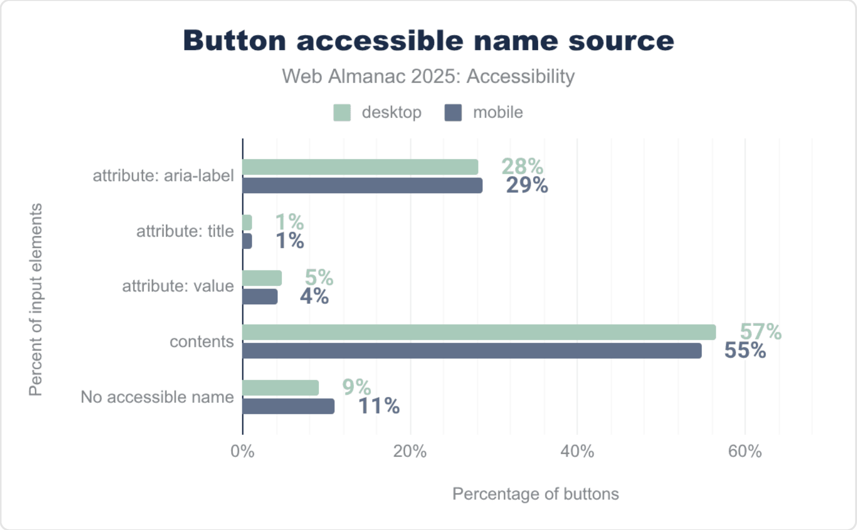


Figure 6.26. Button accessible name source.

We are seeing a concerning trend with the continued increase of 4% to 5% in defining buttons with `aria-label` alone, without corresponding visible labels. This disconnect between what a user sees visually and what assistive technologies announce can create confusion and barriers. This is especially true for people with cognitive disabilities or who use voice input. Ideally, the accessible name and visible label should match to provide a consistent user experience.

Nearly 66% of pages use the `aria-label` attribute, up from earlier years, making it the most frequently used ARIA attribute for accessible names. About a quarter of pages use `aria-labelledby`.

While using ARIA to enhance accessibility is positive, it underscores the importance of testing with assistive technologies and involving users with disabilities to ensure meaningful and accurate naming.

Hiding content

The `aria-hidden="true"` attribute removes an element and all its descendants from the accessibility tree, making the content invisible to screen readers. This is useful for hiding purely decorative or redundant visual elements that would otherwise confuse non-visual users.

66%

Figure 6.27. Pages with at least one element with the `aria-hidden` attribute.

In 2025, use of `aria-hidden` increased by 3% compared to 2024. Approximately 66% of websites have some content hidden using this ARIA attribute.

Similarly, the `aria-expanded` attribute, which signals whether a section of content is expanded or collapsed, also saw increased adoption, reaching 40% of desktop sites and 38% of mobile sites. This attribute is important for communicating the state of disclosure widgets like accordions or expandable menus to assistive technologies.

Thoughtful application of these ARIA attributes remains crucial in 2025. They help with the management of dynamic content and ensure inclusive experiences across devices and user needs.

Screen reader-only text

In 2025, a common and effective technique for accessibility is the use of screen reader-only text. This is text that's visually hidden but remains accessible to assistive technologies like screen readers. This approach is often applied to provide additional context, instructions, or descriptive labels for interactive elements without cluttering the visible interface.

Developers frequently use common CSS classes such as `.sr-only`, `.visually-hidden`, or `.element-invisible` to achieve this effect. These classes typically use off-screen positioning, clipping, or zero-sized boxes to hide the text visually while ensuring it remains in the accessibility tree and readable by screen readers.

16%

Figure 6.28. Desktop websites with a sr-only or visually-hidden class.

Usage of these common screen reader-only classes remained essentially unchanged between 2024 and 2025. Some websites include hidden text to provide context to screen reader users in ways that may not be apparent from the semantic HTML alone.

Dynamically-rendered content

ARIA live regions are critical for making dynamically changing content accessible. They inform screen readers about updates to page content, such as form validation messages, status updates, or live feeds. These updates occur without a full page reload, and are therefore necessary for users to receive important information without disruption.

Role	***Desktop***	***Mobile***	***Implicit `aria-live` value***
status	*15.18%*	*14.51%*	*polite*
alert	*7.12%*	*6.74%*	*assertive*
timer	*0.91%*	*0.84%*	*off*
log	*0.61%*	*0.55%*	*polite*
marquee	*0.09%*	*0.10%*	*off*

Figure 6.29. Pages with live region ARIA roles and their implicit `aria-live` value.

In 2025, about 33% of sites use the `aria-live` attribute, up 4% from 2024. Usage of the `role` value `status`, essentially an `aria-live` value of `polite`, increased by approximately 5%. This signals more widespread adoption of polite notifications that inform users of non-urgent updates.

Additional ARIA roles, such as `alert`, `timer`, `log`, and `marquee`, also have implicit `aria-live` attributes with predefined behaviors, enabling a broad spectrum of live region use cases.

Increased use of ARIA live regions in 2025 reflects progress in communicating dynamic content updates effectively, supporting users who rely on assistive technologies to interact with modern, responsive web experiences.

User personalization widgets and overlay remediation

Accessibility widgets and overlay remediation tools are third-party scripts designed to enhance website accessibility. They offer user personalization options, such as font size or contrast adjustments, and automated fixes for common accessibility issues.

These overlays often promise quick-fix compliance but fall short of addressing complex accessibility challenges that require manual code and design changes. The European Disability

Forum[197] has warned that such tools can interfere with users' own assistive technologies, creating conflicts that reduce accessibility and frustrate users.

Though overlays can help remove some surface-level barriers and provide additional personalization features, reliance on them often leads organizations to stop investing in proper accessibility. Overlays generally have more usability, security, and performance drawbacks than fixing underlying code issues.

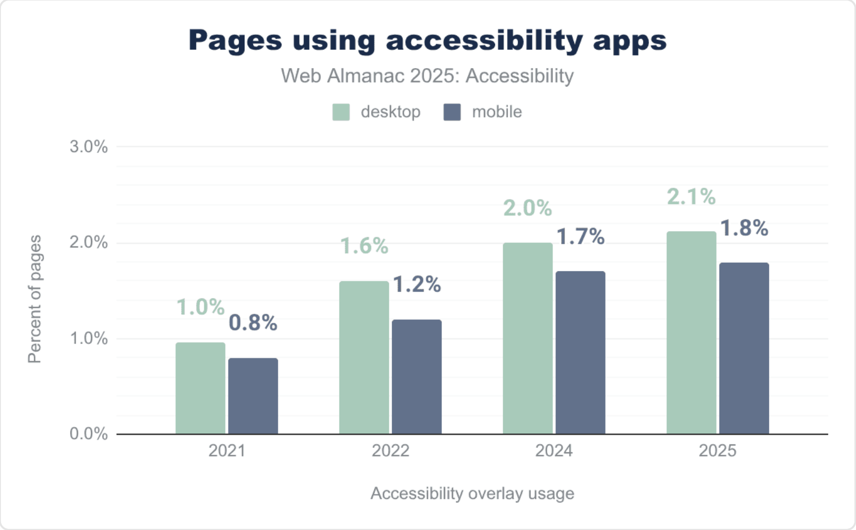


Figure 6.30. Pages using accessibility apps (overlays).

Data shows about 2% of desktop sites use such accessibility apps. Rates are even lower rates among the highest-traffic sites, at 0.2% among the top 1,000. This pattern shows that overlays are mostly adopted by lower-traffic sites and remain a controversial and imperfect solution.

Despite a marginal increase in their use in 2025, the distribution of these accessibility apps remains consistent with 2024, dominated by providers like UserWay[198], AccessiBe[199], AudioEye[200], and EqualWeb[201].

[197] https://www.edf-feph.org/accessibility-overlays-dont-guarantee-compliance-with-european-legislation/
[198] https://userway.org/
[199] https://accessibe.com/
[200] https://www.audioeye.com/
[201] https://www.equalweb.com/

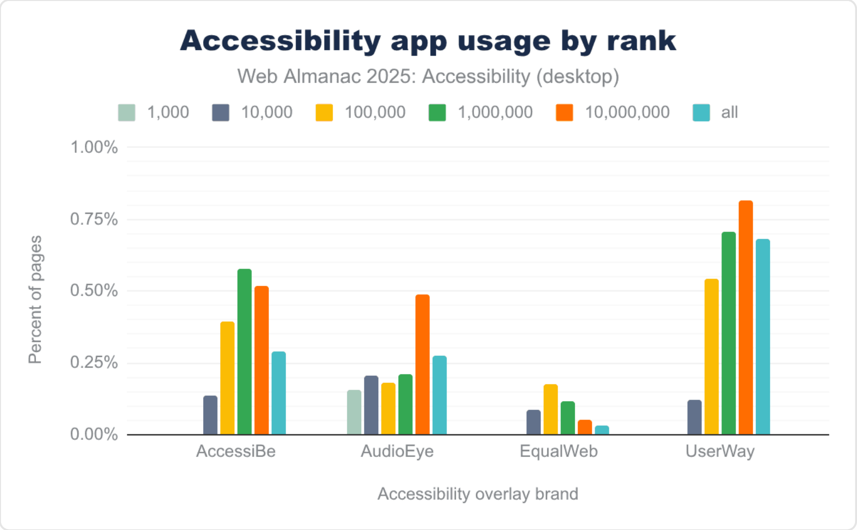


Figure 6.31. Accessibility app usage by rank.

Confusion on overlays

Accessibility overlays and personalization widgets, while growing marginally in use by about 2% in 2025, continue to be a source of significant controversy and confusion.

Leading organizations such as the International Association of Accessibility Professionals (IAAP) and the European Disability Forum (EDF) have explicitly warned that overlays are not a silver bullet. They must never impede users' access or interfere with their assistive technologies and shouldn't be marketed as making a site fully compliant.

Marketing claims often create unrealistic expectations among organizations, leading to legal and practical risks. These tools can't replace inclusive design, manual accessibility testing, and ongoing remediation. All these are essential to meet accessibility standards.

The European Accessibility Act and other regulations emphasize that accessibility must be built into websites at the source. While accessibility overlays exist, they don't meet the detailed criteria of accessibility standards if the underlying website itself isn't accessible. They're therefore not an appropriate solution on their own.

Bitvtest.de has recently decided[202] to stop testing and disallow their certification mark on websites that use an overlay tool. Bitvtest is a professional accessibility testing service built

[202] *https://bitvtest.de/test-methodik/web/beschreibung-des-pruefverfahrens#c761*

around the German BITV-Test, which is a structured way to check how accessible a website or app is.

The Impact of Artificial Intelligence (AI)

Artificial intelligence (AI) is a broad field of computer science focused on building systems that can perform tasks that normally require human intelligence. Within this field, large language models (LLMs) are a specific kind of AI system trained on very large datasets to learn statistical patterns.

LLMs use these patterns to generate and transform text, answer questions, and follow instructions, but they don't have understanding or intent in a human sense. They predict likely words and sentences based on their training data.

Web accessibility tools increasingly rely on LLMs. Many platforms now offer AI-generated `alt` text as a way to address missing image descriptions, though the quality and accuracy of these automated solutions varies widely.

Developers are adding AI tools, like GitHub's Accessibility Scanner[203], to their every day work. These tools give instant feedback and accessibility recommendations, making it easier to fix accessibility issues.

But AI in accessibility isn't without its problems.

Right now, there's no standard way to tell if AI made or improved a website or its content. This makes it harder to evaluate sites and leaves users in the dark about what they're looking at.

Some experts, like accessibility advocate Léonie Watson, talk about an "agentic web[204]" where AI changes how we interact with content online. This raises questions about how accessibility standards need to adapt.

AI-powered browsers and extensions are becoming mainstream fast. Voice assistants and AI agents built into browsers might soon handle most of our basic information searches. People already have the option of choosing AI-generated answers over traditional search results. This shift could help or hurt accessible design. It all depends on whether these AI tools are themselves fully accessible.

Experts like Joe Dolson have explored whether AI can build fully accessible websites on its own[205], highlighting both the potential and current limitations of the technology. Scott Vinkle's

[203] *https://github.com/github/accessibility-scanner*
[204] *https://tetralogical.com/blog/2025/08/08/accessibility-and-the-agentic-web/*
[205] *https://wpbuilds.com/accessibility/5/*

experiences at Shopify[206] shows how AI can improve accessibility in real-world situations.

The A11Y Collective blog on Artificial Intelligence and Accessibility[207] points out that while AI tools that automate `alt` text or real-time captions, and voice assistants can help accessibility at scale, they still struggle with accuracy, context, privacy, and bias.

Research by Dries Buytaert[208] shows AI can tackle huge backlogs of unlabeled images, but human review is still essential for quality. He explores the balance between quality, privacy, cost, and complexity for organizations considering AI-powered alt text.

Digital Accessibility Training[209] outlines the opportunities and challenges of AI for content creators. AI tools enable accessibility features at scale but raise concerns about content validity, bias, and ethical usage.

Hidde de Vries contrasts how humans and language models approach accessible component code[210]. Humans base HTML, CSS, and ARIA decisions on specifications, user needs, assistive technology behavior, and platform quirks, all guided by intentions for the interface. LLMs instead predict likely code from training data, which is problematic because most existing code has accessibility issues, and the models lack intent or understanding of specific users.

Adrian Roselli acknowledges that recent advances in computer vision and LLMs have brought real benefits, such as better image descriptions and improved captions and summaries. However, he argues these tools still lack context and authorship[211]. They can't know why content was created, what a joke or meme depends on, or how an interface is meant to work. Their descriptions and code suggestions can easily miss the point or mislead users.

AI raises significant ethical concerns that go beyond accessibility.

100

Figure 6.32. Comparable number of gasoline cars for carbon emissions needed to train an LLM.

One major issue is environmental impact. A study on carbon emissions and large neural network training[212] estimated that training a single large language model such as GPT-3 consumed more than 1,200 megawatt-hours of electricity and produced around 500–550 metric tons of CO_2-equivalent. This is comparable to the lifetime emissions of over 100 gasoline cars. For more information refer to the Sustainability chapter.

[206] *https://scottvinkle.com/blogs/work/4-ways-i-use-ai-as-an-accessibility-specialist*
[207] *https://www.a11y-collective.com/blog/artificial-intelligence-accessibility/*
[208] *https://dri.es/comparing-local-llms-for-alt-text-generation*
[209] *https://www.digitalaccesstraining.com/pages/articles?p=ai-and-accessibility-opportunities-and-challenges-for-content-creators*
[210] *https://hidde.blog/ai-for-accessible-components/*
[211] *https://adrianroselli.com/2023/06/no-ai-will-not-fix-accessibility.html*
[212] *https://arxiv.org/ftp/arxiv/papers/2104/2104.10350.pdf*

Another core concern is how training data is collected and used. Several high-profile lawsuits[213] allege that AI vendors scraped and used copyrighted material without permission or compensation. This included millions of stock photos and books.

Studies[214] of large models show they can amplify ableist, racist, and other harmful biases present in their training data. This ultimately shapes how disability, race, and gender are described, reinforcing stigma at scale.

Looking forward, it's clear to us that AI will increasingly become a tool web developers and content creators rely on. However, AI should support human expertise, not replace it.

As these tools get better, the accessibility community needs to answer some important questions in 2026 and beyond:

- How do we check if AI-generated content is accurate and accessible?
- What standards should govern AI use in web content?
- How will assistive technologies work with AI-driven interfaces?
- Can AI provide equal accessibility while respecting individual needs?
- What ethical guidelines do we need to prevent AI from reinforcing bias or excluding people?

Sectors and accessibility

This section compares accessibility scores across various industry and community sectors to identify patterns and leaders.

Country

We can identify a website's country of origin either by the server's geographic location (GeoID) or by its Top-Level Domain (TLD). Both methods have limitations. Hosting costs, server location strategies, and domain ownership practices mean that a website's server may not reflect its target audience. Globally-used TLDs like `.ai` or `.io` aren't necessarily tied to their countries of origin.

[213] *https://natlawreview.com/article/two-major-lawsuits-aim-answer-multi-billion-dollar-question-can-ai-train-your*
[214] *https://www.boia.org/blog/ethics-of-ai-in-accessibility-avoiding-bias-when-using-generative-ai-text*

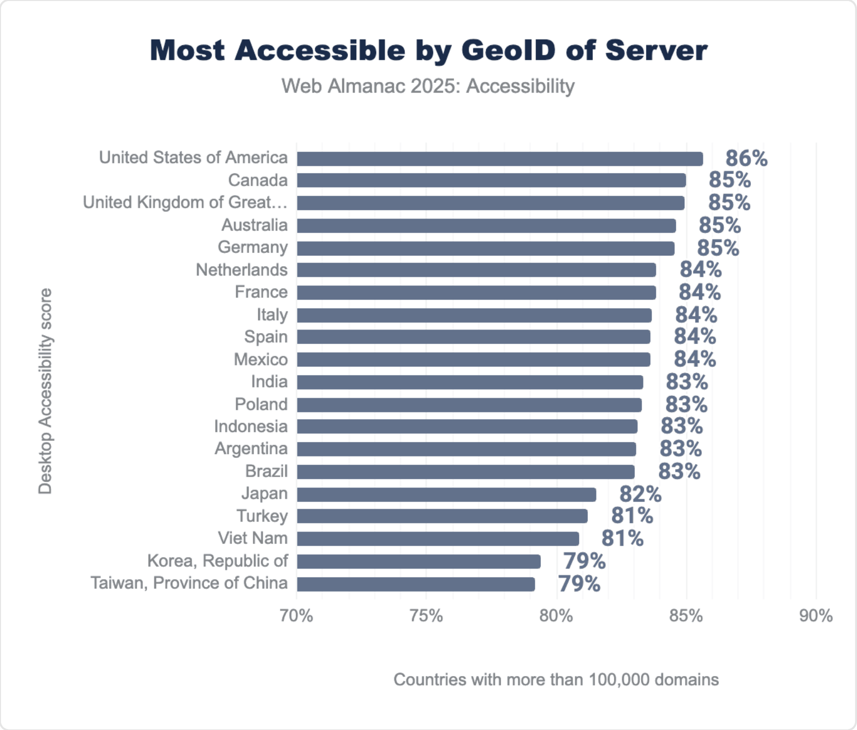


Figure 6.33. Most Accessible by GeoID of Server

In 2025, the United States remains the most accessible country by GeoID, a position driven by decades of Section 508 compliance requirements for federal agencies and ongoing ADA Title III litigation. The `.edu` and `.gov` TLDs also lead accessibility metrics, reflecting mandatory compliance for U.S. government and educational institutions.

We also note the EU Accessibility Act's limited impact in 2025. While the Act became fully effective on June 28, 2025, mandating that private and public sector digital services be accessible across the European Union, preliminary data shows no dramatic spike in website accessibility for European-based sites.

This lag likely reflects implementation challenges, transitional periods for existing services, and the time required for organizations to redesign and audit their digital offerings.

Legal enforcement and the threat of litigation remain the strongest drivers of accessibility compliance, as evidenced by the United States' leading position. The full impact of newer European and global legislation may take years to manifest in web accessibility statistics, as organizations work through implementation timelines and transition periods.

We noticed a notable trend emerging in 2025. `.ai` domains appeared for the first time in accessibility rankings, now outperforming all TLDs except `.edu` and `.gov`. This likely reflects the growing adoption of AI-related businesses, many of which prioritize modern development practices, including accessibility.

Originally assigned to Anguilla[215], a small Caribbean island, in 1995, the `.ai` TLD extension remained relatively obscure for nearly 15 years until 2009, when Anguilla opened direct registrations worldwide. The domain lay dormant for most of its history until the artificial intelligence boom of 2022 onward transformed it into one of the fastest-growing TLDs globally.

The catalyst came with the arrival of ChatGPT in late 2022, which sparked unprecedented interest in AI. Between July 2022 and July 2023, registered `.ai` domains skyrocketed from 75,314 to 196,292, a 161% increase in just twelve months.

Geographically, North America drives the majority of `.ai` registrations[216] at 62.5%, with the United States alone accounting for 62.5% of all registered `.ai` domains. Asia follows at 18.8%, and Europe at 17.2%.

Many `.ai` domain holders are venture-backed, well-funded tech startups which are more likely to use AI. It's still up for debate whether AI can produce more accessible code than human teams, but this is one indicator.

Unlike older traditional companies using `.com` or `.org`, newer AI companies often build with contemporary web standards and tools that consider accessibility from the start. These companies that use an `.ai` domain and sell AI products are very likely to have involved AI in building their website, at least by consulting it for code or content, if not by using it in more automated agentic setups.

Traditional TLDs like `.com`, `.org`, `.net` don't rank as accessibility leaders. This suggests that domain type alone isn't a strong predictor of accessibility compliance.

[215] https://en.wikipedia.org/wiki/Anguilla
[216] https://techjury.net/industry-analysis/the-rise-of-ai-domains/

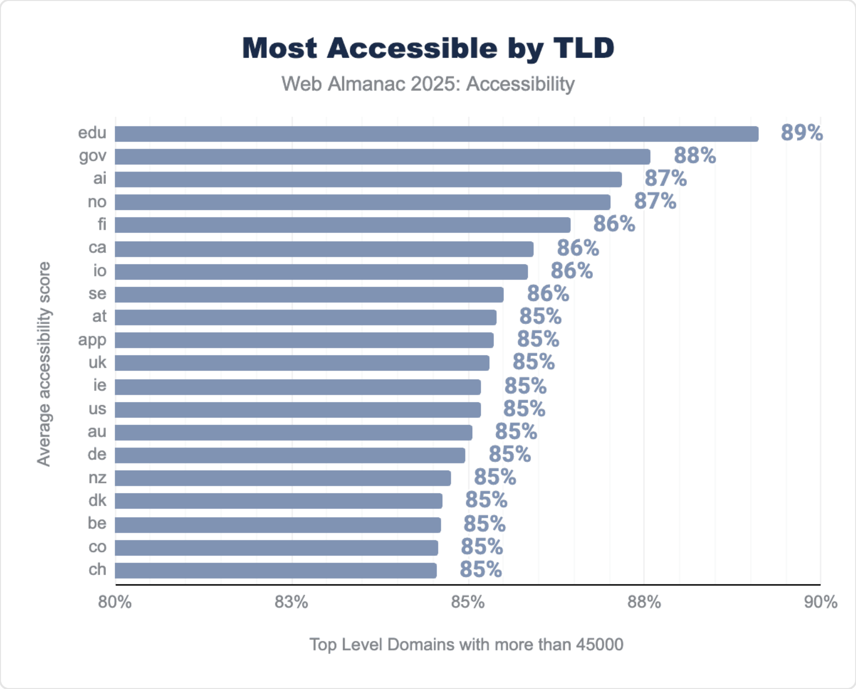


Figure 6.34. Accessible countries by Top Level Domain (TLD).

The map of TLD ranking is very similar to 2024, but obviously doesn't include the increasing number of non-country specific TLDs now available.

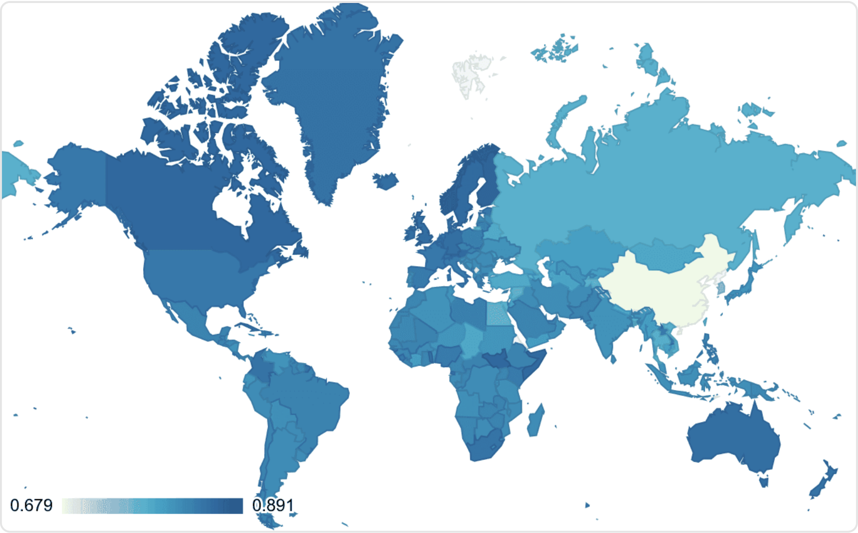


Figure 6.35. Map of ccessible countries by Top Level Domain (TLD).

Government

Government websites remain a critical arena for demonstrating public commitment to accessibility. But implementation varies dramatically across jurisdictions. The 2025 data reveals important trends in global government website accessibility, influenced by recent legislation, methodological changes in data collection, and enforcement mechanisms.

We used a different methodology this year to be able to assess a broader range of domains which fell outside of scans in 2024.

In 2024, we only sampled 79 domains from the government of the Netherlands. In 2025 we queried over 10 times that number with 957 domains. We similarly scanned about twice the number of domains for Luxembourg and Finland. The greater accuracy means we have a more comprehensive dataset, but also a more complex year-over-year comparison.

In the United Kingdom, we saw an improvement to 94% accessibility, up 2% from 2024. This reflects the benefits of standardized design systems, such as the UK Government's Digital Service Standard[217], which prioritizes accessibility across all public digital services. The Netherlands, Luxembourg, and Finland continue to lead, with the Netherlands achieving 98% in previous years. They have maintained this position through consistent governance frameworks and design system prioritization.

[217] https://www.gov.uk/service-manual/service-standard

We also made an effort to include Scotland (gov.scot) and Wales (gov.wales). In 2025, we averaged from 19,568 domains, while in 2024 it was only 16,594.

Monitoring is a key part of prioritizing accessibility, and we applaud dashboards like the French Government's[218] which highlight progress on a number of website quality indicators, including accessibility. Much of the code behind this is open source. The Dutch government also has a dashboard[219] that keeps track of almost 9.000 websites and applications. At the time of writing, 60% of all tracked web properties comply with legal requirements.

The Accessibility Monitoring Reports done by AccessibleEU[220] are important, but much more abstracted. The Web Accessibility Directive[221] in the EU requires member states to monitor and report accessibility compliance every 3 years. These reports[222] are publicly available.

The European Union's evolving regulatory landscape has significantly impacted government website accessibility in 2025.

The EU Web Accessibility Directive requires public sector organizations to meet specific technical standards. The broader European Accessibility Act (EAA), which became fully effective on June 28, 2025, extends requirements to private sector organizations in key sectors such as e-commerce, travel, and banking.

Despite this regulatory momentum, 2025 accessibility data shows no dramatic spike in European government website compliance, suggesting that implementation is still underway and the full impact may not be visible until 2026.

EU Member States are required to publish accessibility statements and provide feedback mechanisms for users to report barriers. Accessibility statements[223] are an important part of the EU's Web Accessibility Directive, but as yet, we don't have a good way to include them in the site scans. The Funka Foundation has reminded us of the limitations of this type of testing for compliance[224].

[218] *https://observatoire.numerique.gouv.fr/observatoire*
[219] *https://dashboard.digitoegankelijk.nl/*
[220] *https://accessible-eu-centre.ec.europa.eu/accessibility-monitoring_en*
[221] *https://en.wikipedia.org/wiki/Web_Accessibility_Directive*
[222] *https://digital-strategy.ec.europa.eu/en/news/web-accessibility-directive-monitoring-reports-2022-2024*
[223] *https://cerovac.com/a11y/2025/07/we-need-to-talk-about-your-accessibility-statement/*
[224] *https://www.linkedin.com/pulse/yes-nordics-score-well-lets-think-one-step-further-funkafoundation-yczzf/*

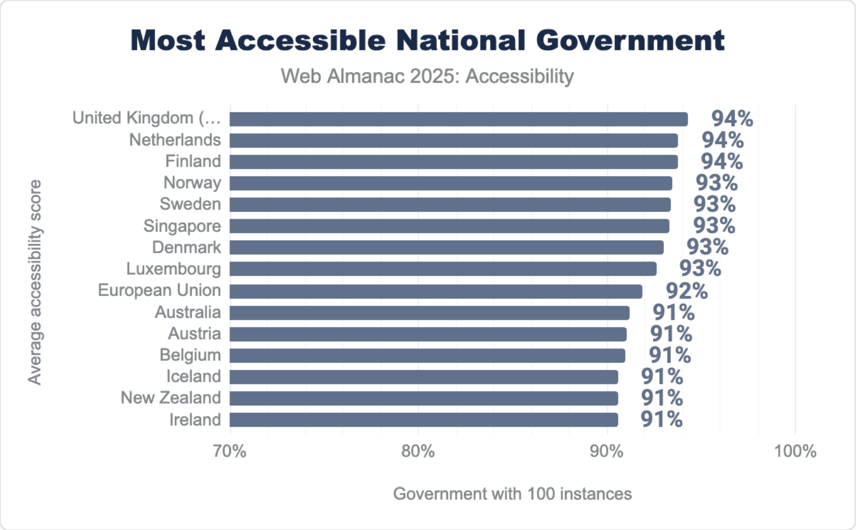


Figure 6.36. Most accessible national government.

The map of 2025 is almost indistinguishable from 2024.

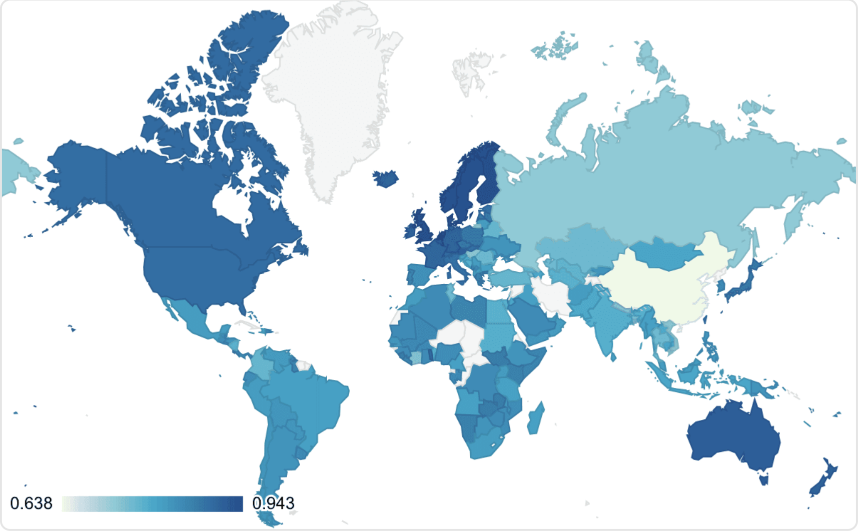


Figure 6.37. Map of the accessibility of global government websites.

In the United States, state government compliance remains inconsistent despite new federal mandates. The Department of Justice's (DOJ) final rule on Title II of the Americans with

Disabilities Act, published in June 2024, requires all state and local government entities to achieve WCAG 2.1 AA compliance by specific deadlines: April 26, 2026, for entities serving populations of 50,000 or more, and April 26, 2027, for smaller entities.

States like Colorado and Vermont have excelled by establishing centralized governance structures. Colorado's Statewide Internet Portal Authority[225] (SIPA) demonstrates how centralized management improves accessibility across multiple agencies.

Nevada, Kansas, California, and New York did well in the samples from both years. But the averages don't indicate that state governments made any significant progress in achieving the new requirements from the 2024 US Department of Justice Final Rule to Strengthen Web and Mobile App Access[226]. State technology leaders at the National Association of State Chief Information Officers (NASCIO) national conference reaffirmed accessibility as a top priority[227].

The Americans with Disabilities Act (ADA), which turned 35 in July[228], was pioneering work that set a precedent globally.

Singapore's recent commitment to accessibility[229] improvement, demonstrated through the open source tool Oobee[230], shows emerging global momentum. Oobee allows organizations to scan hundreds of pages and generate consolidated accessibility reports, positioning it as a Digital Public Good[231].

[225] *https://sipa.colorado.gov/*
[226] *https://www.justice.gov/archives/opa/pr/justice-department-publish-final-rule-strengthen-web-and-mobile-app-access-people*
[227] *https://statescoop.com/accessibility-nascio-state-priority-2025/*
[228] *https://www.access-board.gov/news/2025/07/21/celebrating-35-years-of-americans-with-disabilities-act/*
[229] *https://archive.opengovasia.com/2025/03/11/digital-accessibility-singapores-commitment-to-inclusivity/?c=ca*
[230] *https://github.com/GovTechSG/oobee*
[231] *https://www.digitalpublicgoods.net/r/oobee*

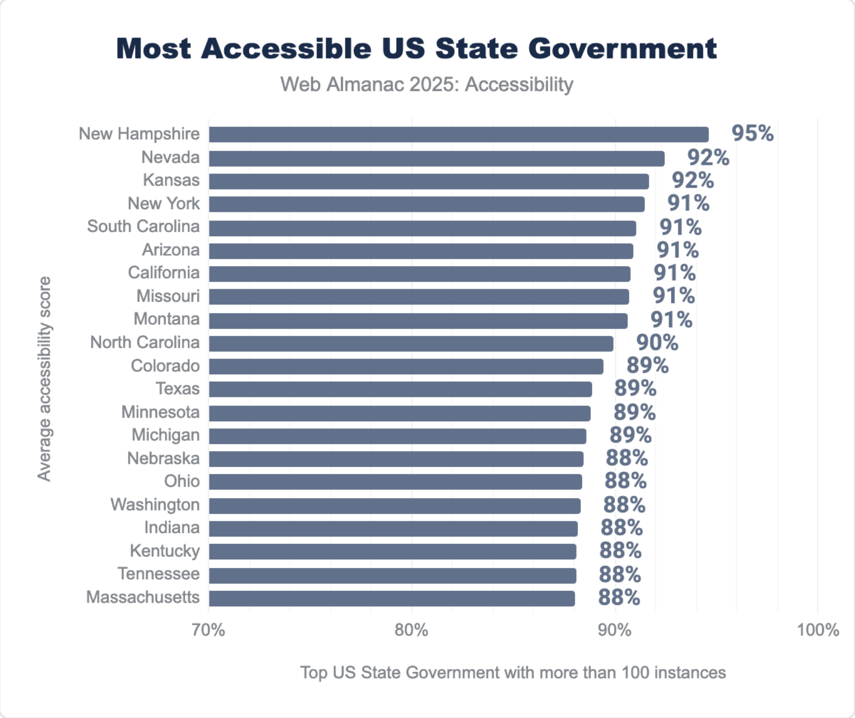


Figure 6.38. The most accessible US state governments.

Content Management Systems (CMS)

A website's choice of Content Management System (CMS) significantly influences its accessibility outcomes. Using Wappalyzer[232] data, the 2025 Web Almanac compared accessibility scores across traditional CMSs, platforms, and specialized website builders. This revealed both consistent patterns and notable outliers.

[232] *https://almanac.httparchive.org/en/2024/methodology#wappalyzer*

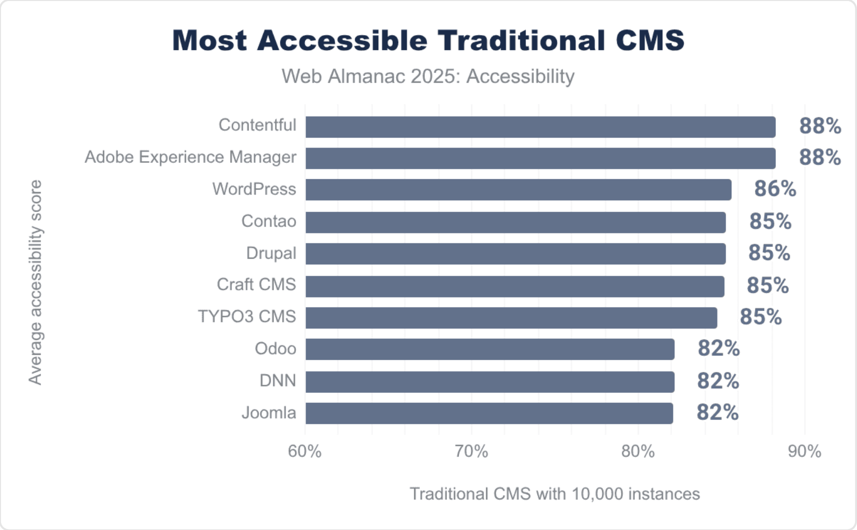


Figure 6.39. Most accessible traditional CMS.

Among traditional CMSs, Sitecore[233] maintained 85% accessibility in 2025, though its instance count dropped below 10,000. Adobe Experience Manager (AEM)[234] and Contentful[235] continue to lead, likely because larger corporations adopting these enterprise solutions have more resources to address accessibility issues. WordPress[236] showed no significant improvement from 2024, but rose to third place, reflecting its market dominance and the growing accessibility consciousness of its user base.

Remarkably, the top five traditional CMSs share consistent error patterns. Color contrast, link names, and heading order dominate as the most common issues. These errors primarily reflect content choices rather than platform limitations, since a CMS can't dictate link naming or color selections. However, AEM stands alone with `label-content-name-mismatch` in its top-five errors. WordPress is unique in having `meta-viewport` errors.

In the top 10 CMS, only DNN[237] has `image-alt` in the top 3 errors. For most traditional CMSs, `image-alt` and `target-size` are consistently in the fourth or fifth place for Google Lighthouse errors.

[233] https://www.sitecore.com/
[234] https://business.adobe.com/products/experience-manager/sites/aem-sites.html
[235] https://www.contentful.com/
[236] https://wordpress.com/
[237] https://www.dnnsoftware.com/

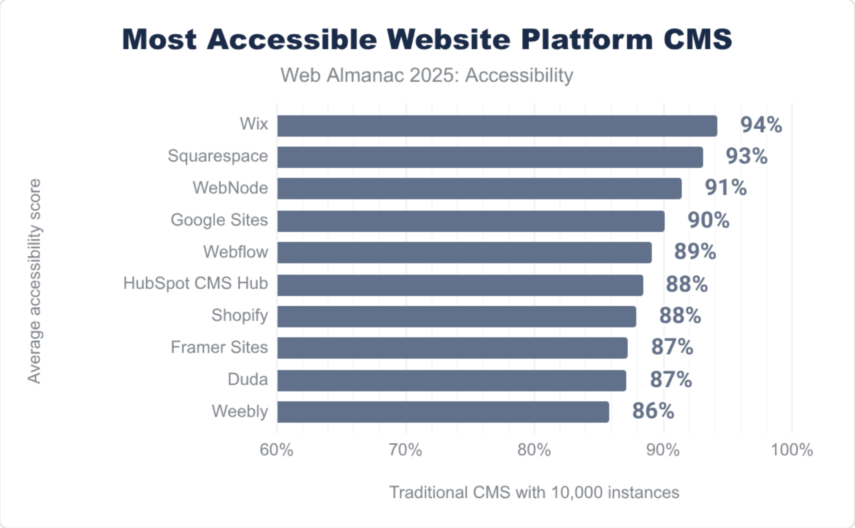


Figure 6.40. Most accessible website platform CMS.

Platform CMS	Most popular	Second most	Third most	Fourth most	Fifth most
Wix[238]	*heading-order*	*link-name*	*color-contrast*	*button-name*	*target-size*
Squarespace[239]	*color-contrast*	*heading-order*	*link-name*	*label-content-name-mismatch*	*frame-title*
Webnode[240]	*heading-order*	*link-name*	*frame-title*	*color-contrast*	*image-redundant-alt*
Google Sites[241]	*image-alt*	*link-name*	*aria-allowed-attr*	*heading-order*	*color-contrast*
Webflow[242]	*link-name*	*color-contrast*	*heading-order*	*html-has-lang*	*target-size*

Figure 6.41. Top accessibility audit issues for popular CMS platforms.

Website platforms like Wix, Squarespace, and Google Sites significantly outperform traditional CMSs in accessibility. This superior performance likely stems from their approach. These platforms often constrain user choices through templated designs and built-in accessibility

[238] https://www.wix.com/
[239] https://www.squarespace.com/
[240] https://www.webnode.com/
[241] https://workspace.google.com/products/sites/
[242] https://webflow.com/

defaults, reducing opportunities for poor accessibility decisions.

The data proves that CMS choice meaningfully impacts accessibility outcomes, even when content creators must take final responsibility for some decisions. Platforms with stricter design constraints and embedded accessibility defaults perform better, while those offering maximum flexibility leave accessibility decisions to users.

JavaScript frontend frameworks

The choice of JavaScript framework also significantly influences a website's accessibility outcomes. Using classifications from the State of JS report[243], we examined how different UI frameworks and meta-frameworks correlate with accessibility performance. This revealed patterns, shifts, and emerging concerns.

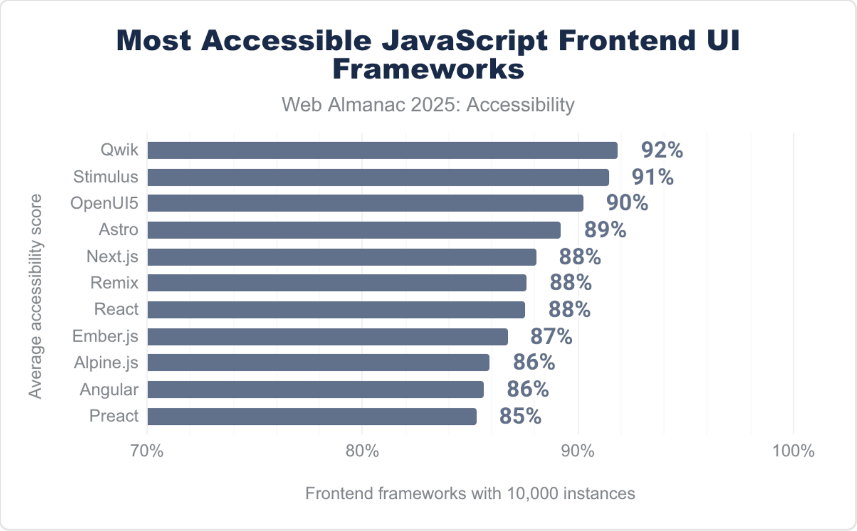


Figure 6.42. Most Accessible JavaScript Frontend UI Frameworks.

In 2025, OpenUI5[244] has risen in accessibility rankings, while frameworks that led in 2024 (Stimulus[245], Remix[246], and Qwik[247]) have shifted positions. Remix appears in both UI and meta-framework categories, but has declined in rankings in 2025, allowing other frameworks to advance. This volatility may reflect sample size changes or real improvements in competing frameworks. The fact that all Remix functionality was integrated right into the next version of

[243] https://stateofjs.com/en-US
[244] https://openui5.org/
[245] https://stimulus.hotwired.dev/
[246] https://remix.run/
[247] https://qwik.dev/

React Router[248], making Remix redundant as a React framework, can also have an impact on rankings.

Historically, Stimulus, Remix, and Qwik outperformed mainstream options like React, Svelte[249], or Ember.js[250] by several percentage points, likely because they prioritize progressive enhancement and semantic HTML.

Among meta-frameworks, Remix, RedwoodJS[251], and Astro[252] led in 2024, with Remix's decline allowing Gatsby[253] to rise to third place in 2025. The rise of server-first meta-frameworks (SvelteKit, Astro, Remix, Qwik, Fresh[254], Analog[255]) reflects a broader industry shift toward better performance and accessibility practices by reducing client-side JavaScript complexity.

The choice of library also has an impact on accessibility.

React offers maximum flexibility and customization, but requires developers to intentionally implement accessibility. Its extensive ecosystem includes accessibility-focused libraries like React Aria[256] and Reach UI[257], but accessibility isn't enforced by default.

Angular[258] provides strong built-in accessibility features, structured conventions that promote semantic HTML, ARIA attribute support, and Material Design[259] components with keyboard navigation and screen reader support out-of-the-box. Angular's opinionated structure tends to guide developers toward more standardized, accessible practices.

Vue.js[260] aims to strike a balance between React's flexibility and Angular's structure. Vue's progressive design, clear template syntax, and component architecture support accessibility, though it relies more on developer discipline and third-party plugins like vue-a11y[261].

We also note that GitHub took the Global Accessibility Awareness Day (GAAD) pledge[262] to improve open source accessibility at scale. This commitment addresses a critical gap: 90% of companies use open source, 97% of codebases contain open source components, and an estimated 70–90% of code within commercial tools derives from open source.

[248] *https://reactrouter.com/*
[249] *https://svelte.dev/*
[250] *http://Ember.js*
[251] *https://rwsdk.com/*
[252] *https://astro.build/*
[253] *https://www.gatsbyjs.com/*
[254] *https://fresh.deno.dev/*
[255] *https://analogjs.org/*
[256] *https://react-spectrum.adobe.com/react-aria/index.html*
[257] *https://reach.tech/*
[258] *https://angular.dev/*
[259] *https://material.angular.io/*
[260] *http://Vue.js*
[261] *https://github.com/vue-a11y*
[262] *https://github.blog/open-source/social-impact/our-pledge-to-help-improve-the-accessibility-of-open-source-software-at-scale/*

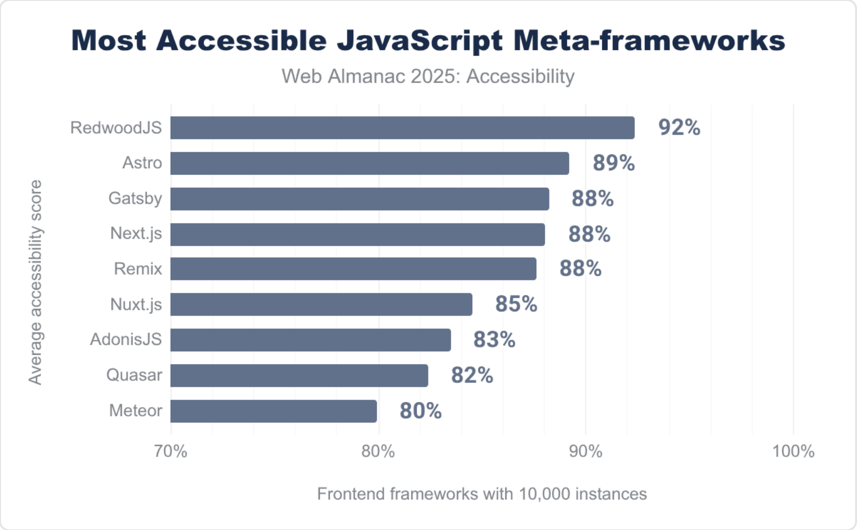


Figure 6.43. Most Accessible JavaScript Meta-frameworks.

RedwoodJS remains the most accessible, followed by Astro and Gatsby.

Conclusion

The 2025 Web Almanac accessibility analysis reveals a year of incremental progress, tempered by significant challenges and missed opportunities.

Automated testing remains essential for assessing accessibility at scale. But the data demonstrates that measurement alone doesn't guarantee meaningful improvement[263]. The web community must move beyond basic compliance metrics to address the systemic issues that continue to exclude millions of users with disabilities.

The most notable improvements in 2025 emerged in sectors and regions where regulatory pressure and enforcement mechanisms are strongest. However, the lack of dramatic improvement following the European Accessibility Act's June 28, 2025, deadline is instructive. The full impact may not be apparent until 2026 and beyond.

The rapid rise of the `.ai` domain to among the most accessible TLDs reflects an important pattern. Newer, venture-backed technology companies tend to build with modern accessibility practices from the start, whereas legacy websites often remain inaccessible. This proves that

[263] *https://www.a11y-collective.com/blog/how-to-check-web-accessibility/*

accessibility is achievable when prioritized early in development.

Despite improvements in specific areas, the core accessibility barriers identified in 2024 persist largely unchanged in 2025.

Color contrast, link naming, heading hierarchy, and image `alt` text remain the top four issues across nearly every platform and framework. These aren't failures directly of the tool, but how the authors use them. Authors have some role in all of these failures according to the W3C's Accessibility Roles and Responsibilities Mapping (ARRM)[264].

This reality reveals a critical insight. CMS platforms, JavaScript frameworks, and web technologies can provide accessibility foundations, but they can't force content creators to make accessible choices. Approaches like the Authoring Tool Accessibility Guidelines (ATAG) 2.0[265] and the new W3C ATAG Community Group[266] could help.

The 2025 metrics suggest stagnation where we expected incremental improvement, highlighting the gap between what's easy to measure and what's easy to fix.

The continued rise of accessibility overlays (now on 2% of sites) is concerning. It seems that organizations often choose shortcuts over genuine accessibility. The IAAP and European Disability Forum have explicitly warned that overlays can interfere with users' assistive technology and must never replace accessible design. The 2025 data confirms overlays remain concentrated in lower-traffic sites, a sign that high-quality, well-resourced organizations are moving away from them toward real solutions.

The 2025 data underscores that automation is necessary but insufficient. Lighthouse and similar tools detect easily measurable violations, yet 50% of images on the web have empty or inadequate `alt` text. Heading hierarchy can be audited, but semantic meaningfulness requires human judgment. Color contrast can be checked, but visual design choices involve subjective artistic decisions informed by accessibility requirements.

Our 2025 findings reveal a web that remains largely inaccessible for millions of people with disabilities.

While incremental improvements in specific areas offer encouragement, persistent gaps in color contrast, link naming, heading structure, and image descriptions demonstrate that the web community hasn't yet made accessibility a genuine priority.

The rise of `.ai` domains, GitHub's open source pledge, and regulatory deadlines like the EU Accessibility Act and ADA Title II final rule offer hope that 2026 may see more substantial change. That's only if organizations move beyond measurement to accountability, from rhetoric

[264] *https://www.w3.org/WAI/planning/arrm/tasks/*
[265] *https://www.w3.org/WAI/standards-guidelines/atag/*
[266] *https://www.w3.org/community/atag/*

to resources, and from compliance to genuine inclusion.

The web should work for everyone. Until that principle guides our design, development, and deployment decisions, the accessibility gaps documented in this report will persist.

Authors

Bogdan Lazar

@bogdanlazar.com tricinel tricinel https://bogdanlazar.com

Bogdan is the accessibility consultant who helps product owners ship accessible websites without blocking ongoing work. He brings over a decade of experience in education and healthcare, with deep expertise in inclusive design and web accessibility. He's been writing a daily newsletter[267] on accessibility since March 2024, driven by one core belief. "Make room for everyone."

Mike Gifford

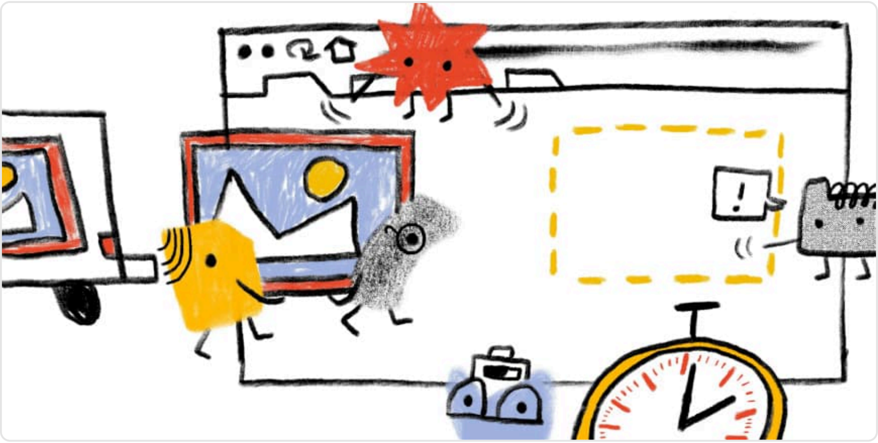
@https://mastodon.social/@mgifford @mgifford.bsky.social mgifford mgifford

https://accessibility.civicactions.com/

Mike Gifford is CivicActions' Open Standards & Practices Lead. He is also a thought leader on open government, digital accessibility and sustainability. He has served as a Drupal Core Accessibility Maintainer and also a W3C Invited Expert. He is a recognized authoring tool accessibility expert and contributor to the W3C's Draft Web Sustainability Guidelines (WSG) 1.0.

[267] *https://dailyaccessibility.com*

Part II Chapter 7
Performance



Written by Himanshu Jariyal, Prathamesh Rasam, Humaira, and Aaron T. Grogg
Reviewed by Barry Pollard, Stoyan Stefanov, and Tanner Hodges
Analyzed by Tanner Hodges
Edited by Barry Pollard

Introduction

Web performance refers to how quickly and smoothly web pages load and also respond to user interactions. Performance plays an important role in shaping engagement, retention, and overall trust, particularly as the web is used across a wide range of devices and network conditions. Pages that feel fast and responsive encourage exploration and continued use, while experiences that feel slow or unpredictable can interrupt flow and reduce confidence. Understanding the factors that influence performance is therefore essential to building web experiences that feel reliable to end users.

Measuring web performance includes a broad set of metrics that describe how pages load, render, and respond to user input in real-world conditions. It is not always possible for the web to feel instantaneous due to device, network, and execution constraints. As a result, performance is not only about speed, but also about how an experience feels while work is in progress. Providing clear feedback while content loads and keeping layouts visually stable when expected helps users understand page behavior and feel in control as they interact with a website.

These considerations have influenced the development and adoption of user-centric performance metrics[268] called Core Web Vitals: Largest Contentful Paint (LCP)[269], Interaction to Next Paint (INP)[270], and Cumulative Layout Shift (CLS)[271], which capture key aspects of loading performance, responsiveness, and visual stability. Over the last year, support for reporting two of the Core Web Vitals—LCP and INP—has expanded[272] beyond Chrome to include other browsers, allowing user experience to be measured more consistently across browser engines.

These metrics are complemented by more traditional indicators such as Time to First Byte (TTFB)[273] and First Contentful Paint (FCP)[274], as well as measures of page resource loading behavior. Together, this broader set of signals helps describe where performance bottlenecks tend to occur and how they relate to overall page behavior. A more comprehensive overview of modern web performance metrics is available at web.dev[275].

This Performance chapter examines these signals at scale across devices and network conditions to provide a data-driven view of the state of web performance. By analyzing real-world data, it highlights where the web is improving, where challenges remain, and which patterns are associated with better user experiences. This year's analysis also includes emerging performance features such as Early Hints[276] and Speculation Rules[277].

Data sources and methodology

This chapter draws on data from the HTTP Archive[278] and the Chrome UX Report (CrUX)[279], combining lab-based measurements with real-user performance data. The HTTP Archive collects Chrome-based page load data via WebPageTest, providing detailed insight into how pages behave under controlled conditions, while CrUX reflects real-world user experiences collected from Chrome users. The primary analysis is based on measurements from July 2025 and spans millions of websites and a large volume of page loads across the web. Additional details on data collection and methodology are available in the Methodology.

[268] *https://web.dev/articles/user-centric-performance-metrics*
[269] *https://web.dev/articles/lcp*
[270] *https://web.dev/articles/inp*
[271] *https://web.dev/articles/cls*
[272] *https://www.debugbear.com/blog/firefox-safari-web-vitals*
[273] *https://web.dev/articles/ttfb*
[274] *https://web.dev/articles/fcp*
[275] *https://web.dev/performance*
[276] *https://developer.chrome.com/docs/web-platform/early-hints*
[277] *https://developer.chrome.com/docs/web-platform/implementing-speculation-rules*
[278] *https://httparchive.org/faq*
[279] *https://developer.chrome.com/docs/crux*

Core Web Vitals summary

Core Web Vitals are Google's main metrics for understanding how a webpage feels to real users. A page is considered good when:

- **Largest Contentful Paint (LCP)**: the main content appears quickly (within 2.5 seconds) so the page feels useful.
- **Interaction to Next Paint (INP)**: the page responds to clicks or taps almost immediately (within 200 milliseconds).
- **Cumulative Layout Shift (CLS)**: the layout stays mostly stable, with very little unexpected movement (score ≤ 0.1).

When a page meets these thresholds for most users, it is classified as delivering a "good" overall page experience.

Of course these are broad measures intended to an estimated categorization across the web as a whole. Individual websites may find that the expectations of "good" for their users may differ.

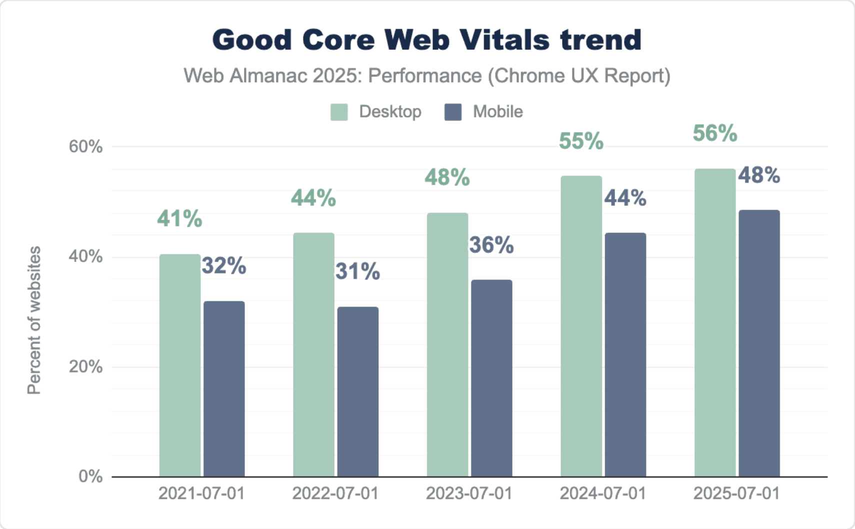


Figure 7.1. Good Core Web Vitals trend.

Mobile Core Web Vitals have shown consistent year-over-year improvement, increasing from 36% in 2023 to 44% in 2024, and reaching 48% in 2025. This rise may reflect improvements in browsers, devices, and networks, alongside site optimizations.

Desktop performance also saw a positive trend, moving from 48% in 2023 to 55% in 2024. However, the improvement for 2025 was marginal, increasing only to 56%.

To better understand these trends, the following section examines how Core Web Vitals vary by page popularity, where more popular pages appear at lower rank values.

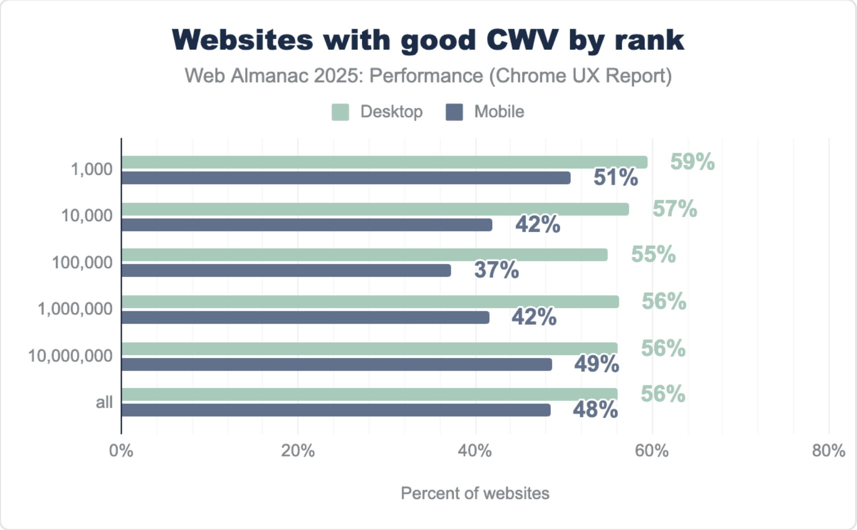


Figure 7.2. Websites with good CWV by rank.

On mobile, the most and least popular sites tend to perform better than those in the middle of the popularity distribution. The most popular sites show better Core Web Vitals results, while performance drops for mid-popularity sites before improving again among the least popular sites.

- 51% of the 1,000 most popular mobile sites have good Core Web Vitals (CWV).
- CWV drops to 42% for the next 10,000 sites and 37% for the next 100,000.
- However, it improves to 42% for the next 1,000,000 sites and 48% for the next 10,000,000.

This pattern may reflect differences in page complexity and performance investment across the popularity tiers.

- Highly popular sites often treat performance as a priority and are more likely to invest in ongoing optimization, given its close correlation[280] to user engagement and business outcomes.

- Mid-popularity sites may combine higher complexity such as additional features and third-party scripts with less sustained focus on performance, leading to drop in results.
- Less popular sites are often simpler, with fewer features and lighter pages, which can benefit from platform defaults and hence offer comparatively better performance.

This U-shaped pattern is more evident on mobile, where slower devices and less stable network conditions tend to amplify the effects of page complexity and limited optimization. On desktop, more powerful hardware and more stable networks can reduce the visible impact of these differences.

Performance can also vary significantly between primary and secondary page navigations. Primary navigations typically occur when a user lands on a site for the home page, requiring more resources to be fetched and executed, while secondary navigations happen as users move between pages within the same site and can benefit from previously loaded and cached resources.

While most CrUX data in this chapter is per origin, our crawler also collects page-level CrUX data at the time of crawl where available allowing us to examine how Core Web Vitals differ between home and secondary page navigations.

[280] *https://www.speedcurve.com/blog/site-speed-business-correlation/*

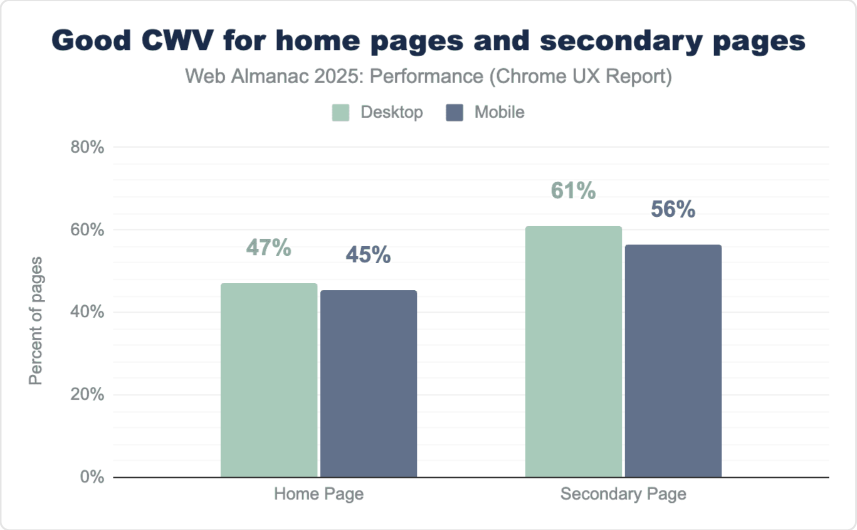


Figure 7.3. Good CWV for home pages and secondary pages.

Secondary pages show higher CWV pass rates than home pages, with a 14% lead on desktop and an 11% lead on mobile. This performance gap suggests that secondary pages often benefit from having cached information, which contributes to faster page loads. Home pages are also updated more frequently and tend to include more dynamic and varied components, while secondary pages are often more templated and consistent, which may make them more stable and easier to optimize.

Modern single page websites often use JavaScript-based navigations, where content changes without a full page reload. While these navigations feel like moving between pages to users, they are not always fully captured by current Web Vitals measurements. Support for soft navigations[281] is expected to improve how Core Web Vitals are captured for these in-page transitions, providing a more accurate view of real user experience beyond the initial page load.

Loading speed

A major factor influencing a user's perception of quality and reliability is the initial loading speed of a website. However, 'speed' is inherently relative and difficult to define with a single value in the context of websites. Because performance varies based on a user's device capabilities and network conditions, we cannot rely on a single 'load time' to capture the user

[281] https://developer.chrome.com/blog/new-soft-navigations-origin-trial

experience. Thus, we look at multiple user-centric metrics[282] that measure not just how fast a site loads, but how fast it *feels*.

The following sections focus on two key loading metrics: First Contentful Paint (FCP) and Largest Contentful Paint (LCP).

First Contentful Paint

To understand the user's first impression of a webpage's speed, we look at First Contentful Paint (FCP)[283]. This metric captures the exact time it takes for a page to begin displaying *any* content, measured from the point the user first requested the page. A page that has a FCP score under 1.8 seconds is considered "good", scores between 1.8 and 3.0 seconds indicate that the page "needs Improvement", and a score over 3.0 seconds is considered "poor" performance.

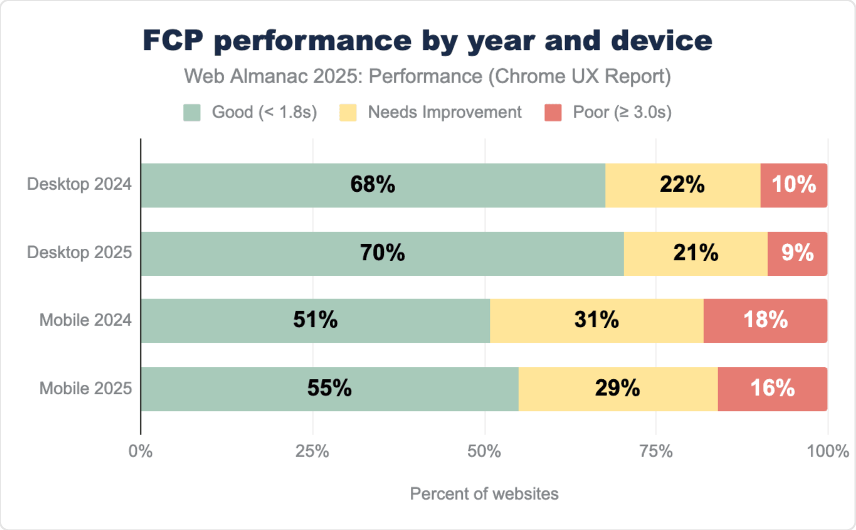


Figure 7.4. FCP performance by year and device.

FCP performance improved across both desktop and mobile since 2024. The share of desktop sites achieving a "Good" FCP increased by 2%, while mobile sites saw a larger gain of 4%. FCP can be broadly understood as consisting of two main parts, each influenced by different aspects of the loading process:

- The first is the network and server overhead, captured by Time to First Byte

[282] https://web.dev/articles/user-centric-performance-metrics
[283] https://web.dev/articles/fcp

(TTFB)[284]. This includes connection setup, redirects, and server processing time, and is largely influenced by network infrastructure and protocol efficiency. When a Service Worker serves a response from cache, the network round trip can be avoided, improving TTFB on repeat visits. However, Service Worker startup can also add latency, which Navigation Preload[285] helps mitigate by starting the network request in parallel during initialization.

- The second part is client-side rendering, which begins after the first byte is received. This reflects the time required for the browser to parse resources and render the first visible content on the page, and is influenced by factors such as browser behavior, render-blocking resources, and user hardware capabilities.

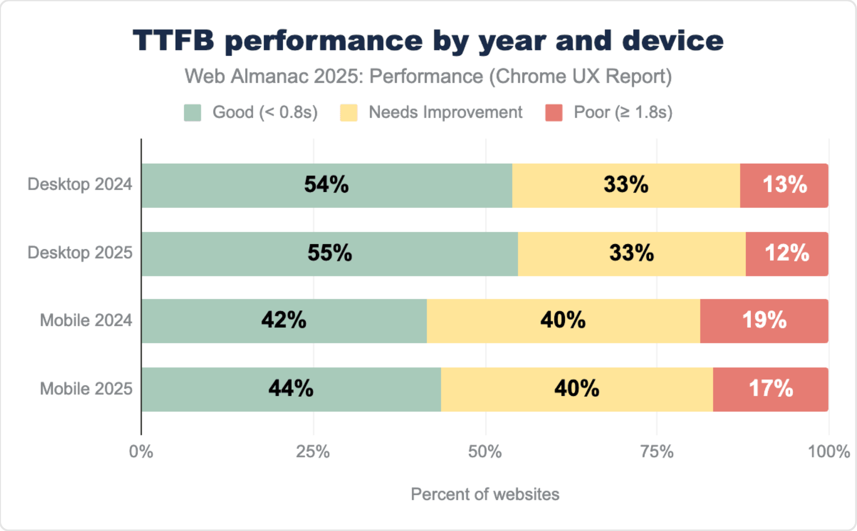


Figure 7.5. TTFB performance by year and device.

Since 2024, the share of sites achieving a "Good" TTFB increased by 1% on desktop and by 2% on mobile.

[284] *https://web.dev/articles/ttfb*
[285] *https://web.dev/blog/navigation-preload*

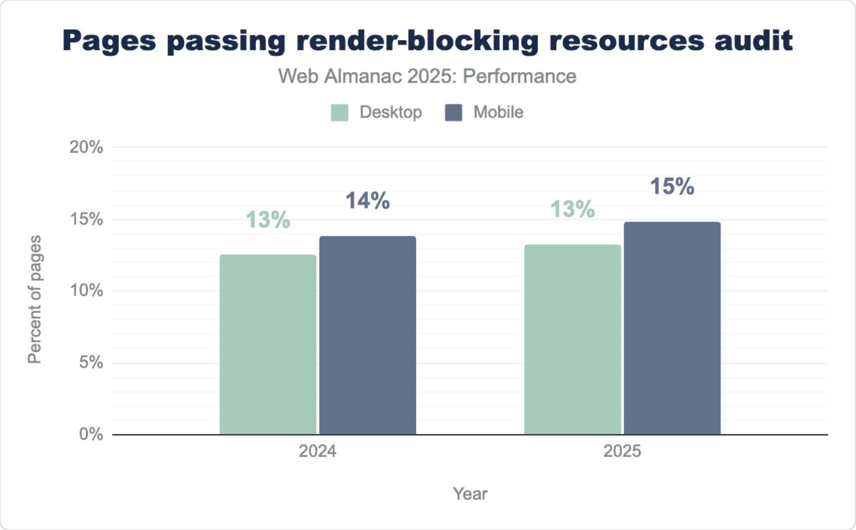


Figure 7.6. Pages passing render-blocking Lighthouse audit.

Over the same period, the proportion of pages passing the "render-blocking resources audit" remained flat on desktop and increased by 1% on mobile.

Taken together, improvements in FCP between 2024 and 2025 align with these modest gains in server response times and small reductions in render-blocking work. This suggests that incremental improvements across both network delivery and client-side rendering are contributing to earlier first paint, with slightly more impact on mobile devices.

While FCP captures the first visual response, LCP reflects when the page's primary content becomes visible and typically involves a longer and more complex critical path. Like FCP, LCP can be understood as the sum of several sequential phases: the time to receive the first byte from the server (TTFB), the delay before the browser begins fetching the LCP resource (resource load delay), the time spent loading that resource (resource load duration), and any delay before the element is rendered (element render delay). Understanding where time is spent across these phases is key to improving LCP, and in turn, overall Core Web Vitals performance[286].

Largest Contentful Paint

To understand when a page feels meaningfully loaded, we look at Largest Contentful Paint

[286] https://web.dev/articles/defining-core-web-vitals-thresholds

(LCP)[287]. This metric measures the time from when the user first requests the page to when the largest visible element—typically a hero image, headline, or prominent text block—finishes rendering on screen. A page with an LCP score under 2.5 seconds is considered "good", scores between 2.5 and 4.0 seconds indicate that the page "needs improvement", and a score over 4.0 seconds is considered "poor" performance.

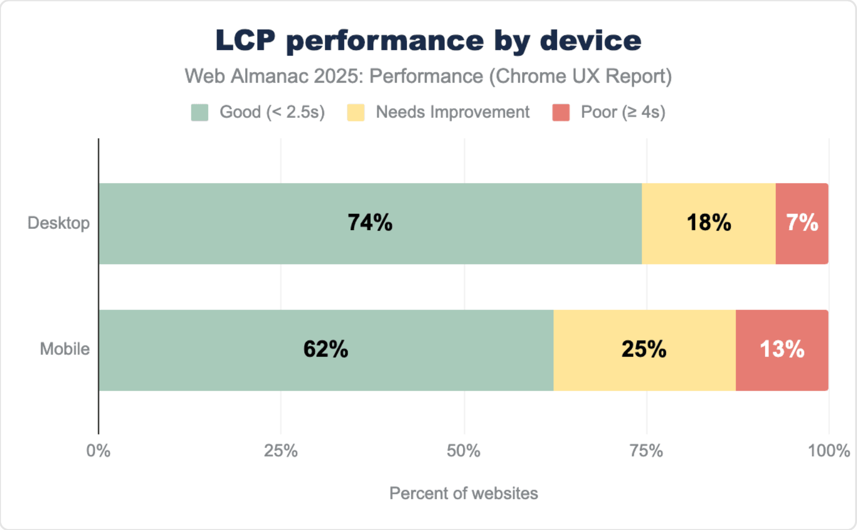


Figure 7.7. LCP performance by device.

Currently, 74% of desktop pages achieve a "good" LCP score compared to 62% on mobile, with mobile also showing nearly double the rate of "poor" experiences (13% versus 7%); a gap that is consistent with the combined effects of slower networks and less capable devices on mobile.

LCP content types

To optimize LCP effectively, we first need to understand what types of content typically become the LCP element.

[287] *https://web.dev/articles/lcp*

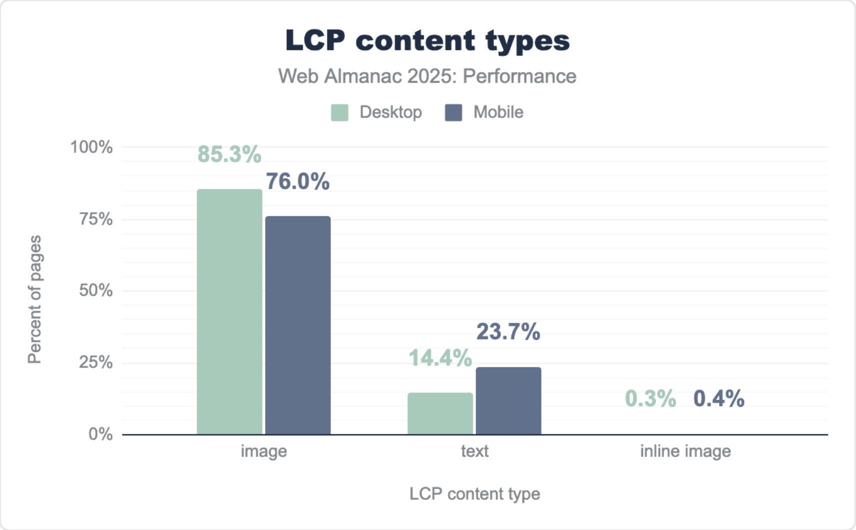


Figure 7.8. LCP content types.

The trend in LCP content types is similar to previous years (see also 2022[288] and 2024[289] data). Images continue to dominate LCP elements across both device types, with 85.3% of desktop pages and 76% of mobile pages having an image as their LCP element. Text-based LCP elements account for much of the remainder—14.4% on desktop versus 23.7% on mobile. This gap likely reflects responsive design practices where hero images are resized, replaced with smaller visuals, or removed entirely on narrower viewports, allowing headline text to become the largest visible element instead.

Inline images (data URIs embedded directly in HTML) remain rare at around 0.5% of pages, indicating limited and careful adoption and awareness of the trade-offs related to larger HTML payloads and caching efficiency.

LCP image formats

Given this continued dominance of images as the LCP element, it becomes relevant to look at the image formats in use, as it directly affects the resource load duration phase of LCP. While the 2024 chapter showed this phase has less optimization potential than others[290], image format efficiency still contributes to overall performance.

[288] *https://almanac.httparchive.org/en/2022/performance#fig-8*
[289] *https://almanac.httparchive.org/en/2024/performance#lcp-content-types*
[290] *https://almanac.httparchive.org/en/2024/performance#lcp-sub-parts*

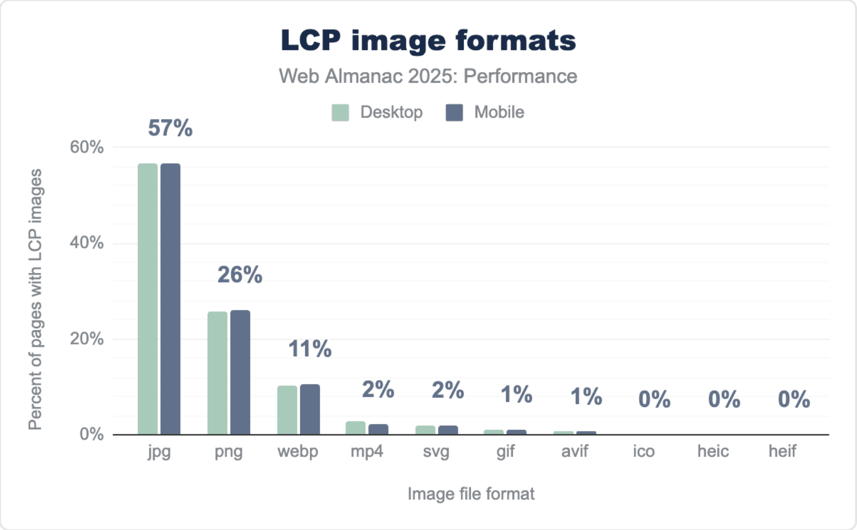


Figure 7.9. LCP image formats.

Modern formats like WebP and AVIF offer better compression than legacy formats, meaning smaller file sizes and faster transfers. However, we see that legacy JPG and PNG are still highly used (JPG accounting for 57% of LCP images and PNG at 26%).

There are some encouraging signs though, such as JPG usage has decreased by 4% since 2024[291] while WebP has increased by 4%.

With PNG and other formats being the same as their 2024 percentages (aside from AVIF reaching 0.7%), it looks like web pages are moving from JPG to WebP, albeit slowly. This slow adoption may reflect the cost of migrating existing image pipelines and content libraries, even as modern formats have broad support.

Cross-origin LCP images

The origin of an LCP image affects how quickly the browser can begin downloading it, impacting the resource load delay phase. When an image is hosted on the same domain as the page, the browser can reuse the existing connection. Cross-origin images may incur additional connection setup (DNS/TCP/TLS), especially when the origin isn't already connected, increasing the time before the download can start.

[291] https://almanac.httparchive.org/en/2024/performance#fig-19

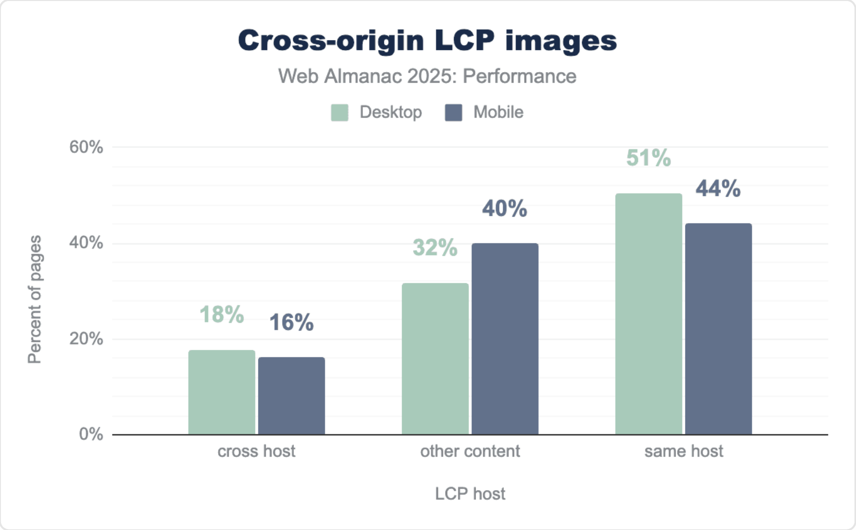


Figure 7.10. Cross-origin LCP images.

51% of desktop pages and 44% of mobile pages serve their LCP image from the same host as the document. Cross-hosted LCP images account for 16-18% of pages—a meaningful portion that may be paying a connection overhead cost unless mitigated with preconnect hints[292].

The “other content” category (32% desktop, 40% mobile) represents pages where the LCP element isn’t an image at all, likely text blocks or background elements. The higher mobile percentage for “other content” may reflect responsive design patterns where hero images are deprioritized on smaller viewports, though we do not conclusively know using this data alone.

LCP resource prioritization

Since resource load delay phase often constitutes a large portion of LCP time, browsers provide tools to help accelerate critical resources. The `fetchpriority="high"` attribute tells the browser to prioritize a resource higher than it normally would—useful since images are typically not considered high priority even when they’re the LCP element. Meanwhile, `<link rel="preload">` instructs the browser to fetch a resource before it would naturally discover it in the HTML.

[292] *https://web.dev/learn/performance/resource-hints#preconnect*

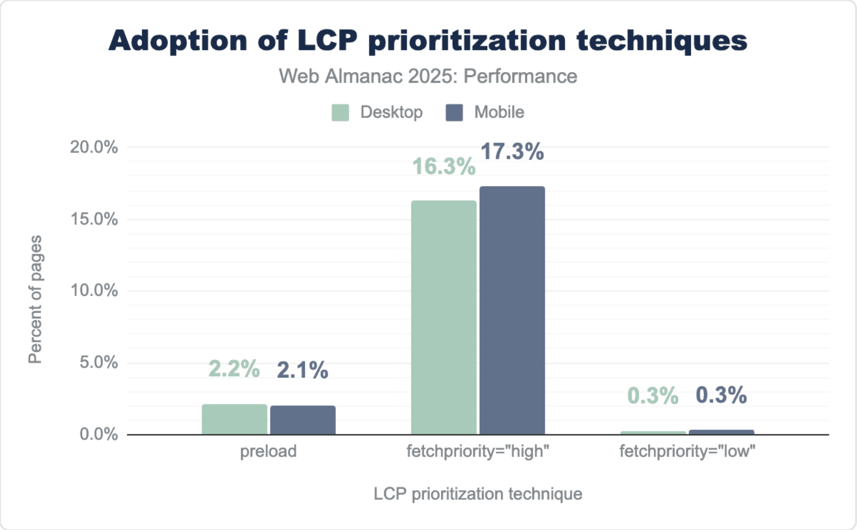


Figure 7.11. Adoption of LCP prioritization techniques.

Adoption of `fetchpriority="high"` has continued its growth, now appearing on 17% of mobile pages with LCP images—up from 15% in 2024[293]. Preload usage remains low at 2.1-2.2%.

Both techniques are relatively simple to implement, though it should be noted that preload is only necessary when the resource is not quickly discoverable in the HTML document. Additionally when preload is used, it should be combined with `fetchpriority="high"` to ensure the image is fetched with high priority—which is not guaranteed by just using preload alone.

The 0.3% of pages using `fetchpriority="low"` on their LCP images is likely unintentional, since identifying which image will become the LCP element at development time can be tricky for developers (varies by viewport and content).

LCP lazy loading

Lazy loading for images means delaying the loading of images until they are needed, for example loading below the fold images only when they are close to entering the user's viewport, instead of the default loading all images at page load. This helps prioritize critical above-the-fold content. Lazy loading is generally a useful optimization, but applying it to the LCP image can be harmful because it delays the main content users are waiting to see.

[293] https://almanac.httparchive.org/en/2024/performance#lcp-prioritization

17%

Figure 7.12. Percentage of mobile pages that lazy load the LCP image.

Overall, about 16%-17% of pages lazy-load their LCP image, a figure that has held steady since 2024. However, the composition has shifted: native `loading="lazy"` usage has increased slightly (from 9.5% to 10.4% on mobile, 10.2% to 11.5% on desktop), while custom approaches like hiding sources behind `data-src` attributes have decreased (from 6.7% to 5.9% on mobile). Native `loading="lazy"` accounts for a larger share of LCP lazy-loading than custom approaches, indicating a shift toward standardized browser features.

Loading speed conclusion

In summary, the loading metrics highlight following key trends:

- FCP and LCP have both improved since 2024, with desktop consistently outperforming mobile.
- Adoption of newer image formats remains limited, despite a gradual shift from JPG to WebP.
- About 16% of web pages still lazy load their LCP image, delaying the display of primary content.

Interactivity

Historically web performance measurement has concentrated on loading speed but there is an increasing recognition—thanks to metrics like INP—that measuring interactivity on the page after it has loaded is equally, if not more important.

Interaction to Next Paint (INP)

Interaction to Next Paint (INP)[294] is calculated by observing all the interactions made with a page during the session and reporting the worst latency (for most sites, there is an additional allowance for pages with many interactions to ignore outliers). An interaction's latency consists of the single longest duration of a group of event handlers that drive the interaction, from the

[294] *https://web.dev/articles/inp*

time the user begins the interaction to the moment the browser is next able to paint a frame.

For an origin to receive a "good" INP score, at least 75% of all sessions need an INP score of 200 milliseconds or less. The INP score is the slowest or near-slowest interaction time for all interactions on the page. See Details on how INP is calculated[295] for more information.

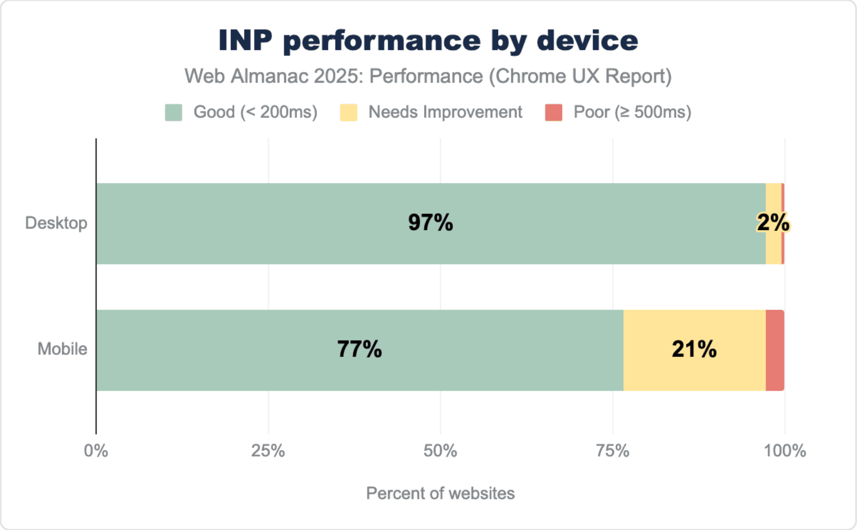


Figure 7.13. INP performance by device.

In 2025, mobile INP performance showed encouraging improvement, with 77% of websites achieving good scores—up from 74% in 2024[296]. This 3 percentage point gain represents meaningful progress, as millions of websites now deliver more responsive experiences to mobile users. Desktop performance remained exemplary at 97%, maintaining the high standard established in previous years.

Notably, the mobile-desktop performance gap has begun to narrow, shrinking from 23 percentage points in 2024 to 20 percentage points in 2025. While a 20 percentage point gap remains substantial, this marks the first measurable step toward closing the divide. The trend demonstrates that mobile optimization efforts are gaining traction across the web.

[295] https://web.dev/articles/inp#good-score
[296] https://almanac.httparchive.org/en/2024/performance#interaction-to-next-paint-inp

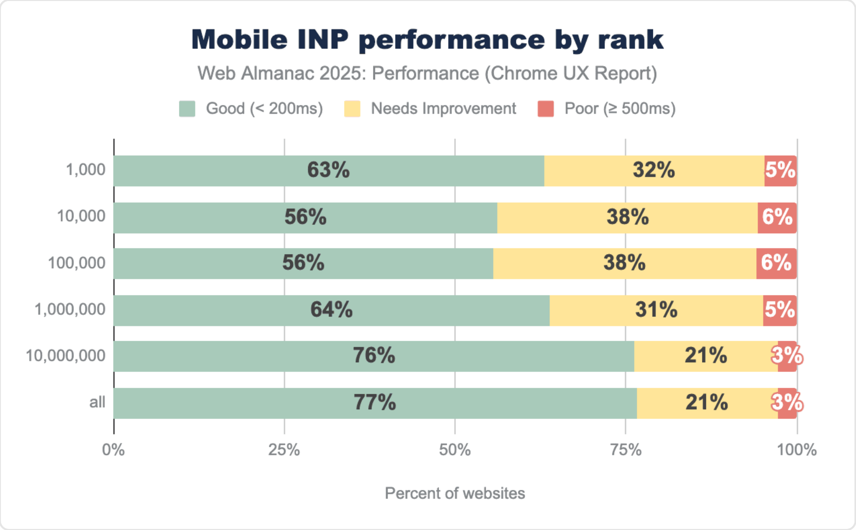


Figure 7.14. INP performance on mobile devices segmented by rank.

The most popular websites showed remarkable INP improvement in 2025, with the top 1,000 sites jumping from 53% to 63% good scores—a 10 percentage point gain that outpaced all other categories. This signals that high-traffic websites are prioritizing interactivity optimization, likely driven by the direct impact on user engagement and business metrics.

While popular sites still lag behind the overall average of 77%, the gap has narrowed significantly. The top 1,000 sites were 21 percentage points below average in 2024 but only 14 percentage points below in 2025—the fastest rate of improvement observed across any category.

This pattern reflects the unique challenges faced by high-traffic websites: more complex functionality, richer interactive features, heavier third-party integrations, and diverse user interaction patterns. E-commerce platforms, social media sites, and news portals inherently require more JavaScript execution than simpler websites, making good INP scores harder to achieve.

The substantial year-over-year improvements suggest that major websites are successfully tackling these challenges through code splitting, interaction optimization, and selective feature loading. As the most visited sites continue to enhance their performance, they set higher standards and provide valuable optimization patterns for the broader web ecosystem.

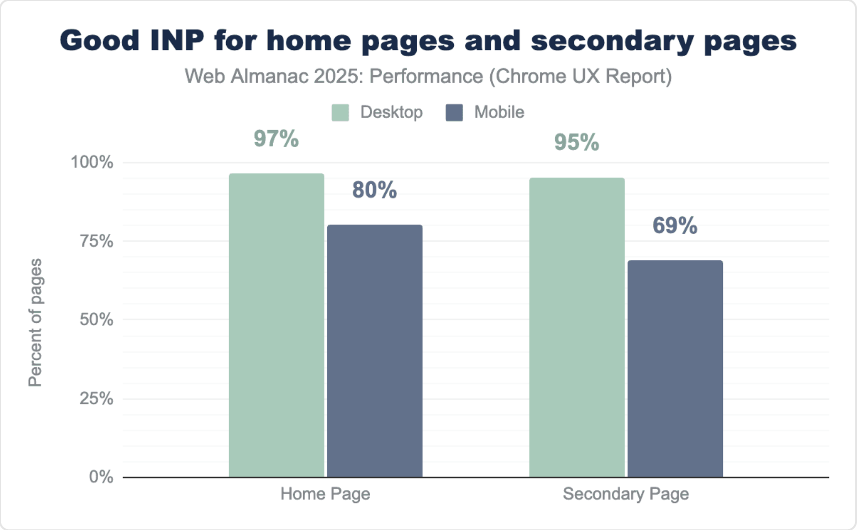


Figure 7.15. Good INP for home pages and secondary pages.

In a notable shift from 2024[297], home pages now demonstrate significantly better INP performance than secondary pages on mobile devices, when looking at page-level CrUX data. Mobile home pages achieved 80% good INP scores—a 7 percentage point improvement over 2024. Secondary pages declined to 69%, creating an 11 percentage point gap. This divergence represents a change from 2024, when home and secondary pages performed nearly identically (73% vs 72% on mobile). Desktop performance remained strong for both page types at 97% and 95% respectively.

The improvement in home page INP likely reflects increased optimization focus on landing pages, where first impressions are critical. However, the decline in secondary page performance warrants attention, as these pages often contain more complex interactions like filters, carousels, and form validation, while also accumulating JavaScript from third-party widgets and analytics that activate deeper in the user journey.

Total Blocking Time (TBT)

Total Blocking Time (TBT)[298] measures the total amount of time after First Contentful Paint (FCP) where the main thread was blocked for long enough to prevent input responsiveness.

TBT is a lab metric and is often used as a proxy for field-based responsiveness metrics like INP,

[297] https://almanac.httparchive.org/en/2024/performance#fig-21
[298] https://web.dev/articles/tbt

which can only be collected using real user monitoring. Lab-based TBT and field-based INP are correlated[299], meaning TBT results generally reflect INP trends. A TBT below 200 milliseconds is considered good, but most mobile websites exceed this target significantly.

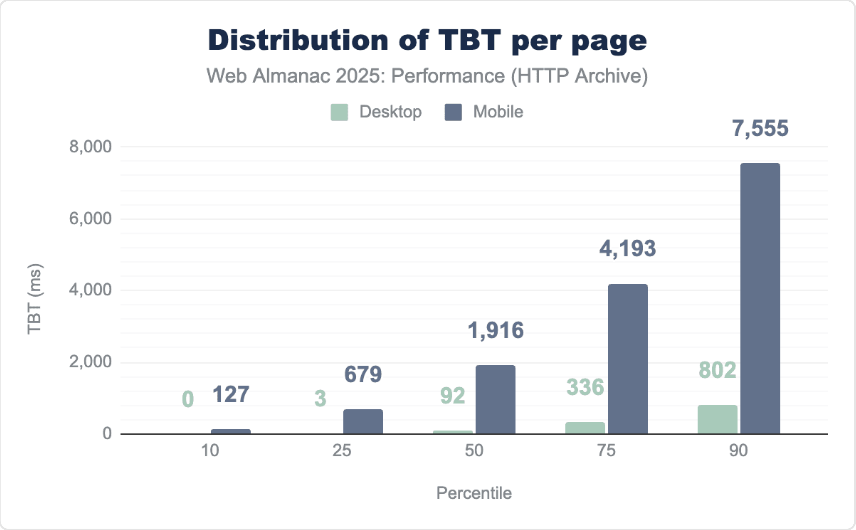


Figure 7.16. Distribution of TBT per page.

The median TBT on mobile increased to 1,916 milliseconds in 2025—up 58% from 1,209 milliseconds in 2024[300]. Desktop TBT also rose from 67 milliseconds to 92 milliseconds. At the 90th percentile, mobile users now face over 7.5 seconds of blocking time before the page becomes fully interactive.

This presents an apparent contradiction: while field-based INP scores improved, lab-based TBT worsened significantly. Several factors could be behind this divergence.

- Real-world devices have become more powerful, masking increased code complexity that lab tests reveal using consistent emulated devices.
- Some sites may be optimizing the interactions that dominate INP while still executing substantial background work that shows up in TBT.
- The INP metric continues to evolve, with upcoming improvements focused on stabilizing measurements and better capturing real-world interaction behavior, as documented in Chromium's INP metric changelog[301].

[299] *https://colab.research.google.com/drive/12lJmAABgyVjaUbmWvrbzj9BkkTxw6ay2*
[300] *https://almanac.httparchive.org/en/2024/performance#total-blocking-time-tbt*
[301] *https://chromium.googlesource.com/chromium/src/+/main/docs/speed/metrics_changelog/inp.md*

The widening gap between desktop (92ms median) and mobile (1,916ms median) reinforces the persistent performance inequality between device classes, suggesting that despite INP improvements, the fundamental challenge of main thread blocking has intensified.

Interactivity conclusion

The main takeaways of the interactivity results are:

- Mobile INP improved to 77% (up from 74%), narrowing the mobile-desktop gap to 20 percentage points.
- Top 1,000 websites achieved the strongest gains, improving from 53% to 63% good INP.
- Home pages now outperform secondary pages significantly (80% vs 69% on mobile).
- TBT increased 58% despite INP improvements, indicating heavier overall JavaScript execution.

Visual stability

Visual stability is measured by Cumulative Layout Shift (CLS) and is an indicator of how predictable and smooth pages feel to users. This section focuses on highlighting progress, device differences, and shifts over the recent years.

Cumulative Layout Shift (CLS)

Cumulative Layout Shift (CLS) measures unexpected layout movement during page load and interaction, with higher scores indicating more disruptive visual shifts. CLS scores are categorized into three thresholds: “good” (≤ 0.1), “needs improvement” (> 0.1 and ≤ 0.25), and “poor” (> 0.25), providing a standardized way to evaluate and compare visual stability across websites.

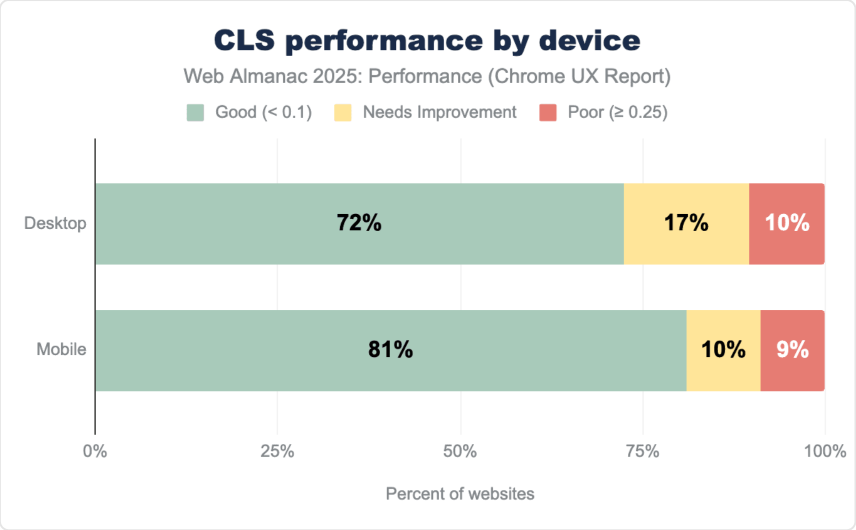


Figure 7.17. CLS performance by device

In 2025, 72% of desktop pages and 81% of mobile pages achieve a "good" Cumulative Layout Shift (CLS) score. Desktop pages show a higher share of "needs improvement" CLS (17%) compared to mobile (10%), while the proportion of pages with "poor" CLS is similar across devices at around 9-10%. This shows that most pages are close to meeting the CLS threshold, with fewer pages experiencing severe layout instability.

Compared to 2024[302], the share of desktop pages with "poor" CLS decreased by 1%, with a similar increase in pages classified as "needs improvement".

[302] *https://almanac.httparchive.org/en/2024/performance#cumulative-layout-shift-cls*

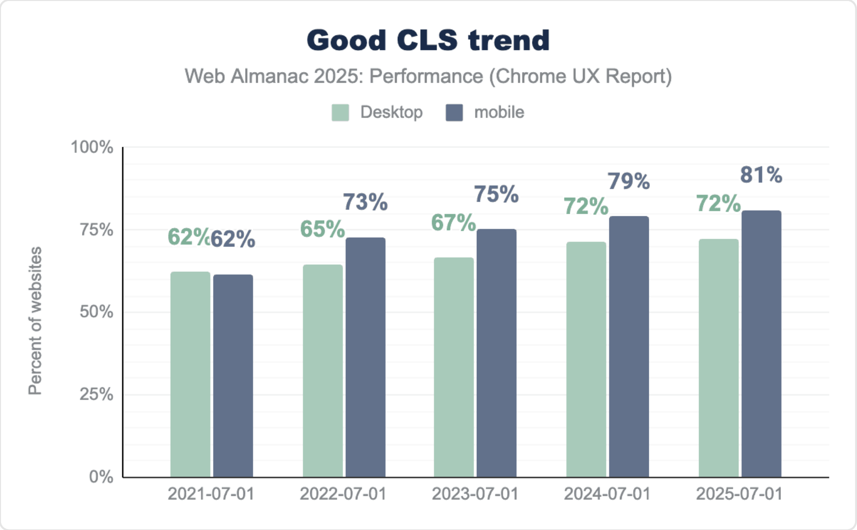


Figure 7.18. CLS performance by device from 2021 to 2025

Looking at the past years, the percentage of websites meeting the "good" CLS threshold has increased each year for both desktop and mobile. Desktop CLS improved gradually from 62% in 2021 to 72% in 2025, while mobile saw stronger gains, reaching 81% over the same period. However, the increase compared to last year is marginal, with the share of sites meeting the "good" CLS threshold on desktop remaining unchanged and mobile improving by only 2%.

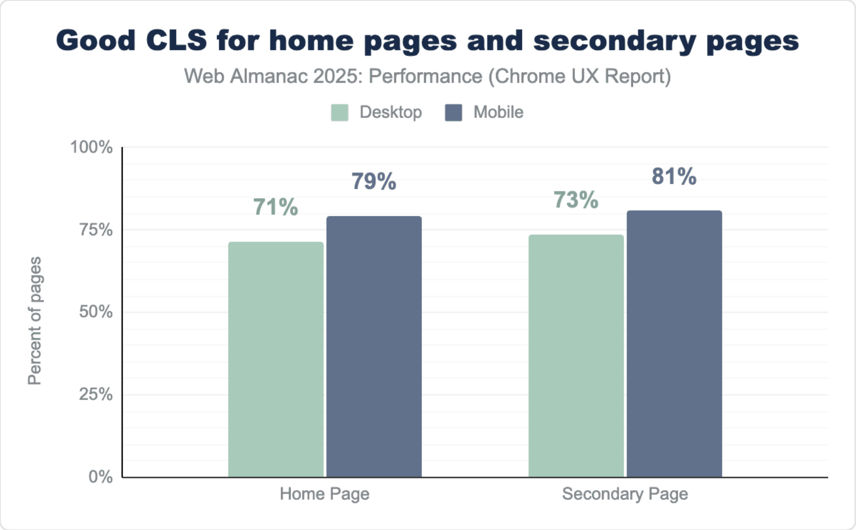


Figure 7.19. The percent of websites having good CWV, segmented by page type.

Pages beyond the homepage show slightly better visual stability than homepages across both desktop and mobile devices when looking at page-level CrUX data. In 2025, 73% of desktop secondary pages achieve “good” CLS compared to 71% of desktop homepages, while on mobile 81% of secondary pages meet the “good” CLS threshold versus 79% of mobile homepages. This suggests that homepages, which often contain more dynamic content such as hero media, banners, and promotional elements, remain more prone to layout shifts than secondary pages.

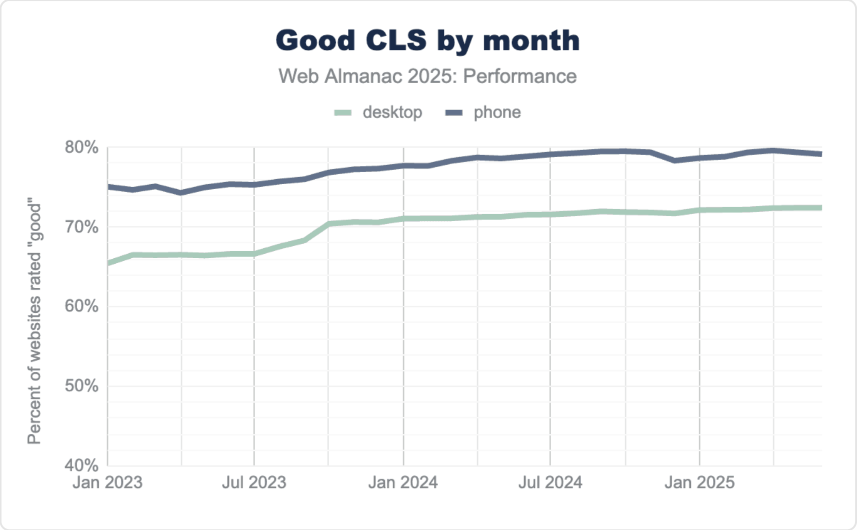


Figure 7.20. Monthly trend of websites with good CLS by device from 2023 to 2025.

From 2023 to 2025, the share of sites with "good" CLS increases steadily across both device types, with mobile consistently outperforming desktop. While there are minor fluctuations over time, both trends show a gradual upward trajectory with no sharp inflection points, indicating sustained improvements rather than sudden changes.

CLS best practices

There are a number of best practices sites can follow to reduce the likelihood of CLS.

Back/forward cache (bfcache)

The back/forward cache (bfcache)[303] allows browsers to instantly restore a page from memory when users navigate using the browser's back or forward buttons. Rather than reloading the page and re-executing JavaScript, the browser preserves the page's state, resulting in near-instant navigations and improved user experience. Because pages are restored in their previous state, bfcache can also help avoid layout shifts that might otherwise occur during re-navigation.

However, not all pages are eligible for bfcache. Eligibility depends on a set of page lifecycle requirements, and pages that violate these constraints fall back to full reloads. You can see a list of ineligible reasons in the HTML specification[304]. While bfcache behavior is ultimately handled

[303] *https://web.dev/articles/bfcache*
[304] *https://html.spec.whatwg.org/multipage/nav-history-apis.html#nrr-details-reason*

by the browser, developers can evaluate page eligibility[305] by using Chrome DevTools.

Pages may be excluded from bfcache due to known lifecycle behaviors, including the use of unload or `beforeunload` event handlers, non-restorable side effects such as active connections or unmanaged timers, and certain third-party scripts that interfere with safe page restoration. Hence, the unload event is deprecated and discouraged due to its negative impact on performance and its incompatibility with the back/forward cache.

The Chrome team recommends avoiding `unload` in favor of alternatives such as `visibilitychange` or `pagehide`, a shift that is reflected in recent usage patterns. Compared to 2024[306], unload handler usage declined across all ranks and both devices in 2025. This reduction suggests that more pages are now eligible for bfcache behavior. Despite this progress, unload handlers remain more common on higher-ranked sites and on desktop, continuing to limit bfcache eligibility for a significant portion of the web, as seen below in the graph.

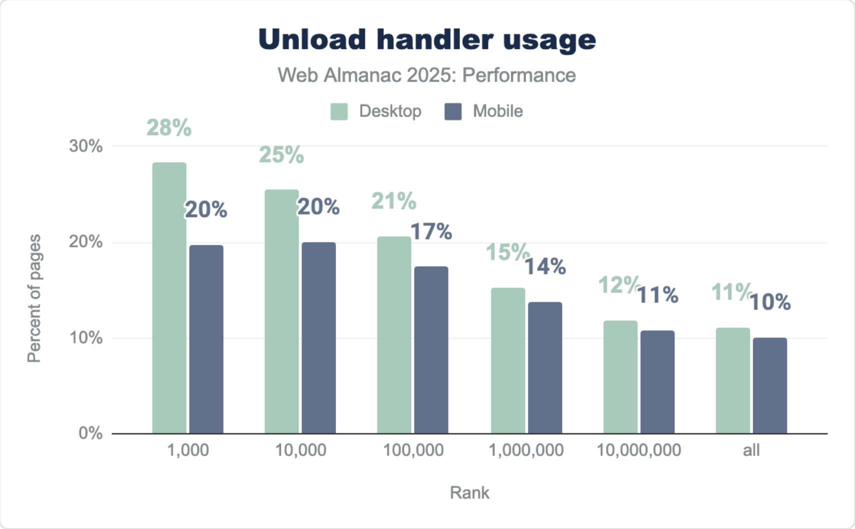


Figure 7.21. Unload handler usage by website rank and device (2025)

It is interesting to see that unload handler usage decreases consistently as the site rank increases. Among higher-traffic websites (top 1,000 sites), unload handlers are present on 28% of desktop pages and 20% of mobile pages, and this share declines steadily across lower-ranked sites, reaching 11% on desktop and 10% on mobile. At every rank, desktop pages exhibit higher unload handler usage than mobile, suggesting that unload handlers remain more common on

[305] https://developer.chrome.com/docs/devtools/application/back-forward-cache
[306] https://almanac.httparchive.org/en/2024/performance#backforward-cache-bfcache

larger, more complex sites than across the long tail of the web. Possibly due to top sites relying more heavily on analytics, advertising, and legacy lifecycle patterns that register unload handlers.

Another common reason for websites to fall in the bfcache ineligibility category is the use of the `Cache-Control: no-store` directive. This cache control header instructs the browser (and any intermediate caches) not to store a copy of the resource, ensuring that the content is fetched from the server on every request.

23%

Figure 7.22. Percentage of sites using `Cache-Control: no-store`.

23% of the sites now use `Cache-Control: no-store`, up from 21% in 2024[307]. This increase may reflect the growing prevalence of authenticated and personalized experiences, stricter security or compliance requirements, and evolving browser behavior that has reduced the performance impact of `Cache-Control: no-store`, particularly with respect to bfcache eligibility.

Note that while historically all browsers have treated `Cache-Control: no-store` as a reason to avoid bfcache, Chrome may allow bfcache[308] for some `no-store` pages when safe. Other browsers including Firefox and Safari generally still treat `Cache-Control: no-store` as a bfcache blocker.

Fixed image sizes

Images are one of the most common causes of Cumulative Layout Shift (CLS) when the browser does not know how much space to reserve for them upfront. If an image loads without explicit dimensions, the browser initially lays out the page as if the image has zero height, and then shifts surrounding content once the image finishes loading.

To prevent this, images should always have intrinsic dimensions defined either via `width` and `height` attributes or by using CSS `aspect-ratio` so the browser can allocate the correct amount of space before the image is fetched.

[307] *https://almanac.httparchive.org/en/2024/performance#backforward-cache-bfcache*
[308] *https://developer.chrome.com/docs/web-platform/bfcache-ccns*

62%

Figure 7.23. The percent of mobile pages that fail to set explicit dimensions on at least one image.

In 2025, a significant share of pages still risk layout instability due to images without explicit dimensions. On mobile, 62% of pages fail to set dimensions on at least one image, an improvement from 66% in 2024, indicating gradual adoption of CLS friendly image practices.

Desktop pages show a similar but slightly worse pattern, with 65% affected in 2025, down from 69% in 2024. While the downward trend is encouraging, the majority of pages still leave the browser guessing image sizes at layout time, making images one of the most persistent and preventable contributors to CLS.

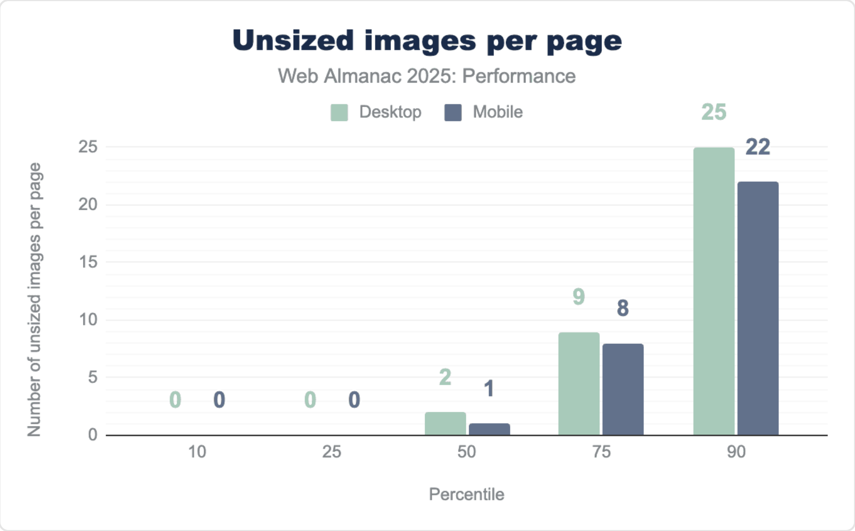


Figure 7.24. Unsized images per page.

The median number of unsized images per web page is two. At the 90th percentile, this number increases sharply to 26 on desktop and 23 on mobile. Unsized images increase the risk of layout shift. However, their actual impact on CLS depends on both the size of the image and how far content shifts when it loads, especially if the shift affects the viewport. CLS is calculated based on the impact fraction (how much of the viewport is affected) and the distance fraction (how far elements move), meaning larger images or shifts closer to the top of the page tend to contribute more heavily to CLS.

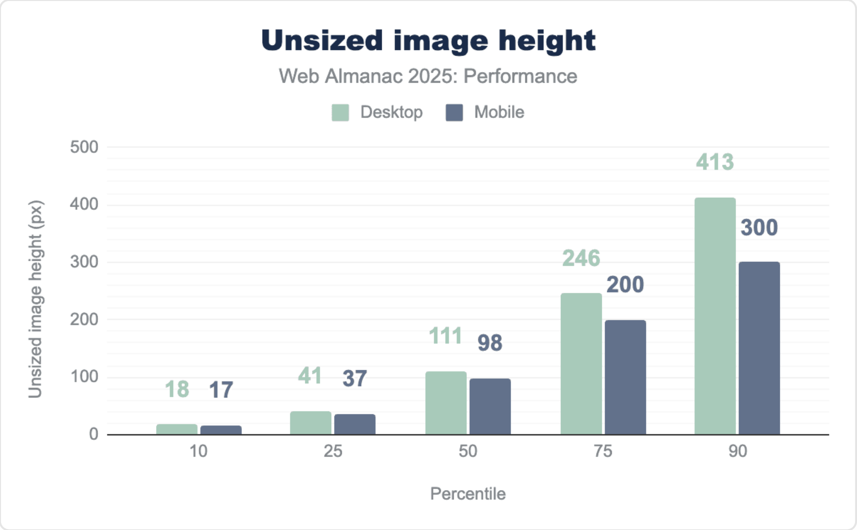


Figure 7.25. Unsized image height.

Unsized images are much taller at higher percentiles. At the median, unsized images are about 100px tall on both desktop and mobile, but by the 90th percentile they grow to around 413px on desktop and 300px on mobile. Taller unsized images increase CLS because they cause larger vertical layout shifts when they load, especially if they appear in the viewport. Since web pages scroll vertically, missing image height has a much bigger impact on CLS than missing width.

Fonts

Browsers often initially display text using a fallback (system) font while a custom web font is still downloading. This temporary rendering can cause a negative impact on the Cumulative Layout Shift (CLS) score. When the custom font finally loads, the browser replaces the fallback font, which can alter the text's size and spacing, leading to content shifting.

87%

Figure 7.26. The percent of mobile pages that use web fonts.

A significant majority 87% of mobile pages utilize at least one web font. This widespread use of custom typography requires downloading and application, substantially raising the potential[309]

[309] https://web.dev/articles/optimize-cls#web-fonts

for layout shifts.

To effectively minimize layout shifts caused by fonts, it is crucial to load essential fonts as early as possible, ideally using resource hints. If a font loads before or very near the first render, the browser can display text using the correct font immediately. This prevents the swap from a default font, which is a common cause of layout shifts. Current data indicates that this opportunity is frequently missed.

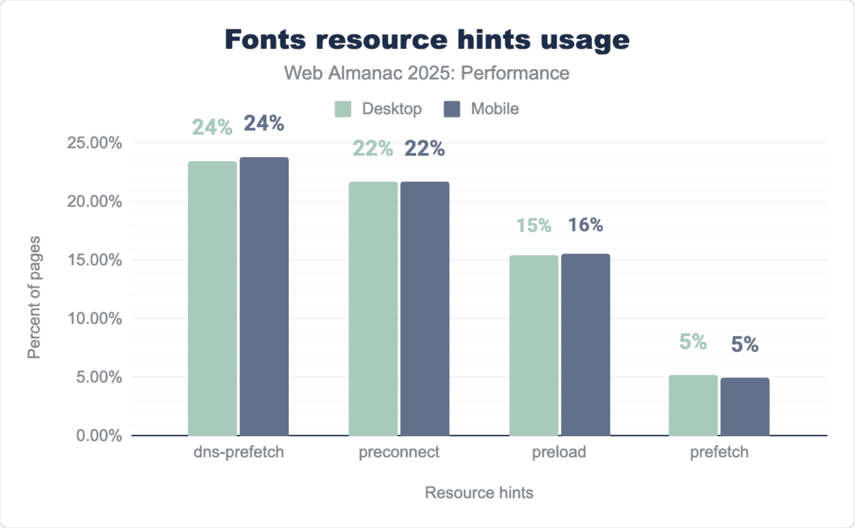


Figure 7.27. Fonts resource hints usage.

Font resource hint usage is very similar across desktop and mobile. About 24% of pages use dns-prefetch and 22% use preconnect, which mainly helps in reducing connection setup time. Preload is a good way to make fonts available early during rendering but is used on only 15-16% of pages. Even fewer pages, around 5%, use prefetch, which is generally less useful for fonts needed during the initial page load. This suggests there is substantial opportunity to improve font performance through more targeted use of resource hints.

Non-composited animations

Non-composited animations[310] contribute to layout shift because they modify layout-affecting properties, triggering reflows that move surrounding content after rendering has begun. Animations using composited properties like `transform` and `opacity` avoid this by updating visual appearance without changing layout, making them safer for visual stability.

[310] *https://developer.chrome.com/docs/lighthouse/performance/non-composited-animations*

40%

Figure 7.28. The percent of mobile pages that have non-composited animations.

Non-composited animations remain common, appearing on 40% of mobile pages and 44% of desktop pages.

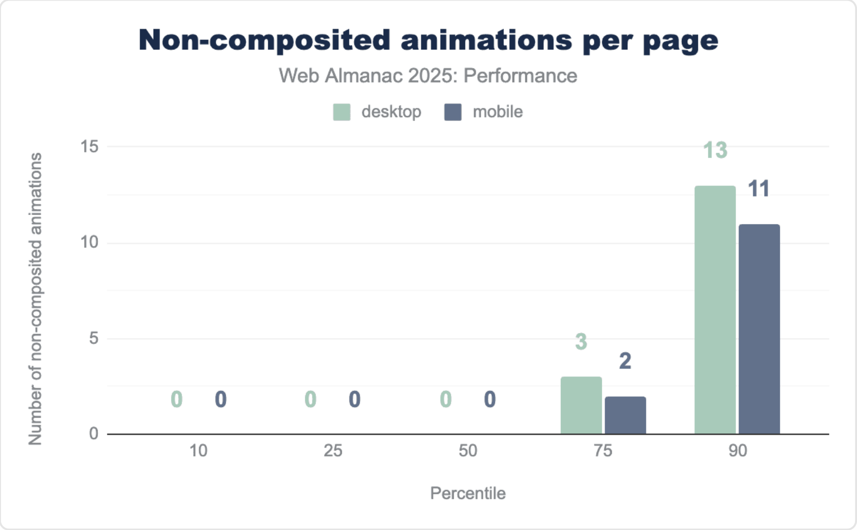


Figure 7.29. Non-composited animations per page.

Non-composited animations impact emerges primarily at higher percentiles, with usage increasing at the 75th percentile and rising sharply at the 90th percentile to 13 animations on desktop and 11 on mobile.

Visual stability conclusion

Visual stability across the web has advanced significantly over the years, particularly on mobile devices. Most pages now deliver stable experiences with minimal unexpected movement, reflecting improved adoption of best practices. However, with around 20-30% of pages still not achieving “good” CLS, especially on desktop, there remains room for continued refinement and optimization.

Despite gradual improvements, unsized images remain common and font-loading patterns still create opportunities for layout shifts, suggesting many sites haven’t fully implemented known

CLS mitigations. Adopting simple best practices[311] like explicit image sizing, preloading critical fonts, and using composited animations, pages can help improve visual stability.

Early Hints

Early Hints[312] provide an early signal to the browser about the resources it will need to load for the requested page.

Early Hints are sent from the server to the browser while the requested page is still being prepared. In this way, browsers can start taking action to optimistically preconnect to other domains or preload an asset before the requested page has been returned to it.

This allows Early Hints to have an absolute impact on the loading performance of the currently requested pages. Consider if, rather than having to wait for the HTML to return to the browser, and the parser to find the link (or even preload link) for the main CSS file and/or LCP asset, it could begin fetching those assets before the HTML was even returned to the browser. This could allow for an FCP that is nearly perfectly rendered, in a single paint.

Since Early Hints can also contain `crossorigin` attribute and CSP header information, it is recommended that they only be used over HTTP/2 or higher for for security reasons[313].

[311] *https://web.dev/articles/optimize-cls*
[312] *https://developer.mozilla.org/docs/Web/HTTP/Reference/Status/103*
[313] *https://www.rfc-editor.org/rfc/rfc8297#section-3*

Early Hints usage

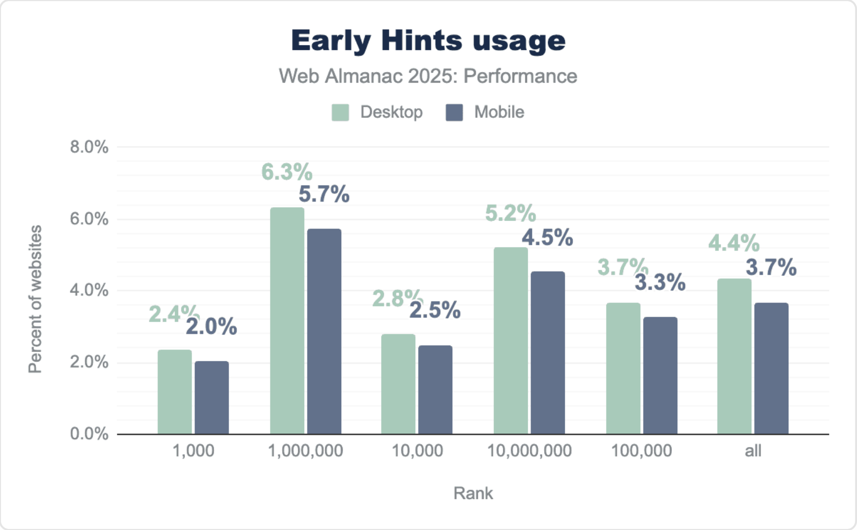


Figure 7.30. Early Hints usage.

We see adoption has not exactly taken off: usage is quite low in all groups, barely breaking 6% on desktop in the top 1,000,000 sites; most other groups are well below 5%.

This is likely related to the complexities of setting up and configuring Early Hints: the assets for any given page must be related to the server before the page is complete and ready for sending; for most CMSs this would be a challenge.

The mobile/desktop parity is also quite noticeable; never more than a 1% difference, and typically closer to 0.5%. Meaning, where Early Hints are implemented, they are likely done so similarly for all device types.

While usage remains low in 2025, there has been a noticeable increase over the past three years.

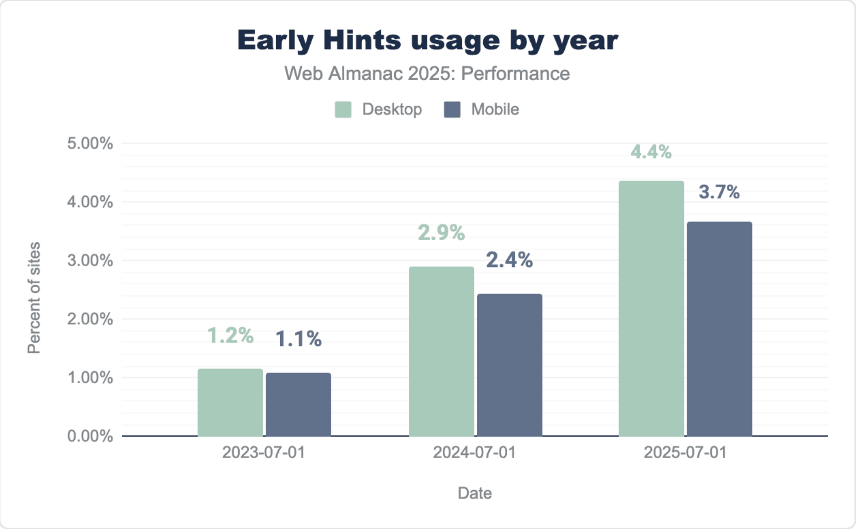


Figure 7.31. Early Hints usage by year.

Early Hints support

Unlike most web performance features, Early Hints relies not only on browsers, but also on servers for support. As of this publication, `preconnect` is supported in all browsers, and `preload` in all except Safari.

With regards to servers, Early Hints are fully supported for Node, H2O and NGINX, and for Apache if you are using `mod_http2`.

Early Hints are available via CDNs including Fastly since 2020[314], Cloudflare since 2021[315], and Akamai since 2023[316].

Speculation Rules

Speculation Rules[317] are an experimental browser API for optimistically prefetching or prerendering complete pages, with the hope that the user will navigate to one of the pages after viewing the current page. These actions happen in the background of the page the user is currently viewing. It currently implemented primarily in Chromium-based browsers, though

[314] *https://www.fastly.com/blog/beyond-server-push-experimenting-with-the-103-early-hints-status-code*
[315] *https://blog.cloudflare.com/early-hints/*
[316] *https://www.akamai.com/blog/performance/akamai-103-early-hints-prototype-the-results-are-in*
[317] *https://developer.mozilla.org/docs/Web/API/Speculation_Rules_API*

Safari has a partial implementation for prefetch behind a flag.

While Speculation Rules do not help the current page's performance, they can greatly improve the loading performance of subsequent pages, often to the point of almost an instantaneous page load.

The intent is for this API to replace `<link rel="prefetch">` for pages and `<link rel="prerender">` with more advanced configuration options. Again, the Speculation Rules API is for full pages only; for individual assets, you would still need to use `<link rel="prefetch">`.

Speculation Rules usage

In the chart below, which shows the percentage of home pages that contain Speculation Rules, we see something interesting: Speculation Rules usage on the top 1,000 sites is quite low, only 3% on desktop and 5% on mobile. And while usage climbs for each subsequent group, it only reaches 15%, mobile and desktop, for the top 1,000,000 sites. It is not until the final group, the top 10,000,000, that we see the percentage jump sharply up, to 24% desktop and 25% mobile:

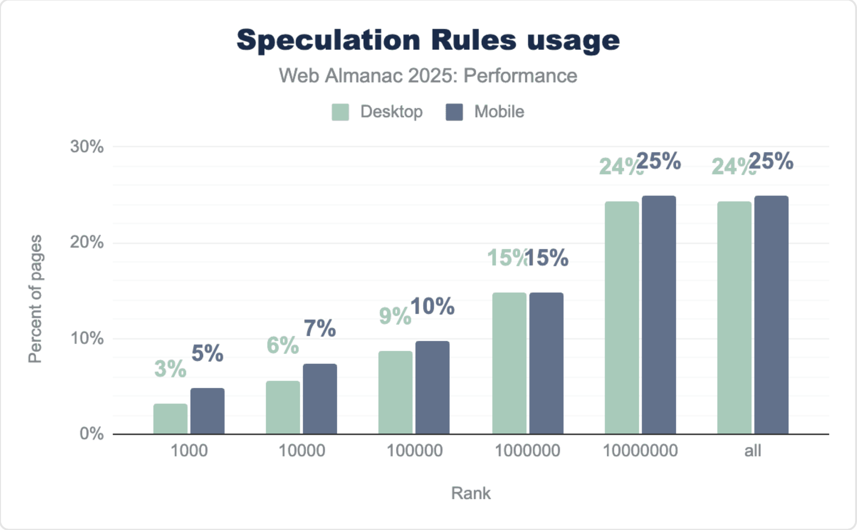


Figure 7.32. Speculation Rules usage.

This could be related to the complexities of configuring Speculation Rules: a site should be careful when prefetching or prerendering pages, since the user's exact intent can never be known, and anything that is fetched and not used is wasteful. So, for a larger site, such as an ecommerce site, and especially a large site with numerous categories and perhaps menu options to jump directly to, Speculation Rules could be difficult to configure properly. They could also be tricky to implement into a legacy or bespoke CMS.

Conversely, Speculation Rules now come baked into WordPress[318], which powers a large share of the Internet, which may help explain higher adoption in the long tail.

Also notable is the parity between mobile and desktop usage; seldom more than a 1% difference. Meaning, where Speculation Rules are implemented, they are likely done so similarly for all device types.

Conclusion

Our analysis of this year's data paints a picture of a web that is becoming more responsive, yet remains complex to optimize. We see clear progress in how the web feels to use: mobile interactivity has improved significantly, with the performance gap between phones and desktop computers finally starting to narrow. This tells us that perhaps the industry's focus on new metrics like Interaction to Next Paint (INP) is working, and developers are trying to prioritize the interactions that matter most to users.

However, we also observe a "performance divide" in how different segments of the web adopt new standards. For example, we saw that the most popular sites lead the way in improving interactivity (INP), likely through manual optimization of complex JavaScript. In contrast, newer standards like Speculation Rules are seeing their highest adoption not at the top, but in the "long tail" of the web, driven by platform-level integrations in popular CMSs like WordPress. This suggests that the future of performance may rely less on individual manual effort and more on smart defaults baked into the tools that build the web.

Despite these advancements, the basics of web performance still pose a challenge. While advanced metrics improve, fundamental issues still persist: nearly 40% of mobile pages still use non-composite animations that risk visual instability, and the majority of pages still lack the correct sizing for images or the resource hints needed to load critical fonts smoothly. This suggests that while frameworks are helping us manage complex JavaScript, we often miss the simpler best practices that ensure good web performance.

Finally, the landscape of measurement itself is maturing. As more browsers extend support for modern metrics like INP, cross-browser comparisons can become more consistent. As we look ahead, the goal for developers is to look past the top-level scores and bridge the gap between potential and practice, leveraging both the manual optimizations used by top sites and the automated tools of the modern web to deliver reliable experiences for every user.

[318] *https://make.wordpress.org/core/2025/03/06/speculative-loading-in-6-8/*

Authors

Himanshu Jariyal

himanshujariyal himanshujariyal https://medium.com/@him_jar

Himanshu Jariyal is a Senior Software Engineer at Microsoft on the Bing Performance team. He specializes in real-user performance measurement and analysis, and in optimizing large, production-critical systems.

Prathamesh Rasam

25prathamesh prathamesh-rasam

Prathamesh Rasam is a web performance architect with over a decade of experience working on large-scale web and mobile systems. He is a public speaker on web performance and builds real-time web and app performance monitoring platforms at scale.

Humaira

hfhashmi

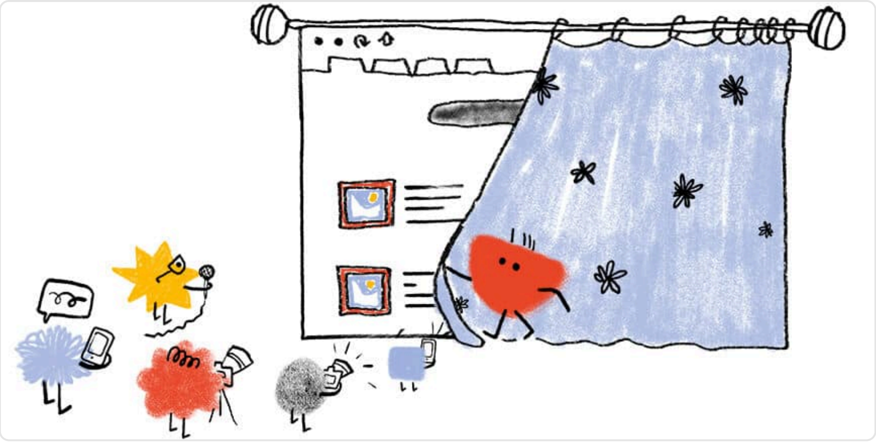
Humaira is a PhD Student in Computer Science at UC Davis. Her research focuses on the intersection of network measurements, policy and privacy.

Aaron T. Grogg

@aarontgrogg.com aarontgrogg aarontgrogg https://aarontgrogg.com/

Aaron T. Grogg started his web development career in 1998. Along the way, he has become a staunch advocate for standards, the semantic web, accessibility and the vital importance of building performant, user-focused digital experiences. Aaron remains devoted to discovering and sharing new web technologies and best practices, and helping to move the web forward for everyone.

Part II Chapter 8
Privacy



Written by Rumaisa Habib, Vinod Tiwari, and Nurullah Demir
Reviewed by Jannis Rautenstrauch
Analyzed and edited by Max Ostapenko

Introduction

The web is the primary interface for digital services, making it a significant source of data as billions of users interact with these systems daily. Consequently, website tracking – the practice of collecting data about visitors – has become a fundamental component of the modern web ecosystem. The motivations for this data collection vary widely, ranging from improving application performance and functionality to enabling targeted advertising and marketing analytics.

However, the scale of this data collection raises significant privacy concerns, making it a widely discussed topic in technical[319] and political spheres[320], and a major area of ongoing research[321]. While developers utilize various technologies to track users, such as HTTP cookies and browser fingerprinting, there is a corresponding rise in privacy measures. These include browser-based restrictions, regulatory compliance tools, and privacy-enhancing extensions.

[319] *https://www.w3.org/TR/tracking-compliance/*
[320] *https://eur-lex.europa.eu/eli/reg/2016/679/oj/eng*
[321] *https://pulse-of-cybersecurity.com/topics?sortBy=total-papers&sortOrder=desc&page=1&pageSize=21&search=web&topic=Web+Tracking+and+Browser+Fingerprinting&conferences=%5B%5D*

In this chapter, we provide a technical overview of the state of web privacy. We analyze the adoption of common tracking mechanisms and examine the prevalence of measures designed to prevent tracking, offering a data-driven look at the current landscape of user data collection.

Online tracking

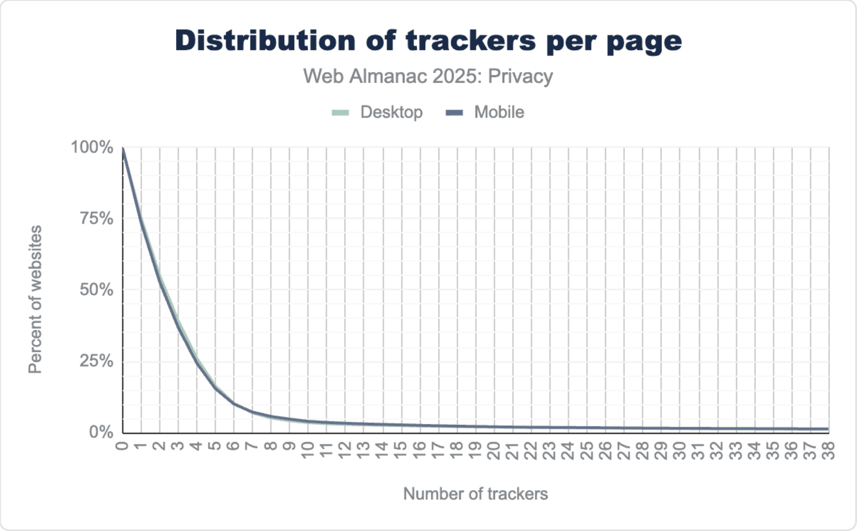


Figure 8.1. Distribution of trackers per page

Our analysis uses the WhoTracks.Me[322] catalog of popular third-party trackers to identify the trackers present on the webpages. To be conservative in our analysis, we only count the WhoTracksMe categories `advertising`, `pornvertising`, `site_analytics` and `social_media` as trackers. This method allows us to determine the distinct third-party trackers at the domain level for each webpage. It is worth noting that the reported numbers represent unique domains, not the total number of HTTP requests.

75%

Figure 8.2. Websites with at least one tracker.

We see at least one third-party tracker in 75% of all webpages (75%: desktop, 74%: mobile),

[322] https://www.ghostery.com/whotracksme/

55% of desktop webpages contain 2 and 39% contain 3 trackers. Up to 6 trackers setup happens more often in desktop pages, while 7 and more trackers are seen more often in mobile pages.

Stateful tracking

Tracking mechanisms are categorized as stateful or stateless. Stateful methods, such as cookies and local storage, store identifying information on the user device. In contrast, stateless methods, like fingerprinting, infer this information at runtime from unique characteristics.

Third-party tracking services

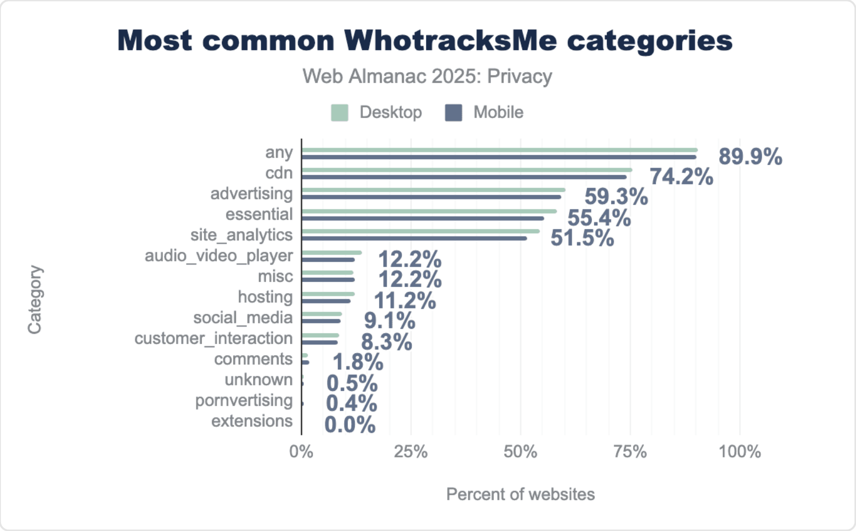


Figure 8.3. Most common WhotracksMe categories

Here, we consider all categories of WhoTracksMe. We observe that most webpages connect to domains categorized as Content Delivery Networks (CDNs) and advertising. At least one CDN-related domain is present on 74% of webpages, followed by advertising-related domains on 59%. Additionally, 55% of webpages include essential domains (such as Google Tag Manager) and 52% contain analytics domains (such as Google Analytics). This high concentration among a few key players effectively sets a baseline for web privacy, where the vast majority of user data flows through a small number of dominant platforms.

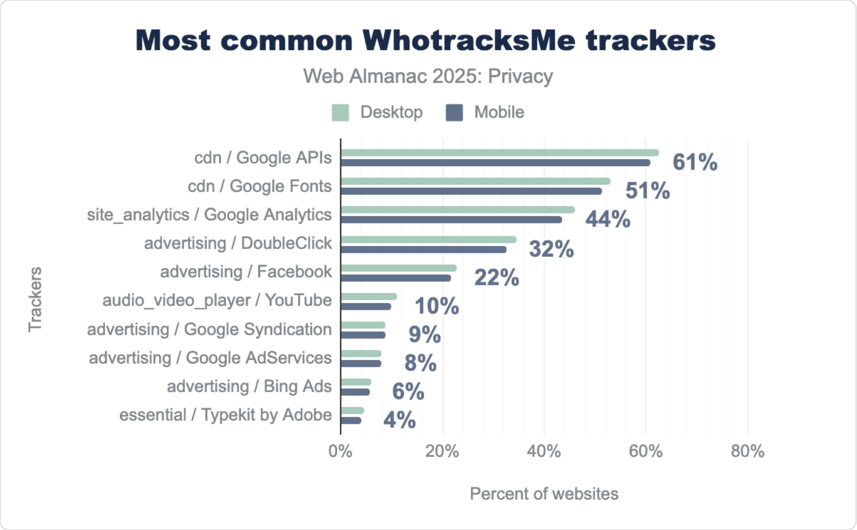


Figure 8.4. Most common WhotracksMe trackers

Our analysis of single trackers shows that Google and Facebook are the majority of tracking services. These entities are the most prominent trackers on the Web, with Google present on 61% of webpages and Facebook on at least 22% of webpages. They are followed by Bing (6%) and Adobe (4%).

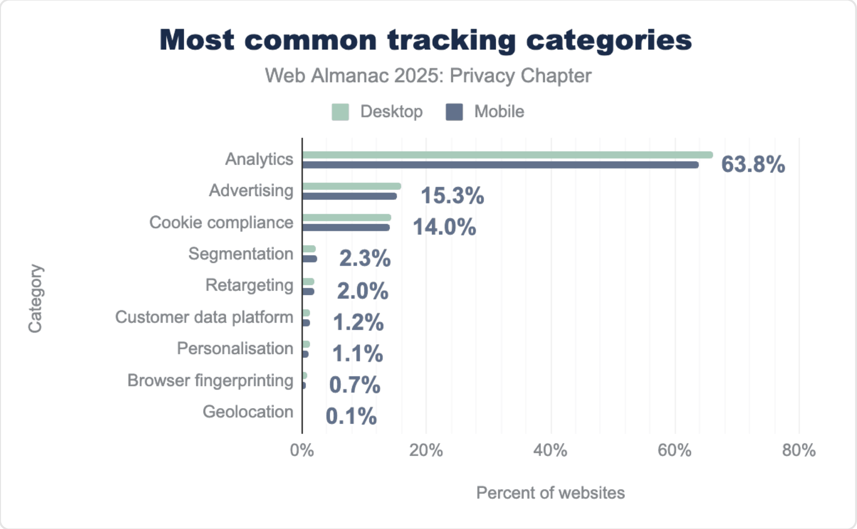


Figure 8.5. Most common tracking categories

Furthermore, when categorizing these services by function, Analytics dominates the landscape, appearing on 64% of desktop webpages. Advertising and Cookie compliance tools follow at 15% and 14% respectively, illustrating that performance monitoring remains the primary driver of data collection. More specialized tracking methods, such as Segmentation and Retargeting, are significantly less common, each found on fewer than 3% of sites.

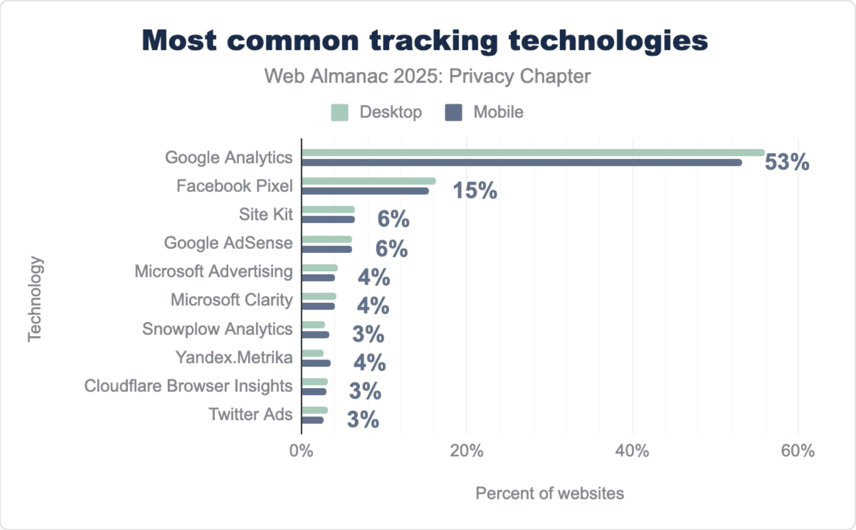


Figure 8.6. Most common tracking technologies

Tracking can happen in different contexts – from understanding user behavior on webpages to building complex advertising profiles. We find that Google Analytics (53%) and Facebook Pixel (16%) are the most popular technologies used to track web users. Beyond these market leaders, adoption drops significantly, with Google's Site Kit (6.41%) and AdSense (6.18%) representing the next tier of usage. Other players like Microsoft also maintain a consistent but smaller footprint, with their Advertising and Clarity tools each present on approximately 4% of websites.

Third-party cookies

Using third-party cookies is an efficient method for tracking and targeting web users. Third parties utilize cookies for user tracking. Despite consistent criticism, this remains a common technique on the web. Although some vendors, like Google, have announced plans to phase out third-party cookies[323] (and later reconsidered[324]), they remain a significant technique for tracking

[323] https://support.google.com/google-ads/answer/14762010
[324] https://privacysandbox.google.com/blog/privacy-sandbox-update

and the majority of the third-party cookies used for tracking purposes.

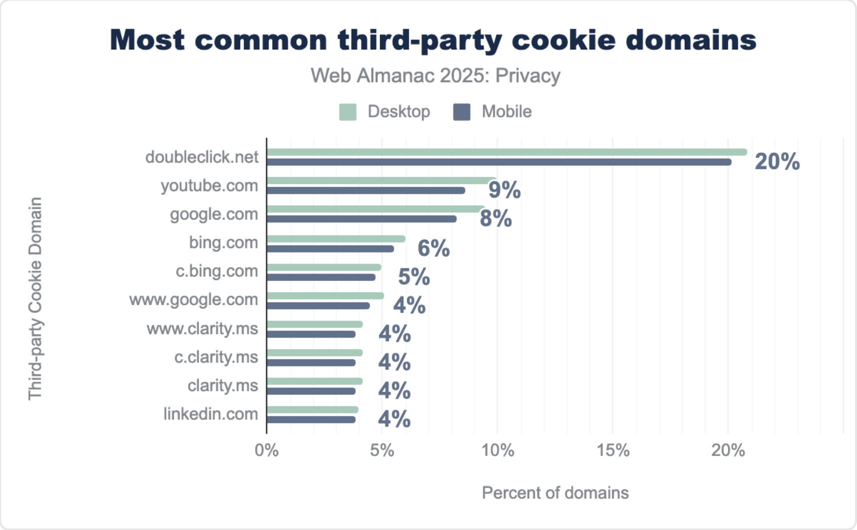


Figure 8.7. Most common third-party cookie domains

Our analysis shows that `doubleclick.net` is the most common third-party cookie domain, appearing on 20% of desktop sites, followed by `youtube.com` (9%) and `google.com` (8%). Overall, while Google entities dominate the top rankings, Microsoft's `bing.com` and `clarity.ms`, along with `linkedin.com`, represent the most significant alternative third-party cookie setters.

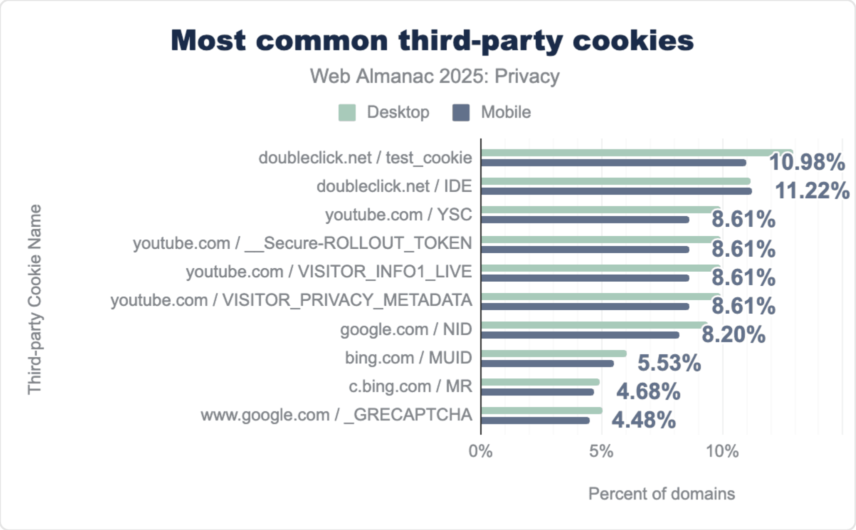


Figure 8.8. Most common third-party cookies

First-party cookies

The following figure shows the most common first-party cookies. While these cookies are set in a first-party context, their names provide evidence that they are primarily used for tracking purposes. The `_ga` cookie is set on 46% of webpages, and `_gid` appears on 18%, both used by Google Analytics, followed by `gcl_au` on 16% of webpages. While the exact purpose of these cookies was not tested, Google publishes[325] their intended functions. Another popular first-party cookie is `_fbp`, used by Meta on 14% of webpages. Meta provides[326] advertisers the option to use first-party cookies with the Meta Pixel. Similar to the results observed for third-party context, Google and Meta remain the dominant entities for tracking in the first-party cookie context.

The usage of cookies on the web remains largely for tracking purposes. Among the functional exceptions, `PHPSESSID` stores a unique session ID for PHP applications on 12% of pages, while `XSRF-TOKEN` handles security against cross-site request forgery and is found on 6% of webpages.

[325] *https://business.safety.google/adscookies/*
[326] *https://www.facebook.com/business/help/471978536642445?id=1205376682832142*

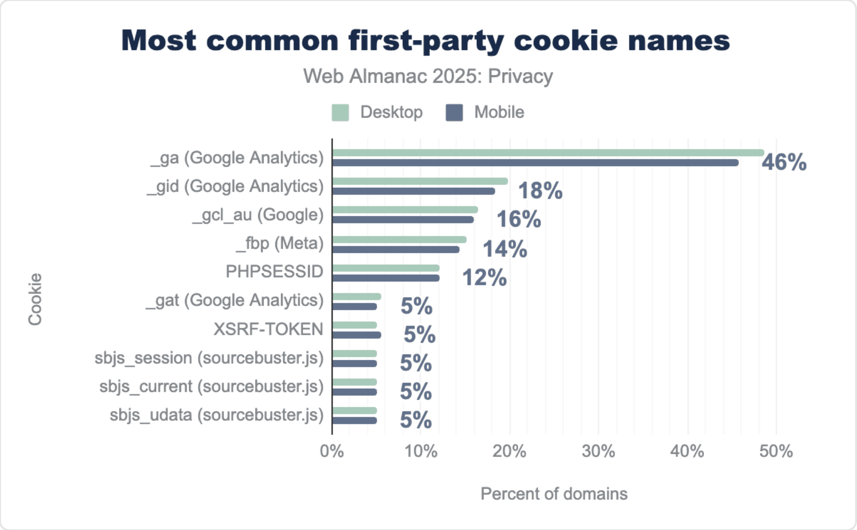


Figure 8.9. Most common first-party cookie names

The Cookies chapter further describes the details and usage trends of cookies extensively.

Stateless tracking

Stateless tracking is the process by which user identifiers are generated on the fly, rather than stored in the browser as state. These identifiers are generally created by using information that can be actively or passively gathered from the target user's device or browser. While it is tricky to correlate the sessions of a user who uses multiple devices, it is effective in that some signals are inherent to the device or website functionality and cannot be easily 'blocked'.

Browser fingerprinting

Browser fingerprinting is a method by which websites can identify a user based on their specific browser information. This information can include[327] system fonts, language settings, hardware configurations, and other such seemingly innocuous datapoints that individually reveal little information, but can be put together to paint a unique picture[328] of a specific user. They are commonly leaked through HTTP headers and JavaScript API calls.

Prior work[329] has shown browser fingerprinting to be highly prevalent in online tracking. Its

[327] https://dl.acm.org/doi/abs/10.1145/3543507.3583333
[328] https://amiunique.org/
[329] https://dl.acm.org/doi/abs/10.1145/3696410.3714548

attractiveness can be attributed to the fact that it is difficult to block, and claims to be effective even if the user is using an Incognito browser. In this report, we identify the most common technologies used to do browser fingerprinting.

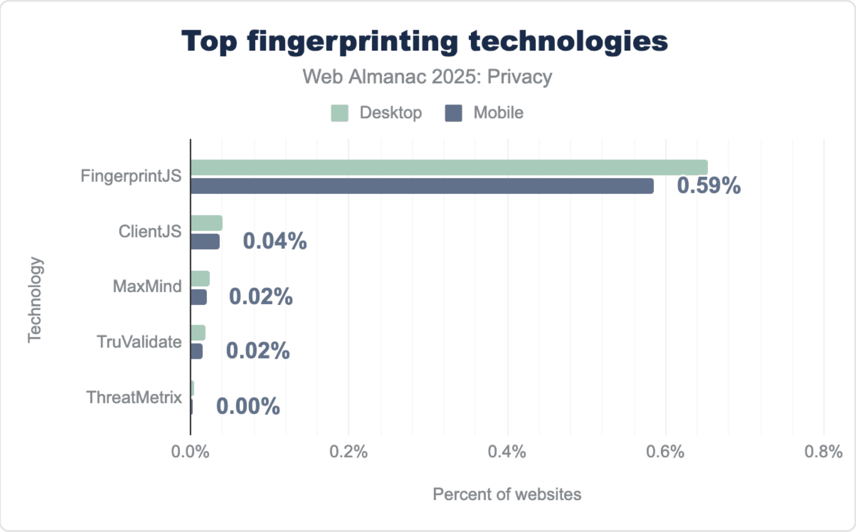


Figure 8.10. Top fingerprinting technologies

Of note, the library FingerprintJS[330] has remained the most popular tool to conduct browser fingerprinting, far surpassing the others. FingerprintJS is used on 0.59% of mobile accessed websites, compared to ClientJS[331] (the next most popular technology) which is present on 0.04%.

The popularity of FingerprintJS can likely be attributed to its thriving open source community, which appears to be more active than that of ClientJS.

Evading tracking protections

As browsers and privacy tools have become more effective at blocking third-party trackers, the tracking industry has adapted. Techniques like CNAME cloaking and bounce tracking allow trackers to disguise themselves as first-party resources or use intermediate redirects to circumvent traditional blocking methods. These approaches exploit the trust browsers place in first-party requests, making them harder to detect and block. In this section, we focus on bounce tracking, which can be observed through redirect chains in our crawl data.

[330] *https://github.com/fingerprintjs/fingerprintjs*
[331] *https://github.com/jackspirou/clientjs*

Bounce tracking

Bounce tracking is a technique where users are briefly redirected through an intermediate domain before reaching their destination. During this redirect, often imperceptible to the user, the intermediate site can set or read cookies, effectively tracking users across sites while appearing as a first party interaction. This sidesteps traditional third-party cookie blocking.

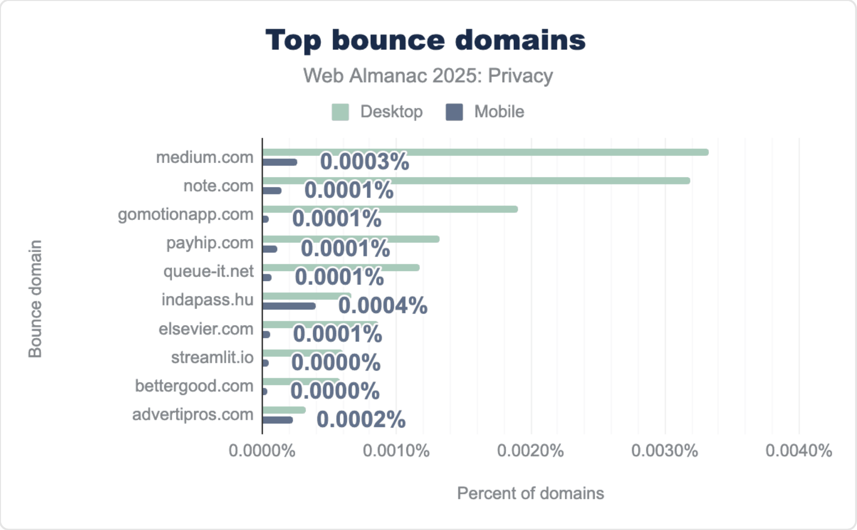


Figure 8.11. Top bounce domains

`medium.com` remains the most common bounce domain in the 2025 dataset at 0.0003%, followed by `note.com` and `indapass.hu`. Year-over-year, prevalence dropped significantly; for instance, `medium.com` fell from 0.009% to 0.0003%, and `indapass.hu` from 0.012% to 0.0004%. This decline likely reflects Chrome's bounce tracking mitigations taking effect. Because the top domains (`queue-it.net`, `payhip.com`, `medium.com`) are legitimate functional services, the data suggests most observed behavior stems from necessary redirects rather than covert tracking.

Browser policies to improve privacy

Browsers have introduced various mechanisms that can influence how much information websites share with third parties. These features operate at the protocol level, controlling headers, limiting data exposure, and standardizing how sites communicate with external resources. Their actual privacy impact depends on implementation, adoption, and whether sites choose to use them.

In this section, we examine three such mechanisms: User-Agent Client Hints, which offer a more controlled alternative to the traditional User-Agent string; Referrer policy, which lets sites limit how much navigation context is passed to third parties; and privacy-related Origin Trials, where browsers experiment with new features before wider rollout.

User-Agent Client Hints

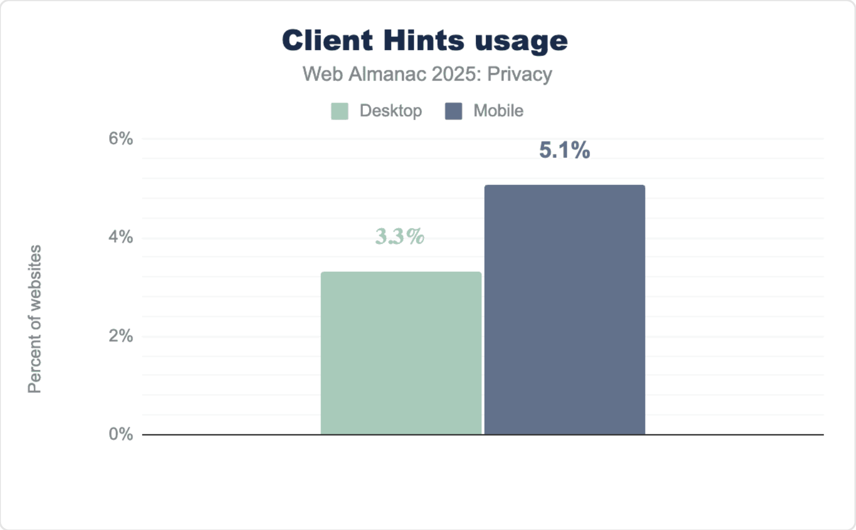


Figure 8.12. Client Hints usage

User-Agent Client Hints offer a privacy-conscious alternative to the traditional User-Agent string, allowing browsers to share device and browser information only when explicitly requested by servers. Instead of exposing a detailed fingerprint by default, sites must opt in to specific hints, reducing passive data leakage. In 2025, adoption sits at 3.3% for desktop and 5.1% for mobile, with mobile's higher rate likely reflecting greater need for responsive design signals. This transition is now permanent; with Chrome 145 removing the `UserAgentReduction` policy, sites must migrate to User-Agent Client Hints[332] for detailed browser or device information.

Last year's data showed a strong correlation between site popularity and Client Hints usage; the top 1,000 sites reached 15.85%, while adoption dropped sharply to around 1.6% at the 100,000 tier. While this year's methodology doesn't break down by rank, the overall figures

[332] *https://web.dev/articles/migrate-to-ua-ch*

suggest adoption remains concentrated among larger sites, with the long tail yet to embrace the standard.

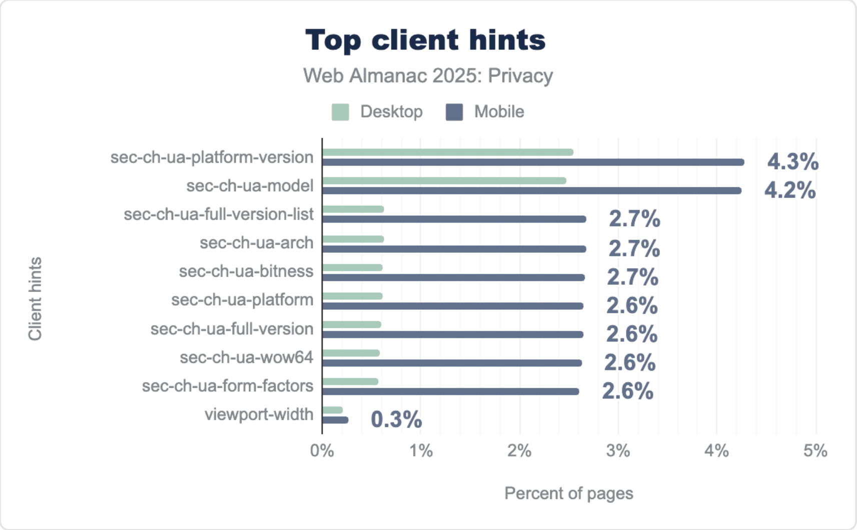


Figure 8.13. Top Client Hints

The most requested Client Hint is sec-ch-ua-platform-version at 4.28%, used to detect OS version for compatibility decisions. Close behind is sec-ch-ua-model at 4.25%, though with a notable skew: mobile usage far exceeds desktop, which makes sense given that device model is primarily relevant for mobile experiences and debugging. The remaining hints, covering architecture, bitness, full version lists, and form factors, cluster tightly between 2.60% and 2.67%, suggesting that sites requesting Client Hints tend to request several together rather than cherry-picking individual signals.

Referrer Policy

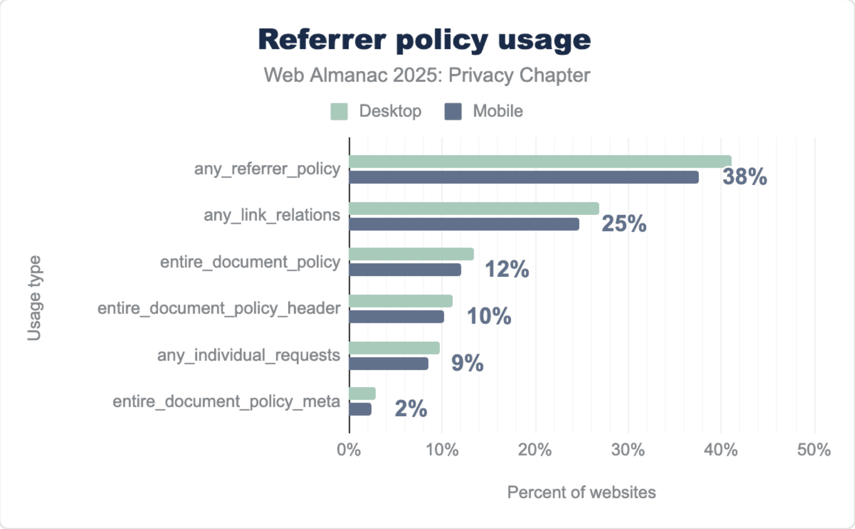


Figure 8.14. Referrer policy usage

When you click a link from one website to another, your browser can reveal where you came from, including the full page URL. Referrer Policy gives website owners control over how much of this information is shared, helping protect user privacy by limiting what third parties can see about your browsing path.

Overall adoption of Referrer Policy rose from 32% in 2024 to 37.66% in 2025, a healthy increase. The most common implementation method remains link-level controls (like `rel="noreferrer"` on individual links) at 24.70%, while document-wide policies set via headers sit at 10.16%. This suggests many sites apply referrer restrictions selectively rather than as a blanket rule.

Meta tag implementations remain the least common at 2.47%, largely unchanged from 2024's 2%. This is expected; headers are generally preferred for security policies since they're harder to tamper with and apply before the page loads.

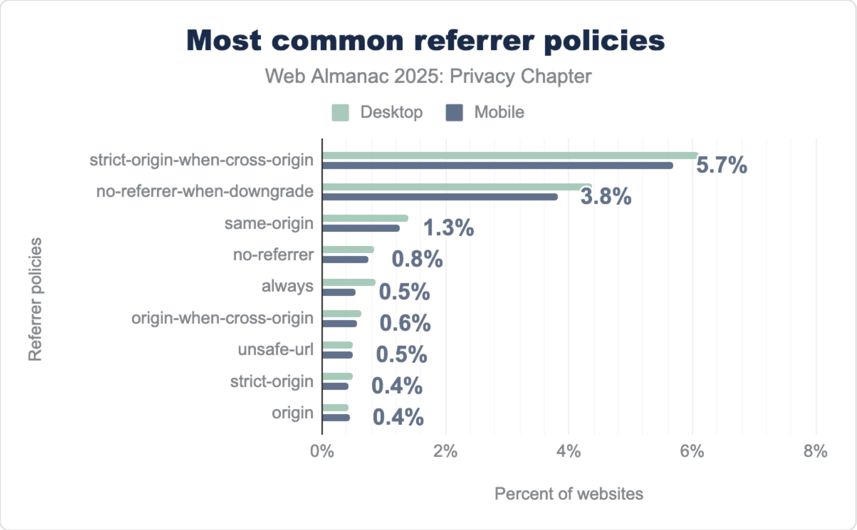


Figure 8.15. Most common referrer policies

The most privacy-conscious policies saw a decline this year, strict-origin-when-cross-origin, which shares the origin but strips the full path when navigating to other sites, dropped from 7.5% to 5.69%. Similarly, no-referrer-when-downgrade fell from 7.0% to 3.81%. These remain the top two policies, but the decrease suggests some sites may have relaxed their settings or shifted implementations.

On the positive side, truly restrictive options like same-origin (1.26%) and no-referrer (0.75%) remain in use, though adoption is low. These policies share nothing with third-party sites, ideal for privacy, but sometimes limiting for analytics and affiliate tracking that sites rely on.

Some sites still specify unsafe-url (0.50%), which exposes the full URL to any destination, though this behavior is Chrome-specific and other browsers have deprecated it. We also see always (0.54%), an invalid value that browsers ignore and fall back to the default strict-origin-when-cross-origin. The presence of these values suggests some sites have misconfigured or outdated referrer policies rather than intentionally choosing privacy-unfriendly settings.

Privacy-related origins trials

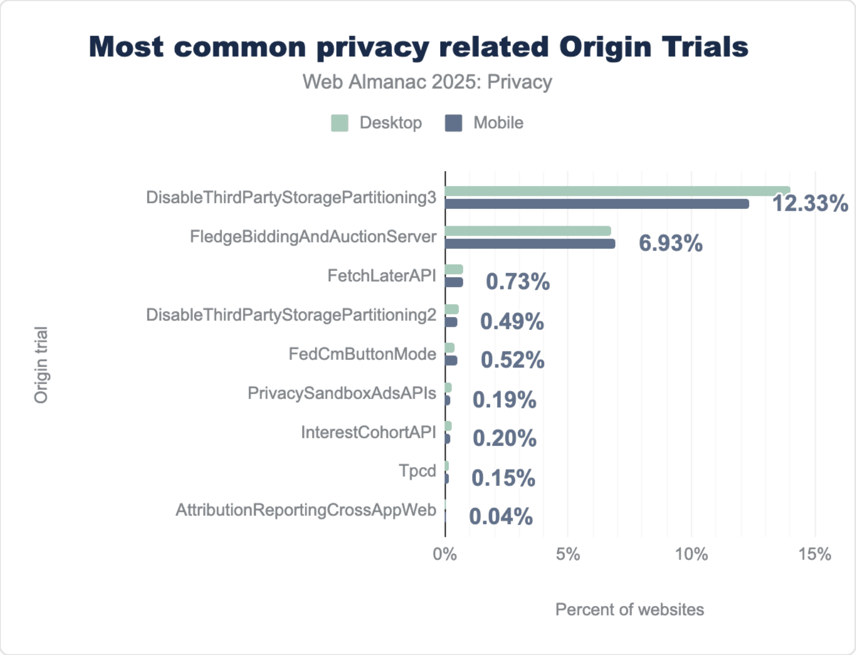


Figure 8.16. Most common privacy related Origin Trials

Origin trials let browsers test experimental features on real websites before committing to a full rollout. Sites can opt in to access new capabilities early, or opt into deprecation trials to temporarily delay changes that would break existing functionality. These trials help browser vendors gather data on how features perform in production while giving developers time to adapt, and as we'll see, most privacy-related adoption falls into the deprecation category.

The most widely adopted trial remains `DisableThirdPartyStoragePartitioning`, which increased from 10.21% in 2024 to 12.33% in 2025 (now in its third iteration). This trial allows sites to temporarily opt out of storage partitioning, a privacy feature that isolates cookies and storage per site, giving developers more time to migrate legacy implementations. Similarly, `FledgeBiddingAndAuctionServer`, part of Google's Privacy Sandbox initiative for interest-based advertising without cross-site tracking, grew modestly from 6.62% to 6.93%.

The biggest shift is `AttributionReportingCrossAppWeb`, which dropped sharply from 2.10% to just 0.04%. This suggests either the trial ended or sites moved away from testing cross-app attribution. New entries include `FetchLaterAPI` (0.73%), deferred requests, and federated identity. Meanwhile, `InterestCohortAPI`, the controversial FLoC predecessor, lingers at 0.20%, largely unchanged and likely residual.

Law and policy

Privacy regulations continue to shape how websites interact with users. In this section, we examine how sites are responding through consent dialogues, and whether privacy signals like Do Not Track and Global Privacy Control are gaining meaningful adoption.

Consent dialogs

Privacy regulations like GDPR[333] and CCPA[334] require websites to obtain user consent before collecting and processing personal data. This has made cookie consent dialogs often managed by Consent Management Platforms (CMPs) a near-universal feature of the modern web. To standardize how consent is captured and communicated across the advertising ecosystem, the Interactive Advertising Bureau developed frameworks like the Transparency and Consent Framework (TCF), US Privacy String (USP), and the newer Global Privacy Platform (GPP).

While these frameworks aim to give users control, adoption and implementation quality vary widely. Some sites fully comply with TCFv2, while others have incomplete implementations or rely on older standards. It’s also worth noting that our crawler is US-based and under TCF, consent banners aren’t required for non-EU visitors, so actual TCF usage is likely higher than what we measure here.

[333] *https://gdpr-info.eu/*
[334] *https://leginfo.legislature.ca.gov/faces/codes_displayText.xhtml?division=3.&part=4.&lawCode=CIV&title=1.81.5*

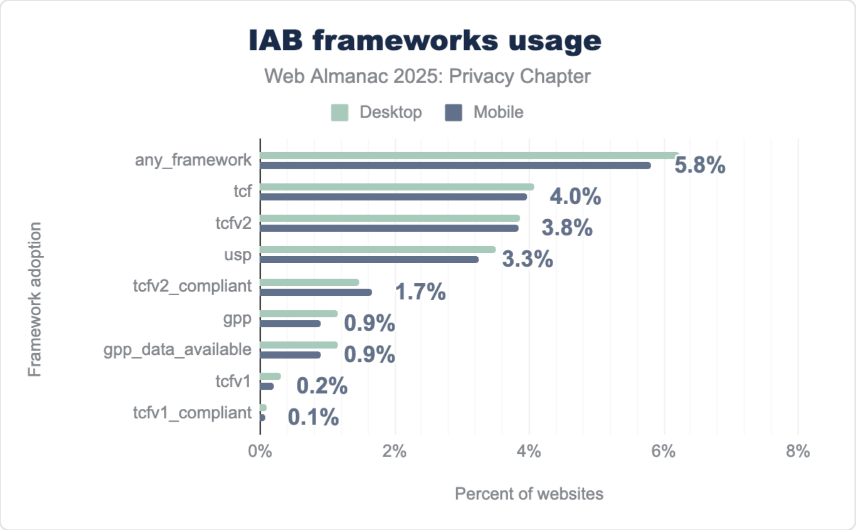


Figure 8.17. IAB frameworks usage

Overall IAB framework adoption remains steady at just above 5.5% for both mobile and desktop. TCF continues to be the most widely adopted framework at 4.0% with TCFv2 accounting for 3.8%. However, only 1.7% of sites are fully TCFv2 compliant, less than half of those claiming to use TCFv2, suggesting that many implementations remain incomplete or improperly configured. USP holds steady at 3.3%, reflecting continued CCPA compliance efforts.

The deprecated TCFv1 has nearly disappeared, sitting at just 0.2% with only 0.1% compliant, indicating the industry has potentially migrated to v2. A notable addition this year is GPP, the IAB's newer unified framework, which appears on 0.9% of sites. Encouragingly, gpp_data_available matches at 0.9%, meaning sites that have adopted GPP are actually using it to transmit user preferences rather than just loading the code.

Comparing year over year, overall framework adoption held flat while TCF usage dipped slightly from 4.2% to 4.0%. This modest decline may reflect early migration toward GPP, though it's too soon to call it a trend. The compliance gap persists, and TCFv2 compliance remained unchanged at 1.7%, highlighting that adoption alone doesn't guarantee proper implementation.

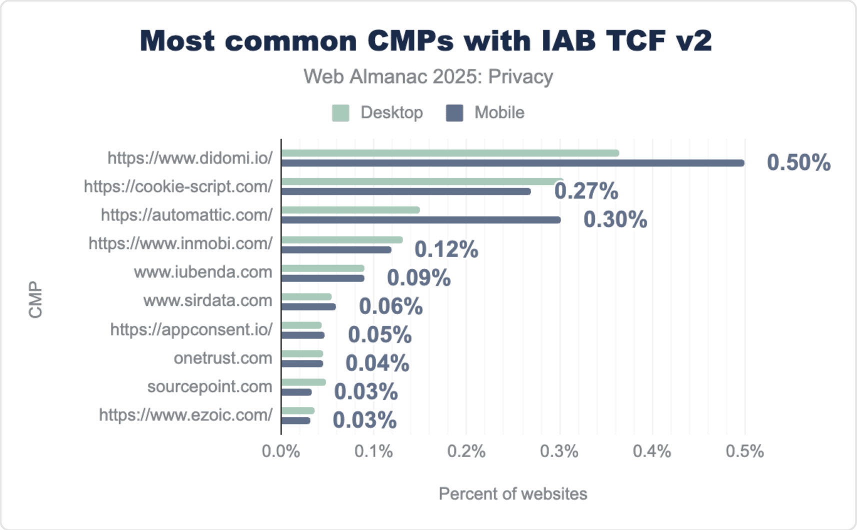


Figure 8.18. Most common CMPs with IAB TCF v2

The CMP landscape shifted notably this year. Automattic, which led in 2024 at 0.67%, dropped to 0.30% in 2025, while Didomi climbed from 0.22% to 0.50%, taking the top spot. Cookie-script emerged as a new entrant at 0.27%, ranking second on desktop. The remaining providers, InMobi, Iubenda, Sirdata, AppConsent, OneTrust, Sourcepoint, and Ezoic, each account for less than 0.12% of sites, showing that TCFv2 CMP adoption remains concentrated among a few major players.

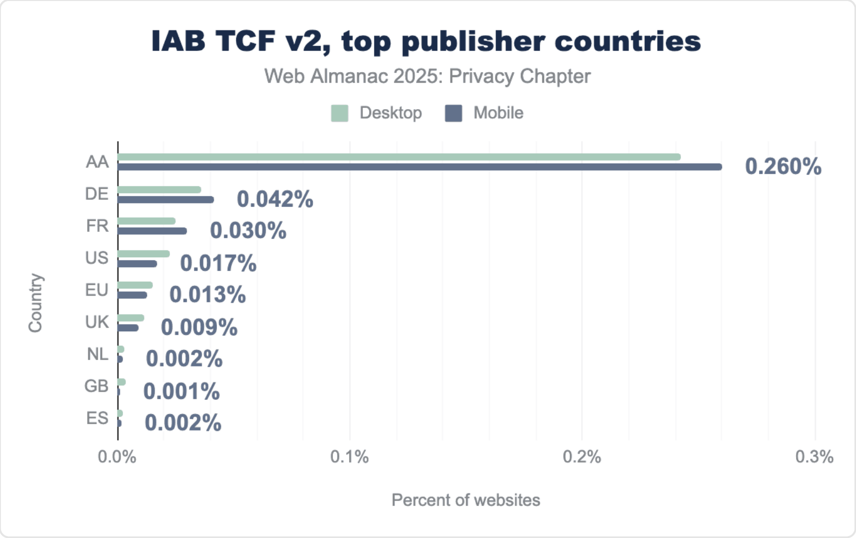


Figure 8.19. IAB TCF v2, top publisher countries

Germany (0.042%) and France (0.030%) lead TCFv2 publisher adoption among EU member states, with the US appearing at 0.017%, notable given TCF isn't required outside the EU. The largest share (0.26%) falls under 'AA', an undefined country code, pointing to gaps in publisher metadata or misconfigured CMP implementations. Overall adoption remains low even among European publishers, suggesting TCFv2 is concentrated among a small subset of sites despite GDPR requirements.

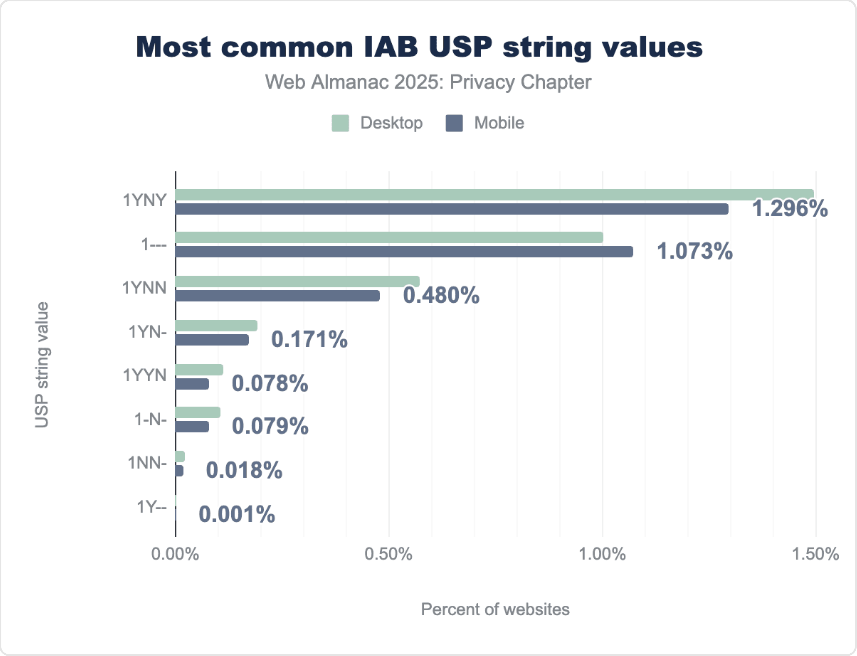


Figure 8.20. Most common IAB USP string values

The most common USP string is 1YNY at 1.296%, indicating that notice was given, the user did not opt out, and the site is covered under the Limited Service Provider Agreement. The second most common value is 1--- at 1.073%, a placeholder string that provides no meaningful signal, suggesting many implementations are incomplete or default. We observed that sites showing `1YYN` have configured their CMP to default new visitors to an opted-out state, a stricter-than-required privacy posture. The low prevalence (0.078%) indicates most sites follow CCPA's standard opt-out model, where consent is assumed until explicitly revoked.

Do Not Track

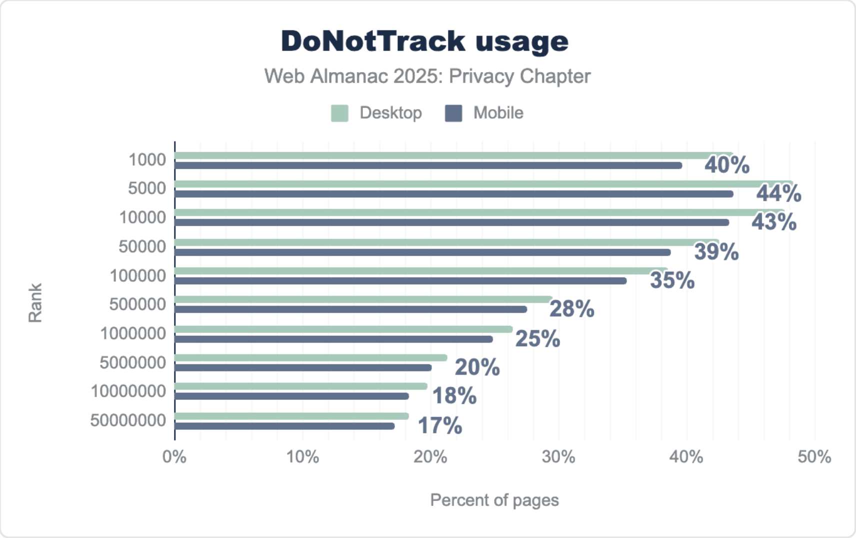


Figure 8.21. Do Not Track usage

Despite being largely abandoned as a standard, with minimum to no legal backing[335] and most advertisers ignoring it, Do Not Track signals persist across the web. Interestingly, adoption correlates strongly with site popularity. Among the top 10,000 sites, DNT detection peaks at around 43%, while the long tail of sites are more likely to maintain legacy privacy signals, even if their practical impact remains questionable.

Mobile adoption consistently edges out desktop across all ranking tiers, though the gap is narrow. The steepest drop-off occurs between the top 100,000 sites (35%) and the 500,000 tier (27%), indicating that mid-tier and smaller sites are far less likely to check for DNT. Whether these sites actually honor the signal, rather than simply detecting it, remains an open question, as DNT compliance has never been enforceable.

Global Privacy Control

Global Privacy Control (GPC) is a browser signal that communicates a user's preference to opt out of having their data sold or shared. Unlike Do Not Track, GPC has legal backing under CCPA/CPRA; websites must treat it as a valid opt-out request. Firefox, Brave, and Safari already support GPC, and Chrome is set to implement it in 2026[336] following California

[335] *https://www.loeb.com/en/insights/publications/2013/10/california-enacts-law-requiring-do-not-track-dis__*
[336] *https://chromestatus.com/feature/5137324344213504*

legislation requiring browsers to offer this setting by 2027. However, like DNT, GPC relies on websites to honor the signal voluntarily at a technical level; the browser sends the header (Sec-GPC: 1), but cannot enforce compliance. The difference is that ignoring GPC carries legal risk, which may prove more effective than DNT's purely voluntary approach.

Conclusion

Online tracking has become the norm on today's Internet. Indeed, we see that 75% (desktop) or 74% (mobile) of the websites we visited contained at least one tracker.

Google continues to dominate the tracking space, followed by Facebook. On the outset, online tracking is lucrative to large companies that can leverage it to serve more targeted ads. However, the consolidation of tracking information amongst a few centralized players is cause for concern to more privacy-conscious users.

Efforts to avoid tracking are constantly being deployed and evaded. For example, we observed `medium.com` in bounce sequences, though likely for functional purposes rather than covert tracking. However, we also discuss safer browser policies, such as sharing user-agent Client Hints instead of the actual user agent string.

Laws and regulations governing online tracking are evolving, along with the mechanisms deployed to comply with them. We see incomplete implementations and poor adoption of the latest version of TCF (v2). However, it comes with a rise in the adoption of the Global Privacy Platform, which is a new addition by the IAB. Moreover, we see a shift in the Consent Management Platform landscape.

Authors

Rumaisa Habib

RumaisaHabib   rumaisahabib   https://rumaisahabib.com/

Rumaisa Habib is a third year PhD candidate working with the Empirical Security Research Group at Stanford University. Rumaisa's primary research interest is to build a better understanding of the web beyond the Western context leveraging large scale Internet measurements.

Vinod Tiwari

𝕏 @@securient securient https://blog.securient.io

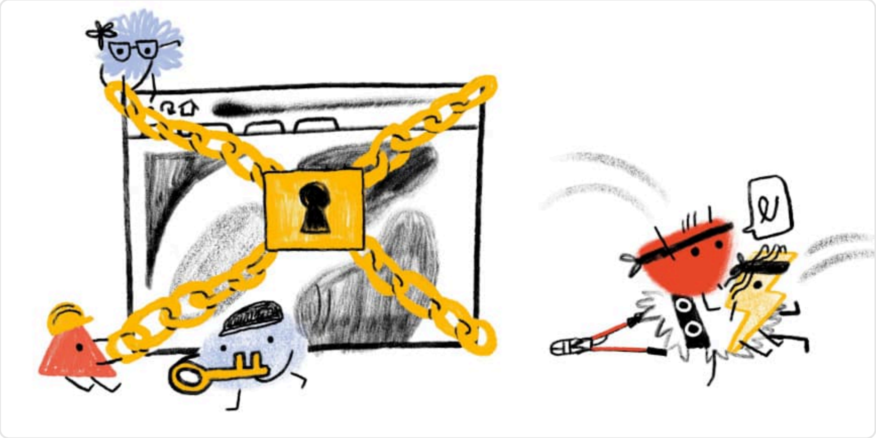
Vinod is a Staff Security Engineer at PIP Labs with over a decade of cybersecurity experience at companies including Amazon, Zapier, and HackerOne. He specializes in penetration testing and cloud security, writes about security on Medium, and actively researches emerging threats in both traditional and Web3 environments.

Nurullah Demir

nrllh nurullahdemir https://ndemir.com

Nurullah Demir is a visiting postdoctoral scholar at Stanford University and the founder of the cyber security platform SecuSeek. His work focuses on internet-scale security and privacy risks.

Part II Chapter 9
Security



Written by Vik Vanderlinden and Gertjan Franken
Reviewed by Anirudh Duggal, Martina Kraus, Gertjan Franken, Brian Clark, Jannis Rautenstrauch, and Vinod Tiwari
Analyzed by Daan V.
Edited by Barry Pollard

Introduction

While more and more parts of many people's lives have moved online, so does their private data, which makes web security ever more important. Many systems we use on a daily basis remain appealing to attackers trying to steal data or cause disruptions. This year has once more demonstrated the scale and complexity of modern threats. The number of DDoS attacks have continued to increase in size and frequency, with the largest attack recorded reaching 31.4 Tbps in November[337]. Supply chain vulnerability grew to unprecedented sizes, with the Shai-Hulud 2.0 attack[338] reportedly compromising over 1,000 npm packages and infecting over 27,000 GitHub repositories. And a critical vulnerability in React known as React2Shell[339] had developers working hard to quickly update their applications.

[337] *https://blog.cloudflare.com/radar-2025-year-in-review/#hyper-volumetric-ddos-attack-sizes-grew-significantly-throughout-the-year*
[338] *https://www.hackerone.com/blog/shai-hulud-2-npm-worm-supply-chain-attack*
[339] *https://www.microsoft.com/en-us/security/blog/2025/12/15/defending-against-the-cve-2025-55182-react2shell-vulnerability-in-react-server-components/*

In this chapter, we analyze the mechanisms that aim to protect the web, and how in some cases they fail to protect the web due to a variety of reasons. We explore core elements of web security such as Transport Layer Security (TLS) and protections against third-party content inclusions. We discuss how the adoption of these security measures evolves, how they help prevent attacks and how misconfigurations can prevent their proper functioning. We further analyze some well-known URIs relating to security.

Beyond measuring adoption, we also explore the drivers behind the adoption of security mechanisms, such as location, technology stack and website industry. By comparing the results to data from previous editions of the Web Almanac, we can notice long-term trends and changes in the state of web security.

Transport security

HTTPS[340] uses Transport Layer Security (TLS)[341] to secure the connection between client and server. Over the years, more and more websites started using HTTPS, thereby better securing their users. This year, the share of all requests sent over HTTPS rose again compared to last year, reaching over 98.8% for mobile connections.

98.8%

Figure 9.1. The percentage of requests that use HTTPS (mobile).

The share of requests that are being sent using HTTPS rather than using plain HTTP creeps ever closer to 100%, rising by another 0.74% on mobile compared to the 2024 edition of Web Almanac[342].

[340] https://developer.mozilla.org/docs/Glossary/https
[341] https://www.cloudflare.com/en-gb/learning/ssl/transport-layer-security-tls/
[342] https://almanac.httparchive.org/en/2024/security#fig-1

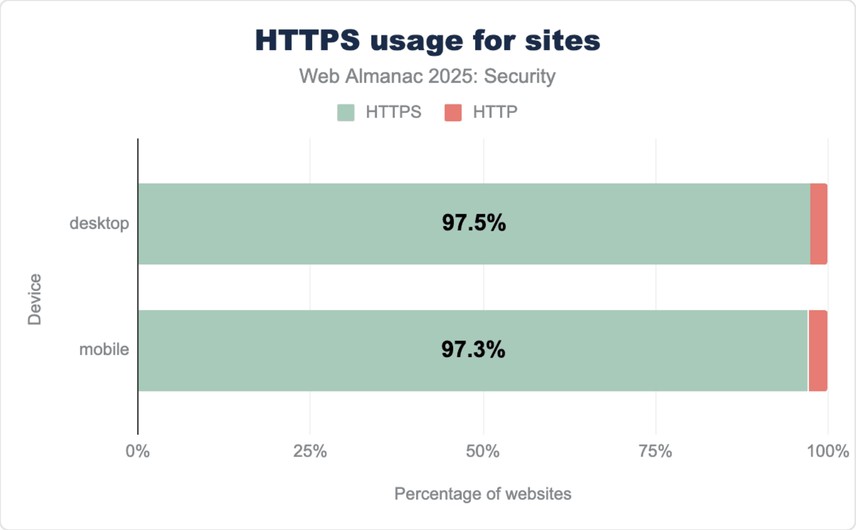


Figure 9.2. The percentage of homepages using HTTPS.

Another positive evolution is visible in the number of homepages served over HTTPS. This number rises from 95.6% to 97.3% on mobile. Because many websites send a number of third-party requests to many (often secure) sites, this number tends to be lower than the share of requests sent over HTTPS, but luckily also keeps rising year after year.

It is good to see that the positive trends in these metrics continues and that the share of sites using HTTPS keeps closing in to 100%. Part of these high numbers can be explained by the decision of browser vendors (Chrome[343], Firefox[344] and Safari[345]) to try to communicate over HTTPS first before falling back to plain HTTP and often showing a security warning to users, thereby encouraging site owners to adopt TLS.

Protocol versions

For a few years now, TLS1.3[346] has been the recommended protocol version to have the highest security. The latest version has deprecated some algorithms that were found to contain flaws[347] in TLS1.2 and provides some stronger security guarantees like forward secrecy. QUIC uses TLS internally as well, thereby providing similar security guarantees as TLS1.3 does[348].

[343] *https://blog.chromium.org/2021/07/increasing-https-adoption.html#:~:text=Beginning%20in%20M94%2C%20Chrome%20will%20offer%20HTTPS%2DFirst%20Mode*
[344] *https://support.mozilla.org/en-US/kb/https-first*
[345] *https://webkit.org/blog/16301/webkit-features-in-safari-18-2/#security-and-privacy*
[346] *https://www.rfc-editor.org/rfc/rfc8446*
[347] *https://www.cloudflare.com/en-in/learning/ssl/why-use-tls-1.3/#:~:text=A%20number%20of%20outdated%20cryptography%20features%20resulted%20in%20vulnerabilities%20or%20enabled%20specific%20kinds%20of%20cyber%20att*
[348] *https://community.cloudflare.com/t/how-is-quic-a-direct-comparison-to-tls-1-3-and-tls-1-2/543349/6#:~:text=TLS%201.2%2C%20TLS%201.3%2C%20and%20QUIC%20share%20similar%20security%20characteristics%20but%20they%20are%20different*

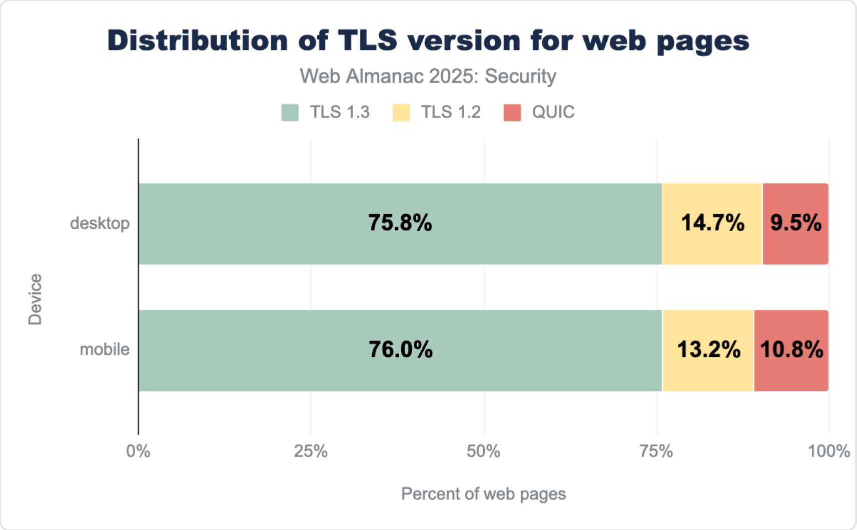


Figure 9.3. The distribution of TLS versions in use.

Like in the 2024 edition of the Web Almanac[349], we find that both QUIC and TLS1.3 see an increase in use. Again we can assume that some sites using TLS1.2 have moved to TLS1.3 and some sites that were using TLS1.2 or TLS1.3 have moved to QUIC. TLS1.2 decreased by another 4.5% this year. We can see that the adoption of QUIC has slowed down a bit this year, with the share of sites using QUIC only rising by 0.8%. In the future, it can be expected that TLS1.2 will slowly phase out over (a long) time and QUIC will continue rising.

Cipher suites

To start communicating over an encrypted channel, both parties need to use the same cryptographic algorithm (or cipher suite[350]) to understand each other. In order to do so, they agree upon a cipher suite to use before communication. We can see that most requests happen over a connection using a Galois Counter Mode (GCM)[351] cipher, which is often preferred because of its resilience against padding attacks[352] and because it provides Authenticated Encryption with Associated Data (AEAD)[353], which is required in TLS1.3[354]. In addition, we see that the use of 128-bit keys has grown by 4% since last year, instead of the more secure 256-variant. Although the 128-bit cipher suites are considered secure by NIST[355], 256-bit

[349] *https://almanac.httparchive.org/en/2024/security#protocol-versions*
[350] *https://learn.microsoft.com/en-au/windows/win32/secauthn/cipher-suites-in-schannel*
[351] *https://www.wikipedia.org/wiki/Galois/Counter_Mode*
[352] *https://blog.qualys.com/product-tech/2019/04/22/zombie-poodle-and-goldendoodle-vulnerabilities*
[353] *https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=2628d946bda9f3d3b087e5c4846e76ae0fb07b6b*
[354] *https://www.rfc-editor.org/rfc/rfc8446#:~:text=Those%20that%0A%20%20%20%20%20%20remain%20are%20all%20Authenticated%20Encryption%20with%20Associated%20Data%0A%20%20%20%20%20%20(AEAD)*
[355] *https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar3.ipd.pdf*

variants provide even more security guarantees. The dataset of this year's Almanac includes more requests than last year's, so it is possible that this change can be attributed to the growing dataset. The cipher suites in use are not very diverse, in fact there are only five cipher suites that were found to be used in the dataset. This can be the case because TLS1.3 only allows GCM or modern block ciphers[356] to be used.

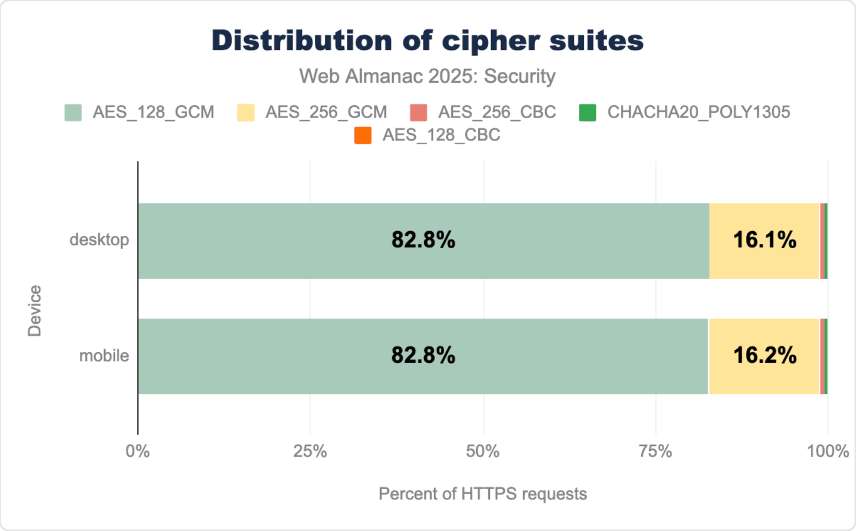


Figure 9.4. The distribution of cipher suites in use.

TLS1.3 only allows the use of algorithms that support forward secrecy[357]. Because of the high adoption of TLS1.3, we expect a large share of requests to fulfill the forward secrecy requirement.

[356] *https://datatracker.ietf.org/doc/html/rfc8446#page-133*
[357] *https://www.wikipedia.org/wiki/Forward_secrecy*

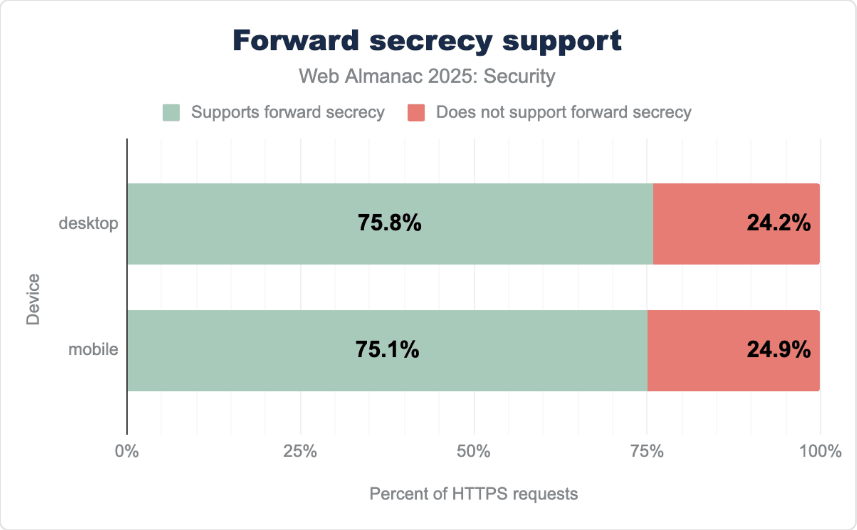


Figure 9.5. The percentage of requests supporting forward secrecy.

Contrary to our expectations, we see a relatively low number of requests that are sent over connections that are forward secret. There is a 20% decrease in forward secret requests since the 2024 Web Almanac[358]. Currently, it seems our metric only includes the forward secret ciphers in TLS1.2 and TLS1.3, but does not include TLS in QUIC which can explain the decline we observer.

Certificate Authorities

In order to use TLS, sites must request a certificate from a Certificate Authority (CA)[359]. Because the browser trusts a number of CAs, a site's certificate can be identified by the browser as a valid certificate. The certificate can then be used for secure communication between the browser and the site's server going forward.

[358] *https://almanac.httparchive.org/en/2024/security#fig-5*
[359] *https://www.ssl.com/faqs/what-is-a-certificate-authority/*

Issuer	Desktop	Mobile
R11	20.80%	21.86%
R10	20.75%	21.73%
WE1	16.35%	17.24%
E6	4.50%	4.65%
E5	4.31%	4.42%
Sectigo RSA Domain Validation Secure Server CA	3.60%	3.52%
Go Daddy Secure Certificate Authority - G2	1.77%	1.60%
WR1	1.16%	1.40%
Amazon RSA 2048 M02	1.37%	1.33%
Amazon RSA 2048 M03	1.30%	1.25%

Figure 9.6. The percentage of certificates issued per issuer (top 10)

Compared to last year, we can see that the then popular R3 intermediate certificate from Let's Encrypt[360] has disappeared as an issuer. This was expected because it has since expired (in September 2025)[361] so even last year the replacement with other intermediates had already started. The R10 and R11 intermediate certificates[362] are the new certificates that are taking over from R3. There are now two intermediate RSA certificates (R10 and R11) and two intermediate ECDSA certificates (E5 and E6) with the explicit goal of trying to prevent intermediate key pinning[363]. The only certificate in the top 5 issuers that is not from Let's Encrypt is WE1, which is part of Google Trust Services (GTS)[364]. Also from GTS in the list is WR1. These certificates seem part of the new generation of intermediate certificates from GTS, expiring among others the GTS CA 1P5 issuer seen last year.

52.6%

Figure 9.7. Percentage of mobile pages that are issued by Let's Encrypt.

The total share of sites using a certificate from Let's Encrypt has gone down slightly to 52.6%

[360] https://letsencrypt.org/
[361] https://crt.sh/?id=3334561879
[362] https://letsencrypt.org/2024/03/19/new-intermediate-certificates.html
[363] https://letsencrypt.org/2024/03/19/new-intermediate-certificates.html#rotating-issuance
[364] https://pki.goog/

from 56% in the last edition. One of the contributing factors, as can be seen in the data, is the larger share of certificates issued by the WE1 certificate from GTS. However, the total share by GTS-issued certificates (WE1 and others) has not been calculated.

HTTP Strict Transport Security

HTTP Strict Transport Security[365] is a response header through which servers can instruct browsers to only visit pages on this domain over HTTPS, instead of trying HTTP first and following the redirect to HTTPS.

36%

Figure 9.8. The percentage of pages using the HSTS header on mobile.

We see a continuing increase in the number of pages using an HSTS header, with a rise of 6 percentage points compared to last edition[366], up to 36% of all pages visited on mobile.

Servers can include a number of directives in the header to communicate additional preferences to the browser. For example, the `max-age` directive tells the browser how long it is required to continue using only HTTPS. The other directives, `includeSubDomains` and `preload`, are optional.

[365] *https://developer.mozilla.org/docs/Web/HTTP/Headers/Strict-Transport-Security*
[366] *https://almanac.httparchive.org/en/2024/security#fig-8*

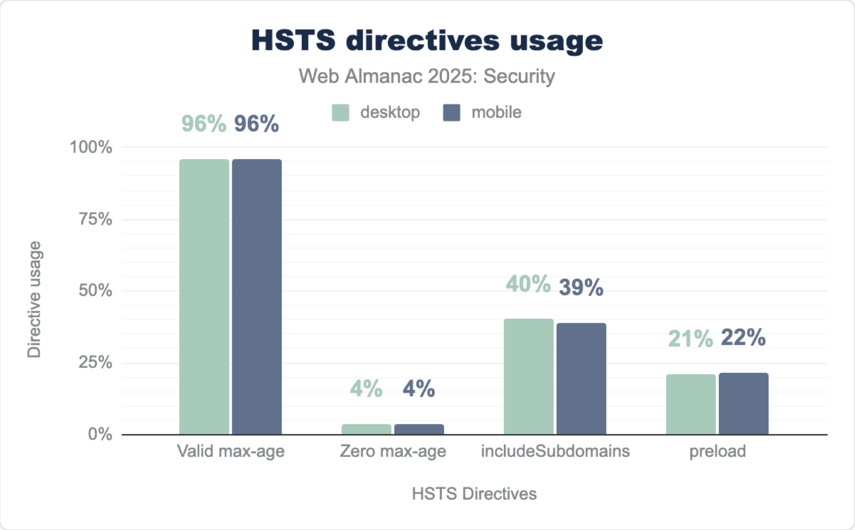


Figure 9.9. The usage of specified HSTS directives.

The share of responses with a valid `max-age` has increased slightly to 96%. The `includeSubdomains` and `preload` directives saw an increase of about 4% each, possibly indicating that certain sites started setting both directives together. The unofficial[367] `preload` directive requires the `includeSubdirectories` to be set and the `max-age` to have a value of at least 1 year. When using the `preload` directive, a site can make sure that a browser will always visit the domain and its subdomains over HTTPS, even when connecting for the first time (which is not necessarily the case when using HSTS without preload).

[367] *https://developer.mozilla.org/docs/Web/HTTP/Headers/Strict-Transport-Security#preloading_strict_transport_security*

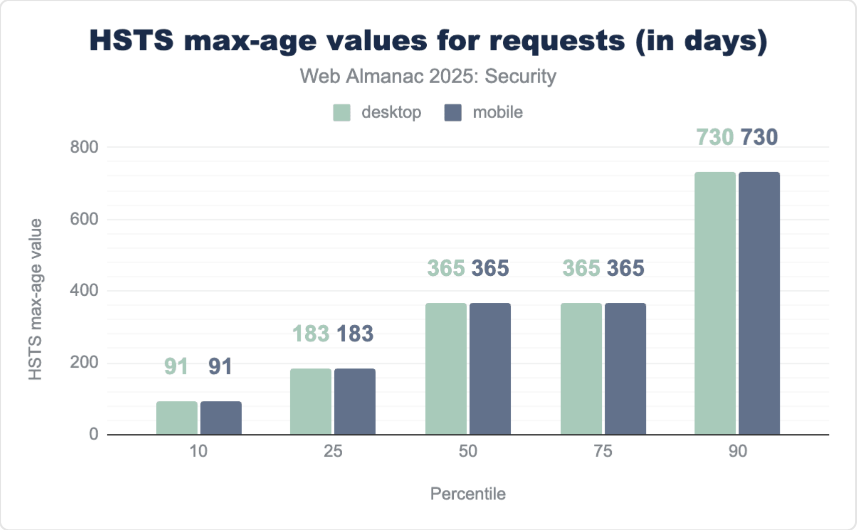


Figure 9.10. The distribution of HSTS `max-age` values by percentile.

The distribution of valid `max-age` values remains largely the same with the exception of the lower percentiles. At the 10th percentile, we see a large increase from 30 days to 91 days, indicating that less sites are setting very low `max-age` values. The median remains at one year or 365 days.

Cookies

Cookies are a vital part of the web. They allow websites to save information for use over multiple stateless requests. In order to secure sites' cookies, there are many features built into browsers that are further reported on in the Cookies chapter. We specifically refer to the Cookie attributes, Cookie prefixes and Persistence (expiration) sections in this chapter.

Content inclusion

Content inclusion is a core component of the web. Being able to include other pages, CSS or JavaScript from a Content Distribution Network (CDN) or images from shared sources is one of the building blocks on which the web was built. It does however introduce certain risks such as sites including content from third parties which places significant trust in those third-party resources. Of course, there is no guarantee that said resource is not malicious or compromised by a malicious actor which can lead to a number of serious attacks such as supply chain attacks.

To reduce this risk, it is important to use security policies to control content inclusion.

Content Security Policy

The Content Security Policy (CSP)[368] allows websites to have fine-grained control over the content that will be loaded on its page. By setting the `Content-Security-Policy` response header or defining it in a `<meta>` html tag, websites can communicate the policy in use to the browser, which will enforce it. The policy has many available directives that allow a website to define from which sources content can be loaded or not.

CSP can be used to block specific resources from being loaded, which can help reduce the impact of potential cross-site scripting (XSS) attacks. In addition CSP can also serve other purposes, such as enforcing the use of encrypted communication channels by means of the `update-insecure-requests` directive or controlling on which pages the current page can load as a subresource using the `frame-ancestors` directive. This allows websites to defend against clickjacking attacks.

+18%

Figure 9.11. Relative increase in adoption of the `Content-Security-Policy` *header from 2024.*

The adoption of CSP continued increasing from 18.5% last year[369] to 21.9% this year, an increase of close to 20%. As predicted in the last edition of the Web Almanac, the adoption has now risen above 20%, indicating the adoption is still steadily rising.

[368] https://developer.mozilla.org/docs/Web/HTTP/CSP
[369] https://almanac.httparchive.org/en/2024/security#content-security-policy

Directives

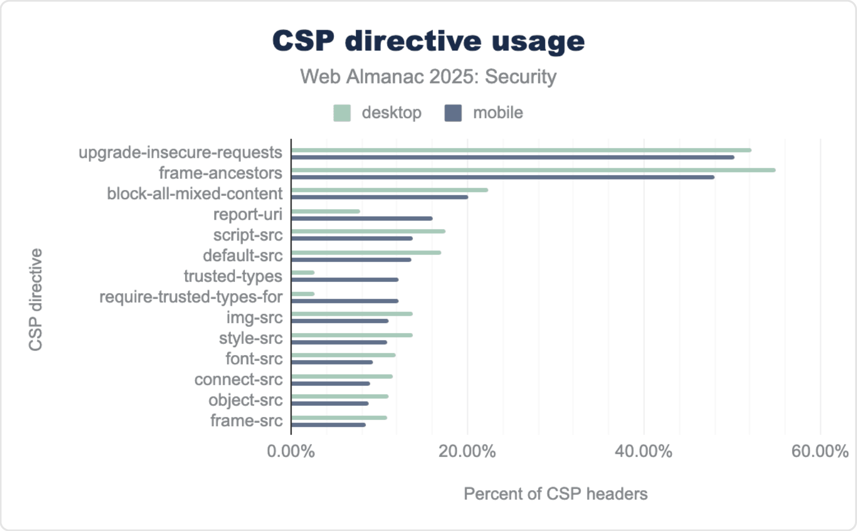


Figure 9.12. Most common directives used in CSP.

Once again, most websites use CSP for the `upgrade-insecure-requests` and `frame-ancestors` directives. The relative share of mobile sites using these directives has decreased slightly, which can be attributed to the higher number of CSP headers scanned. The absolute number has risen by 400,000 and 800,000 CSP headers on desktop and mobile respectively.

The `block-all-mixed-content` directive which has been replaced by `upgrade-insecure-requests` has continued to slightly decrease like it has been over the last few years. This is good news because the directive is deprecated[370].

[370] https://developer.mozilla.org/docs/Web/HTTP/Reference/Headers/Content-Security-Policy/block-all-mixed-content

Policy	Desktop	Mobile
`upgrade-insecure-requests`	21.45%	22.79%
`block-all-mixed-content; frame-ancestors 'none'; upgrade-insecure-requests;`	19.92%	18.06%
`require-trusted-types-for 'script';report-uri /checkin/_/AndroidCheckinHttp/cspreport`	2.67%	12.10%
`frame-ancestors 'self'`	7.55%	6.38%
`upgrade-insecure-requests;`	4.92%	4.30%
`frame-ancestors 'self';`	2.53%	2.29%

Figure 9.13. Most prevalent CSP headers

The same header values we saw last year in the top three appear in this year's top four again. The third most common CSP header this year is a new one, however. This change occurs because this year, we sorted the most common headers by mobile usage instead of by desktop usage like last year. We can see a trusted types policy with a specific `report-uri` endpoint relating to Android appear at the third place.

Trusted types[371] can be used to restrict the parameters passed into injection sinks (like `element.innerHTML`) such that they only allow properly typed values to be passed instead of plain strings. By restricting the values passed to injection sinks, many possible DOM XSS vulnerabilities can be prevented. The trusted types header we observe appears on more than 12% of the mobile pages. We don't have a direct explanation for the high usage of this specific CSP policy value.

In fifth and sixth place we once again see the `upgrade-insecure-requests` and `frame-ancestors 'self'` directives, but this time with a trailing semicolon. A semicolon is used to separate directives but if there is only one directive defined it can be discarded[372], both header values are therefore valid CSP policies with the same effect.

Keywords for `script-src`

One of the most important directives included in the CSP is `script-src`. Through the use of this directive, websites can control which scripts can run on their pages. This can hinder

[371] https://w3c.github.io/trusted-types/dist/spec/
[372] https://w3c.github.io/webappsec-csp/#grammardef-serialized-policy

attackers because it can disallow them from running arbitrary scripts. `script-src` has multiple potential keywords.

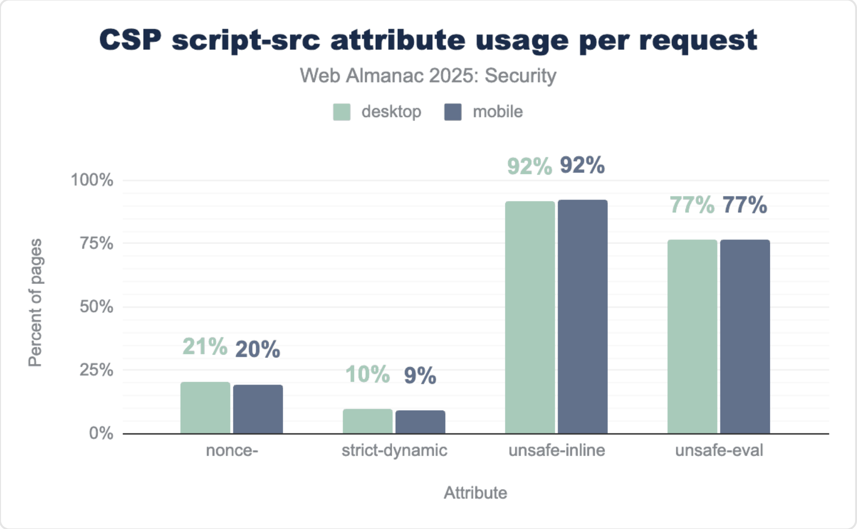


Figure 9.14. Prevalence of CSP `script-src` keywords per request.

We find that the `unsafe-inline` and `unsafe-eval` keywords are used very often. These keywords significantly reduce the security impact of CSP's `script-src` as they allow any inline script to be executed or allow the use of the `eval` -function in JavaScript respectively.

Compared to last year[373] we barely see any changes to the usage of `script-src` keywords. An important note to make is that the presence of `unsafe-inline` does not necessarily mean that inline scripts can be executed. In some cases and following the CSP specification[374] `unsafe-inline` will be ignored. This is for instance the case when a nonce and `strict-dynamic` keywords are added to the CSP policy.

[373] https://almanac.httparchive.org/en/2024/security#keywords-for-script-src
[374] https://w3c.github.io/webappsec-csp/

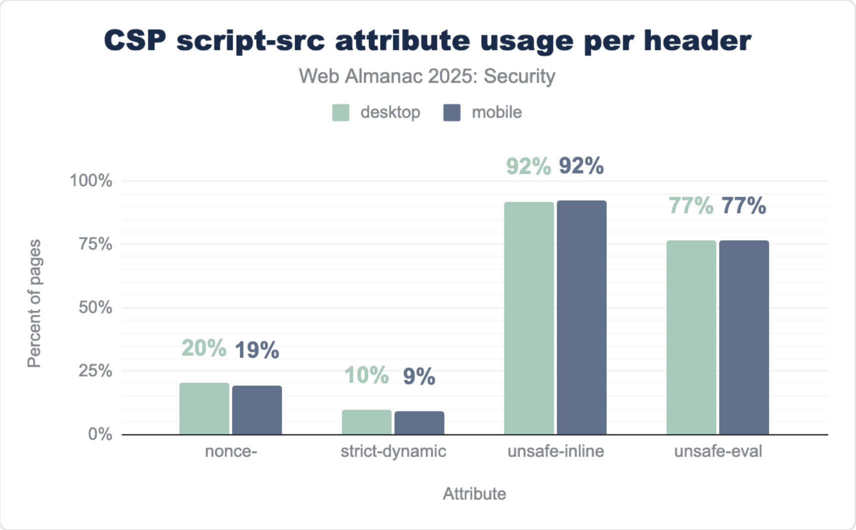


Figure 9.15. Prevalence of CSP `script-src` keywords per header.

We also check the use of keywords per header instead of per page. In CSP, multiple CSP headers can be present in one response and may define different directives. If a directive is defined multiple times, the policies will be combined to create the the most restrictive policy to be used by the browser[375].

We see a very similar distribution compared to the values per request, indicating that either most pages only use one CSP header or only use `script-src` in one of the CSP headers that they set, meaning there are no conflicting `script-src` directives on most pages.

Allowed hosts

CSP is a complex security policy to thoroughly understand and correctly use. When looking at the length of the CSP headers in use, we can see a wide variety in policy sizes.

[375] *https://content-security-policy.com/examples/multiple-csp-headers/*

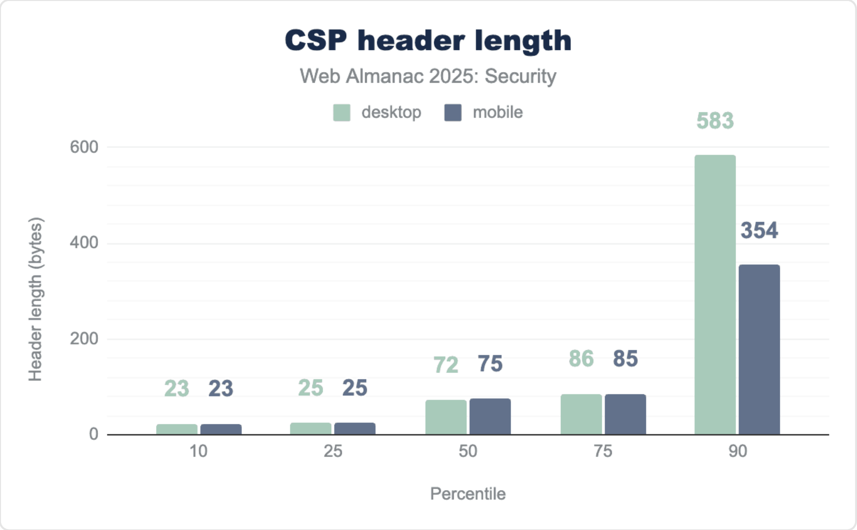


Figure 9.16. CSP header length.

Out of all observed headers, 75% are 86 bytes or less in length. This is slightly more than last year where this was 75 bytes or less. We can see that there are more longer policies in use and in the 90th percentile. The desktop policies have gotten longer while the mobile policies have gotten slightly shorter, increasing the difference between the policy lengths.

Host	Desktop	Mobile
`https://www.googletagmanager.com`	0.74%	0.62%
`https://fonts.gstatic.com`	0.61%	0.49%
`https://fonts.googleapis.com`	0.60%	0.48%
`https://www.google.com`	0.54%	0.46%
`https://www.google-analytics.com`	0.47%	0.38%
`https://www.youtube.com`	0.41%	0.34%
`https://*.google-analytics.com`	0.35%	0.31%
`https://*.google.com`	0.31%	0.30%
`https://connect.facebook.net`	0.33%	0.29%
`https://www.gstatic.com`	0.35%	0.27%

Figure 9.17. Most frequently allowed HTTP(S) hosts in CSP policies

The most commonly loaded HTTPS origins included in CSP headers are those used for ads, fonts and other CDN resources.

Host	Desktop	Mobile
`wss://*.hotjar.com`	0.18%	0.15%
`wss://*.intercom.io`	0.07%	0.07%
`wss://booth.karakuri.ai`	0.08%	0.06%
`wss://www.livejournal.com`	0.04%	0.05%
`wss://*.quora.com`	0.03%	0.05%
`wss://tsock.us1.twilio.com/v3/wsconnect`	0.02%	0.04%
`wss://api.smooch.io`	0.05%	0.03%
`wss://*.zopim.com`	0.05%	0.03%
`wss://*.pusher.com`	0.04%	0.03%
`wss://cable.gumroad.com`	0.04%	0.03%

Figure 9.18. Most frequently allowed WSS hosts in CSP policies

For the secure websocket (`wss://`) origins, we see Hotjar take the first position, doubling in occurrences. Other origins remain low in occurrence.

Hotjar is used for website analytics, indicating a continuing interest in analytical information of websites while Intercom is used for customer services. We also see AI-first companies entering these statistics with karakuri, a Japanese AI company that is closing the top three.

Subresource Integrity

In order to protect themselves from loading tampered resources, developers can make use of Subresource Integrity (SRI)[376]. While CSP allows for developers to restrict the sources from which resources are loaded, SRI makes sure those resources contain the content that is expected by the developer. The opposite can be the case when for instance a CDN is compromised and an attacker succeeds in changing a valid script into a malicious one.

By using the `integrity` attribute in `<script>` and `<link>` tags, developers can communicate to the browser the expected hash of the resource. When loading the specified resource, the browser will then check whether the hash of the resource contents corresponds

[376] https://developer.mozilla.org/docs/Web/Security/Subresource_Integrity

to the provided hash and if not, refuse to load/execute the resource, thereby protecting the website from potentially compromised content.

23.6%

Figure 9.19. Pages using SRI (mobile).

SRI is used on 25.9% and 23.6% of desktop and mobile pages respectively. This is a rise by around 2.5% for both numbers since last year[377], showing that a growing number of developers are using SRI to protect their pages.

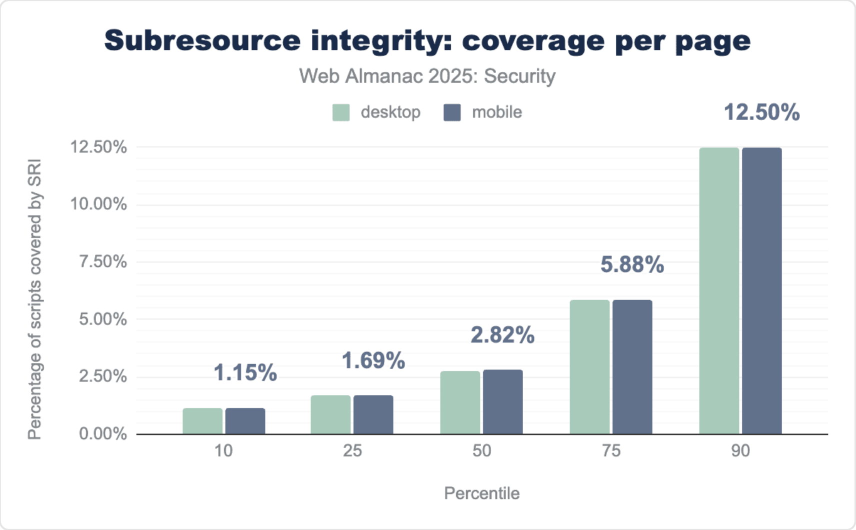


Figure 9.20. SRI coverage per page.

Compared with last year[378], we see a drop in median subresource coverage per page by 0.41%. It is likely that this drop is caused by the larger number of pages being crawled by the web almanac crawler this year.

[377] https://almanac.httparchive.org/en/2024/security#subresource-integrity
[378] https://almanac.httparchive.org/en/2024/security#fig-25

Host	Desktop	Mobile
`www.gstatic.com`	34.70%	34.55%
`static.cloudflareinsights.com`	8.89%	8.37%
`cdnjs.cloudflare.com`	7.43%	7.20%
`cdn.userway.org`	6.10%	6.60%
`code.jquery.com`	4.96%	4.77%
`cdn.shoplineapp.com`	4.62%	5.22%
`cdn.jsdelivr.net`	4.50%	4.06%
`d3e54v103j8qbb.cloudfront.net`	2.03%	2.30%

Figure 9.21. Most common hosts from which SRI-protected scripts are included

The list of most common hosts from which SRI-protected scripts are loaded has remained largely stable over time, with some entries moving relative positions. The `cloudflareinsights` record for instance has increased in relative occurrence, but only by 2.40%.

Like last year, most of these entries are CDNs, showing that most resources loaded and protected by SRI are loaded from CDNs. This makes sense because these scripts are often not under direct control of the developers. Often, CDNs will host a large number of scripts that can then be included by many developers. This is advantageous for developers because the load on their own servers will go down, but also for the client because it can cache scripts from the CDN instead of loading the same script once for every website that includes it.

The risk of including a script that is hosted on a server not under your own control is higher than when you as developer have full control. If the developer of this external script decides to include malicious content or the server is compromised and the script is swapped with a malicious one, without SRI the developer’s users will be loading and executing this malicious content. A good way to protect users against these unexpected changes and to know with certainty which script is allowed to run is to use SRI.

Permissions Policy

The Permissions Policy[379] (formerly Feature Policy) is a policy that enables websites to allow or disallow the use of specific features in the browser, such as the camera, microphone, sensors (like the accelerometer) or geolocation data. Through the `Permissions-Policy` response header, developers can allow or disallow specific feature use by the main page and its embedded content. A specific policy for one embedded resource can be set through the `allow` attribute of the `<iframe>` element.

+50%

Figure 9.22. Relative increase in adoption of the `Permissions-Policy` header from 2024 (mobile).

Compared to last year[380], the use of the `Permissions-Policy` saw a relative increase of almost 60%. Although this is a large relative increase, the absolute percentage of websites using `Permissions-Policy` remains rather small at 3.7% on mobile. The rise in adoption is a good sign for the policy.

[379] https://developer.mozilla.org/docs/Web/HTTP/Permissions_Policy
[380] https://almanac.httparchive.org/en/2024/security#permissions-policy

Header
interest-cohort=()
accelerometer=(), autoplay=(), camera=(), encrypted-media=(), fullscreen=(), geolocation=(), gyroscope=(), magnetometer=(), microphone=(), midi=(), payment=(), picture-in-picture=(), publickey-credentials-get=(), screen-wake-lock=(), sync-xhr= usb=()
geolocation=(),midi=(),sync-xhr=(),microphone=(),camera=(),magnetometer=(),gyroscope=(),fullscreen=(self),payme
accelerometer=(), autoplay=(), camera=(), cross-origin-isolated=(), display-capture=(self), encrypted-media=(), fullscreen=*, geolocation=(self), gyroscope=(), keyboard-map=(), magnetometer=(), microphone=(), midi=(), payment=*, picture-in-picture=*, publickey-credentials-get=(), screen-wake-lock=(), sync-xhr=*, usb=(), x spatial-tracking=(), gamepad=(), serial=()
geolocation=(self)
accelerometer=(), camera=(), geolocation=(), gyroscope=(), magnetometer=(), microphone=(), payment=(), usb=()
accelerometer=(self), autoplay=(self), camera=(self), encrypted-media=(self), fullscreen=(self), geolocation=(self), gyroscope=(self), magnetometer=(self), microphone=(self), midi=(self), payment=(self), usb=(self)
browsing-topics=()
geolocation=(), microphone=(), camera=()
geolocation=self

Figure 9.23. Most prevalent `Permission-Policy` headers

When looking at the top 10 used `Permissions-Policy` values, we find that less developers now use the header to opt out of Google's Federated Learning of Cohorts (FLoC)[381], with only 11.5% of `Permissions-Policy` headers containing the `interest-cohort=()` value. We also see that a value to opt out of many features at once became a popular value with 10% of `Permissions-Policy` headers containing this value on mobile. We suspect this is due to it being replaced by the Topics API with a different permission policy[382]. This is also likely to reduce further in the future since Google is retiring these Privacy Sandbox APIs[383].

[381] https://privacysandbox.com/intl/en_us/proposals/floc/
[382] https://privacysandbox.google.com/private-advertising/topics/web/controls#opt_out_as_a_developer
[383] https://privacysandbox.google.com/blog/update-on-plans-for-privacy-sandbox-technologies

All other observed values in the top 10 are aimed at restricting the permissions of the web page and embedded content. The Permissions Policy is open by default, which means that in order to restrict the use of a feature, it has to be explicitly mentioned in the header. Like last year, 0.27% of `Permissions-Policy` headers on desktop set the `*` wildcard value, thereby explicitly granting all permissions to the page and embedded content that does not define a stricter policy in the `allow` attribute. On mobile, we do not find the wildcard value at all.

As mentioned before, the Permissions Policy can also be defined in the `allow` attribute on an html `<iframe>`. For example an iframe allowing its embedded content to have access to the camera and microphone would look like:

```
<iframe src="https://example.com/" allow="camera 'self'; microphone
'self'">
```

Out of the total 33.3 million iframes on mobile, we observed that 29.2% include an `allow` attribute. The relative decrease in use of the `allow` attribute can be attributed to the more than 10 million more iframe elements that were found in this year's crawl that included more pages than last year's.

Directive	Desktop	Mobile
`encrypted-media`	60.90%	70.66%
`autoplay`	34.99%	41.83%
`picture-in-picture`	26.10%	36.63%
`gyroscope`	23.76%	34.20%
`accelerometer`	23.76%	34.20%
`clipboard-write`	20.39%	26.24%
`join-ad-interest-group`	24.23%	17.35%
`web-share`	12.07%	15.67%
`attribution-reporting`	3.93%	2.35%
`run-ad-auction`	3.84%	2.26%

Figure 9.24. Most prevalent `allow` attribute directives

Interestingly, the three top `allow` attribute values of last year (`join-ad-interest-group`, `attribution-reporting` and `run-ad-auction`) have extremely low adoption compared to last year. It is possible that the large player that was hypothesized to have added these values to their iframe elements has since reverted that change. The other top 10 values of last year have seen a major increase in inclusion into the `allow` attribute value overall, with absolute changes up to +50%.

Iframe sandbox

By employing the `sandbox` attribute on `<iframe>` elements, developers can protect their users against several attacks in which a resource embedded in an `<iframe>` is either compromised or malicious. In the `sandbox` attribute's value, developers can specify which restrictions should be put into place for the content loaded and displayed in the `<iframe>`. For example, the following `<iframe>` would allow the embedded webpage to run scripts:

```
<iframe src="https://example.com/" sandbox="allow-scripts"></iframe>
```

We see the use of the sandbox attribute rise compared to the 2024 edition from 20.0% to 22.7%, showing that more and more developers want to protect their users against potential misuse by embedded content.

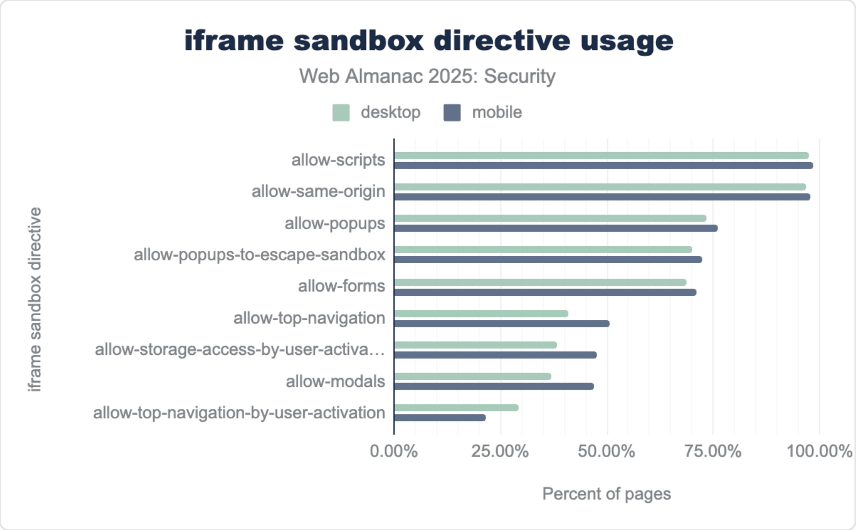


Figure 9.25. Prevalence of sandbox directives on iframes.

Like last year[384], we see that out of all iframes with the `sandbox` attribute, 98.5% (on mobile) use it to allow the embedded webpage to execute scripts by including the `allow-scripts` directive. Following in second place, used in 97.8% on mobile iframes, developers use the `allow-same-origin` to make the loaded resources part of the embedder's origin.

Emerging security headers

In general we see a rise in most of the existing security features such as TLS, HSTS, CSP policies, SRI and the iframe `sandbox` attribute. Besides these existing features, as time keeps moving, new attacks are crafted and more defenses are being implemented. This section goes over the relatively new Document Policy that is starting to show up more often in the wild.

Document Policy

Document Policy[385] is a draft community group report last updated in 2022. It was originally created as a response to proposed additions to Permissions Policy that did not fit the Permissions Policy model or added too much complexity.

Document Policy has several advantages over related mechanisms such as Permissions Policy,

[384] *https://almanac.httparchive.org/en/2024/security#iframe-sandbox*
[385] *https://wicg.github.io/document-policy/*

CSP and sandboxing. It is more fine-grained than Permissions Policy and has a different inheritance model: child resources can overwrite certain parent-chosen policies if they are compatible. It is more general than CSP as it has directives related to the permissions of a resource once it has been loaded instead of only determining from which origins resources can be loaded. It is easier to extend than sandboxing because features that are not mentioned in sandboxing are blocked by default, which makes it very difficult to add new ones.

Header	Desktop	Mobile
`include-js-call-stacks-in-crash-reports`	68.48%	71.99%
`js-profiling`	12.66%	15.24%
`js-profiling; include-js-call-stacks-in-crash-reports`	17.41%	11.94%
`force-load-at-top`	1.25%	0.65%
`no-font-display-late-swap`	0.06%	0.05%

Figure 9.26. Most common document policy header values

We can see that of the Document Policy headers in use, more than two thirds of them are used to include call stacks in crash reports. Combined with the `js-profiling` directive these two features make up the vast majority of current use-cases. Currently in total we find policy values containing 19 different directives. In general there may be more directives defined but as of now we are not aware of the total number them.

We currently only find just over 24,000 and 29,500 pages for desktop and mobile respectively which is 0.10% of the total number of pages visited for both. We expect to see a rise in adoption of Document Policy headers going forward, although future adoption may not happen quickly.

Attack preventions

While there are many defenses for websites implemented by many browsers, it can be challenging to keep an overview of all the possibilities and best practices. In addition, when protections are opt-in and therefore not enabled by default, it becomes even more of a challenge. Developers have to remain up to speed with modern attacks and the defenses that exist to protect users against these attacks. This section assesses which attack prevention measures are in use across the web.

Security header adoption

A multitude of protection mechanisms can be configured through HTTP response headers. Based on the values of these headers the browser will enforce these protections. Not all security mechanisms are relevant for every website, but the absence of all security headers can point to missing urgency towards security.

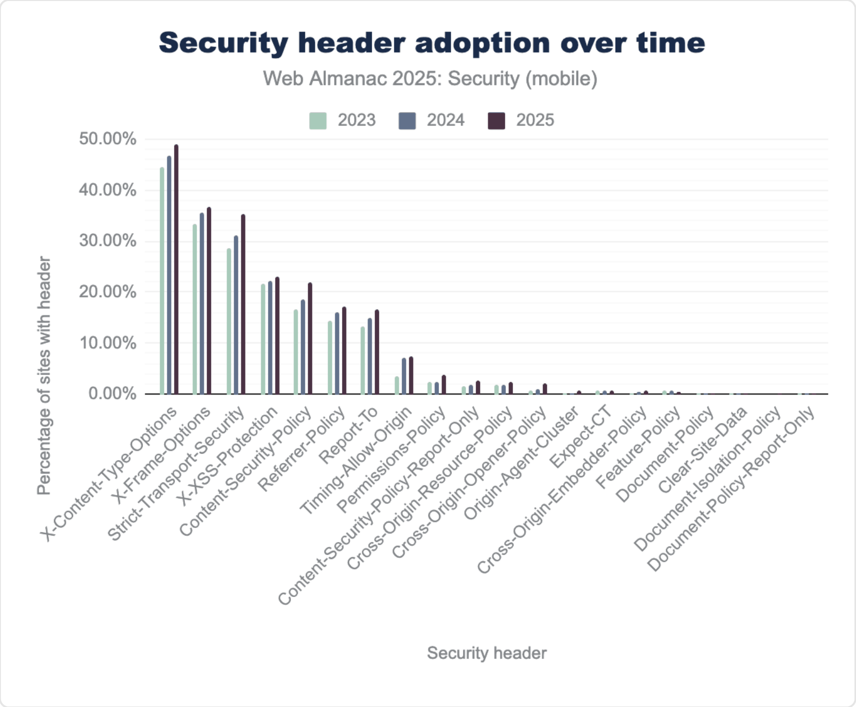


Figure 9.27. Adoption of security headers for site requests in mobile pages over time.

Like last year[386] there are only a few headers for which adoption decreased. The `Feature-Policy` header is deprecated in favor of the `Permissions-Policy`, therefore it's no surprise that the adoption is declining. The other two: `Clear-Site-Data` and `Document-Policy-Report-Only` have such low adoption (0.01% and 0.00001% respectively) that relative changes in adoption may seem large while absolute differences are actually small. This means that the overall adoption of security headers keeps increasing over time, which is a positive sign for web security overall.

The strongest risers since the 2024 edition are `Strict-Transport-Security` (+4.02%),

[386] https://almanac.httparchive.org/en/2024/security#security-header-adoption

`Content-Security-Policy` (+3.39%) and `X-Content-Type-Options` (+2.30%).

`Origin-Agent-Cluster`

The `Origin-Agent-Cluster`, when correctly set, communicates to the browser a request to share the resources used for the document (like the operating system process) with documents of the same origin. The browser may or may not honor the request and the client can verify using JavaScript whether the request was in fact honored.

Usage remains low with only 0.47% of mobile sites, and 0.38% desktop sites using this, but let's dig into what they are using it for:

Header	Desktop	Mobile
?1	74.11%	90.32%
?0	25.79%	9.60%
1	0.08%	0.07%
0	0.01%	0.01%

Figure 9.28. Most common `Origin-Agent-Cluster` header values

A boolean is defined in the HTML living standard[387] as a string starting with a question mark. This means that values like `1` and `0` are invalid for this header. Luckily the use of these values is limited. The `?1` value is the only valid value for the `Origin-Agent-Cluster` and it's used to communicate that the developer wants to opt into the feature, all other values are ignored. On mobile, more than 90% of headers have the valid `?1` value. Unfortunately, 0.07% of header values are `1`, a value that will be ignored while the developer likely wants to request the use of dedicated resources.

Use of `document.domain`

By using `document.domain`, a developer is able to read the domain portion of the current document, as well as set a new domain (only superdomains of the current domain are allowed), after which the browser will use the new domain as origin for the same-origin policy checks. However, the use of this property is now deprecated and browsers may stop supporting the property soon.

[387] https://httpwg.org/specs/rfc8941.html#boolean

Blink Feature	Desktop	Mobile
`DocumentSetDomain`	0.49%	0.36%
`DocumentDomainSetWithNonDefaultPort`	0.16%	0.14%
`DocumentDomainEnabledCrossOriginAccess`	0.0008%	0.0004%
`DocumentDomainBlockedCrossOriginAccess`	0.0002%	0.0001%
`DocumentOpenAliasedOriginDocumentDomain`	0.00008%	0.00001%

Figure 9.29. The use of `document.domain` based on specific blink features

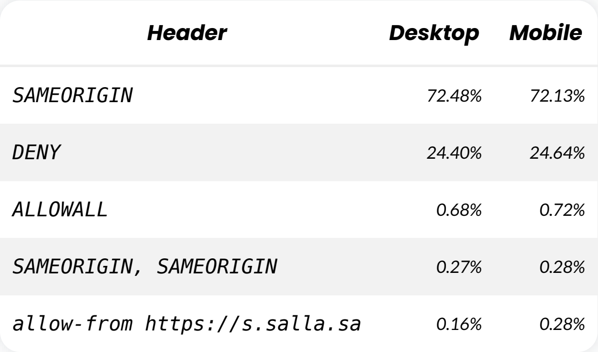
We see that less than 0.5% of websites on desktop and mobile are using the `document.domain` setter to change the origin of a page and even less sites do so with a non-default port. This is a positive trend but still represents a few tens of thousands of websites that should update their code.

Preventing clickjacking with CSP and X-Frame-Options

As previously mentioned, a Content Security Policy (CSP) can be effective against Clickjacking[388] attacks through the use of the `frame-ancestors` directive. Some of the top CSP header values include a `frame-ancestors` directive with a `'none'` or `'self'` value, thereby blocking embedding of the page overall or restricting the embeddings to pages of the same origin.

Another way of defending against clickjacking attacks is through the `X-Frame-Options` (XFO) header. By setting the XFO header, developers can communicate that a document cannot be embedded in other documents ('DENY') or can only be embedded in documents of the same origin (`SAMEORIGIN`).

[388] https://owasp.org/www-community/attacks/Clickjacking



Header	Desktop	Mobile
`SAMEORIGIN`	72.48%	72.13%
`DENY`	24.40%	24.64%
`ALLOWALL`	0.68%	0.72%
`SAMEORIGIN, SAMEORIGIN`	0.27%	0.28%
`allow-from https://s.salla.sa`	0.16%	0.28%

Figure 9.30. Most prevalent `X-Frame-Options` header values

We find that there are barely any changes between the data this year and in 2024[389]. The `X-Frame-Options` header is primarily used to allow same-origin websites to embed the page (72.1%). Secondly, it is used to deny any page from embedding its own content (24.6%).

Examples of other values we observe are shown in the third to fifth row of the table. The `ALLOW-FROM` value used to be valid but is now deprecated and ignored by browsers. Instead of using `ALLOW-FROM` in XFO, developers should switch to using the `frame-ancestors` directive in CSP. These out-of-spec values do not show up often, but may have been set by developers expecting protections to be active due to them setting the header.

Preventing attacks using Cross-Origin policies

Because of the emergence of microarchitectural side-channel attacks like Spectre and Meltdown[390] and Cross-Site Leaks (XS-Leaks)[391], our security perspective relating to use and embeddings of cross-origin resources has changed. In response to these threats, new mechanisms were created to control the rendering of resources on other websites and thereby protect against these new threats.

Multiple new security headers, known as the cross-origin policies, were created as a response to these challenges: Cross-Origin-Resource-Policy (CORP), Cross-Origin-Embedder-Policy (COEP) and Cross-Origin-Opener-Policy (COOP). These headers provide mechanisms that protect against side-channel attacks by allowing developers to control how their resources are embedded across different origins. We observe that the adoption of all of these headers keeps growing year after year, with both CORP and COOP reaching over 2% adoption this year.

[389] https://almanac.httparchive.org/en/2024/security#security-header-adoption
[390] https://spectreattack.com/
[391] https://xsleaks.dev/

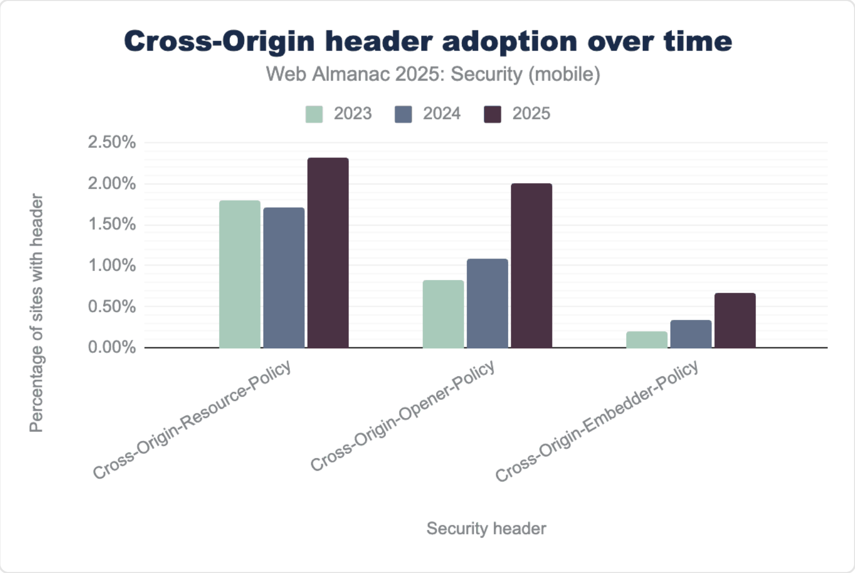


Figure 9.31. Usage of Cross-Origin headers over time.

Cross Origin Embedder Policy

The Cross-Origin-Embedder-Policy (COEP)[392] allows a developer to configure which cross-origin resources can be embedded on the current document. By default (when the header is absent) all cross-origin resources can be embedded on the page, which is the same behaviour as when the header is set with the `unsafe-none` value.

COEP value	Desktop	Mobile
`unsafe-none`	83.26%	86.52%
`require-corp`	6.68%	4.92%
`credentialless`	2.59%	1.89%

Figure 9.32. Prevalence of COEP headers containign a valid varient of the allowed values

Compared to last year[393], most developers still set the COEP header to explicitly allow all content to be embedded onto the current document using the `unsafe-none` value. While the

[392] https://developer.mozilla.org/docs/Web/HTTP/Headers/Cross-Origin-Embedder-Policy
[393] https://almanac.httparchive.org/en/2024/security#cross-origin-embedder-policy

percentage of use of this value is still over 86% on mobile, it has dropped by almost 2% since last year, which can indicate that developers are starting to change their use of the header.

The other values `require-corp` and `credentialless` saw a minor increase of 0.2% and 0.3% respectively in adoption since last year. When using `require-corp`, the browser will enforce that only same-origin content or cross-origin content that is allowed to be embedded by CORP can be embedded onto the page.

For `credentialless`, the browser will allow cross-origin requests in `no-cors` mode regardless of CORS policy of the content, but cookies will not be attached to the request.

Cross Origin Resource Policy

Related to COEP, the Cross-Origin-Resource-Policy (CORP)[394] does not enforce which content can be embedded in the current document, but rather from which documents the current content can be accessed.

The only three possible values are `cross-origin`, `same-origin` and `same-site`. The `cross-origin` value allows any document to access the resource, while the `same-origin` and `same-site` values restrict which documents can access the resource to the documents in the same origin or site respectively. Developers should be aware of the difference between the origin (scheme, host, port) and site (registerable domain)[395]. If the header is present, requests with a mode of `no-cors` will be blocked by the browser.

CORP value	*Desktop*	*Mobile*
cross-origin	*81.36%*	*80.52%*
same-origin	*14.40%*	*15.63%*
same-site	*3.80%*	*3.48%*

Figure 9.33. Prevalence of CORP header values

In most cases, the header is used to allow access to any cross-origin resource. We see a big change in this number this year, dropping more than 10% since last year[396] and landing on 80.5% on mobile. On the other hand, the use of the `same-origin` value went up around 10%, showing that developers are moving to protect their resources against cross-origin access. The share of use for `same-site` remained approximately the same, showing a slight decrease of

[394] *https://developer.mozilla.org/docs/Web/HTTP/Cross-Origin_Resource_Policy*
[395] *https://web.dev/articles/url-parts*
[396] *https://almanac.httparchive.org/en/2024/security#cross-origin-resource-policy*

less than half a percent.

Cross Origin Opener Policy

The final cross-origin policy header, Cross-Origin-Opener-Policy (COOP)[397] allows a developer to control how other pages can reference their page when opening it through browser APIs such as `window.open`.

The default value of `unsafe-none` allows the COOP protection to be disabled, which is also what happens when the header is absent. If a developer uses `window.open` to open a page which uses `unsafe-none`, they can use the returned value to access certain properties of the opened page, which can lead to Cross-Site Leaks.

When `same-origin` is present on both the opener and opened resource, the reference returned by `windows.open` can be used by the opener. The `same-origin-allow-popups` allows a document to open another document with `unsafe-none` while still keeping access to a working reference.

Finally, the `noopener-allow-popups` makes sure the reference is never accessible except for when the opened document also has the same COOP value set.

COOP value	Desktop	Mobile
same-origin	58.22%	61.64%
unsafe-none	28.47%	26.82%
same-origin-allow-popups	11.36%	9.95%
noopener-allow-popups	0.03%	0.03%

Figure 9.34. Prevalence of COOP headers containing a valid variant of the allowed values

The use of the strictest `same-origin` value for COOP has continued to rise from 47.5% to 61.6% on mobile. The `noopener-allow-popups` is a very new value and was not present yet last year[398]. This year we see a small share of adopters using this value. The use of `unsafe-none` had declined by just over 10%. These changes represent a positive evolution in the use of COOP.

[397] https://developer.mozilla.org/docs/Web/HTTP/Headers/Cross-Origin-Opener-Policy
[398] https://almanac.httparchive.org/en/2024/security#cross-origin-opener-policy

Cross-origin isolation

In order to access certain sensitive APIs like `SharedArrayBuffer` or `Performance.now` a site has to be cross-origin isolated[399]. In order to be cross-origin isolated, the developer has to set a COEP of `same-origin` and CORP of either `require-corp` or `credentialless`. The browser will then allow access to these APIs again. This strengthens the protection against XS Leaks. These days, developers can opt into cross-origin isolation using the document isolation policy[400] as well.

Preventing attacks using `Clear-Site-Data`

Using the `Clear-Site-Data` HTTP response header, developers are able to instruct the client to clear browsing data. The value of the header specifies which type or types of data should be cleared. This can be useful when a user logs out of the website, so the developer can be sure that any authentication cookies are most assuredly cleared.

It is difficult to estimate the adoption of `Clear-Site-Data` correctly as its use is usually most valuable when logging out users. The crawler we use does not log into websites and therefore can also not log out to check how many sites use the header after logout. For now, we see 2,024 mobile hosts using the `Clear-Site-Data` header, which is only 0.01% of the total number of hosts crawled.

[399] *https://developer.mozilla.org/docs/Web/API/Window/crossOriginIsolated*
[400] *https://wicg.github.io/document-isolation-policy/*

Clear site data value	Desktop	Mobile
`cache`	30.82%	29.25%
`*`	17.74%	20.61%
`cookies`	7.02%	8.16%
`"cache"`	8.94%	7.19%
`"storages"`	5.66%	6.08%
`cache, cookies, storage`	2.12%	3.23%
`"cache", "cookies", "storage", "executionContexts"`	2.32%	2.51%
`"cookies"`	2.78%	2.46%
`"storage"`	2.27%	2.03%
`"cache", "storage", "executionContexts"`	1.36%	1.54%

Figure 9.35. Prevalence of `Clear-Site-Data` headers

Our data shows that most developers attempt to use the `Clear-Site-Data` header for clearing the cache. The most prevalent value is `cache`, followed by the wildcard character `*` and `cookies`. All values from the top three are invalid according to the specification. The value of this header must be a 'quoted string', which means `"cache"`, `"*"` and `"cookies"` are the equivalent valid values. This is concerning as the top three values combined already represent 58.01% of current header values on mobile.

We see these numbers jump quite a lot year after year, which can likely be explained by the low adoption of the header, where a very small number of hosts can change the relative fraction by a large amount.

Preventing attacks using `<meta>`

Besides being set via response headers, some of the security mechanisms of the web can be configured directly within the html document through the use of the `<meta>` tag. Two examples of this are the `Content-Security-Policy` and the `Referrer-Policy`. The use of a meta tag for these mechanisms has remained largely stable from last year[401], at around

[401] https://almanac.httparchive.org/en/2024/security#preventing-attacks-using-meta

0.60% and around 2.50% for the CSP and Referrer-Policy respectively. A very small decrease in CSP and very small increase in Referrer-Policy could be observed, just like last year.

Meta tag	***Desktop***	***Mobile***
`includes Referrer-policy`	*2.75%*	*2.52%*
`includes CSP`	*0.64%*	*0.59%*
`includes not-allowed policy`	*0.12%*	*0.11%*

Figure 9.36. The percentage of hosts enabling different policies using a meta tag

Other security mechanisms cannot be configured through the use of the `<meta>` tag, however every year we see developers still attempt this. This year we even see a rise in policies that are not allowed to be configured using a meta tag from 0.07% to 0.11% on mobile. These values are ignored by the browser, thus potentially leaving users vulnerable if the correct header is not configured. Keeping up with our running example, this year we found 5,564 meta tags that included the `X-Frame-Options` policy. This is almost 600 pages more than last year, which is a worrying trend.

Web Cryptography API

The Web Cryptography API[402] is a JavaScript API that provides an interface for performing basic cryptographic operations. Examples of such operations are the creation of random numbers, hashing, signing content, verifying signatures and of course encryption and decryption.

[402] *https://www.w3.org/TR/WebCryptoAPI/*

Feature	Desktop	Mobile
`CryptoGetRandomValues`	34.45%	40.95%
`SubtleCryptoDigest`	2.65%	2.98%
`CryptoAlgorithmSha256`	2.36%	2.48%
`SubtleCryptoImportKey`	1.29%	1.68%
`CryptoAlgorithmEcdh`	0.97%	1.39%
`CryptoAlgorithmSha512`	0.17%	0.32%
`CryptoAlgorithmSha1`	0.21%	0.26%
`CryptoAlgorithmAesCbc`	0.21%	0.17%
`SubtleCryptoSign`	0.13%	0.14%
`SubtleCryptoEncrypt`	0.13%	0.12%
`CryptoAlgorithmHmac`	0.10%	0.11%

Figure 9.37. The usages of features of the Web Cryptography API

The `CryptoGetRandomValues` function remains the most widely used feature of this API, however it is still declining in use, just like it was last year[403]. Its use on mobile dropped by more than 12% this year, landing just under 41%. The other features continue to rise, with the second most popular feature `SubtleCryptoDigest` growing by 1.2% to just under 3%.

Bot protection services

Bots have been present on the web for a long time, and malicious bots are a large part of that. Because of these issues, many products have been created by different vendors to protect websites against bots. The use of these products continues to rise year after year, including this year, where we see a jump from 26.5% to 31.1% adoption on mobile, a rise of over 4.5%.

[403] *https://almanac.httparchive.org/en/2024/security#web-cryptography-api*

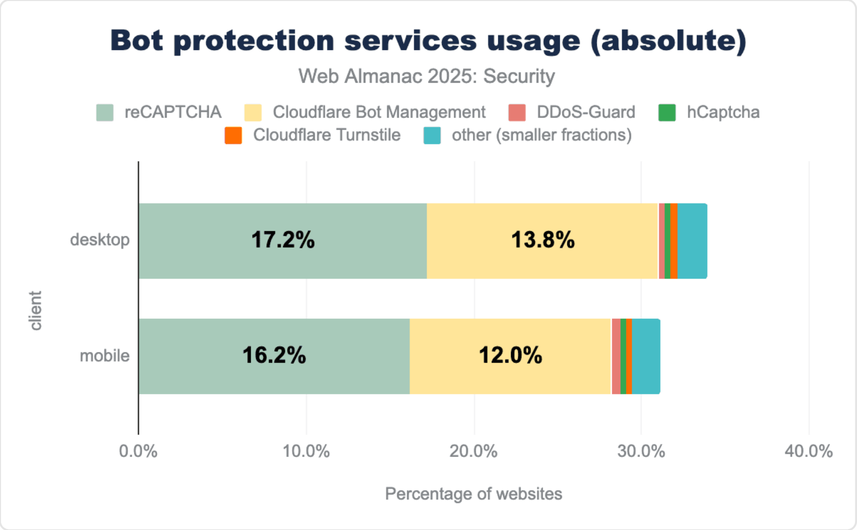


Figure 9.38. The absolute distribution of bot protection services in use.

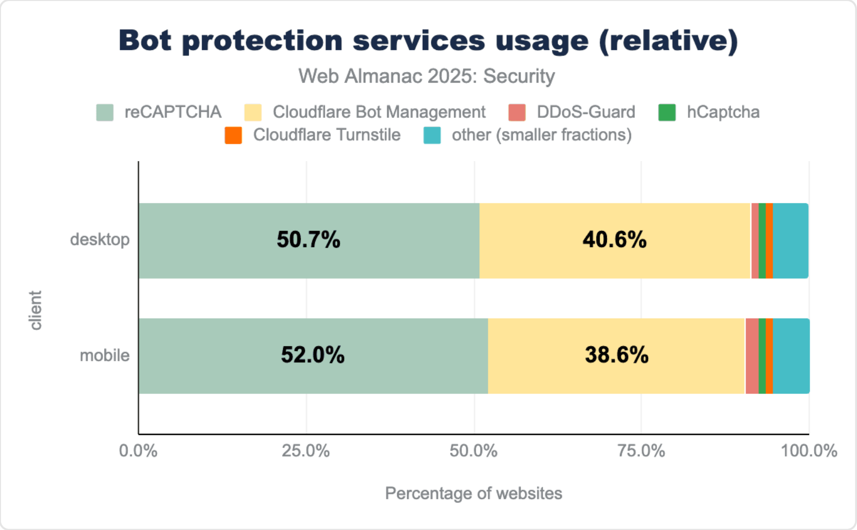


Figure 9.39. The relative distribution of bot protection services in use.

We see that reCAPTCHA remains the largest bot protection service, but Cloudflare Bot Management is growing more rapidly, thereby taking up a larger relative share of websites using bot protection services. In case these trends continue over the next year, we may see Cloudflare Bot Management closing the gap on reCAPTCHA.

HTML sanitization

With the `SetHTMLUnsafe` and `ParseHTMLUnsafe` APIs, two relatively recent additions to browsers, developers can use a declarative shadow DOM from JavaScript[404].

When a developer attempts to use `innerHTML` to place a custom HTML component that includes a definition for a declarative shadow DOM onto the page (for example `<template shadowrootmode="open">...</template>`), this will not work as expected. By using `SetHTMLUnsafe` or `ParseHTMLUnsafe` the developer can ensure that the declarative shadow DOM is properly instantiated by the browser.

As the name implies, the developer is responsible for making sure that only safe values are passed to these 'unsafe' APIs. In other cases, the developer runs the risk of allowing dangerous content to end up on the page, which can lead to XSS attacks.

Feature	Desktop	Mobile
`ParseHTMLUnsafe`	*19869*	*17147*
`SetHTMLUnsafe`	*443*	*449*

Figure 9.40. The number of pages using HTML sanitization APIs

Since last year[405], we see a big rise in the use of these APIs. On mobile, the number of pages using `SetHTMLUnsafe` rose from 2 to 449 pages and the number using `ParseHTMLUnsafe` rose from 6 to 17,147 this year. The latter still only accounts for 0.06% of the crawled pages, but it is an interesting change and we can expect the adoption to keep rising in the following year. However, it is not expected that these APIs will gain widespread adoption anytime soon.

Drivers of security mechanism adoption

There may be various reasons that web developers choose to adopt more security practices. Some of the most noteworthy are:

- **Geographical**: depending on the region, there may be more security-oriented education or knowledge, or in some cases there can be local laws that mandate stricter security hygiene.
- **Technological**: the technology stack in use can influence the adoption of security

[404] *https://developer.chrome.com/blog/new-in-chrome-124#dsd*
[405] *https://almanac.httparchive.org/en/2024/security#html-sanitization*

mechanisms. Depending on the technology in use, some security features will be enabled out of the box without the developer having to think about enabling them.

- **Popularity**: very popular websites may have larger budgets for security, in part because they are more likely to be the target of certain cyberattacks. In addition, these sites are likely to attract more security professionals and in some cases bug bounty hunters to help them implement security features and strengthen their defenses.

Location of websites

The location where a website is hosted or where its developers are based can have a big impact on the adoption of certain security features. Security education for developers plays a big role, as developers cannot implement features of which they don't know they exist or which they don't understand. In addition, local laws can sometimes require the adoption of certain security practices.

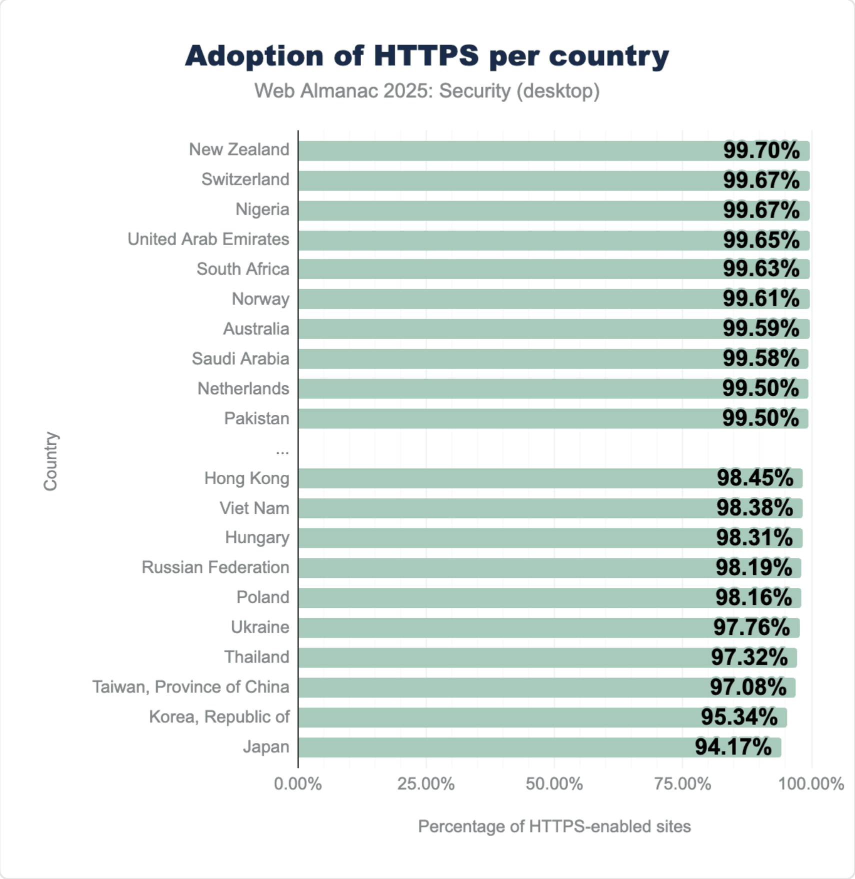


Figure 9.41. The adoption of HTTPS per country; top and bottom 10 countries by adoption.

HTTPS adoption has been increasing year after year, a trend that luckily continues this year as well. We see even adoption in the top countries continued to increase by a few tenths of a percentage. The bottom countries saw slightly larger rises in HTTPS adoption, with Japan now being the only country with adoption under 95% and only five countries having less than 98% adoption of HTTPS, an extremely good result. In the following years, we can expect the bottom countries to slowly catch up with the rate of adoption of the top countries, although a full 100% adoption seems unlikely in the near future.

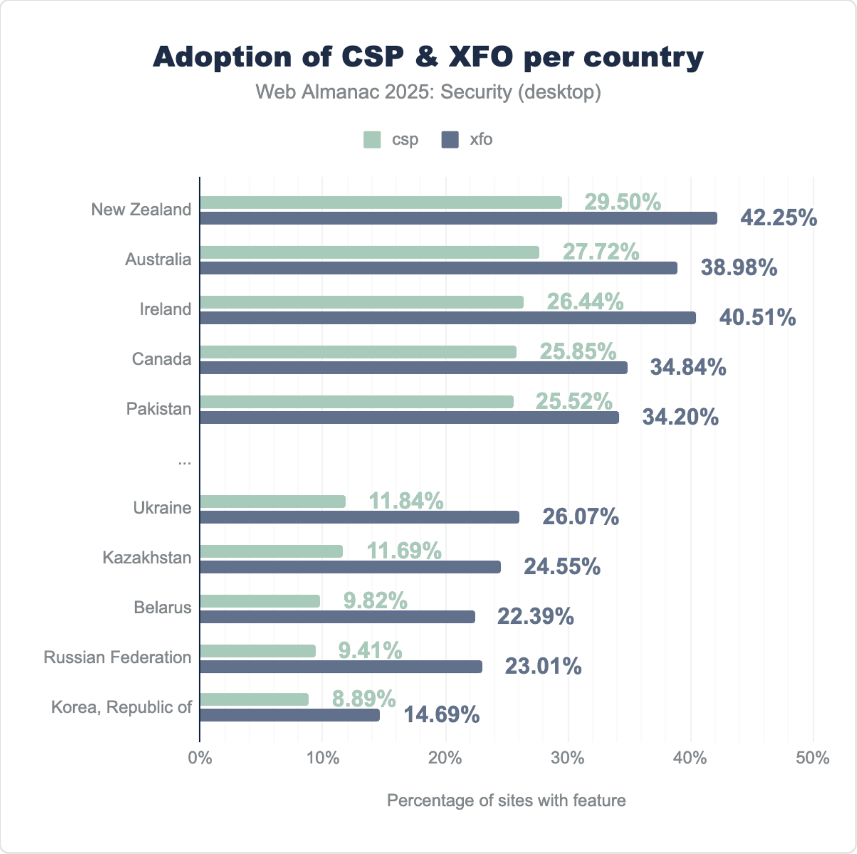


Figure 9.42. The adoption of CSP and XFO per country; top 5 and bottom 5 countries by CSP adoption.

We see a more varied picture when looking at more complex security mechanisms. With a small rise in adoption for most leading countries, some of the bottom countries saw a decline in adoption of these policies. The leading countries still have CSP configured on just over a quarter of the websites. The gap between CSP and XFO remains large, although it has gone up slightly, reaching only up to 14% instead of 15% last year[406].

Technology stack

A website's security can vary depending on the technology in use. Because frameworks include security features by default and it is in the best interest of large vendors to keep their users

[406] https://almanac.httparchive.org/en/2024/security#location-of-website

secured, using these underlying technologies may boost a website's security.

Technology	***Security features***
LiveJournal	*`Content-Security-Policy` (99.99%), `Permissions-Policy` (99.99%), `Referrer-Policy` (99.99%)*
Weblium	*`X-Content-Type-Options` (97.31%), `X-XSS-Protection` (97.31%)*
GoDaddy Website Builder	*`Strict-Transport-Security` (95.97%), `Content-Security-Policy` (95.97%)*
Sitevision CMS	*`X-Frame-Options` (81.54%)*
Microsoft Sharepoint	*`X-Content-Type-Options` (57.44%)*
Liferay	*`X-Content-Type-Options` (52.65%)*

Figure 9.43. Security features in use by selected CMS systems.

We see that a number of blogging websites and website builders have some important security mechanisms configured almost throughout all of their systems, with HSTS, CSP and `X-Content-Type-Options` being some of the most popular ones.

Website popularity

Popular websites with a large user base often have a good reason to protect their users to the best of their abilities so they will not lose users and their trust. Protecting the often sensitive data they store while they likely are the target of more directed attacks requires significant investments in securing their website, but will also lead to a more generally secure website as a trade-off.

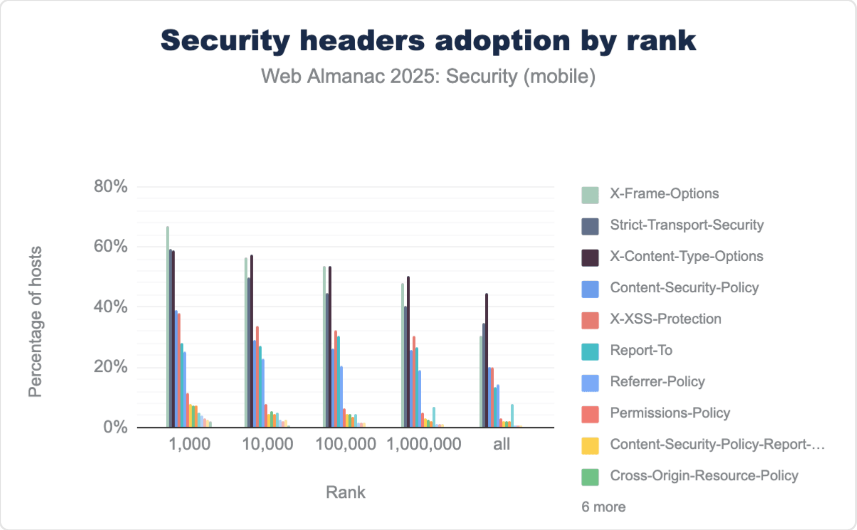


Figure 9.44. Security header adoption by website rank according to CrUX.

We find that the most popular security headers like `X-Frame-Options`, `Strict-Transport-Security`, `X-Content-Type-Options` and `Content-Security-Policy` always have a higher adoption among more popular websites on mobile. The most widely adopted header, `X-Frame-Options` is used on 67% of the top 1,000 sites yet only on 30% of all sites visited by the browser. We see the gap between the adoption in more popular sites and less popular sites remains practically identical since last year[407].

Website category

Depending on the industry, more importance may be attributed to keeping a website more secure. We attempt to estimate the efforts in securing websites per industry based on the average number of security headers set by websites. While the number of security headers does not necessarily indicate whether a website is better secured (after all, security mechanisms can be misconfigured), it provides a good estimate for the effort taken to implement security features.

[407] *https://almanac.httparchive.org/en/2024/security#website-popularity*

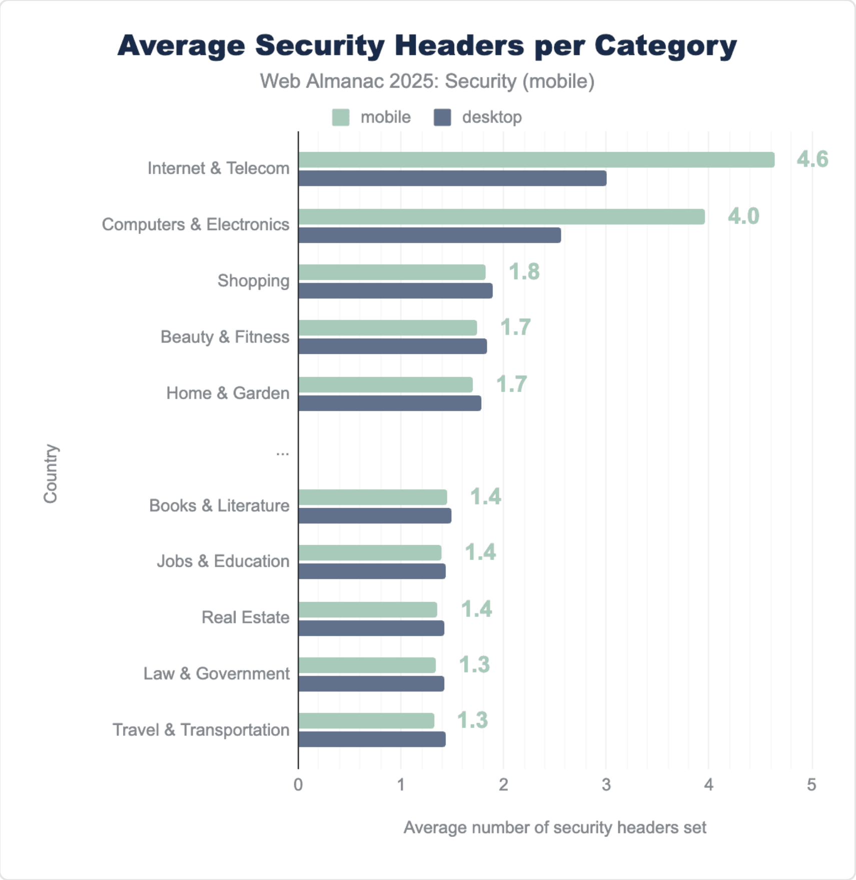


Figure 9.45. The average number of security headers by website category; top 5 and bottom 5 categories.

We see a big difference occurring in the average number of security headers since last year. Sites in the *Internet & Telecom* and *Computers & Electronics* categories use significantly more security headers. Especially on mobile clients the difference with these categories is clearly visible. While these two categories seem to be outliers in the good sense, the average number of security headers over other industries has remained largely the same, with a very minor change of about 0.1 header per site on average.

The two leading industries in terms of security headers happen to be industries related to the field of internet and computer security. It is possible that due to the relevance of security in these industries, developers of these sites are more aware of the potential risk and therefore more willing to use certain security mechanisms.

Malpractices on the web

Cryptocurrencies are more popular than they have ever been. Cryptomining has been a large business for a number of years and adversaries have been known to install cryptominers as a form of malware on victim websites. Over the past years, the use of cryptominers on the web has been steadily declining.

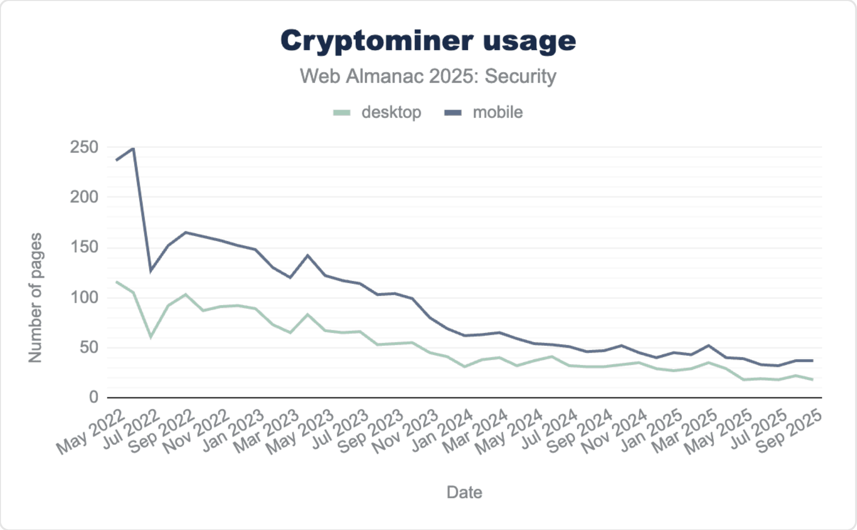


Figure 9.46. The number of cryptominers in use over time; from May 2022 to Sep 2025.

We see the number of pages with cryptominers decreased to only 37 pages on mobile, a 42% decline since last year[408]. It is also a 83% decline since September 2022, only three years earlier.

[408] https://almanac.httparchive.org/en/2024/security#fig-47

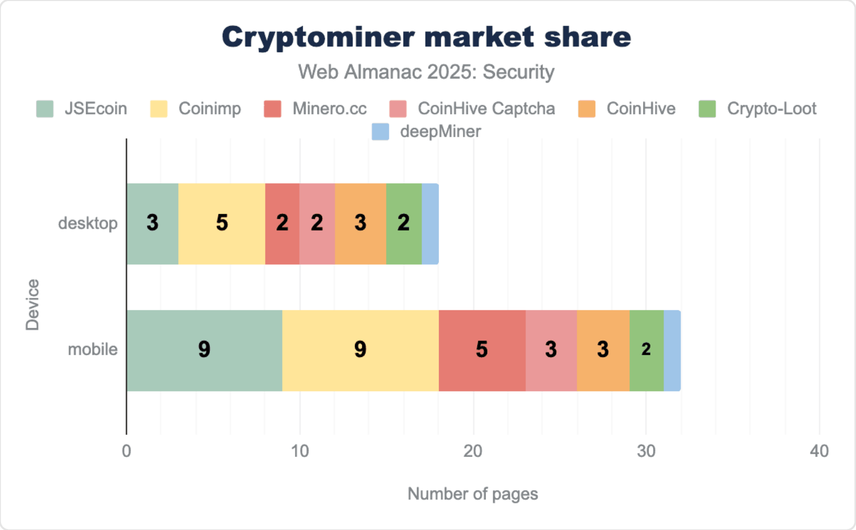


Figure 9.47. The cryptominer market shares.

Although we find a very low absolute number of cryptominers on the web, we still take a look at the shares that different cryptominers represent. Compared to last year[409], we see the number of pages with Coinimp has dropped to match the nine pages that have JSEcoin. An interesting note to make is the difference between the number of cryptominers on desktop and mobile pages, with the number of pages on mobile being almost double the number on desktop.

Security misconfigurations and oversights

While there are many security mechanisms available on the web and in browsers, it is vital to configure these mechanisms correctly and as expected. Misconfigured security mechanisms create a false sense of security for the developer that thinks their users are protected. In this section we highlight the occurrence of several misconfigurations that can compromise a website's security.

Unsupported policies in `<meta>`

When configuring security policies, it is important that developers understand how they have to define the policy. Some policies can be defined through both a header and the HTML `<meta>` tag. However, many policies cannot be defined through `<meta>`. Developers

[409] *https://almanac.httparchive.org/en/2024/security#fig-48*

sometimes make the mistake of trying to configure these policies through the `<meta>` tag anyway. Unfortunately browsers will ignore these policies, resulting in inactive security policies.

Policy	Desktop	Mobile
`X-Frame-Options`	4,584	5,564
`X-Content-Type-Options`	2,440	2,854
`Permissions-Policy`	1,983	2,236
`X-XSS-Protection`	1,691	1,702
`Referrer-Policy`	1,630	1,644

Figure 9.48. Top 5 security policies mistakenly configured through `<neta>`

We find that around 0.11% of mobile sites attempt to set security headers that browsers explicitly do not support via `<meta>` tags. The most frequently attempted policies are `X-Frame-Options`, `X-Content-Type-Options` and `Permissions-Policy`. Compared to 2024, we see that the number of mobile sites setting these policies in `<meta>` rose in absolute numbers by more than 5,000 pages, showing that these misconfigurations are still actively being set by developers.

Unsupported CSP directives in `<meta>`

While CSP *can* be configured via `<meta>` tags and this behaviour has been observed on 0.59% of mobile pages, certain CSP directives are explicitly disallowed in `<meta>` tag implementations and will be ignored by browsers. These directives can only be set by using the `Content-Security-Policy` response header.

Directive	Desktop	Mobile
`frame-ancestors`	2.37%	2.11%
`sandbox`	0.004%	0.003%

Figure 9.49. Percentage of CSP policies defined in `<meta>` with disallowed directives

We find that over 2% of mobile pages setting a CSP policy via a `<meta>` tag include the `frame-ancestors` directive and 0.003% include the `sandbox` directive. The latter boils down to only three pages out of the entire crawled dataset. Compared to last edition, the misconfiguration of `frame-ancestors` shows up on 600 more pages, thereby rising by over 0.8%. This represents a slow but negative evolution for these types of misconfigurations.

COOP/COEP/CORP confusion

Because the cross-origin policies COEP, CORP, and COOP have a similar naming and are related in purpose, they are sometimes confused by developers. Assigning the wrong values to these headers has the effect that the browser will apply the default policy for the header as if no header was supplied at all, thereby disabling additional defenses wanted by the developer.

Invalid COEP value	*Desktop*	*Mobile*
`same-origin`	4.43%	4.02%
`cross-origin`	0.55%	0.46%
`*`	0.13%	0.11%
`(unsafe-none|require-corp); report-to='default'`	0.09%	0.11%
`: require-corp`	0.09%	0.10%

Figure 9.50. Prevalence of invalid COEP header values

In total, 5.6% of the observed COEP headers contain an invalid value on mobile. Over 4% of these headers contain the `same-origin` value that is only a valid value for the CORP or COOP headers. Another almost 0.5% contain `cross-origin`, a value destined for the CORP header. Unfortunately, these misconfigurations also saw a rise since last year[410], by almost 1% for the `same-origin` value in the COEP header.

In addition to these misconfigurations, we also observed several values with syntactical errors that the browser will not be able to parse and therefore will also revert to the default value for the header. These syntactical errors represent a minority of cases.

Timing-Allow-Origin Wildcards

The `Timing-Allow-Origin` (TAO) response header allows a server to specify a list of origins that can access values of attributes obtained through features of the Resource Timing API[411].

[410] https://almanac.httparchive.org/en/2024/security#coep-corp-and-coop-confusion
[411] https://developer.mozilla.org/docs/Web/API/Performance_API/Resource_timing

Any origin listed in this header can thus access detailed timestamps relating to the connection to the server, such as the time at the start of the TCP connection, start of the request, and start of the response. Granting origins this access should be done with care, as this opens up the possibility for the listed origins to execute timing attacks or other cross-site attacks against the website.

In a case when CORS is configured, many of these types of timing values (including those listed above) are returned as 0 to prevent cross-origin leaks. By listing an origin in the `Timing-Allow-Origin` header, these values retain their original value and are no longer zeroed out.

84%

Figure 9.51. The percentage of `Timing-Allow-Origin` headers that are set to the wildcard (`*`) value.

Besides returning a list of origins, developers can also set the `Timing-Allowed-Origin` header's value to the wildcard character `*` to indicate that any origin may access the timing information. We find that 84.6% of the TAO headers contain the wildcard `*` value, rising by 2% since last year[412]. This indicates that many developers have no problem with universally exposing finegrained timing information to any origin.

Missing suppression of server information headers

When websites publish excessive information about their infrastructure, such as the server and specific version thereof, they may run a higher risk of being targeted by automatic vulnerability scanners. Hiding this information is a form of security by obscurity which is a tactic that is generally frowned upon because it does not address the core vulnerability of a system, however because it may aid in remaining under the radar of certain attacks we include an analysis.

We track the use of headers that are commonly used to report this type of information, namely: `Server`, `X-Server`, `X-Backend-Server`, `X-Powered-By`, and `X-Aspnet-Version`.

[412] https://almanac.httparchive.org/en/2024/security#timing-allow-origin-wildcards

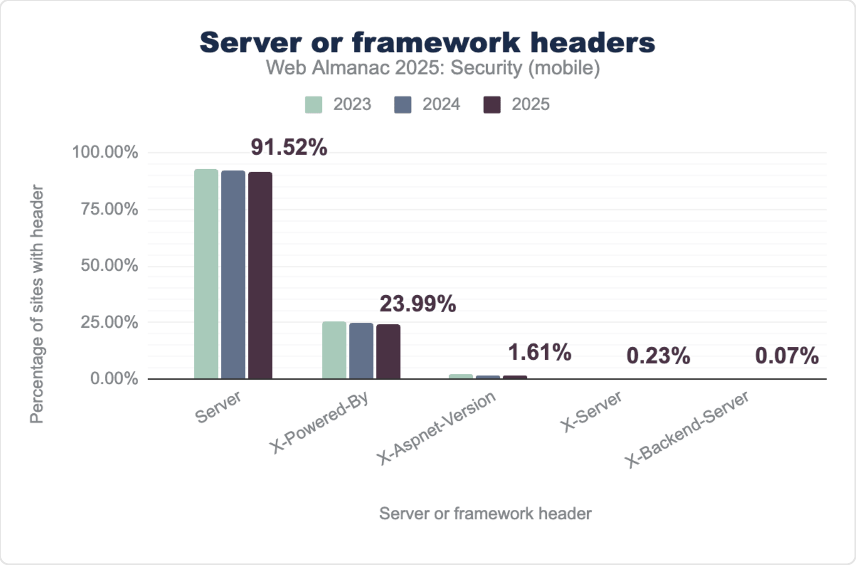
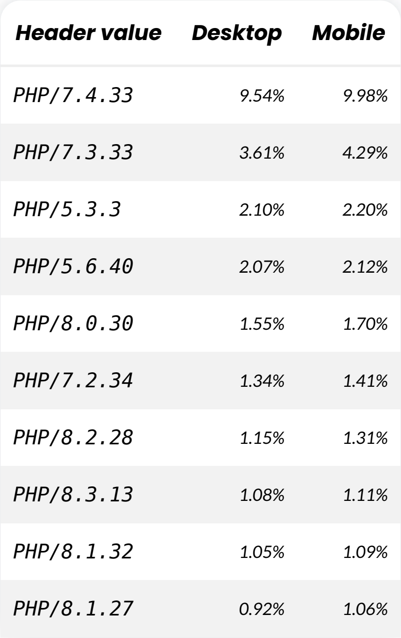


Figure 9.52. Prevalence of headers used to convey information about the server.

Just like the past years, the `Server` header remains the most widely used header by a large margin over `X-Powered-By`. These headers show up on 91.5% and 23.9% of hosts on the web. For each of the headers we see a slight decrease in the amount of hosts they show up on. We don't expect these values to see major changes over time, as many web technologies automatically set some of these headers and developers may not have a lot of interest in removing these headers as the gains in terms of security are small.



Header value	Desktop	Mobile
`PHP/7.4.33`	9.54%	9.98%
`PHP/7.3.33`	3.61%	4.29%
`PHP/5.3.3`	2.10%	2.20%
`PHP/5.6.40`	2.07%	2.12%
`PHP/8.0.30`	1.55%	1.70%
`PHP/7.2.34`	1.34%	1.41%
`PHP/8.2.28`	1.15%	1.31%
`PHP/8.3.13`	1.08%	1.11%
`PHP/8.1.32`	1.05%	1.09%
`PHP/8.1.27`	0.92%	1.06%

Figure 9.53. Most prevalent `X-Powered-By` header values with specific framework version

If we look at the most common values within the `Server` and `X-Powered-By` values, we see that PHP has the tendency to expose the exact version running on the server particularly in the `X-Powered-By` header. For both desktop and mobile, we find that over 27% of the `X-Powered-By` headers contain version information. It is likely that the header is automatically returned by the platforms that we observe in our data. Interestingly, we see a slight decrease in the old PHP versions 7.x and lower and a slight increase in the new PHP versions 8.x, which is an indication that at least some developers are updating their servers.

Missing suppression of `Server-Timing` header

The `Server-Timing` HTTP header is a response header defined in a W3C Editor's Draft[413] which can be used to communicate server performance metrics. Developers can send metrics containing zero or more properties. One of the specified properties is the `dur` property, which can be used to communicate millisecond-accurate timings that contain the duration of specific actions on the server.

[413] https://w3c.github.io/server-timing/

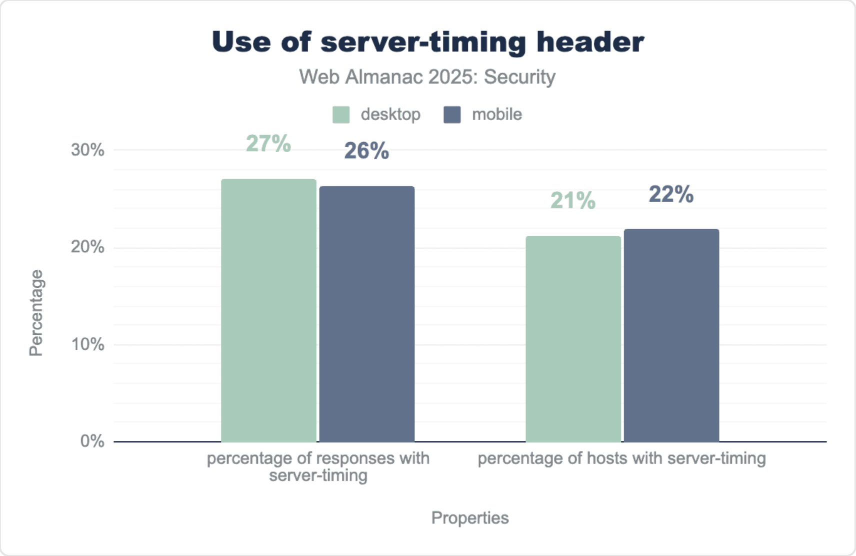


Figure 9.54. The usage of the `Server-Timing` header.

The percentage of hosts returning a `server-timing` header has increased by over 15% in use by over a fifth of hosts that have been visited by the crawler. This is a very steep increase since last year[414].

[414] https://almanac.httparchive.org/en/2024/security#missing-suppression-of-server-timing-header

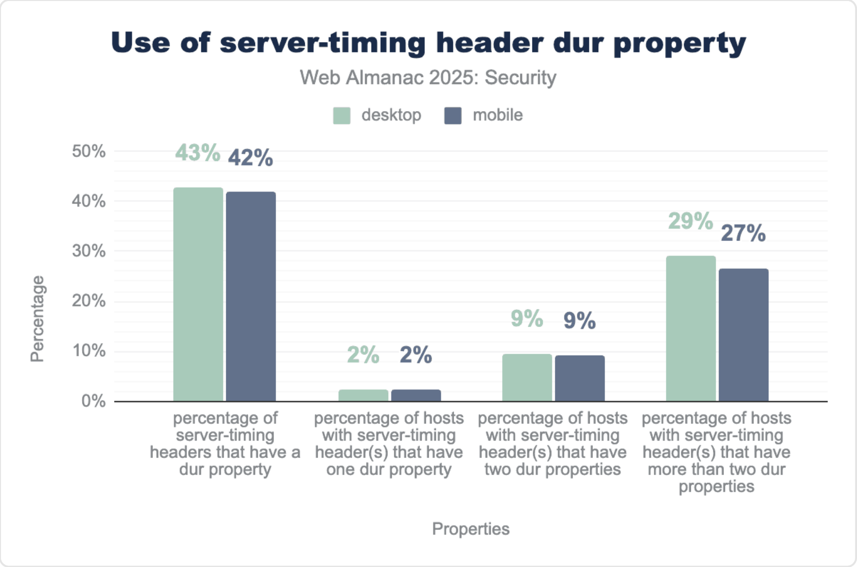


Figure 9.55. The relative usage of `dur` properties in `Server-Timing` headers.

Of the `server-timing` headers on the web, we find 42% of them have at least one `dur` property. This is a relative decrease compared to last year[415], but given the steep incline in header use over the year, the absolute number has risen. This also means that more headers do not include a `dur` property and use the header for other purposes, possibly through the use of the `desc` property that allows developers to set a description for certain metrics.

Because the information included in the `server-timing` header can be sensitive, access to the values is restricted to the same origin and to origins listed in the `Timing-Allow-Origin` header. As we have shown above, many websites configure the `Timing-Allow-Origin` with a wildcard character, allowing all origins to access this potentially sensitive information. Even without cross-origin access, timing attacks can still be executed directly against servers exposing sensitive timing information outside of the browser context.

Well-known URIs

Well-known URIs provide a standardized mechanism for designating specific locations for site-wide metadata and services. Defined by RFC 8615[416], a well-known URI is identified by a path component that begins with the prefix `/.well-known/`. This allows clients to discover

[415] https://almanac.httparchive.org/en/2024/security#fig-53
[416] https://www.rfc-editor.org/rfc/pdfrfc/rfc8615.txt.pdf

specific resources without needing prior knowledge about a site's URL scheme.

`security.txt`

The well-known `security.txt` file is a standardized file format that websites use to communicate vulnerability reporting information. White hat hackers and penetration testers can use this file to find contact details, PGP keys, policies, and other information for responsible disclosure.

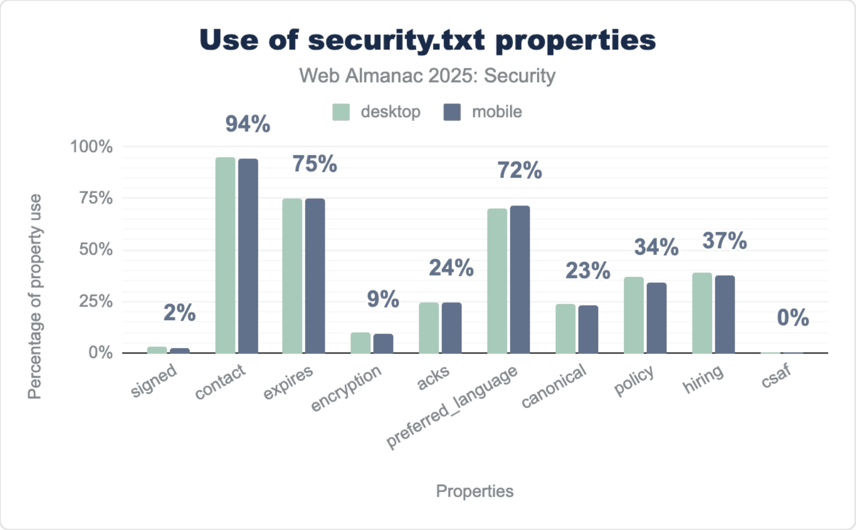


Figure 9.56. The usage of `security.txt` properties.

Adoption has increased to 1.82% of desktop and 1.72% of mobile websites, both up from 1% in 2024, showing growing recognition of standardized security disclosure practices.

Among sites implementing `security.txt`, contact information remains nearly universal at 95% (desktop) and 94% (mobile), up from 92% and 89% in 2024, respectively. Interestingly, 75% now define an expiry date, a significant jump from 51% for desktop and 48% for mobile websites in 2024. Preferred language is specified by 70-72% of implementations, while policy (which defines vulnerability reporting procedures) appears in only 37% desktop and 34% mobile files, down from 39% in 2024. However, the absolute number of `security.txt` files defining a policy has risen by two thirds.

This analysis shows that at least 25% of `security.txt` files are not fully valid, because including an expiry date along with contact information is required, as stipulated in RFC 9116[417].

`change-password`

The `change-password` well-know URI is a W3C specification draft from 2021, which has not been updated since. The purpose of the URI is for users and external resources to quickly find the correct location at which they can change their passwords for the specific site.

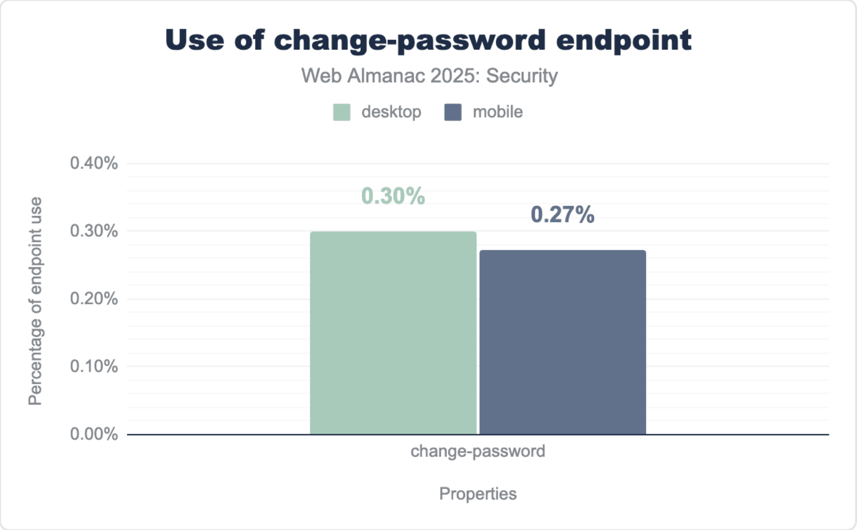


Figure 9.57. The usage of the change-password .well-known endpoint.

We see a very minor rise to 0.30% in desktop sites (up from 0.27%) and no change for the mobile endpoints, which remain at 0.27%. This slow adoption is not unexpected, especially taking into account that not all websites require authentication mechanisms.

Detecting status code reliability

The specification[418] draft to check on the reliability of a website's HTTP response status code also remains unchanged since 2021. The purpose of this specific well-known endpoint is that it should never exist and thus the response status code should never be an ok status[419]. If a redirect occurs after which the site responds with an ok status, this would be considered as incorrect behavior.

[417] *https://www.rfc-editor.org/rfc/rfc9116.txt*
[418] *https://w3c.github.io/webappsec-change-password-url/response-code-reliability.html*
[419] *https://fetch.spec.whatwg.org/#ok-status*

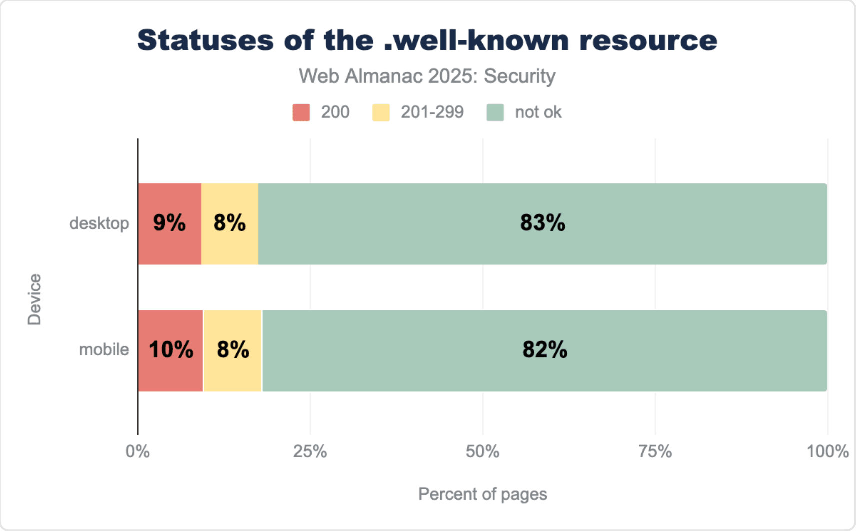


Figure 9.58. The distribution of statuses returned for the `.well-known` endpoint to assess status code reliability.

We find similar results to 2022's[420] and 2024's[421] edition of the Web Almanac. However, the number of faulty ok status responses has grown slightly, from 84% in 2024 for both desktop and mobile pages to 83% and 84%, respectively. Web developers should continue to make their application use the correct status codes to respond to incoming requests in order to avoid these status codes from losing meaning.

Sensitive endpoints in `robots.txt`

Finally, we check whether sensitive endpoints are disallowed to be visited by crawlers in the well-known `robots.txt` file. By checking the disallowed endpoints in this file, attackers may find pages to target. This year, 81% of desktop sites and 80% of mobiles sites hosted a `robots.txt` file, which is very similar to last year's edition of the Web Almanac[422].

[420] https://almanac.httparchive.org/en/2022/security#change-password
[421] https://almanac.httparchive.org/en/2024/security#change-password
[422] https://almanac.httparchive.org/en/2024/security#sensitive-endpoints-in-robotstxt

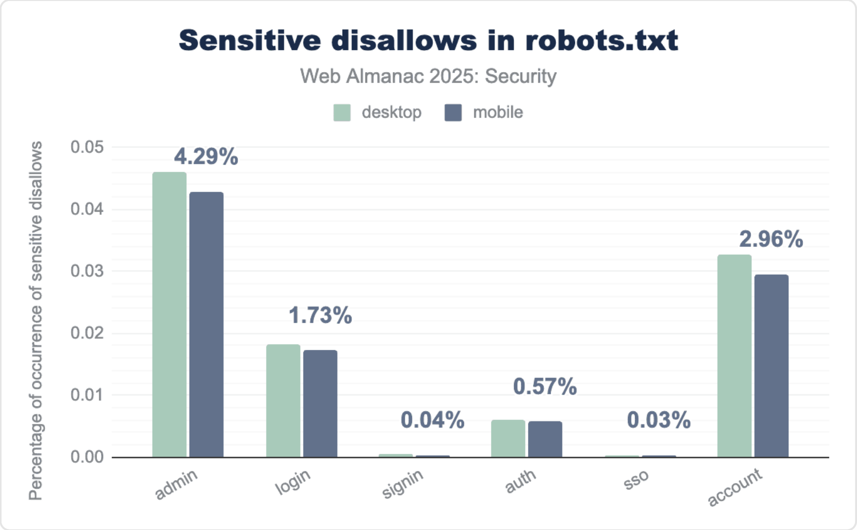


Figure 9.59. The percentage of sites including specified endpoints in their `robots.txt`.

Also the contents of those files remain very similar. The largest increase is recorded for the `auth` endpoint with 0.2% for desktop sites, while the largest decrease was recorded for the `login` endpoint with 0.2% for desktop sites.

Conclusion

This security chapter shows positive trends in the adoption of web security policies. HTTPS is reaching near-100% adoption overall, and per-country metrics show every country is moving towards the goal of a universal use of HTTPS. We saw growing adoption of many modern security policies aiming to better protect users against modern attacks such as the `Content-Security-Policy` which saw an increase in use by over 18% and the `Permissions-Policy` which was used 50% more than last year. We also see newer policies like the Document Policy appear in the wild, showing that developers are actively working on adoption of new security features.

Despite these positive trends, developers must remain vigilant when leveraging these security mechanisms. Due to the growing complexity of the many available security mechanisms, we saw growth in the number of misconfigurations on the web. We saw that 0.1% of pages configure security policies in the `<meta>` HTML tag while this is not supported by browsers. Another problem is the confusion between related protections: 5% of values of the COEP header are invalid values that are only valid in the related CORP or COOP header. We also observe a form of developer fatigue where the least strict value of a protection is configured in order to make deployment more manageable or prevent potential problems, such as the wildcard value in the `Timing-Allow-Origin` header showing up in over 84% of these headers. Luckily, developers can easily mitigate these issues once they are aware of the problems.

New attacks in the future will inevitably drive the design of even more protection mechanisms to keep users safe worldwide. Policy makers will have to focus on reducing complexity in these new mechanisms to avoid developer confusion, but while the adoption of new security features takes time, we see relatively new policies being picked up and getting more adoption over time, thereby creating a more secure web for everyone.

Authors

Vik Vanderlinden

𝕏 @vikvanderlinden   vikvanderlinden   vikvanderlinden   https://vikvanderlinden.be/

Vik Vanderlinden is a PhD candidate in Computer Science at the DistriNet Research Group[423] at KU Leuven. His research focuses on web and network security, primarily focusing on timing leaks in web applications and protocols.

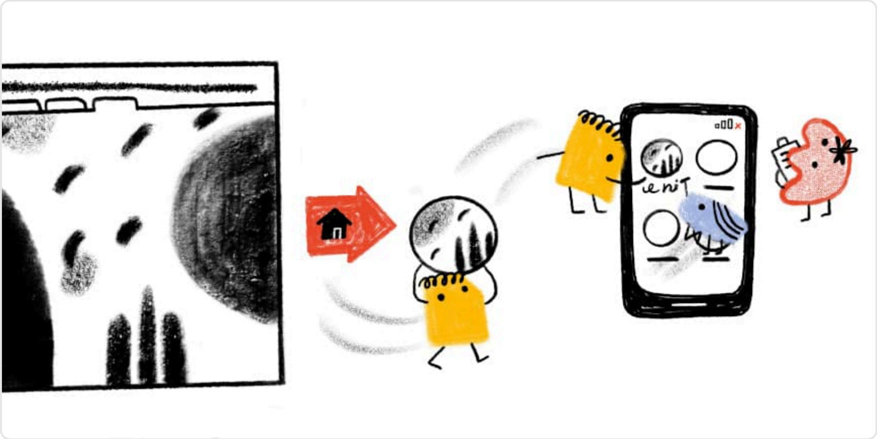
Gertjan Franken

𝕏 @GJFR_   GJFR   gertjan-franken   https://gjfr.dev

Gertjan Franken is a postdoctoral researcher with the DistriNet Research Group[424] at KU Leuven. His research spans various aspects of web security and privacy, with a primary focus on the automated analysis of browser security policies. As part of this research, he maintains the open-source tool BugHog[425] for pinpointing bug lifecycles.

[423] *https://distrinet.cs.kuleuven.be/*
[424] *https://distrinet.cs.kuleuven.be/*
[425] *https://github.com/DistriNet/BugHog*

Part II Chapter 10
PWA



Written by Diego Gonzalez and Michael Solati
Reviewed by Maxim Salnikov, Kai Hollberg, and Aaron Gustafson
Analyzed by Onur Güler
Edited by Barry Pollard

Introduction

The year 2015 was the first time we read about Progressive Web Applications[426]. Nine attributes—responsive, connectivity independent, app-like-interactions, fresh, safe, discoverable, re-engageable, installable and linkable—were what defined the cutting edge of what could be achieved with web technologies back then. *The PWA concept was coined 10 years ago*, and it is with great pride that we look at the state of this set of technologies, one decade after its ideation.

The concept of a PWA has evolved a lot in these 10 years, and different browsers support it in different variations and with different names. An idea that started as a way of enabling access to a web application via "Add to home screen" on mobile browsers is now present on multiple platforms and devices. This set of technologies allow to integrate web content directly into the underlying platform, enabling at the same time advanced capabilities and a more native look

[426] https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/

and feel. Let's dive into what the last couple of years have brought to PWAs.

Changes to PWA/web apps

The last couple of years have seen new features coming to web apps that enable more customization, advanced control of the application's behavior and better performance. But above all, there has been progress in supporting web apps (to an extent or another) on multiple engines!

Chromium based browsers prompt for an application installation once a minimum set of requirements is identified. This used to be the case on apps that had a manifest file, a service worker and were served over a secure connection. For a while this was the "trifecta" for PWA installability. This has changed and nowadays only the manifest is required (the HTTPS connection is still there, do not worry); service workers are no longer required for browsers like Edge and Chrome to display the installation prompt.

On the other hand, Safari does not prompt for web app installation. It does however allow any web page to be installed as an application by adding it to the dock[427] on macOS 14.

Another great news for PWAs is that Firefox now supports web apps[428] from version 143! It is available for PWAs on Windows only, following what has been a "top request from the Mozilla Connect community"[429].

With web apps, being *web*-based, one of the concerns of not having a service worker is offline support. It is true using service workers allow developers to provide a great offline experience by caching resources that make up the UI of the web app. The change in requirements (for Chromium) or default behaviors (for Safari and Firefox) means that the browser may create a default offline experience that shields users from lack of connectivity.

Bear in mind that service workers are still required for the best offline UX. For an up-to-date summary of web app installability, see this MDN article[430].

With this summary, we are now ready to jump into the data and understand the state of PWAs right now. Where possible we will also by comparing this year's data with the data from 3 years ago[431]—which was the last time there was a chapter on PWAs.

[427] *https://support.apple.com/en-us/104996*
[428] *https://support.mozilla.org/en-US/kb/web-apps-firefox-windows*
[429] *https://connect.mozilla.org/t5/discussions/how-can-firefox-create-the-best-support-for-web-apps-on-the/m-p/60561/highlight/true#M21220*
[430] *https://developer.mozilla.org/docs/Web/Progressive_web_apps/Guides/Making_PWAs_installable*
[431] *https://almanac.httparchive.org/en/2022/pwa*

Service worker

Service workers remain essential to allow advanced capabilities like background sync, offline support and push notifications to web apps. This year, the data suggest around one fifth of web properties are using service workers.

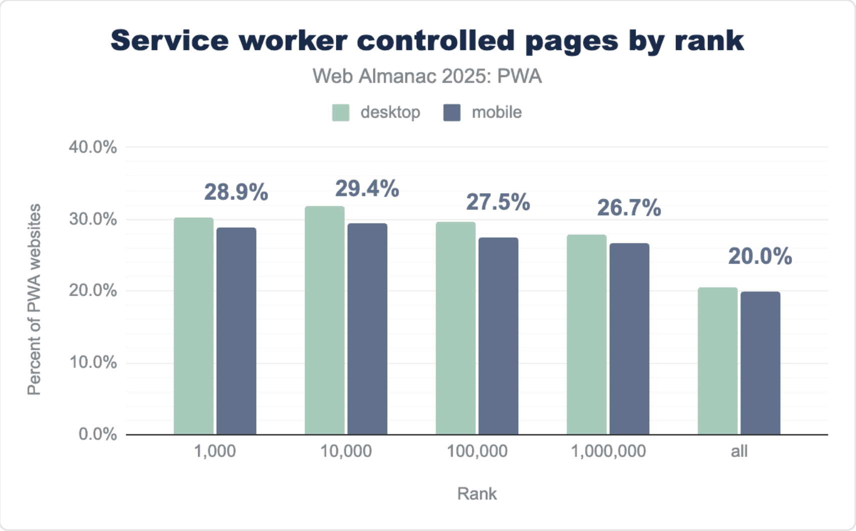


Figure 10.1. Service worker controlled pages by rank.

To start strong, we will look at the service worker controlled pages by rank. From the top 1,000 pages, 30.3% (desktop) and 28.9% (mobile) are managed by a service worker. This is around a 20% increase compared to the data from 3 years ago.

Overall, across every rank grouping there was a strong increase, when looking at *all* the percentage of PWA websites going from 1.4% (desktop and mobile) in 2022 to 20.5% (desktop) and 20.0% in mobile.

Following we have usage data for capabilities of service worker by events, methods and objects.

Service worker events

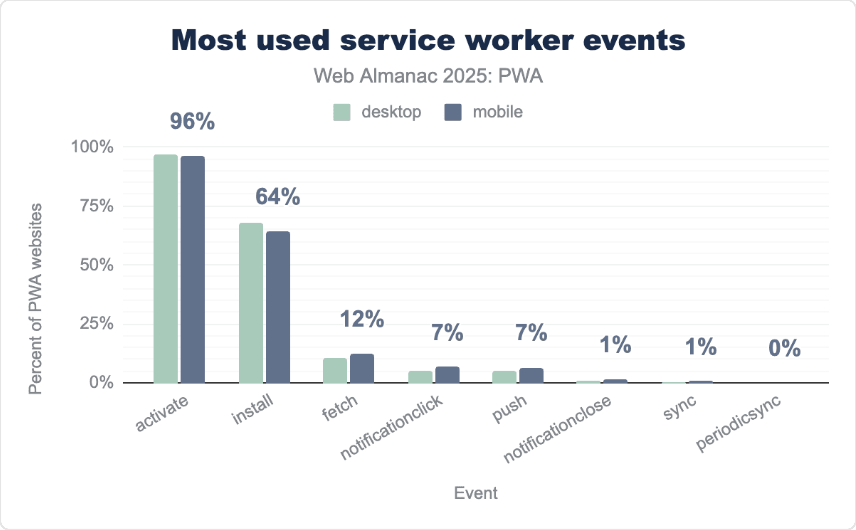


Figure 10.2. Most used service worker events.

The most used event for these service workers is the activation (`activate`) with almost every service worker using it, around 96% of PWAs using it. The `install` event takes second place with around 64% usage. Both `install` and `activate` are core lifecycle events so these numbers do not come as a surprise. This might suggest that applications are caching resources to speed up their loading times and performing service worker management.

Usage of other advanced events, like `fetch`, `notificationclick` and `push` falls considerably, possibly due to these being capabilities that fall into more advanced scenarios, like intercepting network requests, bypassing the default offline UX or delivering notifications from a push service.

Service worker methods

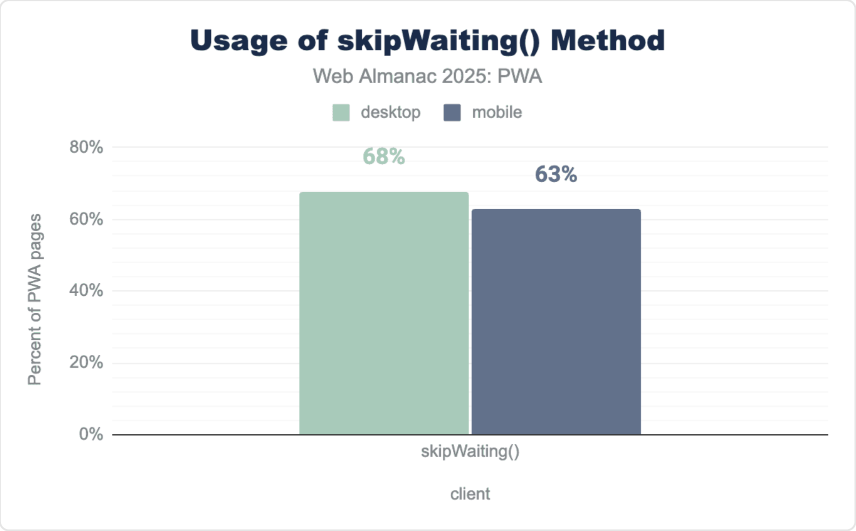


Figure 10.3. Most used service worker methods.

Looking at the most used service worker method, `skipWaiting()` has a notable use, with 68% usage on desktop and 63% on mobile. The browser will activate the service worker and replace the old one immediately. This implies developers are keen to ensure users don't get stuck with stale assets. This is beneficial for applications that require frequent updates, like dashboards and messaging apps. It basically helps users get the latest version of the application right away.

Service worker objects

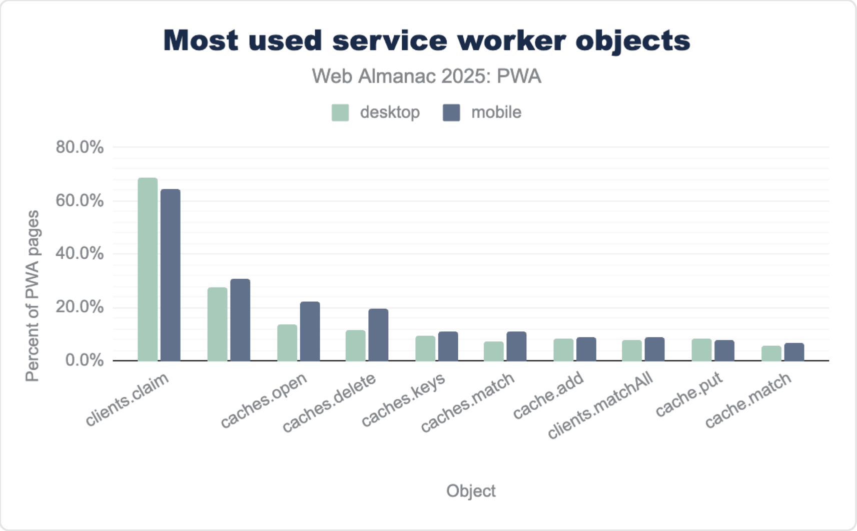


Figure 10.4. Most used service worker objects.

The most used service worker objects are `clients`, `caches` and `cache`. For clients, this can be expected as it is the way to tell the service worker to take control of all open pages by calling `clients.claim()`. Regarding `caches`, management methods appear on top, also not surprising considering that these are the methods that developers would use to make sure their assets are up to date to achieve that speedy page load.

As hinted before, the main methods from these correspond to `claim`, `open`/`delete`/`keys`/`match`, and `add`.

Registration properties

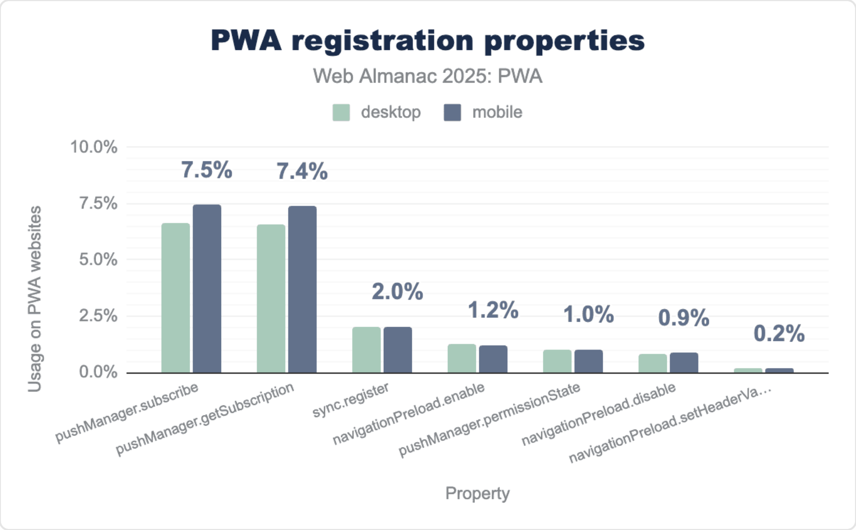


Figure 10.5. Most registered service worker properties.

Diving deeper into service worker functionality, the data sheds some light into advanced capabilities that PWAs are using. For all PWAs using service workers across desktop and mobile, ~7% are registering for pushManager, 2% register for sync and 2% register for navigationPreload.

Web application manifest

The web application manifest is now, more than ever, the most important part of a web app. It defines a look and feel, enables advanced capabilities that are gated behind an installation and is becoming an integral part that identifies a web application as a whole. But in order to be effective, the manifest file needs to be well formed.

94.9%

Figure 10.6. Percent of mobile manifest files parsable on desktop.

For the current year, 94.5% of desktop sites and 94.9% of mobile sites are parseable. There is no change from the last data set[432], and same as last time around, the fact that the manifest file was able to be parsed does not imply completeness or minimum availability of features. Many values in the manifest, as important as they may seem, have reasonable fallbacks in place.

From those parseable manifests, we will now look at individual present fields. This can give us an understanding of how developers are using the manifest file and if there have been changes since 2022.

Manifest properties

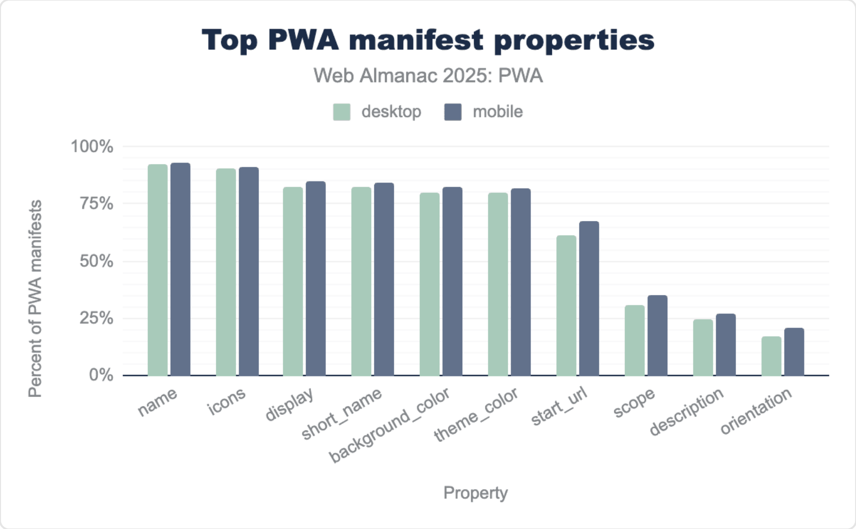


Figure 10.7. Most used manifest properties.

Straight up, these are the most used PWA manifest properties: `name`, `icons`, `short_name`, `display` and `background_color`. The top 4 most used properties are the same ones from 2022[433], with subtle notable changes regarding their order.

Let's examine how individual members rate in the totality of manifest files scanned by the Web Almanac. Unless noted otherwise, values are very similar so I will refer to both mobile and desktop sites.

[432] https://almanac.httparchive.org/en/en/2022/pwa#fig-9
[433] https://almanac.httparchive.org/en/2022/pwa#manifest-properties

Manifest field	*Sites*	
name	92%	93%
icons	90%	91%
short_name	82%	85%
display	82%	85%
background_color	80%	82%
theme_color	80%	82%
start_url	61%	68%
scope	31%	35%
description	25%	27%
orientation	17%	21%

Figure 10.8. Manifest properties.

For the manifests that specify the `categories` member, the top categories are:

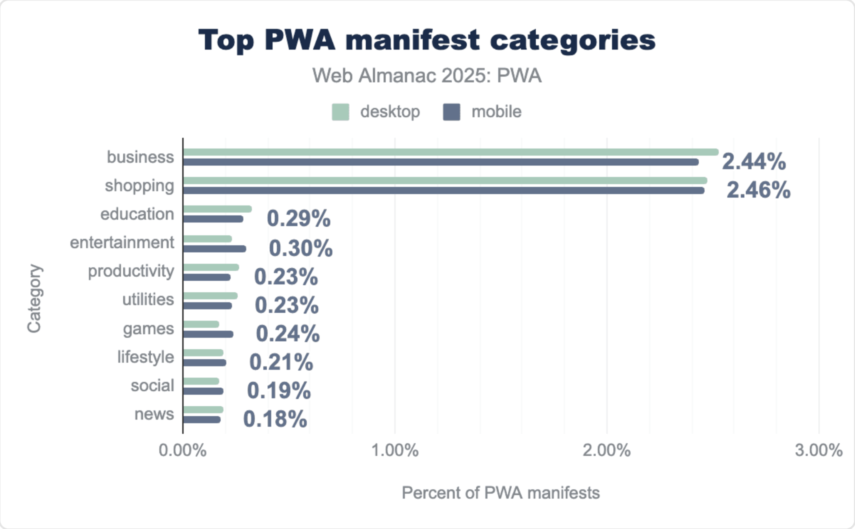


Figure 10.9. Most used manifest category values.

While the manifest key `categories` is not used in many PWAs, these results hint at the top verticals on which web apps are more popular.

Manifest `display` values

The display member is used to specify the preferred display mode for the web app. Different browsers have different interpretations of these values, but overall it hints to the UA how much of the browser UX to show/hide.

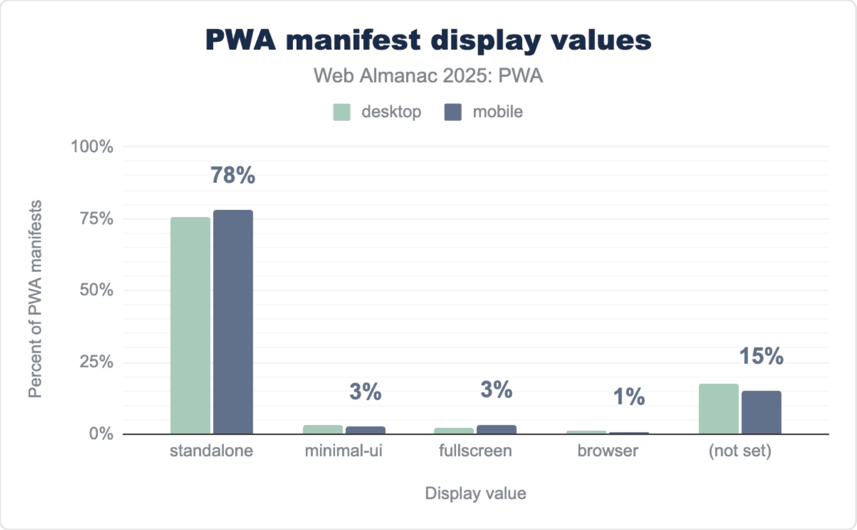


Figure 10.10. Most used display values in the manifest (standalone).

Most web apps (78%) opt for a `standalone` value for the display member. From the documentation, `standalone` "opens the app to look and feel like a standalone native app". This is not surprising as the general intention of a web app for a developer would be to have the app look more native-like, removing the browser chrome and some other UX. On the end this is something that varies with the implementation, as for example in Chromium browsers you get a `...` menu and on Firefox some browser UI like the URL bar still remains for installed apps.

Manifest `icons` sizes values

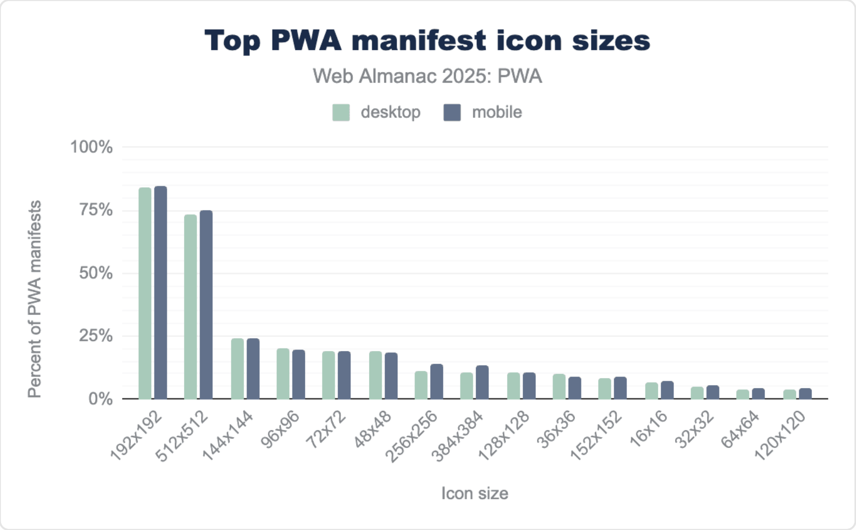


Figure 10.11. Most used icons size values in the manifest.

Top sizes include 192x192 and 512x512.

Manifest `orientation` values

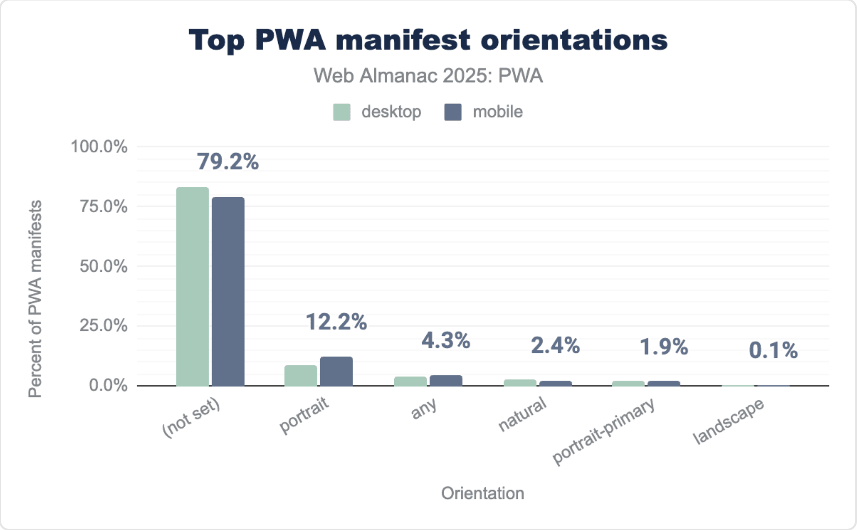


Figure 10.12. Most used orientation values in the manifest.

Unsurprisingly, around 79.2% of PWAs do not set orientation. Responsive design and modern development make defining the app's orientation less necessary, but portrait takes second place with around 12.2%.

Service worker and manifest usage

We've seen the latest data on what the most used service worker and manifest features are. In 2025, we can see that roughly one fifth of sites use service workers, and roughly one tenth use manifests.

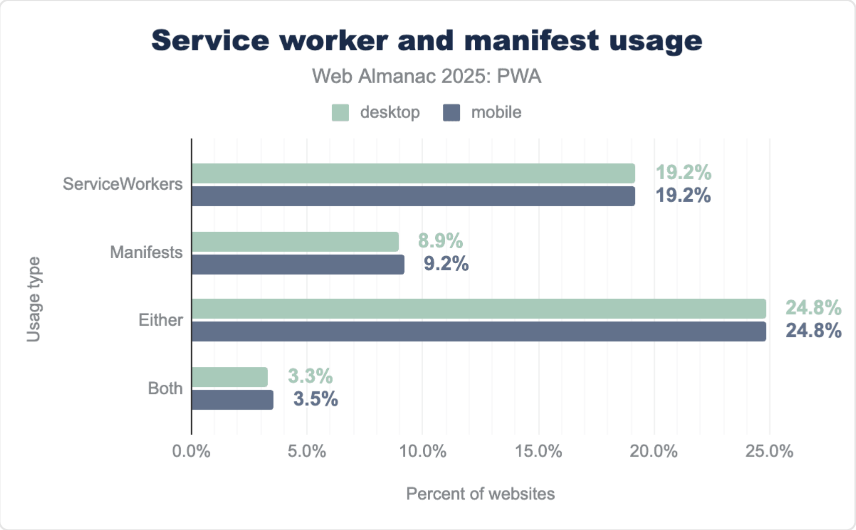


Figure 10.13. PWA service worker and manifest file usage.

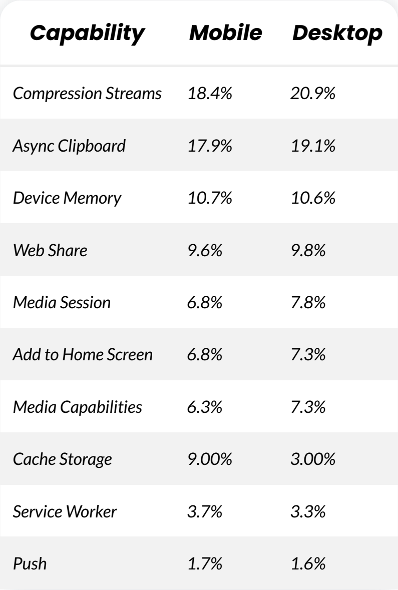
Overall, there are considerable changes to the data from the 2022 Web Almanac[434]. Service worker usage has taken a huge leap from around 1.7% to 19.2%. That is around a 10 times increase in adoption. Manifest usage is slightly higher but remains at a very similar spot as it was 3 years ago.

Digging into this more, it looks like a lot of the growth is due to Google Tag Manager enabling service workers.

PWAs and Fugu APIs

These are the top 10 used advanced capabilities in PWAs for 2025.

[434] *https://almanac.httparchive.org/en/2022/pwa#fig-10*



Capability	Mobile	Desktop
Compression Streams	18.4%	20.9%
Async Clipboard	17.9%	19.1%
Device Memory	10.7%	10.6%
Web Share	9.6%	9.8%
Media Session	6.8%	7.8%
Add to Home Screen	6.8%	7.3%
Media Capabilities	6.3%	7.3%
Cache Storage	9.00%	3.00%
Service Worker	3.7%	3.3%
Push	1.7%	1.6%

Figure 10.14. Top 10 used advanced capabilities in PWAs.

There is a complete separate chapter dedicated to capabilities to dive deeper in the adoption that these sort of APIs have had in 2025.

Notifications and PWAs

Notifications make sense for apps as they allow the user to re-engage with the application. This is a controversial capability as there is considerable bad UX and dark patterns to try to get users to accept them. The data shows that in both desktop and mobile, the most common action a user takes is to ignore these requests.

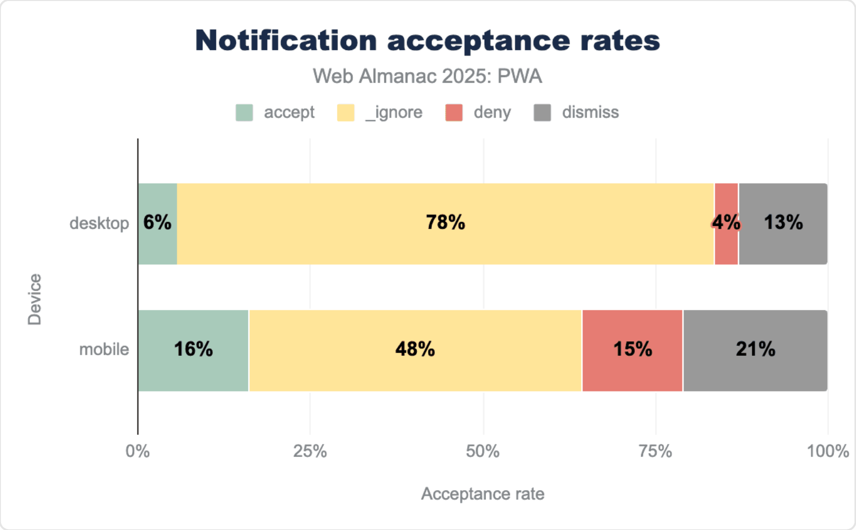


Figure 10.15. PWA notification acceptance rates.

Desktop notification acceptance is overwhelmingly ignored: 78% of them are disregarded, with a slightly lower but still meaningful 48% on mobile. This is not a surprise considering overall the notification fatigue users deal with on a regular basis.

Conclusion

It has been a decade since PWAs. The landscape has changed a bit as the technology and capabilities reach maturity; we now have all major browsers (Edge, Chrome, Firefox and Safari) supporting web apps to a degree. Web apps continue to show growth and almost one fourth of web sites crawled have either a service worker or a manifest file.

If you have noticed a slow down in pace of new features, you are correct. There are less web app features being launched today than a few years ago. I believe this has to do with the fact that capability-wise the platform has reached a level of maturity that allows a wide range of use cases to be created using a web tech stack. On the other hand there is a question of adoption that is tightly coupled with cross-engine capability support. It is until (very) recently that some major browsers are starting to support (some) advanced features, yet alone the possibility to acquire or install a web app.

Whilst advanced capabilities continue to be used sparingly, likely due to slow implementor support, the oldest capabilities like Web Share are starting to show up as well, and even more niche UX features like 'window-controls-overlay' are starting to show up in the data. Slowly but surely these features are being commoditized.

If we look back to the last Web Almanac that had a PWA chapter, these are the things we can notice:

- 2 more browsers (Safari and Firefox) have support for web apps.
- Service worker controlled pages have gone up around 20% for all PWA websites from 2022.
- There is less diversity of service worker events being used than 3 years ago. Less percentage of PWAs are using `notificationclick`, `push`, and `fetch`.
- Percentage of `pushManager` registrations has dropped since 2022, likely due to the rise of usage (see note below) of service worker to focus on performance rather than pushing notifications.
- Between 2022 and 2025 the total of PWA technology usage (desktop 12.5 million and mobile 15.5 million) has around doubled (desktop 5.4 million and mobile 7.9 million - 2022). The percentage of usage for service workers has surged around 10 times, and manifests have largely stayed around the same 8-9% usage.
- Ignoring notification prompts is still the most common behavior for users.

Since these are percentages, it is worth noticing that "diversity of calls" or "drop in registrations" isn't particularly a sign of rejection or lack of use, but instead is a result to the fact that the usage of the core technology has risen. *As an example, for PWA registration properties, the total for* *`pushManager.getSubscription`* *goes up from 52 thousand pages to roughly under 3 million in the last 3 years.*

As we close the 2025 Web Almanac PWA chapter, there's ongoing work and agreement towards looking at a way to democratize web app distribution by baking installation capabilities directly into the platform. We hope in the next 10 years the PWA technologies will follow the same fate as responsive design, where they have been commoditized and web apps that have good UX for application lifecycle management will be the norm, in conjunction with newer capabilities that redefine what a web app can do. Here's to the next decade for web apps!

Authors

Diego Gonzalez

@diekus diekus https://diek.us

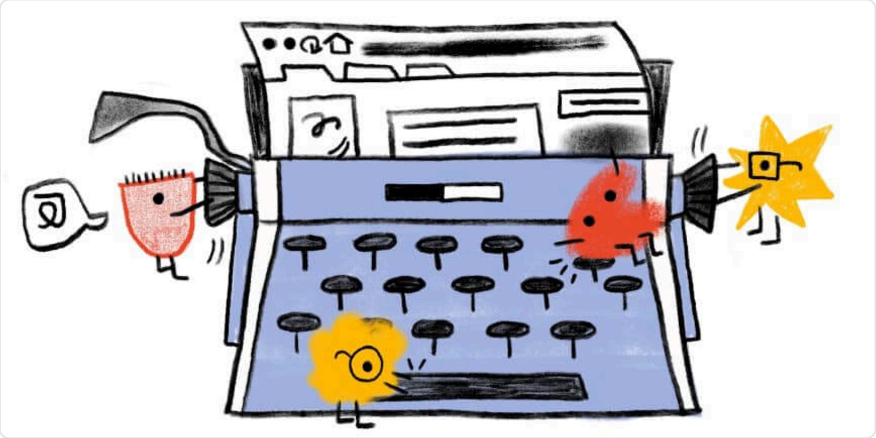
Diego Gonzalez is a Costa Rican engineer working on web platform features for the Microsoft Edge browser.

Michael Solati

@MichaelSolati MichaelSolati https://michaelsolati.com

Michael is a Developer Advocate at Amplication, focusing on helping developers build APIs and drink IPAs. Additionally, he is a Web GDE and has found his love in creating compelling experiences on the web and the voodoo ways of the web.

Part III Chapter 11
CMS



Written by Vahe Arabian
Reviewed by Dan Knauss
Analyzed by Onur Güler and Alon Kochba
Edited by Samuel Fatola and Sreemoyee Bhattacharya

Introduction

Content Management Systems (CMSs) now power almost all websites. By 2025, they do far more than simply let people create and publish pages. They shape how sites are built, how easy they are to use, and how well content surfaces in search engines and AI-powered tools. As sites grow larger, more active, and more personalized, a CMS can be the difference between a site that feels fast, intuitive, and reliable, and one that feels slow and frustrating. This chapter examines the CMS landscape using data from the HTTP Archive.

We look at how many sites use CMSs, how top sites differ from the wider web, how CMS-powered sites perform, and which new capabilities are emerging as these platforms run at scale. Rather than just comparing CMS feature lists, this chapter focuses on how CMS defaults, hosting models, and build approaches shape the real experience of the web—for users and search engines—across both the top million and the top ten thousand sites.

The data shows that CMSs are now used by the majority of sites, and that sites without them are declining every year. To understand the trade-offs this creates for speed, usability, and discoverability, the chapter first explores overall CMS adoption, then examines what that means for performance and user experience, and finally looks at the new tools and patterns emerging alongside widespread CMS use.

What is a CMS?

A Content Management System is software that allows people to create and update content on the web without editing code for every change. Most CMSs separate content from presentation, so editors can change text, images, and structure without writing HTML, CSS, or JavaScript. Developers can extend a CMS by building themes, plugins, or modules.

Broadly, there are two main types of CMSs. A classic monolithic CMS combines content storage, presentation, and delivery into a single system.

A headless or composable CMS separates content management from the site or app that displays it. In practice, most CMSs share a common set of tasks: modeling content, editing it, managing media, extending functionality, and integrating with other systems. The differences in how they approach these tasks become clearer when we look at adoption and implementation patterns across the web.

CMS adoption

By 2025, CMS adoption reflects a web that's both mature and increasingly specialized. Almost every site now runs on some kind of CMS, but how these platforms are used varies a lot by region, traffic level, and underlying technology. Instead of everyone standardizing on a single go-to system, the market is clearly fragmenting, with different platforms leading in different parts of the world, for different use cases, and at different scales.

Overall adoption

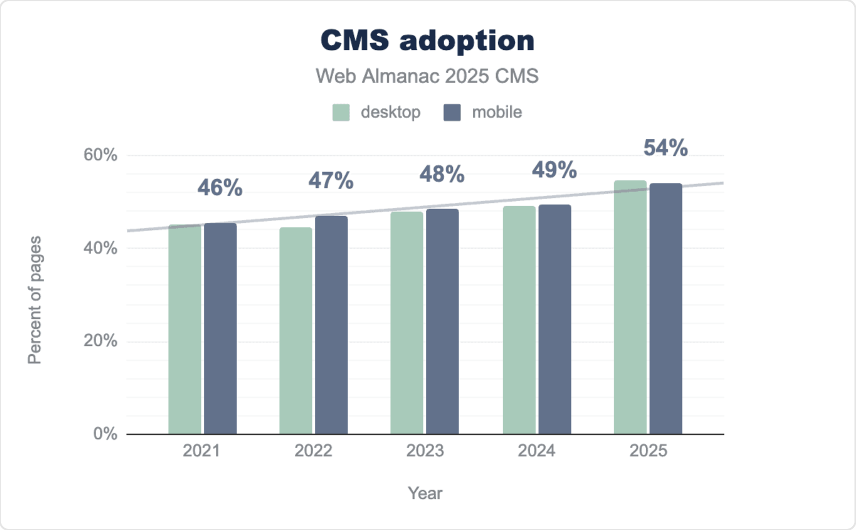


Figure 11.1. CMS adoption.

Overall adoption HTTP Archive measurements show that CMS-driven sites account for over 54% of observed websites in 2025, reinforcing CMSs as the default infrastructure for the web.

Adoption by geography

CMS adoption looks different from one region to another.

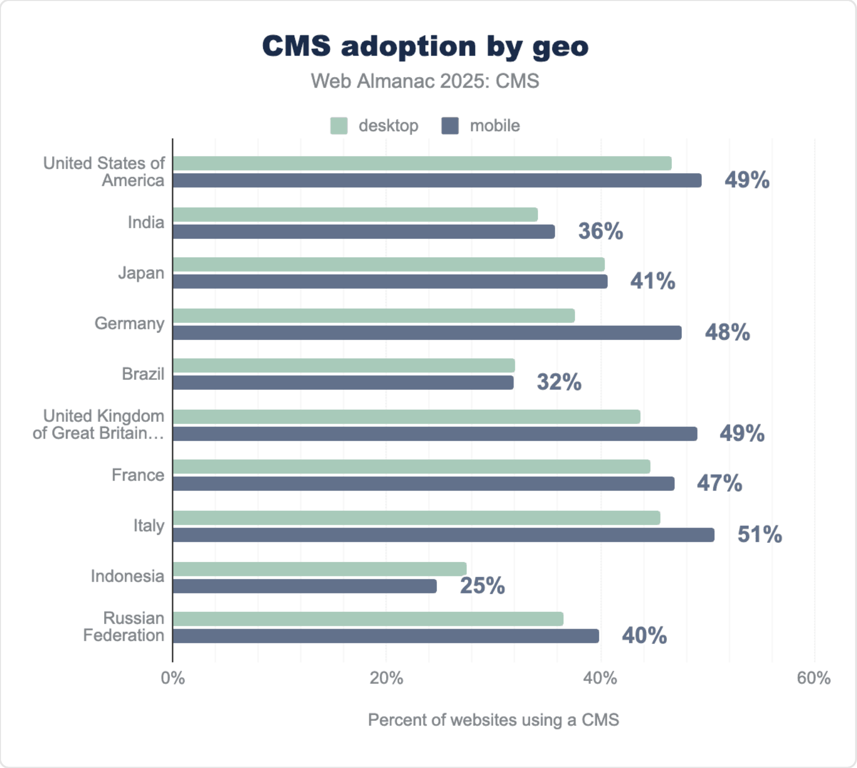


Figure 11.2. CMS adoption by geo.

HTTP Archive data shows higher concentrations of hosted CMS platforms in North America and Western Europe. In contrast, open-source systems hold a relatively stronger position in parts of Asia and Eastern Europe.

Adoption by rank

Website rank continues to shape CMS adoption patterns. High-traffic sites tend to choose platforms that support complex workflows, deep customization, and long-term scalability. Lower-traffic sites are more likely to use hosted or all-in-one solutions that reduce operational overhead.

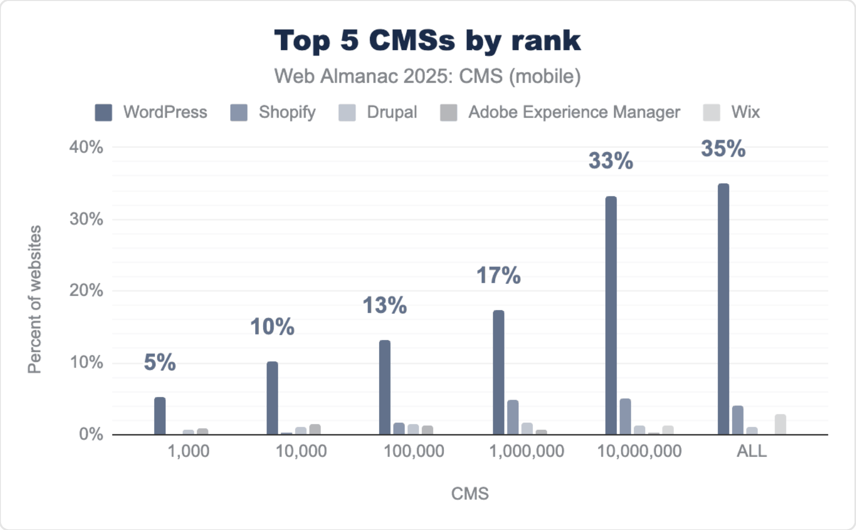


Figure 11.3. Top 5 CMSs by rank.

Among the top 10,000 websites, WordPress accounts for roughly 58% of CMS usage, while Drupal represents about 6–7%, far higher than its 1% overall market share. Platforms like Wix and Shopify are almost absent at this traffic level.

Most popular CMSs

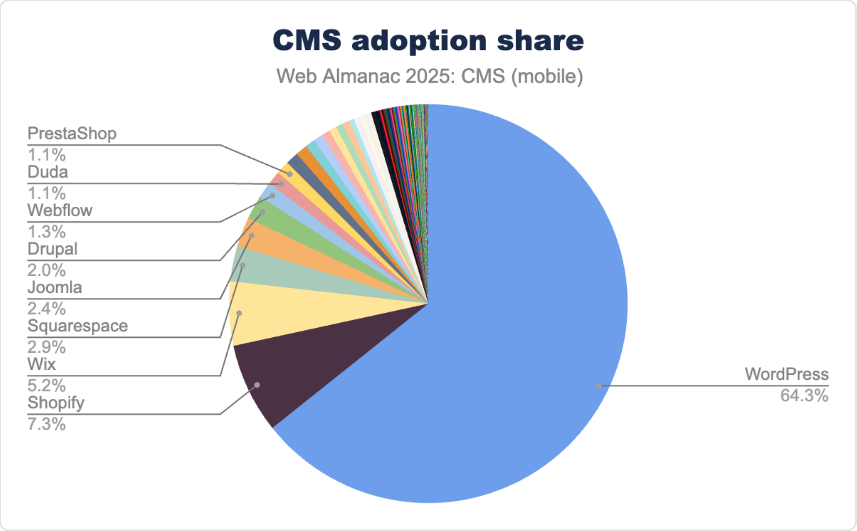


Figure 11.4. CMS adoption share.

WordPress remains the dominant CMS in 2025, powering roughly 64% of CMS-driven sites. Its growth, however, has slowed to under one percentage point year over year, which points more to market saturation than to a major competitive threat from any single rival.

The data shows that this slowdown is not caused by one platform displacing WordPress. Instead, small gains are spread across several CMSs. Shopify has grown to around 7.3–7.8% CMS share, Wix to about 5%, and Squarespace to roughly 3%. No single platform's growth is large enough to offset WordPress's slowing expansion, suggesting a more fragmented ecosystem rather than a consolidating one.

Overall, the data indicates a more diverse CMS landscape shaped by market maturity and greater choice. WordPress remains broadly adopted, but new growth is increasingly distributed across multiple platforms instead of accruing to a single default CMS.

Fastest growing CMSs

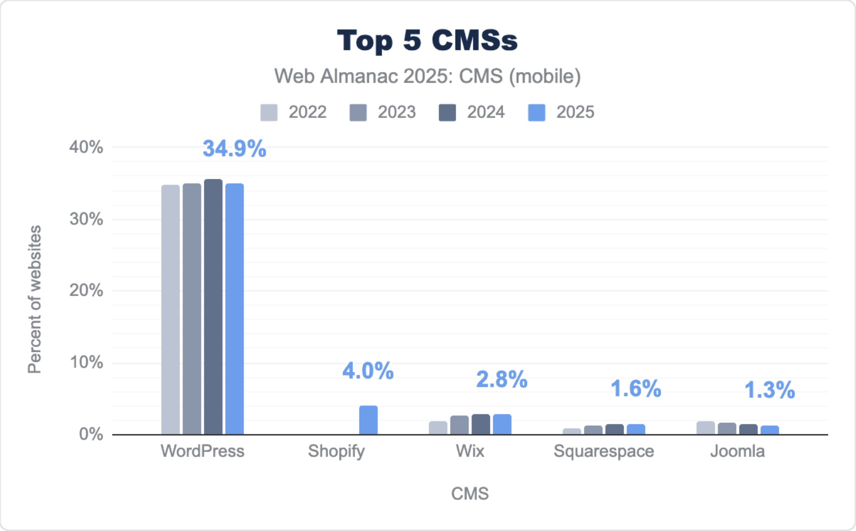


Figure 11.5. Top 5 CMS'.

Year-over-year growth among CMS platforms in 2025 is modest and uneven. Shopify is a new entry this year having previously just been classified as under the Ecommerce category but now is included under CMS as well. Squarespace shows small but positive growth of roughly 0.2–0.3 points, reaching around 3.0–3.3%. Wix's growth slows sharply compared with prior years and is now effectively flat at about 5.2% after a period of rapid expansion. These gains are incremental and spread out, not indicative of a new wave of explosive growth.

By contrast, WordPress's share declines slightly year over year—by less than one percentage point—from its 2024 peak, marking its first sustained slowdown after decades of expansion. Even so, WordPress still powers most CMS-driven sites and remains the clear leader across the web. Traditional open-source platforms like Joomla and Drupal continue long-term declines in overall share, though Drupal is still disproportionately represented among high-traffic sites. In short, the fastest-growing CMSs in 2025 are making gains in specific niches, while WordPress remains dominant in absolute terms and continues to anchor the broader ecosystem.

WordPress in 2025

Given its dominance in the CMS world, it's worthwhile discussing it a bit more.

Market and ecosystem

WordPress's importance in the CMS ecosystem is now less about raw growth and more about influence. Its sheer scale makes it the reference point for understanding architectural choices, performance patterns, and real-world implementation trade-offs. Because WordPress is used across almost every traffic tier and use case, its behavior in production has an outsized impact on overall web performance and CMS-driven outcomes.

That influence is supported by a highly extensible ecosystem. Rather than enforcing a single development or hosting model, WordPress supports many approaches through its plugin, theme, and integration layers. This flexibility lets sites evolve gradually—adding new features, monetization models, or editorial complexity—without migrating platforms. As a result, WordPress often stays in place across multiple generations of a site's lifecycle, even as other platforms grow in narrower segments.

The same openness, however, introduces significant variability. WordPress sites show a wide range of architectures, performance profiles, and operational complexity compared with more tightly integrated CMSs. Its dominance does not lead to uniform outcomes; instead, it produces a broad distribution shaped by implementation choices. The next sections examine how this flexibility appears in practice, in terms of architecture, performance metrics, and constraints at scale.

Technical performance and improvements

Recent WordPress development has focused on reducing performance bottlenecks that appear as sites grow in size, editorial complexity, and block usage. Rather than redefining interaction models, core engineering work emphasizes making existing systems faster, more cache-friendly, and less sensitive to configuration differences.

A major area of progress has been editor performance. Optimizations to template loading, block rendering, and pattern reuse have reduced editor startup and interaction lag, delivering up to 35% faster template loading in block-heavy setups. Persistent caching of reusable block patterns and global styles cuts down on repeated computation, addressing long-standing scalability issues for large sites with complex layouts.

Media handling has also improved. Updates to image processing pipelines result in roughly 20% faster AVIF image generation[435], better auto-sizing, and more reliable lazy loading. These changes reduce server-side processing time and frontend layout shifts, improving load performance without requiring individual site-level tuning.

[435] *http://core.trac.wordpress.org/ticket/61758?st_source=ai_mode#:~:text=AVIF%20generation%20has%20improved%20from,:%2095.81%20MB%20(100463863%20bytes)*

On the frontend, performance work has centered on cutting unnecessary payloads and redundant processing. Core now loads block styles and scripts more selectively, avoids assets for hidden blocks, and caches generated styles more aggressively. Speculative loading techniques—such as prefetching and prerendering likely navigations—further improve perceived speed and Largest Contentful Paint in supported browsers. Collectively, these optimizations target common real-world bottlenecks, improving consistency across a wide variety of configurations. HTTP Archive data supports this pattern, showing that WordPress performance variance is driven more by configuration than by core limitations.

Accessibility improvements run alongside performance work. Enhancements to semantic markup, keyboard navigation, labeling, and editor usability aim to make accessibility a baseline property of WordPress output, not something that depends entirely on theme or plugin choices.

WordPress progression in 2025

These changes suggest that WordPress is shifting from a focus on expansion to one on stabilization. The emphasis on editor responsiveness, cache reuse, asset reduction, and variance control aligns with HTTP Archive findings that show WordPress sites spanning a wide performance range rather than clustering at the top or bottom. Core changes are increasingly aimed at lifting the worst-performing implementations, narrowing the gap between well-tuned and poorly tuned sites.

Ongoing investment in block-based architecture and Full-Site Editing (FSE) looks less like experimentation and more like long-term commitment. Rather than stepping back from the block model, WordPress is absorbing its operational costs through steady optimization. Blocks are now treated as core infrastructure, with performance work focused on making them viable for large, long-lived sites.

Equally important is what WordPress core does not try to do. Despite widespread experimentation with AI, collaboration tools, and advanced workflows in the broader ecosystem, core continues to prioritize APIs, primitives, and backward compatibility over bundled, opinionated end-user features. This follows WordPress's historical pattern: let innovation happen at the ecosystem edge while keeping core conservative and predictable.

In a CMS market that is becoming more polarized, these choices reinforce WordPress's position as durable web infrastructure. While other platforms differentiate themselves through tightly managed, vertically integrated experiences, WordPress continues to favor adaptability, longevity, and risk reduction at scale. Its technical trajectory in 2025 is less about chasing competitors feature-for-feature and more about remaining operable across the broadest and most diverse slice of the web.

Page builders

Page builders have become a dominant interface layer within the WordPress ecosystem. Estimates suggest that around 60% of WordPress sites use a page builder, reflecting demand for faster iteration, less reliance on developers, and greater editorial autonomy. Elementor has the largest observed footprint, followed by WPBakery and Divi, alongside broad use of the native Block Editor.

Page builders appeal because they speed up workflows. WYSIWYG editing, reusable components, and template libraries allow non-technical users to adjust layouts without writing code, aligning with broader no-code and component-based development trends. For many organizations, this shrinks bottlenecks and shortens iteration cycles.

These advantages come with trade-offs. Page builders often generate more complex DOM structures and larger CSS and JavaScript bundles, raising performance risks at scale. Older builders, in particular, have created long-term maintenance and lock-in issues. As performance expectations rise, these costs are increasingly visible.

In response, major builders are moving closer to WordPress core conventions. Newer efforts prioritize block-native output, reduced reliance on shortcodes, and deeper integration with the Block Editor and FSE. Rather than trying to replace core, many builders are converging around shared primitives and competing mainly at the UX layer.

Page builders change how WordPress pages are put together and rendered, with clear consequences for performance. By abstracting layout and design into visual components, they reduce the need for custom code and speed up site creation. But this extra abstraction also affects DOM complexity, asset-loading behavior, and runtime execution, making page builders a major contributor to performance variation across WordPress sites.

Historically, many page builders were associated with heavy markup and large CSS and JavaScript payloads, which often led to slower loads and weaker Core Web Vitals. More recent builders have tried to improve this by adding conditional asset loading, minification, and reduced shortcode use. These efforts help, but they do not completely erase the performance gap between builder-heavy sites and more tightly controlled builds.

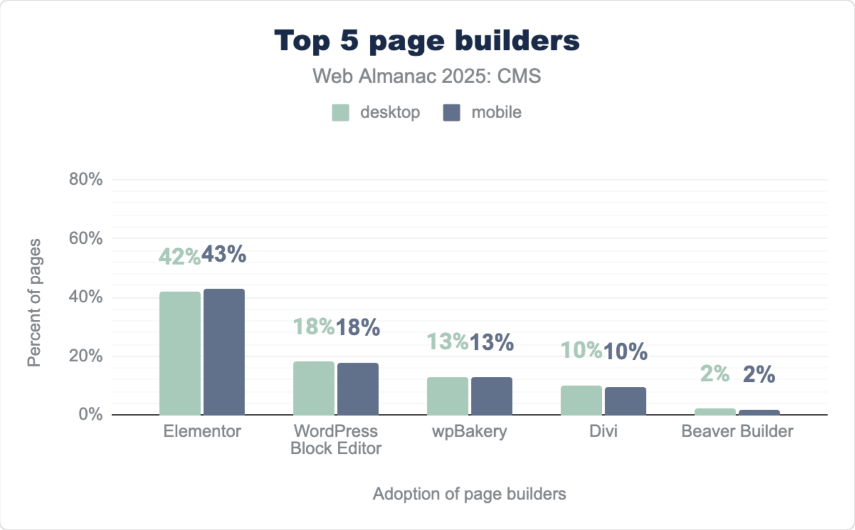


Figure 11.6. Top 5 page builders.

Usage patterns are also shifting. Between 2024 and 2025, Elementor remained the most widely used page builder, but its share dropped from about 56% to 43%, pointing to a more fragmented ecosystem. The WordPress Block Editor grew to around 18%, while WPBakery fell from roughly 21% to 13%. Divi declined from about 14% to 10%, and Beaver Builder held a small but steady share of around 2%. These trends point to a gradual move away from older builders toward block-native or more performance-focused approaches.

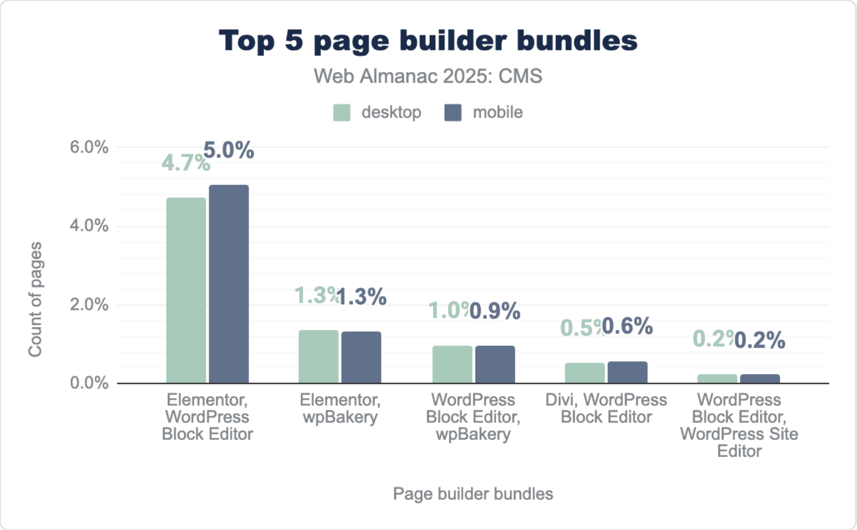


Figure 11.7. Top 5 page builder bundles.

Overall, the data suggests that page builders are still a core part of the WordPress ecosystem, but their performance impact now depends heavily on how closely they align with WordPress core. Builders that reduce markup overhead, avoid global asset loading, and integrate deeply with the Block Editor tend to limit performance costs more effectively. As expectations continue to rise, the technical differences between builders may matter more than their visual feature sets.

CMS user experience

The architectural and performance trade-offs described so far are shaped not just by platform design, but by how CMSs are actually used. The user experience of CMSs—especially for editors and administrators—remains a critical but often underexplored driver of real-world outcomes.

Across platforms, editorial UX often emphasizes flexibility over structure, presenting users with many options but few guardrails. Effective CMS UX instead leans on structured components, sensible defaults, and role-based permissions to reduce cognitive load and support consistent publishing. As sites grow, additional needs—such as governance, approval workflows, taxonomy management, and localization—turn CMS UX from a design preference into an operational requirement.

Although invisible to most visitors, CMS UX directly affects frontend quality. Poorly constrained editorial tools can result in inconsistent layouts, heavy pages, accessibility problems, and outdated content. In this way, backend UX indirectly shapes performance, discoverability, and accessibility for end users.

One common response to these challenges has been the adoption of page builders, which aim to simplify layout and design through editor-friendly visual interfaces. Their impact echoes HTTP Archive findings that show wide performance variation for WordPress sites driven more by configuration and tooling choices than by core itself.

Core Web Vitals

Core Web Vitals offer a practical way to see how CMS platforms perform under real-world conditions. Rather than showing theoretical best-case performance, these metrics capture the combined effects of platform defaults, hosting choices, themes, plugins, and page builders as experienced by actual users.

This section looks at year-over-year Core Web Vitals performance across major CMS platforms, focusing on mobile, where constraints are tighter and differences are easier to see. The goal is not to crown a single “fastest” CMS, but to understand how varying levels of platform control and ecosystem complexity affect performance at scale.

Year-over-year trends

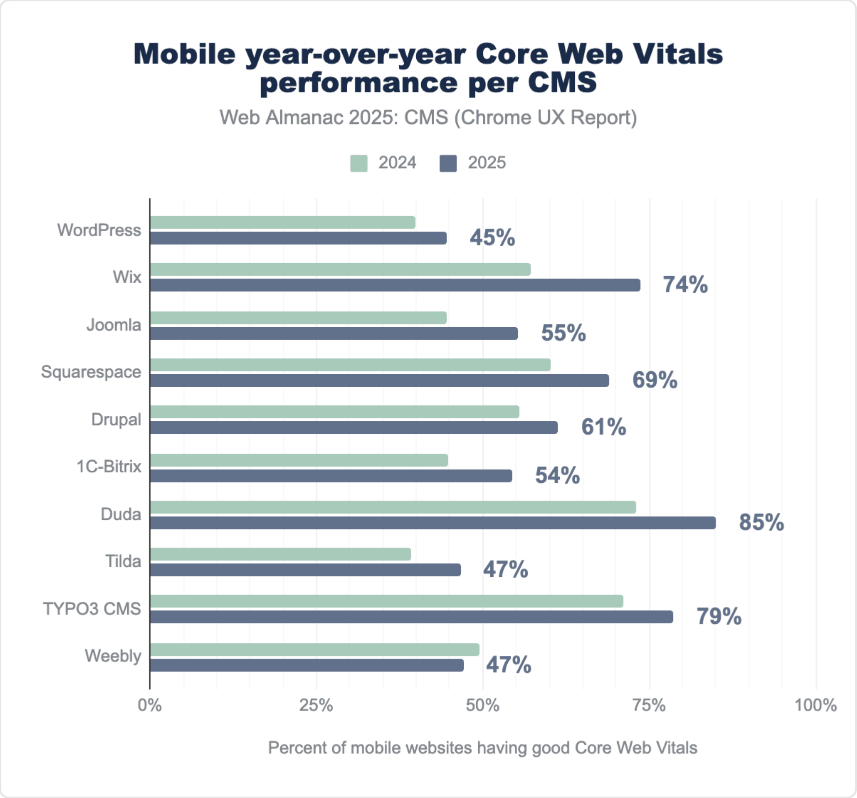


Figure 11.8. Mobile year-over-year Core Web Vitals performance per CMS.

From 2024 to 2025, most CMS platforms improved their overall Core Web Vitals performance, though by varying amounts. Platforms with more tightly managed environments see the largest gains—led by Wix (around +14% year over year) and Duda (+11%), followed by Squarespace (+8%) and Joomla (+7%). Several smaller platforms, including 1C-Bitrix, Tilda, and TYPO3, improved by about 5%.

More extensible platforms improved less. WordPress and Drupal each gained around 4% year over year, while Weebly declined slightly (about -1%). These patterns highlight how difficult it is to propagate improvements evenly in ecosystems that allow more customization and diversity in implementation.

Largest Contentful Paint (LCP)

Largest Contentful Paint measures how quickly the main content of a page appears and is one of the most important metrics for perceived load speed.

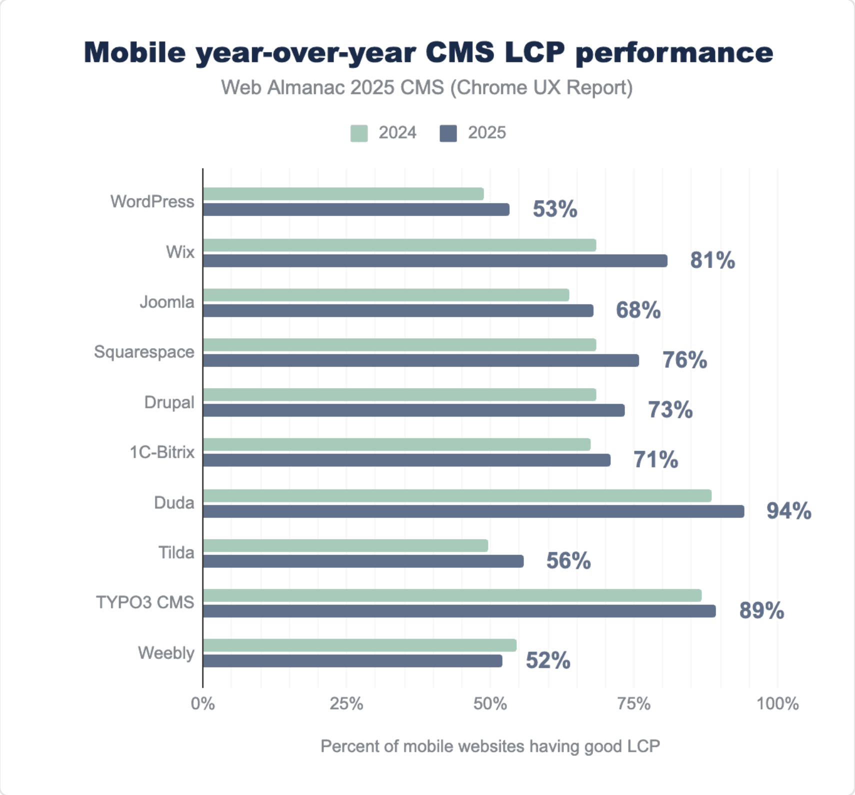


Figure 11.9. Mobile year-over-year CMS LCP performance.

In 2025, 54% of sites achieved a "good" LCP score, reflecting ongoing progress but also significant remaining variability.

Most CMS platforms show year-over-year LCP improvements. Wix leads with roughly a 10% gain, followed by Squarespace (+7%) and Duda (+5%). WordPress, Joomla, and Drupal each improve by about 4%, while Weebly slips slightly (around -1%). These differences align with platform-level choices around asset loading, image optimization, and default configurations.

Cumulative Layout Shift (CLS)

Cumulative Layout Shift captures how visually stable a page is while it loads, by measuring unexpected layout movements.

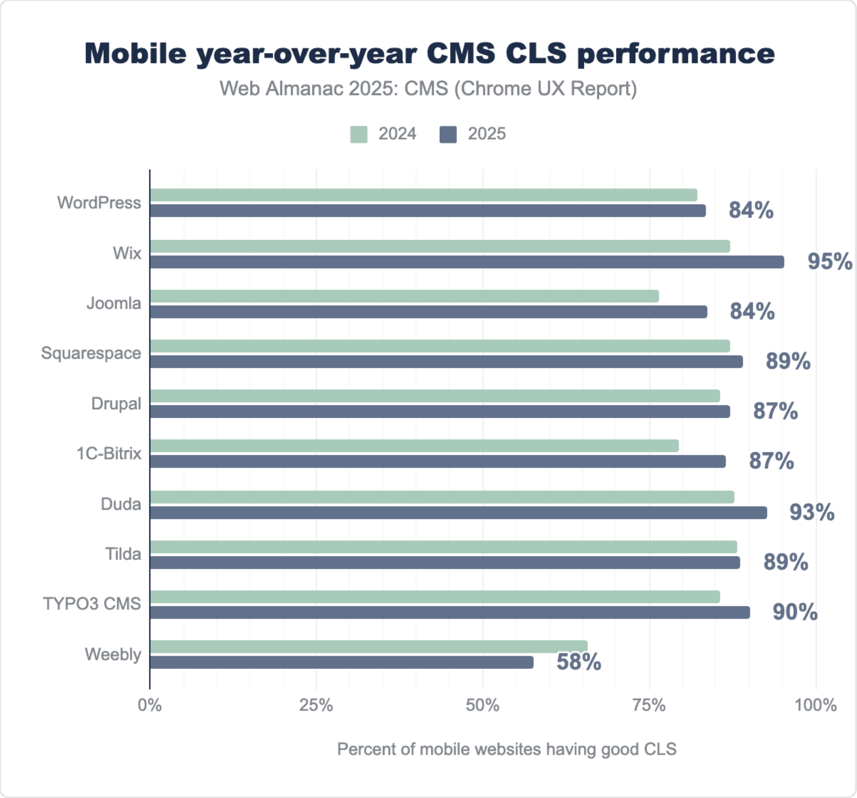


Figure 11.10. Mobile year-over-year CMS CLS performance.

CLS remains one of the more uneven metrics across platforms, reflecting challenges in managing late-loading content, embeds, and dynamic layouts.

Wix again shows the strongest year-over-year improvement (around +8%), followed by Duda (~+4%), then Joomla and 1C-Bitrix (each around +3%). Other platforms show little change, while Weebly experiences a notable decline (about -8%). Overall, CLS outcomes seem to depend heavily on implementation discipline, not just platform choice.

Interaction to Next Paint (INP)

Interaction to Next Paint measures how responsive a page feels across all user interactions, not just at initial load.

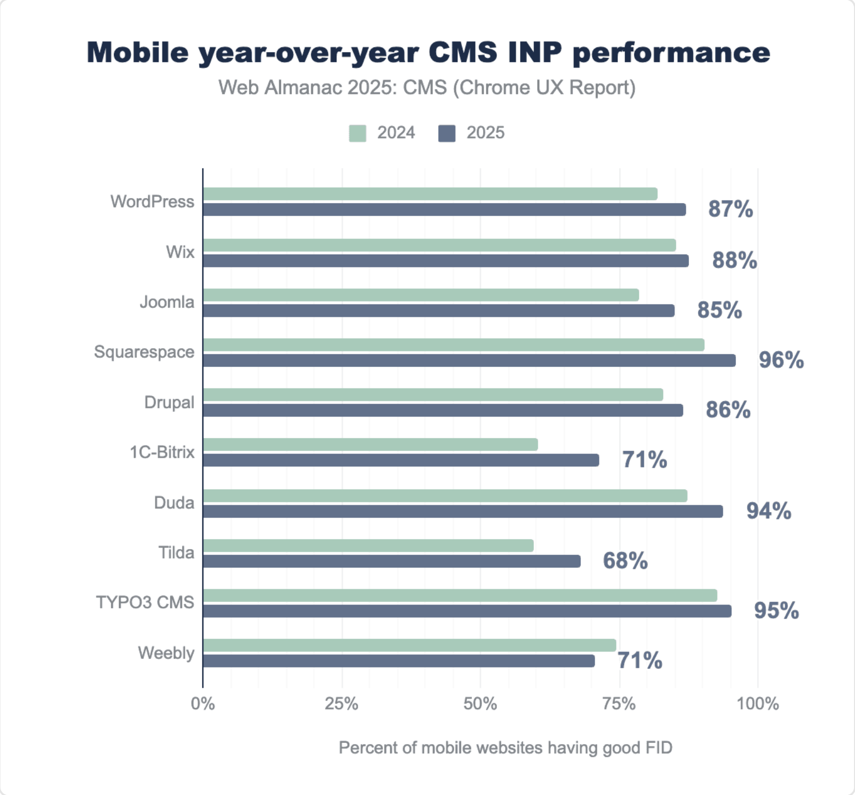


Figure 11.11. Mobile year-over-year CMS INP performance.

Compared with LCP and CLS, INP improvements are modest, underscoring how difficult it is to manage JavaScript, long tasks, and third-party scripts.

In 2025, 1C-Bitrix leads in INP improvement (about +10%), followed by Squarespace and Duda (around +6%), Joomla and Tilda (about +5%), and Drupal (+3%). Weebly again sees a decline (-3%). Across the board, no CMS consistently delivers excellent INP at scale, suggesting that interaction latency remains a shared problem.

Lighthouse quality metrics

Lighthouse offers a complementary, lab-based perspective on site quality across Performance, Accessibility, SEO, and Best Practices. While Lighthouse scores do not directly reflect real-user experience, they help compare typical implementations under consistent test conditions.

Performance

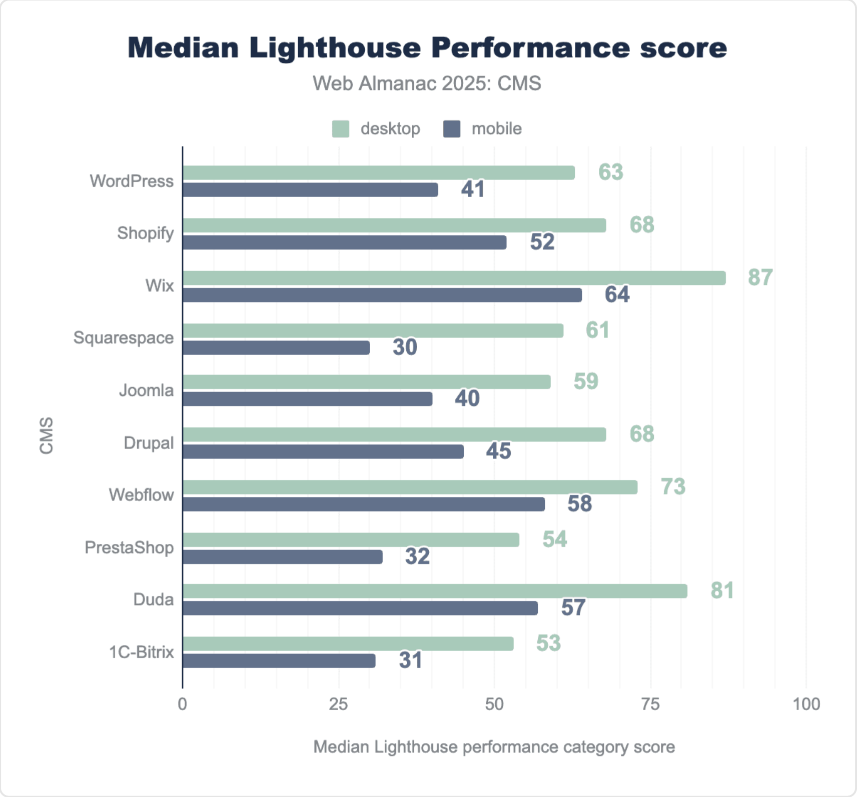


Figure 11.12. Median Lighthouse Performance score.

Median Lighthouse performance scores improved between 2024 and 2025 on both desktop and mobile, with desktop consistently scoring higher. On desktop, Wix (87) and Duda (81) are out in front, followed by Webflow (73). WordPress records a median score of 63, with several platforms close by.

On mobile, scores are lower across the board. Wix leads with 64, followed by Webflow (58), Duda (57), and Shopify (52). WordPress (41) and Joomla (40) sit behind this group, while PrestaShop and 1C-Bitrix post the lowest scores. Year over year, Wix shows the biggest jump on mobile (from 55 to 64), while most other platforms see only modest movement or remain stable. These lab scores are best read as context, not as a proxy for real-user experience.

SEO

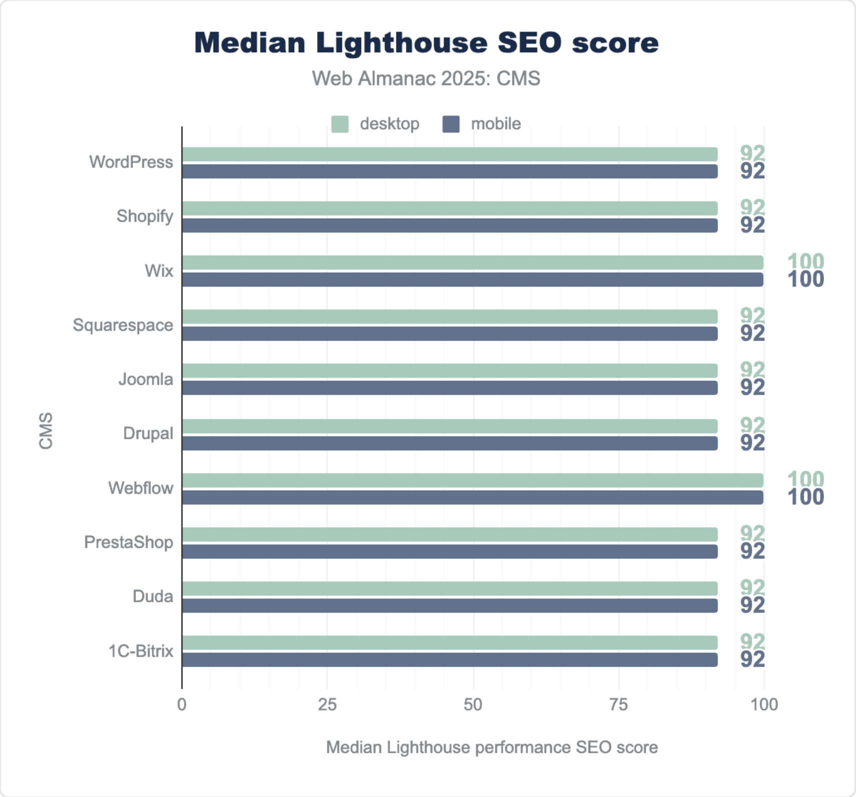


Figure 11.13. Median Lighthouse SEO score.

In 2025, Lighthouse SEO scores remain high across CMS platforms, with most clustered between 92 and 100 on both mobile and desktop. Webflow and Wix achieve perfect scores, while WordPress, Duda, and Joomla remain around 92. The small year-over-year changes suggest that basic SEO best practices are now widely baked into modern CMS platforms.

Accessibility

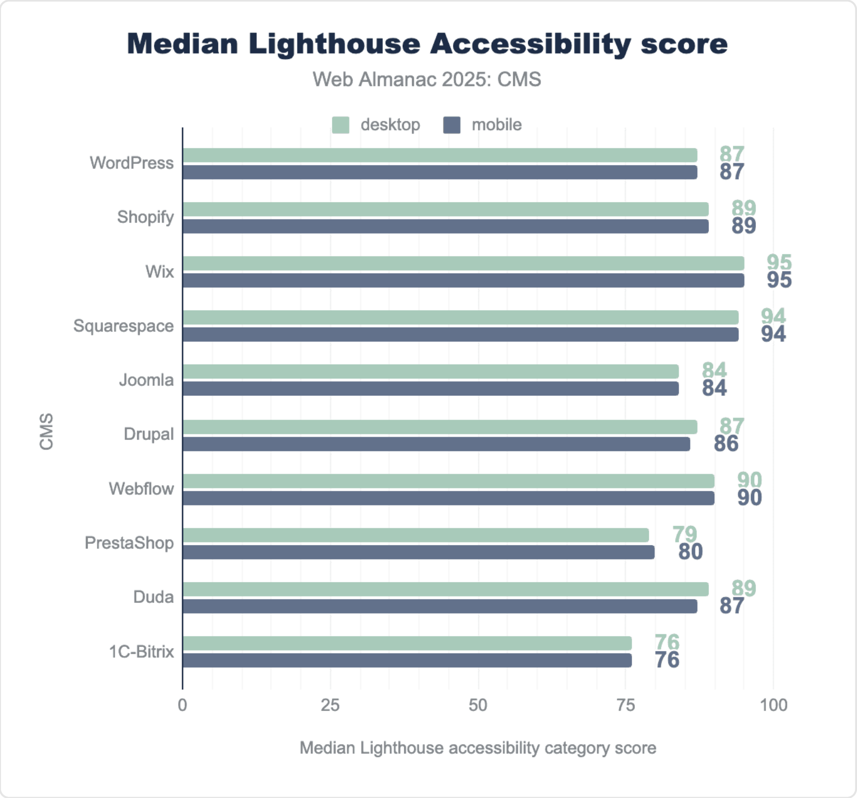


Figure 11.14. Median Lighthouse Accessibility score.

Accessibility scores vary more than SEO but still show limited change year to year. In 2025, median scores range from 76 to 95, with Wix (95) and Squarespace (94) leading. WordPress and Joomla remain stable, while 1C-Bitrix trails at 76. Overall, improvements are gradual, indicating steady but not transformative progress.

Best Practices

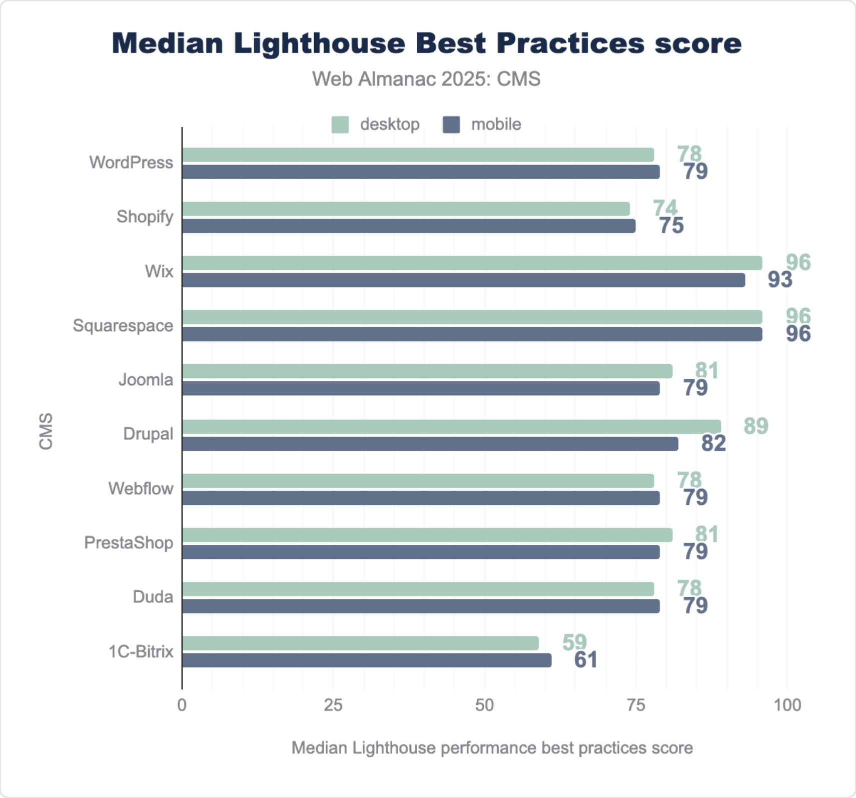


Figure 11.15. Median Lighthouse Best Practices score.

Best practices scores show clearer separation. On mobile, Squarespace (96) and Wix (93) lead, followed by Drupal and PrestaShop (both 82). WordPress, Duda, and Joomla cluster around 79, and 1C-Bitrix ranks lowest at 61. Desktop results follow a similar pattern.

Year-over-year, some platforms improve notably—particularly Wix (from 79 to 93) and Drupal (from 79 to 82)—while others barely move. These differences point to ongoing gaps in areas like modern API usage, security configuration, and error handling.

Page weight and resource composition

"Page weight" is the total size of all resources a browser needs to download to render a page. Over the past decade, page weight has steadily increased.

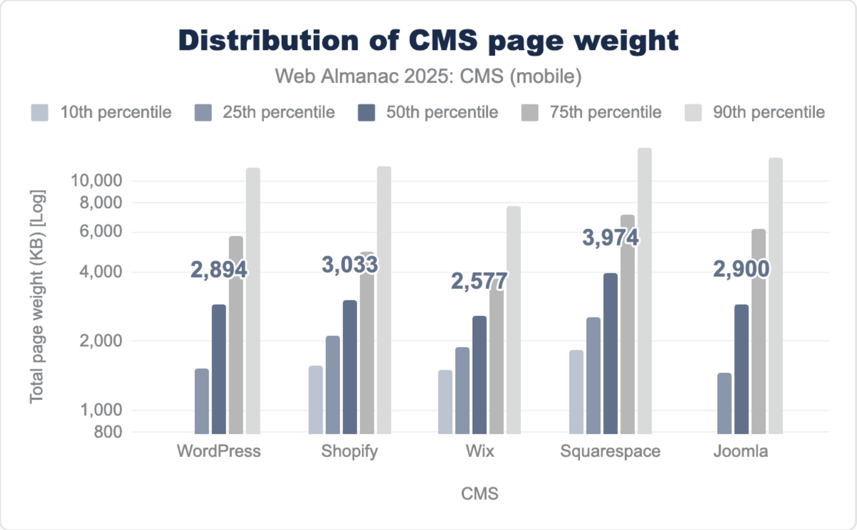


Figure 11.16. Distribution of CMS page weight.

In 2025, the average page is roughly 2.67 MB on desktop and 2.28 MB on mobile. Both figures exceed the commonly recommended range of 1–1.5 MB, reflecting continued bloat across the web.

Most of this growth comes from images and JavaScript, while HTML remains a relatively small part of total transfer size. Despite growing awareness of performance issues, page weight keeps rising. Pages that take longer than three seconds to load tend to have much higher bounce rates, and additional delays are strongly linked to lower conversion rates. For mobile users on slow or metered connections, heavy pages can be a real barrier to access.

Page weight is therefore more than a technical detail. It shapes user perception, accessibility, and business outcomes. Organizations that treat page weight as a first-class constraint generally see more predictable performance, while those that do not often struggle with sluggish experiences and reduced engagement.

Total page weight by CMS

Total page weight varies widely between CMS platforms, influenced by default themes, asset pipelines, plugin ecosystems, and hosting models. While page weight has gone up across all platforms, CMS choice still affects both the median size and the spread of page weights.

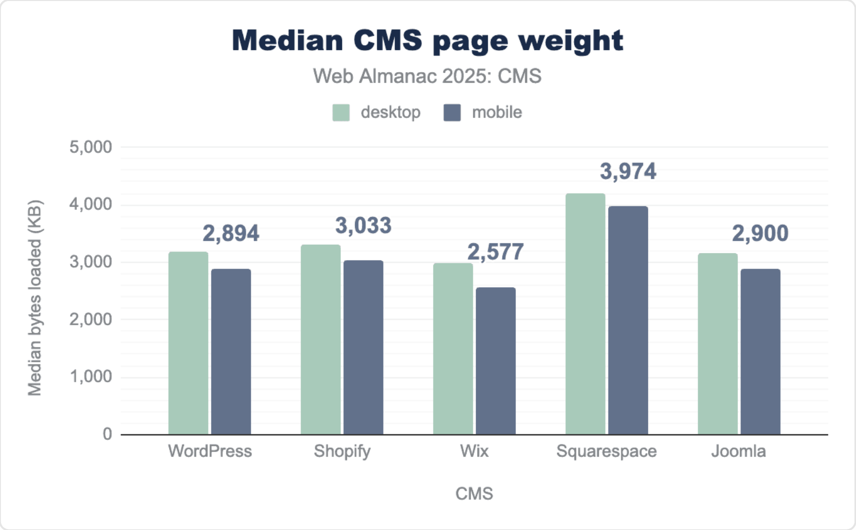


Figure 11.17. Median CMS page weight.

Platforms in more controlled environments tend to show tighter distributions. More extensible CMSs show wider variation, with page weight driven less by core and more by the accumulated impact of themes, plugins, page builders, and third-party tools. As a result, two sites on the same CMS can have very different total transfer sizes.

This variability mirrors the earlier performance patterns, reinforcing the link between page weight and broader performance outcomes.

Resource composition: images

Images remain the largest contributor to page weight on all CMS platforms. Even as modern formats and responsive image techniques become more common, their benefits are often offset by a growing number of images and larger visual assets—especially on template-heavy and visually rich sites.

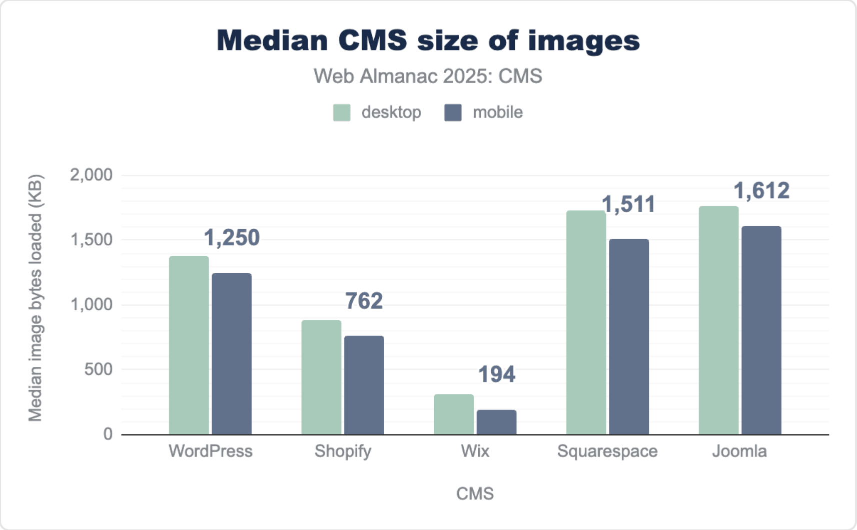


Figure 11.18. Median CMS size of images.

CMS platforms with stronger defaults for image handling generally produce smaller, more consistent image payloads. In more flexible systems, image optimization depends heavily on theme choices, plugin setups, and editorial habits, leading to greater variability.

The continued growth in image payloads helps explain how some platforms can improve loading metrics yet still serve heavier pages overall.

Resource composition: JavaScript

JavaScript is the fastest-growing part of page weight and one of the most important factors for runtime performance. JavaScript payloads reflect the combined cost of core CMS logic, themes, page builders, analytics, ads, and other third-party services.

Page builders and component-based systems often increase JavaScript use by adding client-side rendering and interaction layers. While newer approaches try to load fewer unnecessary scripts, JavaScript remains a major driver of interaction latency and responsiveness issues discussed earlier.

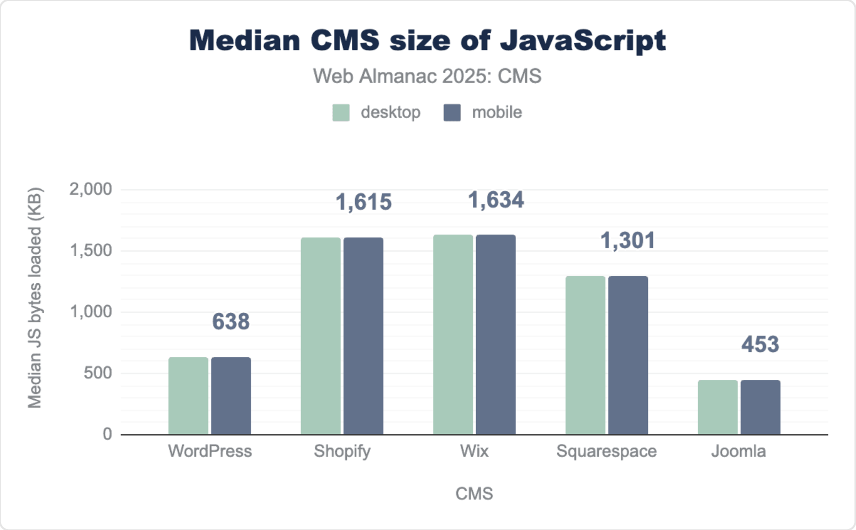


Figure 11.19. Median CMS size of JavaScript.

As with overall page weight, JavaScript costs vary significantly within extensible CMS ecosystems. Implementation choices matter more than platform choice alone.

CSS and HTML

CSS and HTML make up a smaller share of total page weight than images and JavaScript, but they still influence rendering. Large global stylesheets, duplicate rules, or unscoped styles can delay rendering and add blocking time.

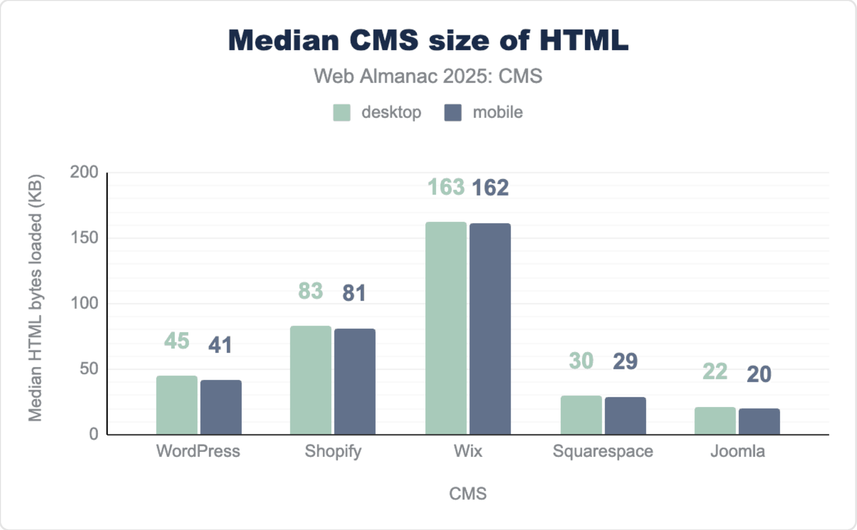


Figure 11.20. Median CMS size of HTML.

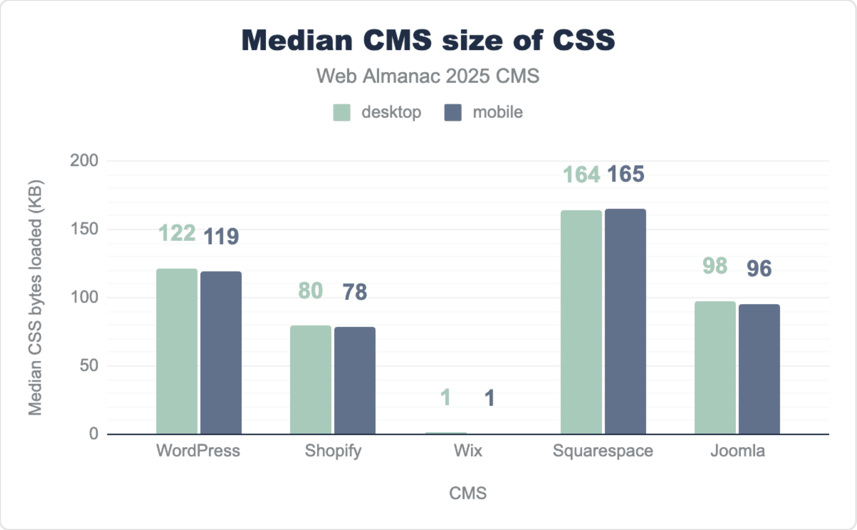


Figure 11.21. Median CMS size of CSS.

Block-based and component-driven systems increasingly generate CSS dynamically. When caching and reuse are done well, this can reduce unused styles. When handled poorly, it adds complexity and overhead. These trade-offs reflect the broader architectural themes explored earlier in the chapter.

Page weight, performance, and variance

The relationship between page weight and performance is not strictly linear, but it is strong. Heavier pages are more likely to miss Core Web Vitals thresholds, especially on mobile devices where network and CPU limitations magnify inefficiencies.

Across CMSs, page weight acts as a compounding factor rather than a single point of failure. Large image payloads, heavy JavaScript, and globally scoped assets accumulate over time, raising the chances of slow loads, visual instability, and sluggish interactions. Platforms that limit customization tend to reduce this risk through consistent defaults, while extensible systems shift more responsibility onto site owners and implementers.

Overall, page weight reinforces a core idea of this chapter: CMS platforms shape performance indirectly, by influencing how resources are assembled and delivered. As pages grow more complex, managing what goes into them—and how it's delivered—becomes critical to maintaining reliable performance

Emerging web APIs

As CMS platforms mature, browser capabilities play a bigger role in shaping performance and user experience. New web APIs increasingly offer improvements that work across frameworks and CMSs, shifting some optimization work from servers to the browser itself. These APIs do not remove underlying costs, but they can reduce perceived latency and improve responsiveness when used thoughtfully.

This section highlights browser-level capabilities that are starting to influence navigation speed, interaction responsiveness, and perceived performance across CMS-powered sites.

Speculation Rules API

Navigation latency remains one of the most noticeable sources of slowness on content-heavy, multi-page sites—the kind many CMSs power. The Speculation Rules API addresses this by letting browsers anticipate likely navigations and prepare pages in advance, via prefetching or prerendering.

Unlike traditional preloading, speculation rules allow developers to declare which navigations are likely and under what conditions the browser should act. Early usage shows measurable improvements in navigation performance, including lower First Contentful Paint and Largest Contentful Paint, and smoother page transitions for common browsing paths.

For CMSs, the key value of the Speculation Rules API is that it improves perceived navigation speed without changing backend rendering logic. Because speculative loading happens in the browser, it can reduce the impact of database latency, plugin overhead, or page builder complexity—particularly on sites with predictable navigation flows. The API is currently supported in Chromium-based browsers and safely ignored elsewhere, making it suitable for progressive enhancement.

Long Animation Frames API (LoAF)

While load performance has improved on many CMS platforms, interaction responsiveness is still a consistent pain point. The Long Animation Frames API builds on earlier long-task measurements by tracking delayed animation frames that block visual updates and user feedback. This directly feeds into understanding Interaction to Next Paint (INP).

LoAF shifts attention from individual JavaScript tasks to full-frame delays, making it easier to pinpoint what truly affects responsiveness. By highlighting which scripts, layout operations, or rendering steps most often cause slow frames, LoAF helps teams uncover issues that may not show up in initial page-load tests.

This is particularly relevant on CMS-driven sites, where sluggishness often results from cumulative complexity rather than a single blocking operation. Page builders, analytics, ads, and third-party tools can interact in ways that degrade responsiveness over time. LoAF enables teams to see these patterns in real-user monitoring and understand how interactions perform throughout a session, not just at load.

View Transitions API

The View Transitions API enables smoother visual transitions between page states by allowing the browser to animate content changes. While this is more about visual continuity than raw speed, it can significantly improve perceived performance—especially on content-heavy sites where users navigate frequently.

For CMS platforms, view transitions are important because they narrow the experiential gap between traditional multi-page sites and single-page applications. Combined with techniques like speculative loading, they let server-rendered, CMS-powered sites feel more fluid without adopting complex SPA architectures. Support is still uneven across browsers and usage is early, but initial adoption suggests growing interest in perceived performance improvements.

Priority Hints and Scheduling APIs

As pages become more complex, resource prioritization plays a bigger role in performance.

Priority Hints let developers indicate which resources (e.g., images, fonts, scripts) are more important, so browsers can adjust loading order under constrained conditions. When used well, this can improve key rendering milestones without shrinking total page weight.

Similarly, new scheduling APIs provide more control over when main-thread work happens, making it easier to defer less critical tasks in favor of user-visible interactions. On CMS-driven pages with many layers of functionality, these tools help reduce contention during key interaction moments.

Implications for CMS platforms

Together, these APIs signal a broader shift in how web performance improves over time. Instead of relying exclusively on backend changes or CMS-specific optimizations, platforms can increasingly lean on browser intelligence that adapts to user behavior and device capabilities. These APIs do not erase the costs of large payloads or heavy execution.

For CMS ecosystems, this reinforces a familiar pattern: browser APIs can smooth some of the variability introduced by extensibility, but they cannot eliminate core trade-offs between flexibility and predictability. Their impact depends on careful configuration, realistic assumptions about user behavior, and fit with the existing architecture.

As browsers continue to evolve, CMS platforms are likely to adopt these APIs gradually, using them to smooth navigation, improve responsiveness, and reduce perceived latency—without rebuilding their core designs. In that sense, emerging Web APIs act less as disruptive forces and more as multipliers for existing performance strategies.

Artificial Intelligence in CMS

AI capabilities are becoming more common across CMS platforms, but they currently affect content workflows more than core delivery performance. In most cases, AI is used to support how content is created, organized, and enriched, rather than how it is rendered or delivered to users.

Across ecosystems, AI adoption is uneven and mostly optional. AI typically appears as an assistive layer—through plugins, extensions, or external services—rather than as a fundamental change to CMS cores. Common uses include drafting and editing content, summarization, translation, image generation, tagging, and SEO metadata suggestions.

Most AI workflows happen during authoring or on asynchronous backends, so they do not significantly affect page weight, resource composition, or Core Web Vitals. Where AI is used at runtime—for personalization, recommendations, or dynamic content—its impact is more likely to show up indirectly, through additional JavaScript or network requests, depending on implementation.

One area where AI intersects with other CMS trends is editorial scale. By lowering the cost of uploading produced and updated content, and time to distribute to other channels, AI can lead to more pages, richer media, and more dynamic experiences. These shifts can intensify existing performance challenges around page weight, client-side execution, and caching, echoing patterns seen elsewhere in this chapter.

For now, AI in CMS platforms is best understood as a workflow accelerator rather than a performance optimizer. Its effects on user experience are mediated through editorial practices, governance, and implementation discipline—not through direct, measurable gains in loading speed or responsiveness. As AI moves closer to runtime systems, its performance implications may grow, but current evidence suggests its main impact remains operational.

CMS in the era of LLM search

The rise of large language model (LLM)-based search and answer engines is changing how CMS-generated content is evaluated and used. Unlike traditional search engines that focus on ranking pages, LLM systems extract, summarize, and recombine content. This increases the importance of structure, clarity, freshness, and machine-readable signals.

From an HTTP Archive standpoint, these changes show up as greater use of structured data, clearer content hierarchies, and more consistent semantic markup. CMS platforms that promote well-structured HTML, logical heading order, and rich metadata are better positioned for their content to be interpreted and reused accurately by LLM systems.

Indexing behavior also shifts. LLM-based tools tend to favor content that is regularly updated, clearly attributed, and easy to extract. CMSs that support rapid updates and programmatic metadata generation may gain indirect advantages. On the other hand, pages that rely heavily on client-side rendering, large JavaScript bundles, or delayed content hydration may be less visible, as extraction becomes slower or incomplete.

These dynamics align closely with the performance patterns discussed earlier. CMSs that already struggle with JavaScript bloat, slow LCP, or poor INP may face compounding discoverability issues as LLM systems increasingly reward efficiency and clarity. LLM search does not replace traditional indexing, but it amplifies existing trade-offs between flexibility, performance, and content structure.

Structured data evolution

Structured data has become a critical layer between CMS platforms, search engines, and AI-driven consumers. By 2025, it is no longer just a tool for rich snippets or enhanced search results. It increasingly serves as the machine-readable context that governs how content is interpreted, summarized, and surfaced across automated systems.

HTTP Archive data shows steady growth in the use of structured data across CMSs, though implementation quality varies widely. Platforms with native schema support tend to generate more consistent and predictable markup. Those that rely heavily on plugins or custom implementations show greater variability, mirroring broader performance patterns where flexibility often comes at the cost of consistency.

Structured data also interacts with performance and page weight. Poorly implemented schema can inflate HTML size and add DOM complexity without tangible benefits. Well-scoped structured data, by contrast, adds minimal overhead while making content far more interpretable. As CMSs automate more schema generation—often via AI-assisted tools—the risk shifts from not having structured data to having too much of it, where redundant or unnecessary markup adds weight without improving outcomes.

In a world of evolving search and discovery systems, structured data acts as a stabilizing signal. CMS platforms that validate schema, limit redundancy, and integrate structured data into core templates—rather than scattering it across ad hoc plugins—are more likely to produce durable, machine-friendly content. Increasingly, structured data maturity is becoming a differentiator not only for SEO, but for long-term content resilience as automated consumption grows.

Conclusion

The CMS landscape in 2025 reflects a mature, increasingly polarized web. HTTP Archive data shows that CMS platforms now power the vast majority of sites, while non-CMS sites keep shrinking as a share of the web. At the same time, adoption patterns, performance outcomes, and resource usage vary widely according to platform choice, hosting model, and implementation discipline.

Market share data confirms that WordPress is still the dominant CMS, powering more than 60% of CMS-driven sites. Platforms like Shopify continue to grow quickly in specific niches, especially ecommerce, while many other hosted builders show signs of slowing down. Rank-based analysis reveals that these trends are uneven: open-source CMSs remain overrepresented among high-traffic sites, while SaaS builders dominate the long tail of smaller properties. This suggests that CMS selection is increasingly driven by organizational needs and constraints, not just general popularity.

Performance data reinforces this split. Vertically integrated platforms tend to deliver more consistent Core Web Vitals, especially on mobile. Self-hosted CMSs show much wider variance, influenced by themes, plugins, and third-party integrations. Page weight and resource composition data further show that implementation decisions often matter more than platform defaults. Even within a single CMS, sites range from highly optimized to severely bloated, underscoring that performance is an ongoing practice, not a built-in guarantee.

Together, the Core Web Vitals and Lighthouse results highlight a recurring theme: CMS choice influences performance mainly through the constraints, defaults, and tooling it provides. Platforms with more controlled environments tend to produce steadier outcomes. More extensible systems, in turn, allow for both excellent and poor implementations.

These patterns do not point to simple winners and losers. Instead, they reflect trade-offs between flexibility, control, and predictability. As the CMS ecosystem diversifies, outcomes depend less on the logo at the bottom of the site and more on how well each platform's tools and constraints fit the site's complexity, the team's capacity, and long-term maintenance goals.

The rising importance of modern Web APIs, AI-assisted workflows, and LLM-based content consumption adds further pressure. Structured data, semantic markup, and efficient content delivery are no longer nice-to-have optimizations; they increasingly determine whether content is discoverable, interpretable, and performant across both traditional search and new AI interfaces. CMS platforms that promote consistent structure, limit unnecessary complexity, and embrace modern browser capabilities are better positioned to adapt.

Overall, the 2025 CMS ecosystem is defined less by raw growth and more by trade-offs. Flexibility often increases performance variance. Ease of use can produce heavier pages. Automation introduces new governance and quality challenges. CMS platforms will continue to evolve, but the dominant factor in real-world outcomes remains implementation. As the web places greater emphasis on speed, stability, and machine-readable content, CMSs will remain core infrastructure—not because they hide complexity, but because they determine how that complexity shows up at scale.

Author

Vahe Arabian

VaheSODP ahearabian https://www.stateofdigitalpublishing.com/

Vahe Arabian is a digital publishing entrepreneur, growth strategist, and the founder of State of Digital Publishing[436] and SODP Media[437]. He works with publishers, startups, and media organizations to drive sustainable growth across SEO, audience development, monetization, and product strategy. Vahe is a frequent speaker and writer on digital media trends, AI in publishing, and the evolving future of online journalism.

[436] https://www.stateofdigitalpublishing.com/
[437] https://www.sodpmedia.com/

Part III Chapter 12
Ecommerce

Written by Amandeep Singh
Reviewed by Barry Pollard
Analyzed by Amandeep Singh
Edited by Barry Pollard

Introduction

Ecommerce is no longer a special case on the web-it *is* the web. In 2025, buying journeys start in search results, social feeds, and live streams; they continue in voice assistants, messaging apps, and lean-back surfaces like smart TVs; and increasingly, they can be completed by AI agents acting on a shopper's behalf. An ecommerce website is still an online store that sells physical or digital products, but it now sits at the intersection of product pages, payments, performance, accessibility, and trust.

When building an online store, there are a few common platform models:

1. **Software-as-a-Service (SaaS)** platforms (e.g., Shopify) minimize the technical knowledge required to run a store by controlling the codebase and abstracting hosting.
2. **Platform-as-a-Service (PaaS)** solutions (e.g., Adobe Commerce / Magento) provide an optimized technology stack and hosting environment while still allowing full

code access.

3. **Self-hosted** platforms (e.g., WooCommerce, OpenCart) run on infrastructure managed by the merchant or their agency.
4. **Headless / API-first** platforms (e.g., Commercetools, Medusa) provide the commerce backend as a service, while the merchant owns the frontend experience and hosting.
5. **Agentic commerce (agent-ready commerce)** layers sit alongside (or on top of) the storefront: structured product data, inventory, policies, identity, and payment flows exposed through APIs and standards so assistants and AI agents can safely discover products and execute purchases-with clear user consent and guardrails.

Platforms may fall into more than one category. For example, some vendors offer SaaS, PaaS, and self-hosted options, and many headless builds still rely on SaaS backends under the hood. The important variables are who controls hosting, who controls the runtime and upgrade path, and how much freedom you have to change the frontend and backend.

Platform detection

We used a tool called Wappalyzer to detect technologies used by websites. It can detect ecommerce platforms, content management systems, JavaScript frameworks, analytics, and more.

For this analysis, we considered a site to be ecommerce if we detected either:

- Use of a known ecommerce platform, or
- Use of a technology that strongly implies an online store (for example, enhanced ecommerce analytics).

Limitations in recognizing ecommerce sites

Our methodology has limitations that affect accuracy.

- We can only recognize ecommerce sites when Wappalyzer detects an ecommerce platform or a strong ecommerce signal.
- Detecting a payment processor alone (for example, PayPal) is not sufficient to classify a site as ecommerce, because many non-store sites also take payments.
- If the store is hosted in a subdirectory, it may be missed. We crawl the home page

and one other page (the largest internal link) per site and per client (desktop and mobile).

- Headless implementations reduce platform detectability because the traditional fingerprints in HTML/JS often disappear.
- Apparent trends can be influenced by improvements or regressions in detection, not just industry shifts.
- Crawl geography matters: results may differ when sites redirect based on location.
- The underlying site set is drawn from Chrome's field data ecosystem, which biases toward sites visited by Chrome users.

Overall adoption

19.2%

Figure 12.1. Percent of mobile pages that are ecommerce sites.

In the 2025 dataset, we detected 19.9% of all analyzed desktop sites and 19.2% of all analyzed mobile sites where ecommerce sites.

That headline number is the first reminder that ecommerce is not just "a vertical"-it's a major slice of what real users experience on the open web.

Adoption by rank

In general, the most popular sites are more likely to be professionally engineered, heavily optimized, and backed by larger budgets.

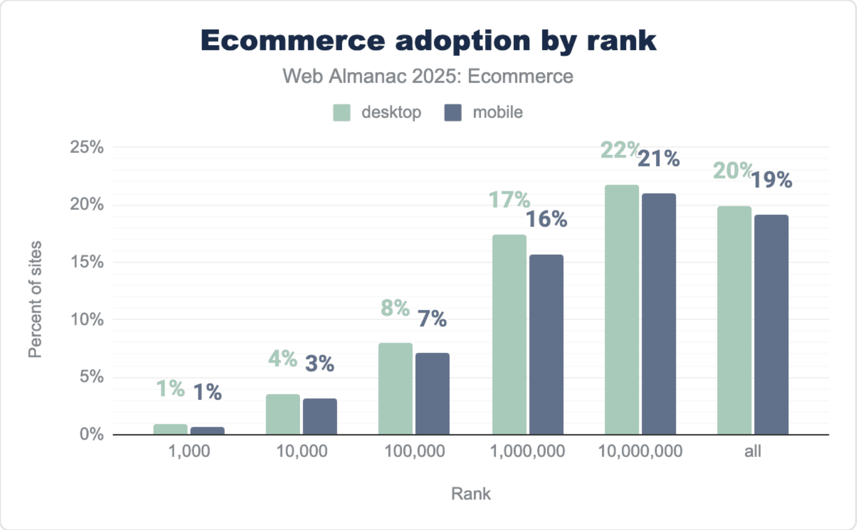


Figure 12.2. Ecommerce adoption by rank.

The pattern is consistent:

- Share of sites that are ecommerce increases as we include less-popular sites.
- At the very top of the web (top 1,000), ecommerce is present but rare.
- This grows over each rank.
- By the time you reach the top 10 million, roughly one in five sites is an online store.

Adoption trend

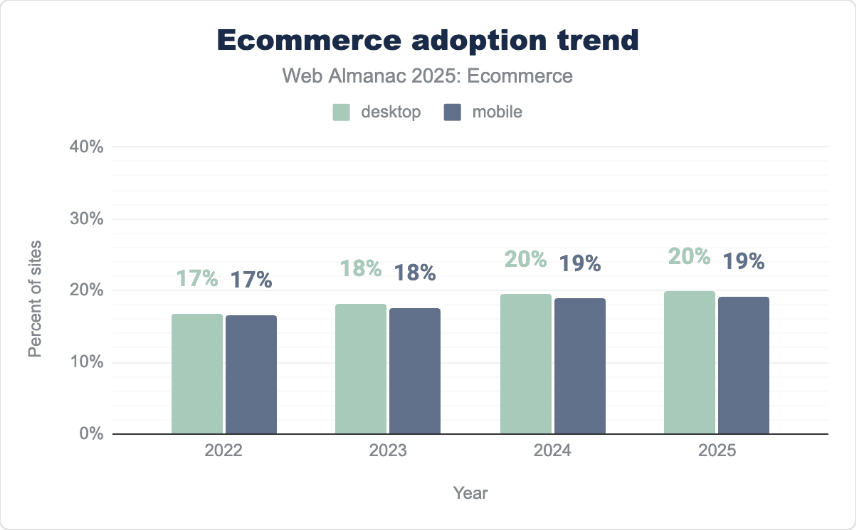


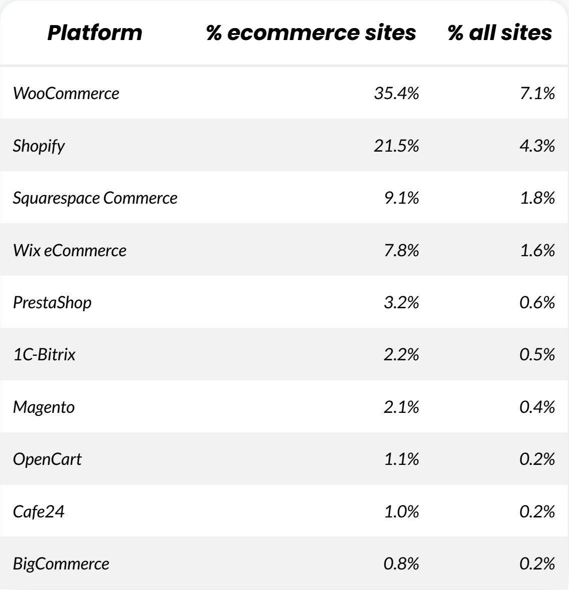
Figure 12.3. Ecommerce adoption by year.

Looking at the trend over time we see a gradual increase year on year.

Platform market share

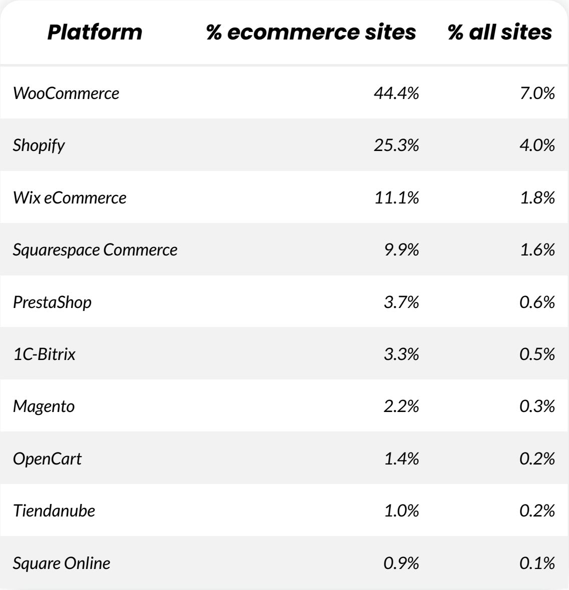
Across both desktop and mobile, the platform landscape remains top-heavy: a small set of systems account for the majority of detected stores, while a long tail of niche and regional platforms fills the rest.

The following tables show the share of detected ecommerce sites within ecommerce (platform market share), and also how often each platform appears across all sites in the dataset.



Platform	% ecommerce sites	% all sites
WooCommerce	35.4%	7.1%
Shopify	21.5%	4.3%
Squarespace Commerce	9.1%	1.8%
Wix eCommerce	7.8%	1.6%
PrestaShop	3.2%	0.6%
1C-Bitrix	2.2%	0.5%
Magento	2.1%	0.4%
OpenCart	1.1%	0.2%
Cafe24	1.0%	0.2%
BigCommerce	0.8%	0.2%

Figure 12.4. Most popular ecommerce platforms on desktop.



Platform	% ecommerce sites	% all sites
WooCommerce	44.4%	7.0%
Shopify	25.3%	4.0%
Wix eCommerce	11.1%	1.8%
Squarespace Commerce	9.9%	1.6%
PrestaShop	3.7%	0.6%
1C-Bitrix	3.3%	0.5%
Magento	2.2%	0.3%
OpenCart	1.4%	0.2%
Tiendanube	1.0%	0.2%
Square Online	0.9%	0.1%

Figure 12.5. Most popular ecommerce platforms on mobile.

Trends since 2024

If you zoom out to the last few years, the story is less about disruption and more about slow consolidation

- WooCommerce remains the largest ecosystem, staying roughly flat (about 36% → 36% of ecommerce sites from 2024 to 2025).
- Shopify continues to gain share (about 20% → 21%).
- Wix eCommerce is the fastest climber in the top 5 (about 7% → 8%).
- PrestaShop continues to trend down in share (about 4% → 3%).

In other words: the default choices are getting more default, and smaller open-source ecosystems are having to compete harder on developer experience, hosting simplicity, and performance out of the box.

Top platforms by tier

Different tiers have different top platforms.

Position	Top 1,000	Top 10,000	Top 100,000	Top 1,000,000	Top 10,000,000	All
1	*Amazon Webstore*	*Amazon Webstore*	*Shopify*	*Shopify*	*WooCommerce*	*WooCommerce*
2	*Magento*	*Salesforce Commerce Cloud*	*Magento*	*WooCommerce*	*Shopify*	*Shopify*
3	*Pattern by Etsy*	*SAP Commerce Cloud*	*Salesforce Commerce Cloud*	*Magento*	*Squarespace Commerce*	*Squarespace Commerce*
4		*Magento*	*Amazon Webstore*	*PrestaShop*	*PrestaShop*	*Wix eCommerce*
5		*Shopify/td>*	*WooCommerce/td>*	*1C-Bitrix/td>*	*Wix eCommerce/td>*	*PrestaShop*

Figure 12.6. Top platforms by rank tier (desktop).

Position	Top 1,000	Top 10,000	Top 100,000	Top 1,000,000	Top 10,000,000	All
1	*Magento*	*Amazon Webstore*	*Shopify*	*Shopify*	*WooCommerce*	*WooCommerce*
2	*Amazon Webstore*	*Salesforce Commerce Cloud*	*Magento*	*WooCommerce*	*Shopify*	*Shopify*
3	*Pattern by Etsy*	*SAP Commerce Cloud*	*Salesforce Commerce Clous*	*Magento*	*Squarespace Commerce*	*Wix eCommerce*
4		*Magento*	*Amazon Webstore*	*PrestaShop*	*PrestaShop*	*Squarespace Commerce*
5		*Shopify/ td>*	*WooCommerce/ td>*	*1C-Bitrix/td>*	*Wix eCommerce/ td>*	*PrestaShop*

Figure 12.7. Top platforms by rank tier (mobile).

- In the very top ranks, enterprise and bespoke ecosystems show up more often.
- In the broader web, the long-tail winners (especially WooCommerce) dominate by volume.

Top platforms by geography

Platform dominance changes by region because of language, local payment rails, agency ecosystems, and the historical footprint of vendors.

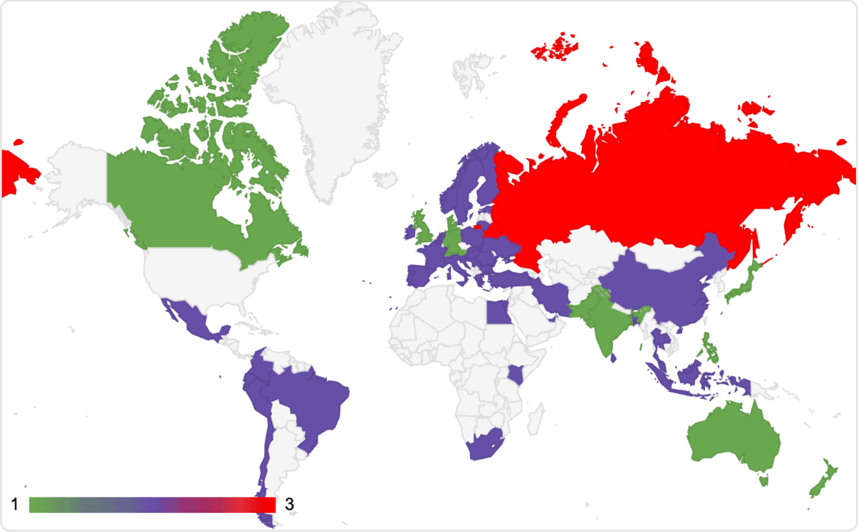


Figure 12.8. Top ecommerce platform by country in 2025.

The three leading platforms take the top spot in most countries: WooCommerce (violet), Shopify (green), and 1C-Bitrix (red).

- On desktop, WooCommerce is the most common platform in 42 of 59 geographies in our country-level view (excluding the ALL aggregate).
- On mobile, WooCommerce leads even more often: 71 of 89 (excluding the ALL aggregate).

There are also meaningful regional exceptions:

- 1C-Bitrix leads in parts of Eastern Europe and Central Asia (e.g., Russian Federation, Belarus, Kazakhstan, Kyrgyzstan).
- Tiendanube stands out in Argentina.
- Shoptet appears as a leader in Czechia.
- Cafe24 leads in South Korea.
- Salla shows up strongly in Saudi Arabia.

Core Web Vitals in ecommerce

Ecommerce sites are unusually sensitive to performance because every extra second compounds: slower category pages reduce product views; slower product pages reduce add-to-carts; slower checkout flows reduce conversion.

We use Core Web Vitals (CWV) field metrics to summarize real-user experience:

- **LCP (Largest Contentful Paint):** measures *loading* performance. It captures how quickly the main content becomes visible. In ecommerce, it often maps to hero imagery, product grids, and critical CSS/JS that blocks rendering.
- **INP (Interaction to Next Paint):** measures *responsiveness*. It captures the delay between a user action (tap/click) and the next visual update. It is sensitive to heavy JavaScript, third-party tags, and main-thread contention.
- **CLS (Cumulative Layout Shift):** measures *visual stability*. It captures how much content shifts as the page loads. It's especially relevant to ecommerce because late-loading product images, personalization widgets, and promo banners can cause mis-clicks.

A site is considered “good” on CWV when it passes all three thresholds.

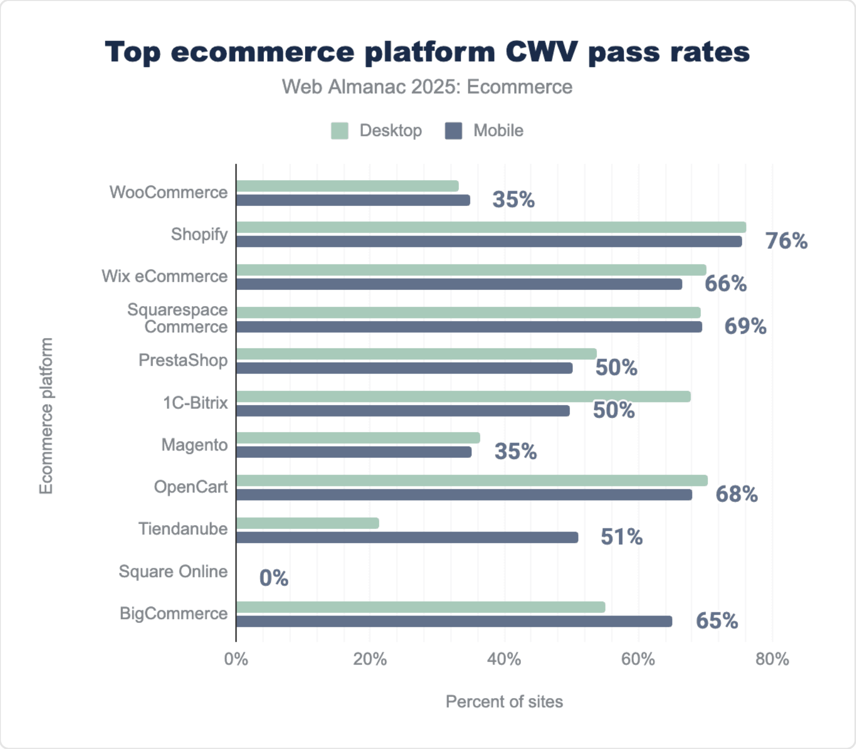
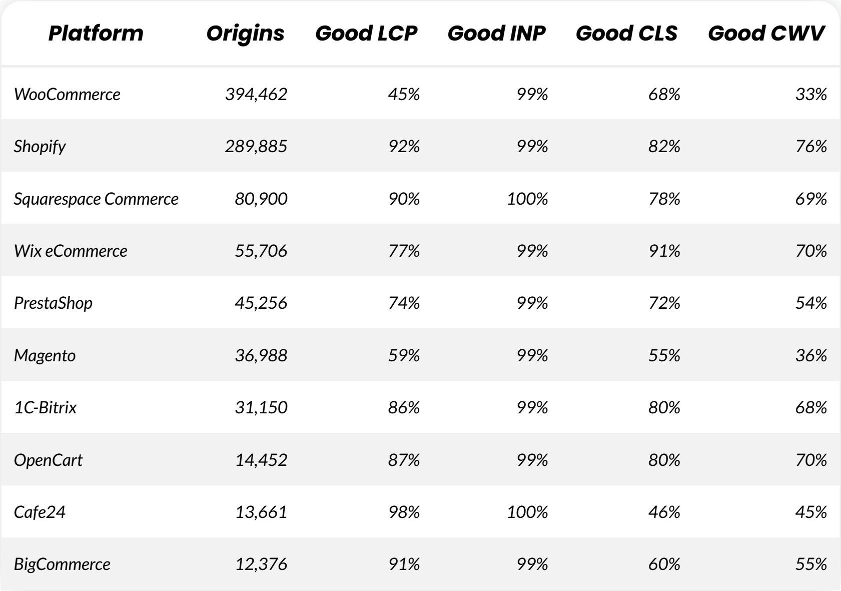


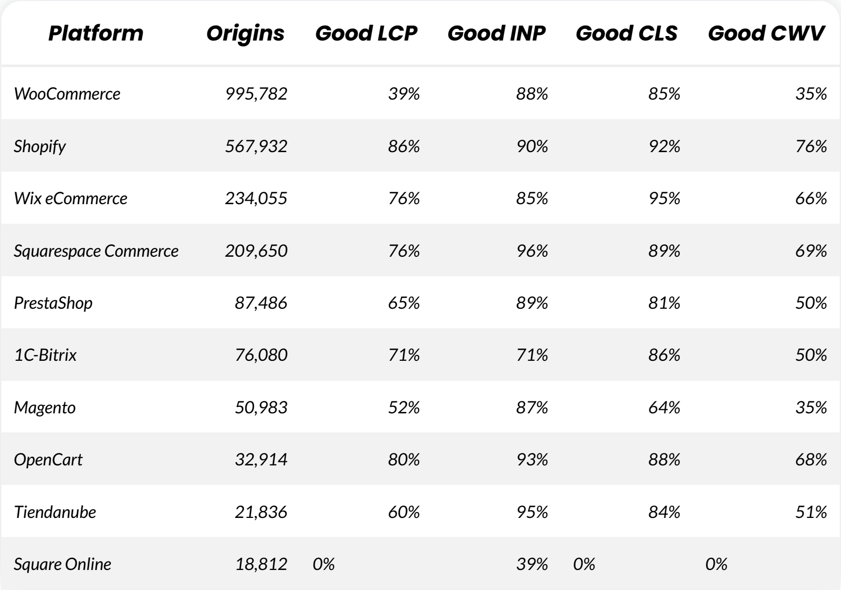
Figure 12.9. Top ecommerce platform CWV pass rates.

CWV by platform



Platform	Origins	Good LCP	Good INP	Good CLS	Good CWV
WooCommerce	394,462	45%	99%	68%	33%
Shopify	289,885	92%	99%	82%	76%
Squarespace Commerce	80,900	90%	100%	78%	69%
Wix eCommerce	55,706	77%	99%	91%	70%
PrestaShop	45,256	74%	99%	72%	54%
Magento	36,988	59%	99%	55%	36%
1C-Bitrix	31,150	86%	99%	80%	68%
OpenCart	14,452	87%	99%	80%	70%
Cafe24	13,661	98%	100%	46%	45%
BigCommerce	12,376	91%	99%	60%	55%

Figure 12.10. Top eCommerce platforms Core Web Vitals good rates (desktop).



Platform	Origins	Good LCP	Good INP	Good CLS	Good CWV
WooCommerce	995,782	39%	88%	85%	35%
Shopify	567,932	86%	90%	92%	76%
Wix eCommerce	234,055	76%	85%	95%	66%
Squarespace Commerce	209,650	76%	96%	89%	69%
PrestaShop	87,486	65%	89%	81%	50%
1C-Bitrix	76,080	71%	71%	86%	50%
Magento	50,983	52%	87%	64%	35%
OpenCart	32,914	80%	93%	88%	68%
Tiendanube	21,836	60%	95%	84%	51%
Square Online	18,812	0%	39%	0%	0%

Figure 12.11. Top eCommerce platforms Core Web Vitals good rates (mobile).

A few patterns show up repeatedly:

- INP is generally strong on desktop across most major platforms, suggesting that modern JS stacks and browser improvements are helping responsiveness.
- LCP is the biggest differentiator-platforms that ship fast themes and tightly controlled app ecosystems tend to score better.
- WooCommerce has scale, but not automatic speed: its CWV pass rates lag behind SaaS-heavy ecosystems, which is consistent with its infinite customization nature.

Lighthouse

Lighthouse is the HTTP Archive's lab-based audit. Unlike Core Web Vitals (field data), it runs in a controlled environment (simulated device, throttled network/CPU) and produces scores for Performance, Accessibility, SEO, and Best Practices:

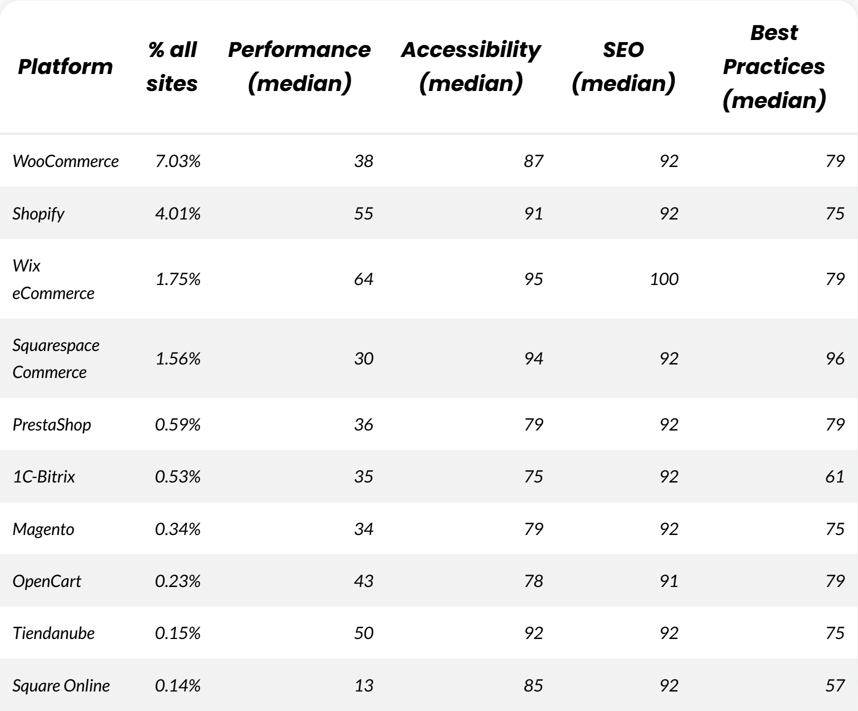
- **Performance**: Lighthouse Performance is a lab score (0–100) summarizing load and responsiveness under a controlled test profile. It's most useful for relative comparisons across platforms.
- **Accessibility**: Lighthouse Accessibility is based on automated checks (it cannot catch everything), but it's a useful baseline signal for common issues like missing labels, low contrast, and incorrect semantics.
- **SEO**: The Lighthouse SEO score reflects technical SEO fundamentals (e.g., title/meta, basic crawlability signals). High medians are common because these checks are straightforward to pass.
- **Best Practices**: Best Practices is a grab bag of security and reliability checks (HTTPS, safe JS patterns, modern APIs). It often reflects platform defaults and theme quality.

Lighthouse is useful for comparisons across large sets of sites because it standardizes the test profile, but it won't perfectly match what real users experience since field data reflects the real mix of devices, networks, geographies, and user behavior.

Median Lighthouse scores by platform

Platform	*% all sites*	*Performance (median)*	*Accessibility (median)*	*SEO (median)*	*Best Practices (median)*
WooCommerce	*7.10%*	*61*	*87*	*92*	*78*
Shopify	*4.32%*	*71*	*90*	*92*	*74*
Squarespace Commerce	*1.84%*	*65*	*94*	*92*	*96*
Wix eCommerce	*1.57%*	*88*	*96*	*100*	*78*
PrestaShop	*0.64%*	*59*	*78*	*92*	*81*
1C-Bitrix	*0.47%*	*58*	*75*	*92*	*59*
Magento	*0.41%*	*60*	*78*	*92*	*74*
OpenCart	*0.23%*	*63*	*79*	*91*	*78*
Cafe24	*0.20%*	*39*	*69*	*85*	*56*
BigCommerce	*0.15%*	*72*	*81*	*92*	*74*

Figure 12.12. Most popular ecommerce platforms Lighthouse scores (desktop).



Platform	% all sites	Performance (median)	Accessibility (median)	SEO (median)	Best Practices (median)
WooCommerce	7.03%	38	87	92	79
Shopify	4.01%	55	91	92	75
Wix eCommerce	1.75%	64	95	100	79
Squarespace Commerce	1.56%	30	94	92	96
PrestaShop	0.59%	36	79	92	79
1C-Bitrix	0.53%	35	75	92	61
Magento	0.34%	34	79	92	75
OpenCart	0.23%	43	78	91	79
Tiendanube	0.15%	50	92	92	75
Square Online	0.14%	13	85	92	57

Figure 12.13. Most popular ecommerce platforms Lighthouse scores (mobile).

A few high-level patterns:

- SaaS storefronts tend to cluster higher on the Performance category (especially on desktop), consistent with tighter control over themes and defaults.
- Accessibility medians are generally strong across top platforms, but medians can hide long-tail variance.
- SEO and Best Practices scores are high for most platforms-where teams usually win or lose is performance and implementation discipline, not basic technical SEO.

Payment providers

Payments are where ecommerce becomes real. They also represent a major dependency surface area: third-party scripts, redirects, fraud tooling, and compliance constraints.

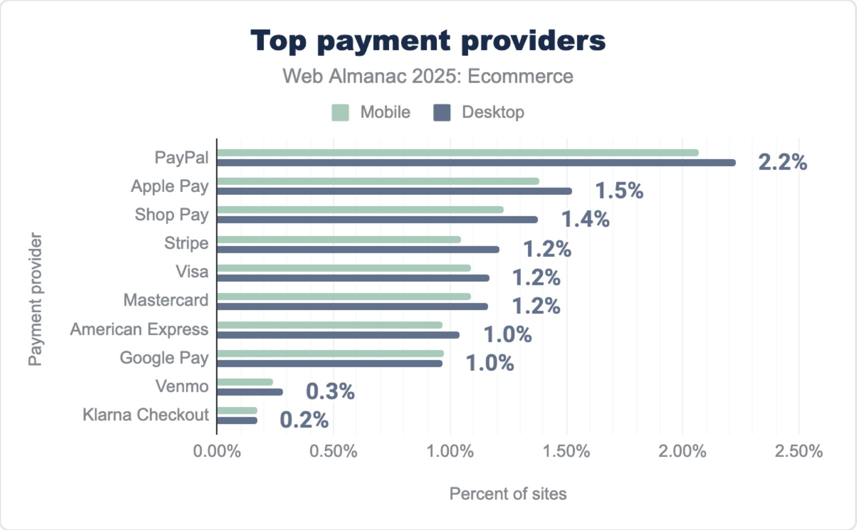


Figure 12.14. Payment provider distribution on ecommerce sites in 2025.

The top 10 payment technologies account for the bulk of payment detections on both devices-another reminder that payments consolidate quickly. However it's worth noting that no single provider or two stands out as overall dominant.

What changed since 2022?

The most noticeable trend is that PayPal's share of payment detections declines, while wallets and processor-first ecosystems grow:

- PayPal falls from ~39% of payment detections in 2022 to ~30% in 2025 (desktop and mobile both show this pattern).
- Stripe and Google Pay gain share over the same period.
- Apple Pay and Shop Pay stay relatively stable at high levels.

This does not mean PayPal is dying. It means the payment layer is becoming more diversified-especially as native wallets, link-based checkout, and platform-native accelerators become standard.

Payment providers by geography

70 out of 83

Figure 12.15. Countries in which PayPal is the top payment provider.

- On mobile, PayPal is the top payment provider in 70 of 83 geographies.
- On desktop, leadership is more split: Stripe leads in 31 geographies while PayPal leads in 22.

Notable examples:

- Apple Pay leads in New Zealand (both desktop and mobile).
- Braintree stands out in Taiwan.
- Several African and Middle Eastern markets show Stripe as the most common top provider in this dataset (e.g., Nigeria, Kenya, UAE).

Conclusion

Ecommerce in 2025 remains both concentrated and diverse. A handful of platforms account for most detected ecommerce sites-led by WooCommerce and Shopify-while a long tail of regional and niche systems continues to matter in specific markets. Website rank adds another layer: enterprise-oriented platforms show up disproportionately in higher-traffic tiers, while long-tail sites skew toward easier-to-adopt and lower-cost solutions.

Performance remains a differentiator, not a footnote. Field metrics (Core Web Vitals) and lab audits (Lighthouse) both show that tighter platform control can correlate with better median outcomes, but the gap is not destiny-self-hosted and heavily customized stacks can perform well when engineering discipline is strong. Payment technologies also consolidate quickly: a small set of providers dominates detections, while wallets and processor-first ecosystems keep gaining share.

The next chapter of ecommerce is not just which platform, but which channels: voice, live commerce, and agentic commerce are pushing stores to become faster, more accessible, and more machine-consumable. The winners will be the ones who treat catalog quality, performance, and trust as product features-because increasingly, the shopper may not be a human clicking around a page at all.

Author

Amandeep Singh

𝕏 @amanvirdi amandeepsinghvirdi amandeepsinghvirdi https://byaman.com/

Amandeep Singh has been developing for the web since 2009 and writes about front end development, UI/UX, Shopify, BigCommerce, WordPress, and programming at byaman.com[438]. He is a writer, mentor, and speaker.

[438] https://byaman.com/

Part IV Chapter 13
Page Weight

Written by Richard Barret and Jamie Indigo
Reviewed and analyzed by Dave Smart
Edited by Montserrat Cano

Introduction

In the early days of the web, every byte was a luxury. Developers spent hours dither-mapping GIFs and hand-optimizing scripts to ensure pages could crawl through 56k modems. Today, in an era of gigabit fiber and 5G dominance, that scarcity mindset has largely vanished. However, as our “bandwidth pipes” have grown wider, the content we push through them has expanded to fill the space.

Page weight—the total volume of bytes transferred to a user’s device—remains one of the most critical metrics for understanding the health of the web. While it is tempting to view a few extra megabytes as a negligible cost of modern rich experiences, the reality is far more complex.

What is page weight?

Page weight (also called page size) is the total volume of data—measured in kilobytes (KB) or megabytes (MB)—that a user must download to view a specific webpage.

When you navigate to a URL, your browser doesn't just download one file; it sends dozens or even hundreds of requests for various assets required to make the page look and function correctly. The sum of all these "shipped" bytes constitutes the page weight.

A modern webpage is an assembly of several different types of resources. Each contributes to the total "heaviness" of the site:

- **Images & video:** High-resolution photos, background videos, and GIFs can quickly balloon page size.
- **JavaScript:** Files that provide interactivity (like menus, tracking pixels, or animations). While often smaller in KB than images, JavaScript is "heavy" because the browser has to spend significant CPU power parsing and executing it.
- **CSS:** Stylesheets that determine the layout, colors, and fonts of the page.
- **Fonts:** Custom web fonts can add several hundred KB if multiple weights (bold, italic, light) are used.
- **HTML:** The structural code of the page, which is usually the smallest part of the total weight.
- **Third-party scripts:** These include ads, analytics, and social media widgets that are fetched from other servers.

Not all bytes are created equally. In this chapter we will explore page weight by bytes and by request volume of file types.

Why weight matters

Page weight is a direct proxy for performance and accessibility. A "heavy" page creates several negative ripple effects:

1. **The performance gap:** Larger payloads can require more CPU cycles and use more device memory to parse and render, often leading to sluggish experiences on low-end devices, regardless of connection speed.
2. **The economic toll:** In many parts of the world, data is a metered commodity. A 5 MB page isn't just a slow experience; it is an exclusive one. If a user cannot afford the data to load your page, your site is, by definition, inaccessible to them.
3. **The accessibility barrier:** If a page is "heavy," it doesn't just load slowly—it becomes physically and cognitively harder to use. Excessive page weight creates significant

inequities, heavily penalizing users who rely on less powerful devices or expensive, slow connections with limited data caps. Refer to the Accessibility chapter to learn more about how page weight is a silent but significant barrier to entry for millions of users with disabilities.

4. **The environmental impact:** Every megabyte transferred requires energy—from data centers to cooling systems to the device in the user's hand. As the web grows, so does its carbon footprint.
5. **Speed & SEO:** Heavier pages take longer to load, especially on slower connections. Google uses page speed (via Core Web Vitals) in their core algorithm, meaning bloated pages can rank lower in search results. See also the SEO chapter.

The weight effects can be divided into three main categories: storage, transmission, and rendering.

Storage

Storage refers to how assets (images, scripts, HTML) sit on a web server or CDN. At this stage, page weight is about file size on disk.

- **Compression at rest:** Developers often store files in highly compressed formats (like WebP for images or Brotli-compressed text). A 1 MB file can be stored as 300 KB.

- **The database bottleneck:** For dynamic sites, the "weight" starts with database queries. If a server has to retrieve 2 MB of raw data from a database to generate one page, the initial response time (Time to First Byte (TTFB)[439]) increases before a single byte is even sent.

- **The cost:** Inefficient storage doesn't just affect speed; it increases hosting costs and the carbon footprint of the data center.

Transmission

Transmission is the process of moving those stored files across the internet. This is where network constraints turn page weight into a performance barrier.

- **Transfer size versus actual size:** Thanks to "on-the-fly" compression (like Gzip), the number of bytes sent over the wire is often much smaller than the original file size.

[439] *https://web.dev/articles/ttfb*

- **Latency and round trips:** It's not just about how much data is sent, but how many files. Each separate file requires a "round trip" to the server. A page with 50 small images (totaling 1 MB) can actually feel slower than a page with one large 2 MB image because of the transmission overhead of 50 separate requests.
- **The bottleneck:** On mobile 4G/5G, signal interference can cause "packet loss." The heavier the page, the more likely a packet will drop, forcing the browser to ask for it again and causing a visible hang.

Rendering

Rendering is what happens once the data arrives. This is the most misunderstood part of page weight: Once a file is downloaded, it must be "unpacked" and processed by the device.

- **Memory inflation:** An image might only take up 200 KB during transmission, but once the browser "renders" it, it must be decoded into raw pixels in the device's RAM. That 200 KB file can easily take up 5 MB of memory.
- **The JavaScript tax:** This is the "heaviest" part of rendering. 100 KB of an image is just pixels, but 100 KB of JavaScript is work. The CPU must parse, compile, and execute that code. On a low-end smartphone, this "weight" can freeze the screen for several seconds.
- **DOM complexity:** Every HTML tag adds a "node" to the browser's memory. A page with 5,000 nodes (a "heavy" DOM) will make scrolling feel laggy, regardless of how fast the internet connection is.

Websites may be changing their rendering strategies, sometimes prompted by rethinking how websites are accessed and consumed by the growing AI chatbots and other large language model tools. Not all of these changes are aimed at AI crawler accessibility. Some testing has been done to identify the technical requirements for AI crawler accessibility, as these may vary between crawlers. We do know that all key information should be present in the initial raw HTML, since AI crawlers do not render JavaScript[440].

Given the growing awareness of these factors, we could expect broader adoption of such strategies, including a reduction in the use of JavaScript files in future editions.

[440] *https://vercel.com/blog/the-rise-of-the-ai-crawler*

What are we shipping?

In this chapter, we analyze trillions of bytes across millions of websites to answer a fundamental question: Are we getting more value out of our payloads, or are we simply succumbing to "weight creep"? We will explore the dominance of media—images and video—which continue to claim the lion's share of the transfer budget, and the steady rise of JavaScript, which carries a performance tax far heavier than its file size suggests.

Request bytes

Median page weight over time shows that page size growth is accelerating, since October 24 there has been a noticeable upward trend, in particular for mobile devices. This is shown on a grander scale with the home page, which on average is close to 45.8% larger than inner pages.

Home page page weight has similarly accelerated but earlier, around early 2023/late 2022.

This does have wider implications:

Page weight implications for AI

There is an increase in energy consumption and computation costs, heavier pages mean more bandwidth, rendering and CPU load per visit, which all increase the energy footprint of web crawling and indexing.

This results in slower or shallower crawling, as AI-driven crawlers that try to "understand" the web by performing actions such as extracting entities or content summarization, may limit the depth or frequency of crawling to manage their computation budgets.

This could lead to a bias towards lighter-footprint web content with faster loading or more semantically clear sites, as they are easier to parse and therefore model. Sites overloaded with JavaScript and client-side rendering could end up being underrepresented in large-scale training sets since the crawlers AI models employ do not render. Finally, AI may skip or truncate what they see on slow or very heavy pages as performance becomes more critical.

Page weight implications for users

As pages get heavier, more sites will likely fall short on LCP and INP, which could directly reduce their visibility in search results. More importantly, this can make a frustrating user experience, potentially leading to lower conversion rates, as convenience has become an increasingly important factor.

Heavier pages widen the digital divide between users on slower connections or lower-end devices, as they would be disproportionately impacted by these increases. This highlights the importance for search marketers to optimize websites with these users in mind.

Page weight implications for publishers

Slower performance correlates with lower online traffic, conversion rates and ad views. Ad tech and analytics scripts drive additional weight, creating additional friction between user experience and monetization.

In summary: AI systems may adapt by scraping less and summarizing more, filling gaps with "hallucinations". This could reduce user confidence in brands and lead to lower online traffic and conversions. User behavior could shift away from traditional page visits toward AI-mediated browsing experiences, as convenience continues to play a key role in how people access online information.

Page weight over time

Since the inception of the Web Almanac, we've consistently observed two trends:

1. The total volume of requests increases.
2. Desktop page loads result in more requests than mobile page loads.

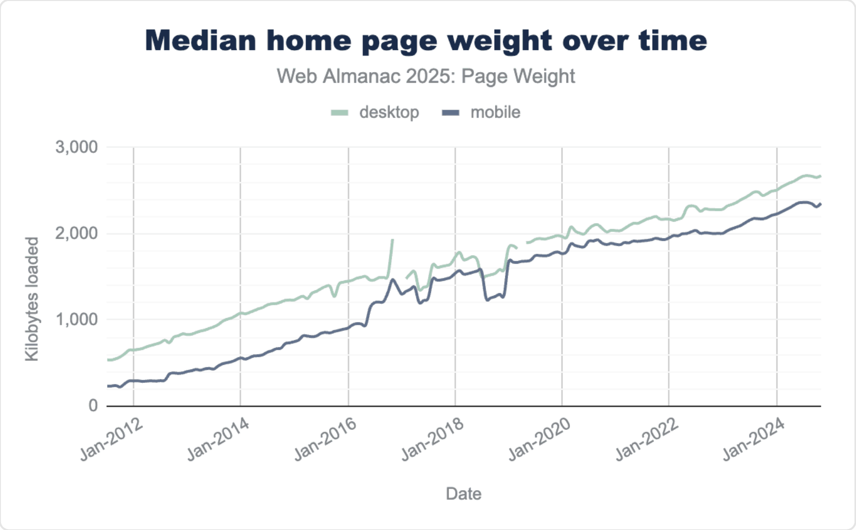


Figure 13.1. Median home page weight over time.

In July 2015, the median mobile home page was a meager 845 KB. As of July 2025, the same median page is now 2,362 KB. The page decade brought a 202.8% increase. The median desktop home page saw a 110.2% increase in the same time frame.

Year over year, the median home page size grew 7.8% to 2.7 MB. The median mobile home page is 2.6 MB, growing 8.4% from 2.4 MB in 2024. In 2025, the median desktop page size reached 2.9 MB, up 7.3% from 2024's median of 2.7 MB.

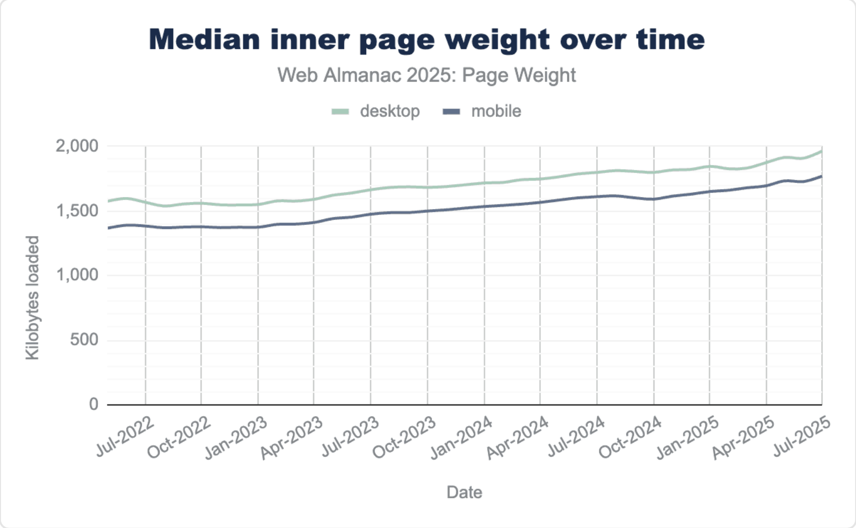


Figure 13.2. Median inner page weight over time.

Inner page weight continued to increase, up 9.5% year over year. Since data for tracking inner page weight began in 2022, the median mobile inner page weight has increased by 27.8% to 1.8 MB, while desktop has grown by 25.2% to reach 2 MB during the same period.

Page weight in bytes

Every update to page functionality introduces additional weight and file types designed to enhance performance while preserving core features. We've examined common file types, their frequency, and response sizes to gain a clearer understanding of their implementation, including comparisons across different devices and page types.

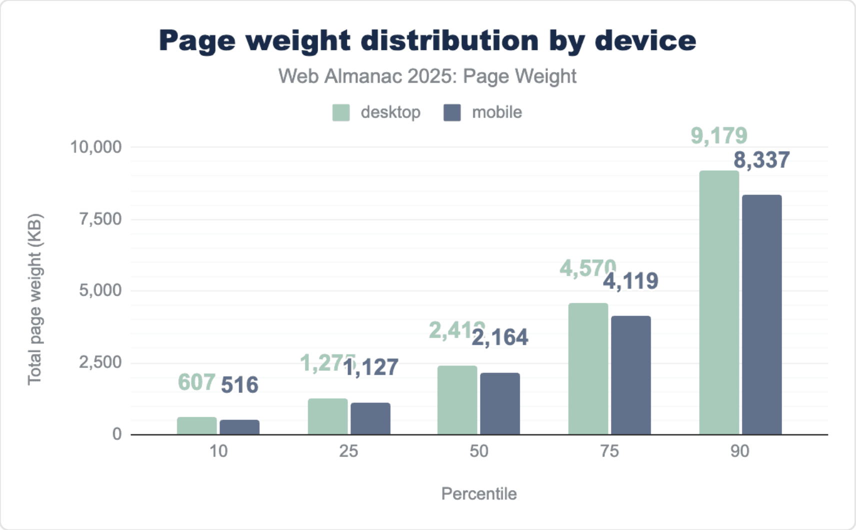


Figure 13.3. Page weight distribution by device.

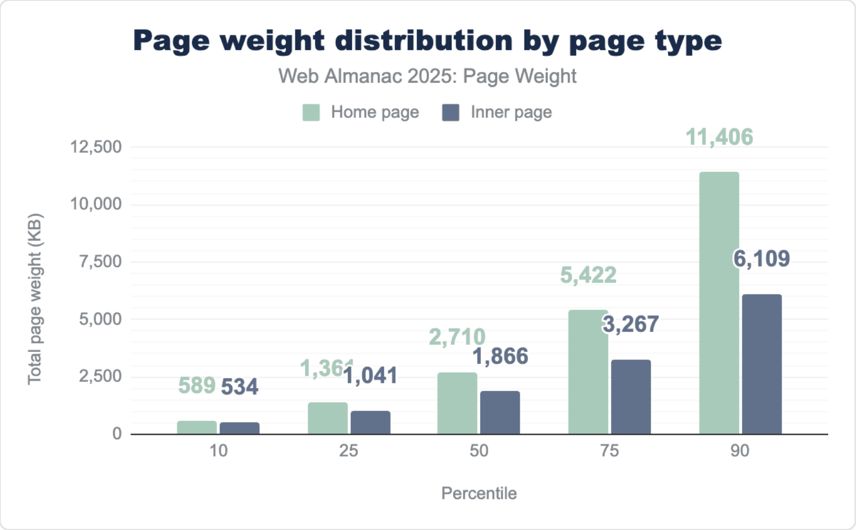


Figure 13.4. Page weight distribution by page type.

Median page by the bytes

To better understand the web, we can examine the 50th percentile. The median represents the typical value and context to study relative page weights.

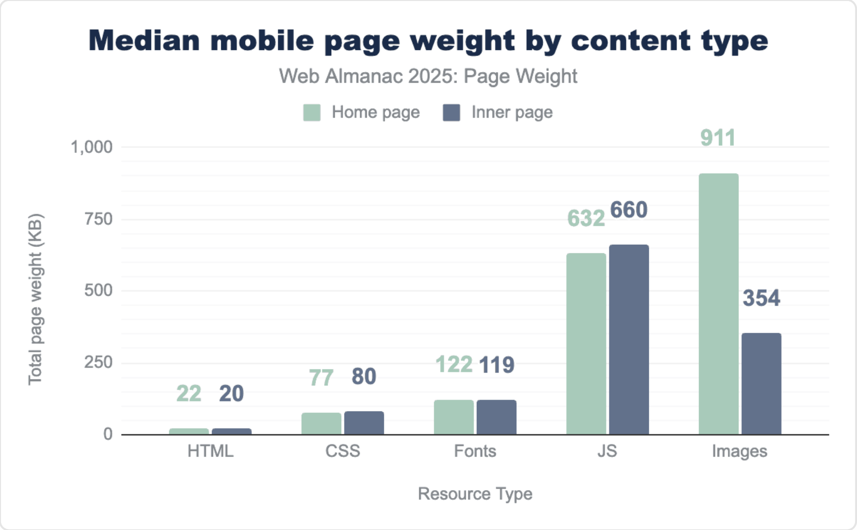


Figure 13.5. Median mobile page weight by content type.

In 2025, the median mobile home page used:

- 22 KB of HTML resources
- 77 KB of CSS resources
- 122 KB of fonts
- 632 KB of JavaScript
- 911 KB of images

Inner pages were similar, with the exception of images, which decreased from 911 KB to 354 KB.

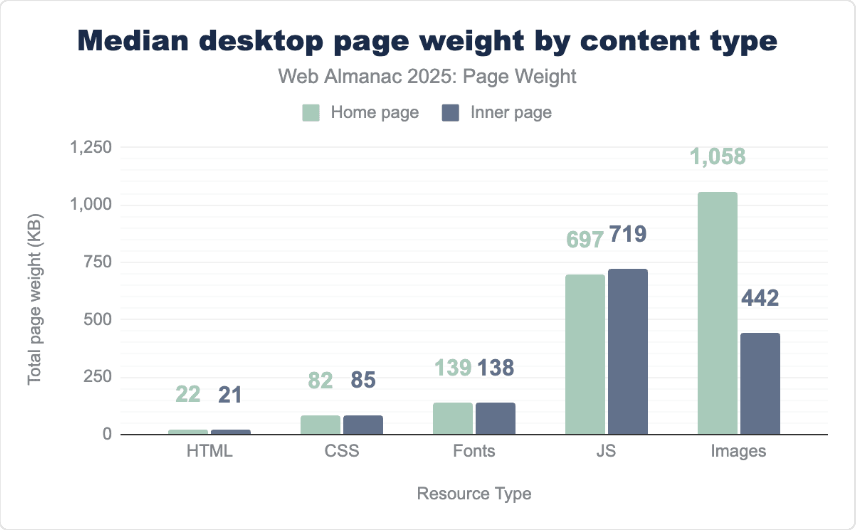


Figure 13.6. Median desktop page weight by content type.

The median desktop page's weight by resource type matched mobile, though the resources were slightly larger in KB. In 2025, the median desktop home page used:

- 22 KB of HTML resources
- 82 KB of CSS resources
- 139 KB of fonts
- 697 KB of JavaScript
- 1,058 KB of images

For both device types, images saw the greatest gap in kilobytes with 1.06 MB used on home pages and 442 KB on inner pages.

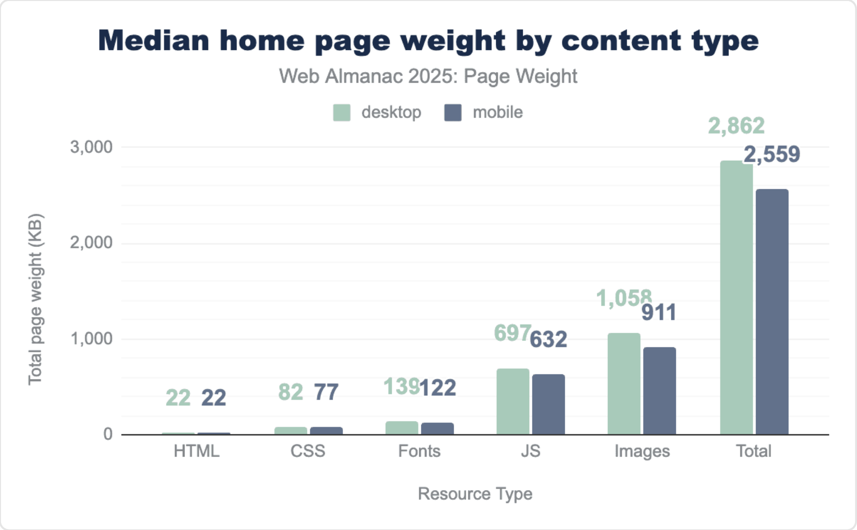


Figure 13.7. Median home page weight by content type

The median home page in 2025 was 2.86 MB on desktop and 2.56 MB on mobile. Images accounted for the most bytes on both mobile and desktop, followed by JavaScript and fonts.

Response size by content type

Not all file types are created equally—or equally sized. File types define how data is stored and encoded, telling a program how to open and display a file's contents.

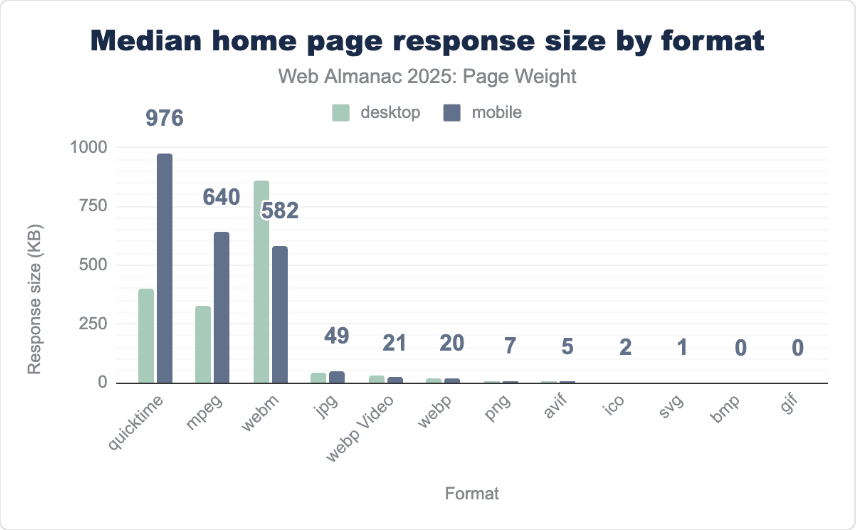


Figure 13.8. Median home page response size by format.

Mobile devices are typically more challenging for performant experiences. Their smaller, varied screen sizes and weaker connections mean that content can take longer to load. This is what makes it so strange that mobile home pages also had some of the heaviest files.

The largest file types, QuickTime and MPEG, are used for video. The median mobile home page used 976 KB of QuickTime, 244% more than desktop's 400 KB. Similarly, the median mobile home page used 196% more MPEG bytes than desktop. The third-largest file type was WebM, a video format optimized for faster loading and lower bandwidth usage. Despite being ideal for mobile connections, more WebM bytes were shipped for desktop.

Upon further investigation, we were surprised to find that video files were particularly susceptible to duplicate downloads where browsers were making multiple requests, often with different range headers. This inflated the bytes for videos, and seemed worse on mobile—perhaps due to slower network downloads? We could find no documentation for this behavior, but would be interested to hear more if any readers can explain this.

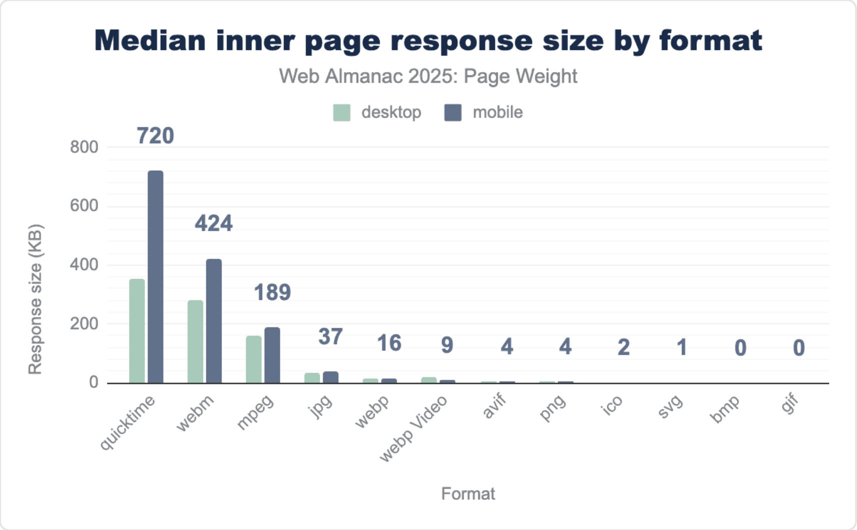


Figure 13.9. Median inner page response size by format

For median inner pages, QuickTime files remained the largest file type in bytes. Mobile saw 720 KB compared, 202% the 352.6 KB found on desktop. WebM surpassed MPEG as the second-largest file type for inner pages, at 423.6 KB on mobile and 280.8 KB on desktop. These were followed by MPEG, with more even distribution between device types (161.6 KB desktop; 188.8 KB mobile) than seen on home pages.

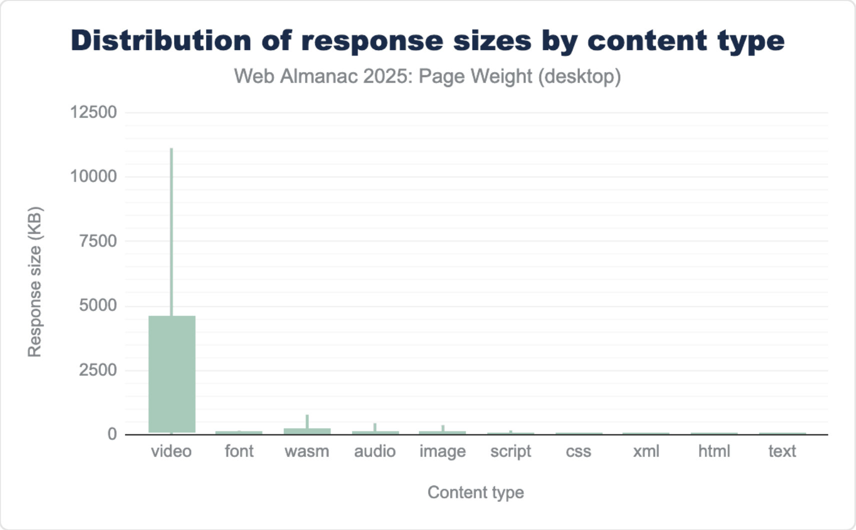


Figure 13.10. Distribution of response sizes by content type.

Videos, by their very nature, tend to be the largest of files. They also have the largest range in size—from 7 KB at the 10th percentile up to 1GB at the 100th. That range is only surpassed by image file types which saw 0 KB on the 10th percentile on 1.2GB at the 100th. XML files saw the smallest range with the 10th to 50th percentiles having 0 KB of the file type and the most extreme desktop pages having 52.1 MB at the 100th.

Image bytes

Image bytes refer to the binary data required to render visual elements such as photographs, icons, and illustrations. Unlike text-based files (like HTML, CSS, and JavaScript), which are highly efficient and easily compressed, image data is inherently dense. The "weight" of these bytes is determined by three primary factors:

1. **Resolution:** The total pixel count of the asset.
2. **Encoding:** The mathematical method used to store the data (for example, JPEG, PNG, or modern formats like WebP and AVIF).
3. **Compression:** The degree to which redundant data is removed to shrink the file size, often at the cost of visual fidelity.

The accumulation of image bytes creates a "heavy" page, which directly correlates with increased latency and slower Largest Contentful Paint (LCP) scores. For users on mobile devices or limited-bandwidth connections, high image byte counts can lead to prohibitive load times and increased data costs.

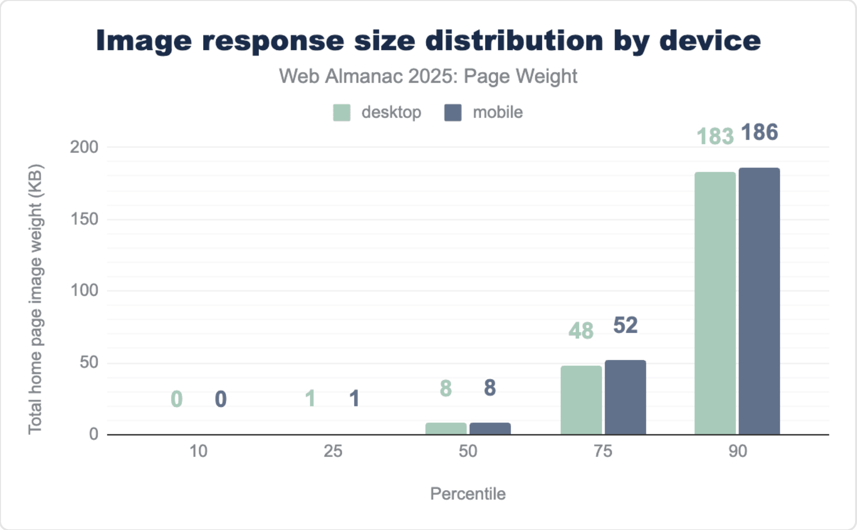


Figure 13.11. Image response size distribution by device

At the 50th percentile, individual image files were 8 KB on both mobile and desktop. It is worth noting that many sites use small images, known as tracking pixels, as part of their analytics. These tiny, often invisible 1x1 pixel images are embedded in web pages and trigger a server request when loaded. Since these files are not differentiated from standard, user-focused image assets, they may cause the median image size to appear smaller than expected.

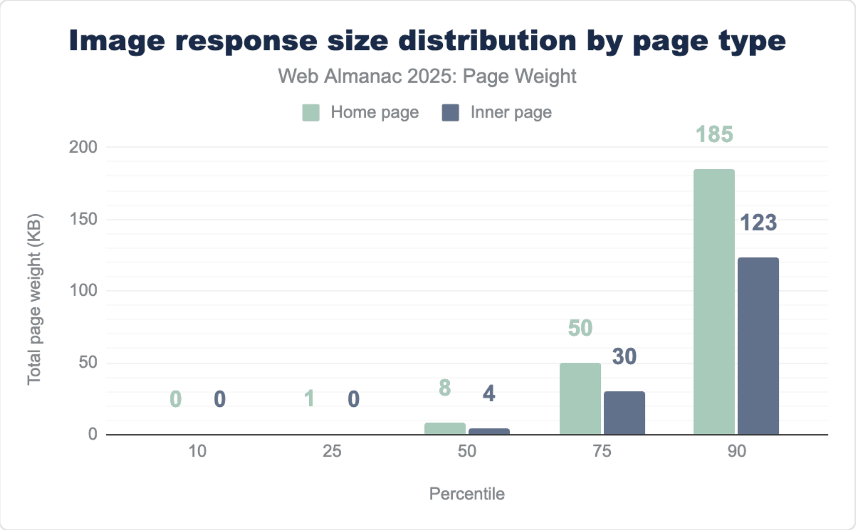


Figure 13.12. Image response size distribution by page type.

Inner pages had smaller individual image file sizes than home pages with a median size of 4 KB. At the 90th percentile, home page images were 185 KB while inner images were 123 KB.

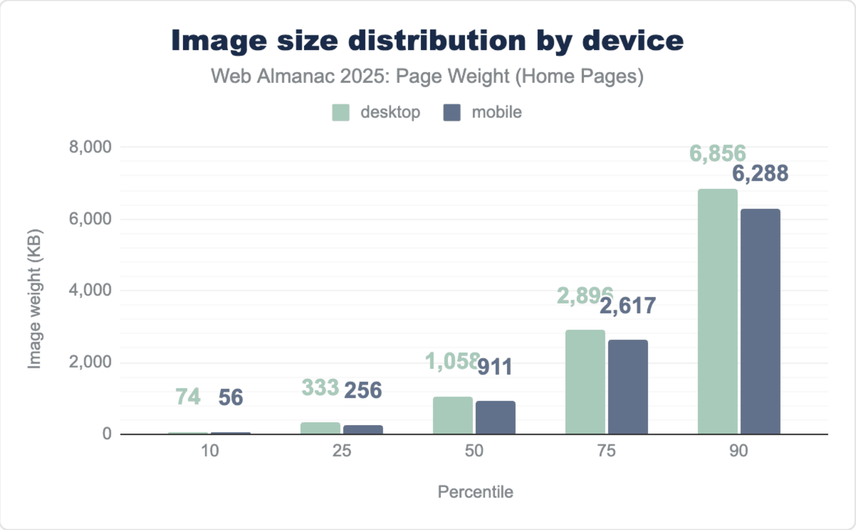


Figure 13.13. Image size distribution by device

A web page rarely uses a single image. When aggregated together, the median desktop page used 1,059 KB of images. Mobile pages used 911 KB. Desktop pages consistently loaded more total image bytes.

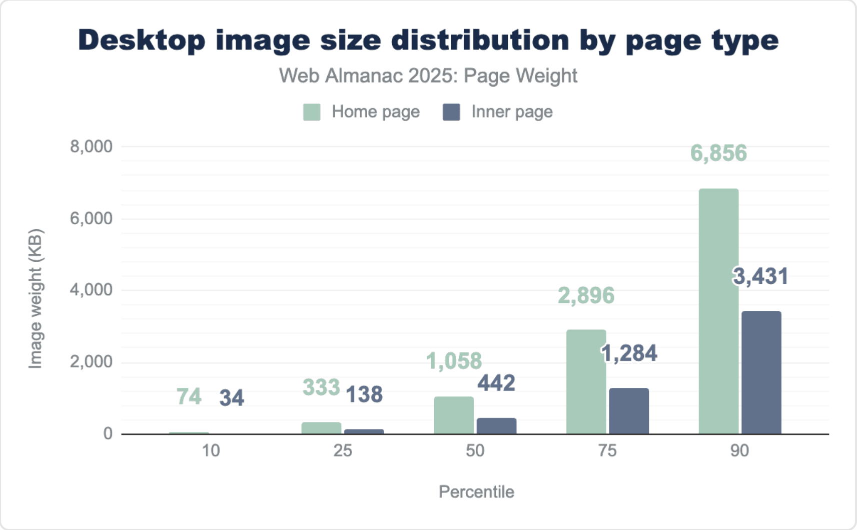


Figure 13.14. Desktop image size distribution by page type.

Desktop home pages consistently used more images than their inner page counterparts. The median home page used 239% the image bytes of similar inner pages. At the 90th percentile, home pages used nearly twice the image bytes at 6,856 KB compared to inner pages at 3,431 KB.

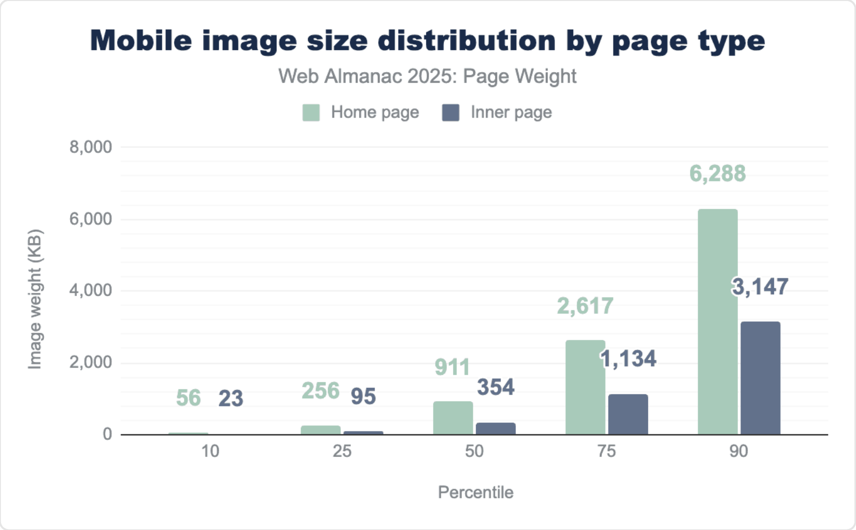


Figure 13.15. Mobile image size distribution by device

In narrowing the scope of inner versus home page image use to only mobile devices, the trend continues. Mobile devices often have slower connection speeds and limited computational power when compared to desktop. This doesn't appear to deter mobile home pages from celebrating their topic with image bytes. The median mobile home page used 911 KB of images while inner pages used 354 KB. At the 90th percentile, mobile home pages used 6,288 KB of images, nearly matching their desktop equivalent of 6,856 KB.

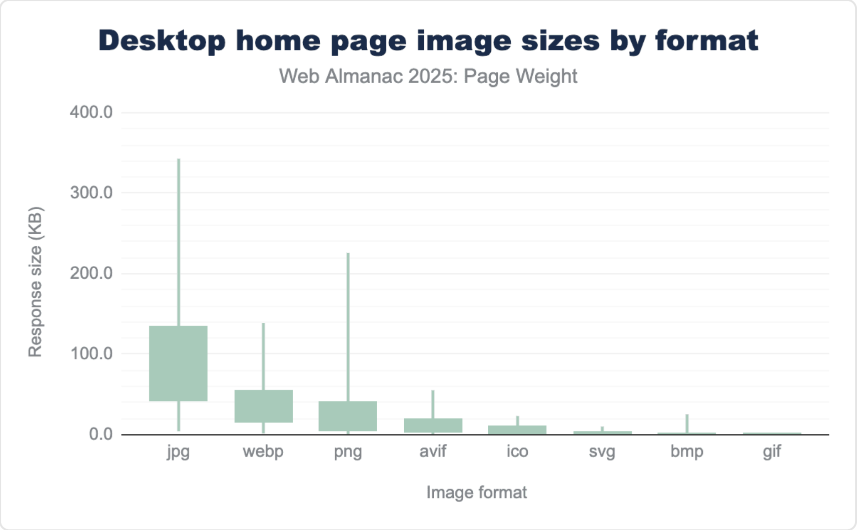


Figure 13.16. Distribution of desktop home page image sizes by format.

The impact of file type on image size comes down to how the digital information is organized and discarded. While every image starts as a grid of raw pixels, the way a file format stores those pixels determines if the file is a few kilobytes or several megabytes.

- **JPG** files are lossy file types. They permanently delete “unnecessary” data to achieve massive size reductions. How much is deleted is at the discretion of the file creator. This is likely why JPG file size spans the largest range, from 3 KB at the 10th percentile to 278 KB at the 90th.

- **WebP** is considered the all-purpose file type for images. Available as both lossy and lossless, these file types spanned from 1 KB at the 10th percentile to 97 KB at the 90th.

- **AVIF** is the newest of the widely adopted image file formats and was introduced in 2019 for its state of the art compression efficiency. This file format can be used for both lossy and lossless compression while producing high quality web images. AVIF ranged from 1 KB at the 10th percentile to 37 KB at the 90th percentile

- **PNGs** are used to create large but perfect image files, making the format ideal for logos and icons. PNG ranged from 0 KB at the 10th percentile to 278 KB at the 90th percentile.

JavaScript bytes

The growing weight of JavaScript is another significant trade-off as websites continue to lean on JavaScript to provide interactive features. While these scripts enhance user engagement, the consistent rise in payload size poses a challenge to site performance

It's important to note that this data set represents bytes for compressed JavaScript files. Depending on how effective compression was, this can be a significant difference. The larger the transfer (assuming compression) the discrepancy between uncompressed and compressed can be exponential. Additionally, inlined JavaScript in the HTML page is not represented on this data.

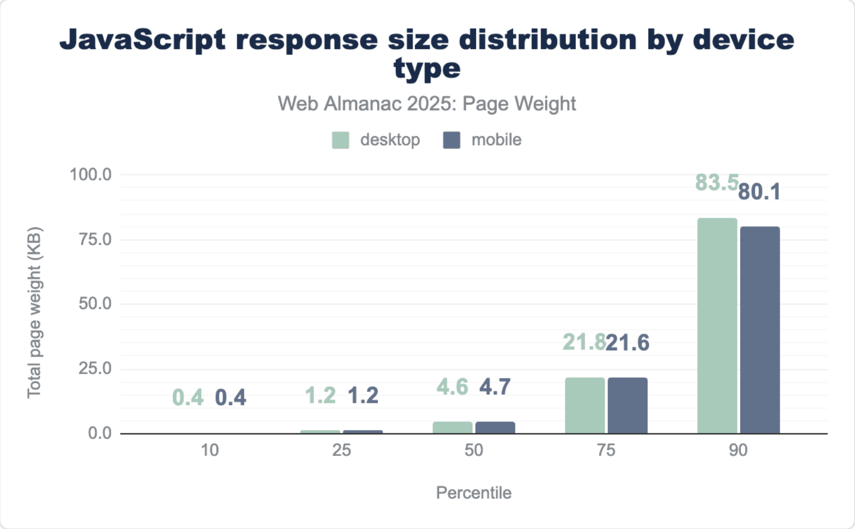


Figure 13.17. JavaScript file size distribution by device.

Much in the way that no one raindrop believes it caused the flood, the median individual JavaScript file is quite small at 4.6 KB for desktop and 4.7 KB for mobile. Smaller JavaScript files are ideal as they allow for small tasks to be completed without delaying response to user interaction. At the 10th percentile, the daintiest JavaScript files were 0.4 KB for both device types. At the 90th percentile, desktop saw JavaScript files at 83.5 KB and mobile saw 80.1 KB.

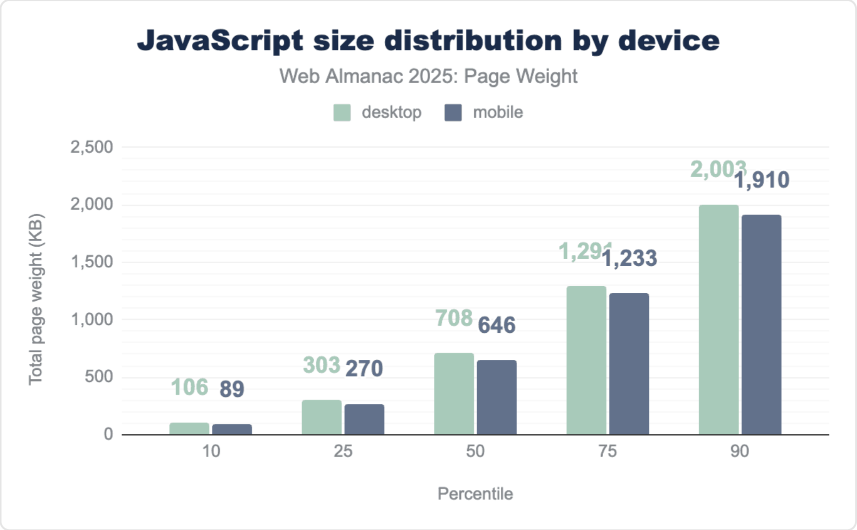


Figure 13.18. JavaScript size distribution by device.

When the relatively small individual JavaScript files are aggregated together, we get a better view of use on pages at large. JavaScript use was relatively consistent between mobile and desktop. At the 10th percentile, desktop used 106 KB and mobile 89 KB. The median desktop page uses 708 KB of JavaScript. This is slightly higher than mobile's 646 KB. These numbers quickly balloon past the median. At the 90th percentile, desktop used just over 2 MB and mobile used 1.9 MB.

Desktop pages at the 100th percentile used 189.9 MB of JavaScript and mobile used 181.1 MB. For perspective, downloading a standard definition 30 minute television show is about 150 MB.

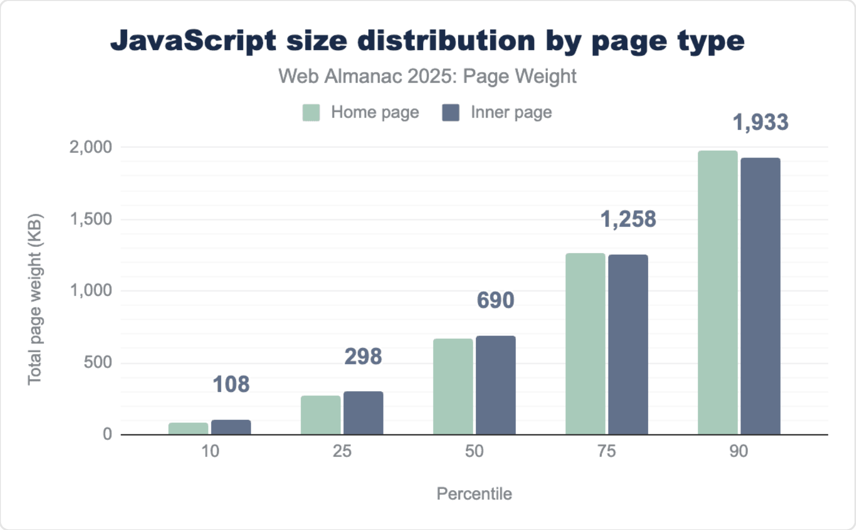


Figure 13.19. JavaScript size distribution by page type.

Unlike images, JavaScript use was consistent between home pages and inner pages. At the 10th percentile, home pages used 87 KB and mobile used 108 KB of JavaScript. Median home pages used 664 KB, while their inner page counterparts used 690 KB. At the 90th percentile, home and inner pages used 1,979 and 1,933 KB respectively.

Unused JavaScript

Unused JavaScript files represent not only a wasted opportunity but also page bloat. Every script called must be downloaded, parsed, compiled, and executed—regardless of whether it contributes to the page.

It's important to note that this data set comes from Lighthouse tests run in tandem with the Web Almanac crawl. Unused is unused javascript after it's been uncompressed. For example, amazon.co.uk downloads 423 KB of JavaScript, but when it's uncompressed that's 1,721 KB.

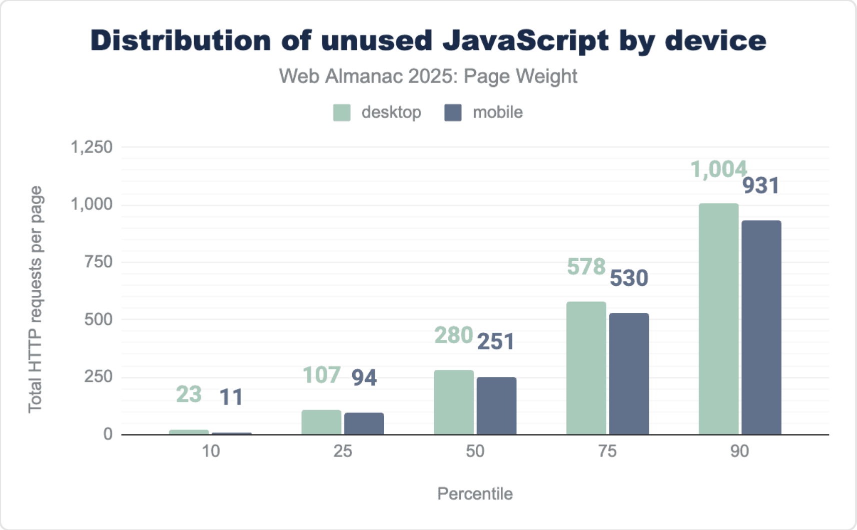


Figure 13.20. Distribution of unused JavaScript by device.

The median page saw a reasonable 280 KB of uncompressed, unused JavaScript on desktop and 251 KB on mobile. At the 100th percentile, desktop pages saw an astonishing 203.2 MB of uncompressed, unused JavaScript.

Video bytes

While images are often the most numerous assets on a web page, video bytes are the undisputed heavyweights of digital payload. As the web shifts toward "video-first" experiences—utilizing background "hero" videos, looping product previews, and embedded clips—the sheer volume of data transferred has reached unprecedented levels.

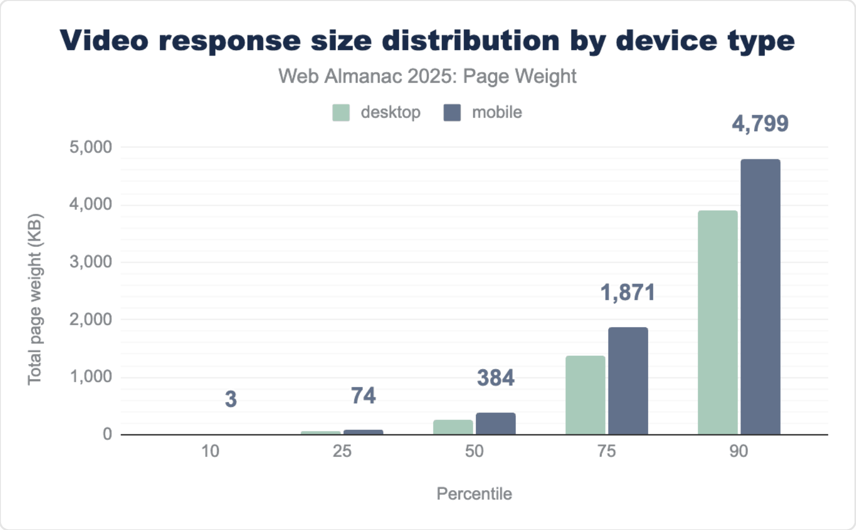


Figure 13.21. Video response size distribution by device.

Video bytes for the median page increased 28% year over year from 246 KB in 2024 to 315 KB in 2025. Mobile consistently used more video bytes across all percentiles. At the 100th percentile, pages shipped 695 MB of video. To put this in context, if a user on a mobile device attempted to load one of these pages without a data plan in place (leaving them to pay $0.15 cents per MB[441]), the experience could cost them $104.10 for video bytes alone.

[441] https://www.telus.com/en/mobility/prepaid/plans

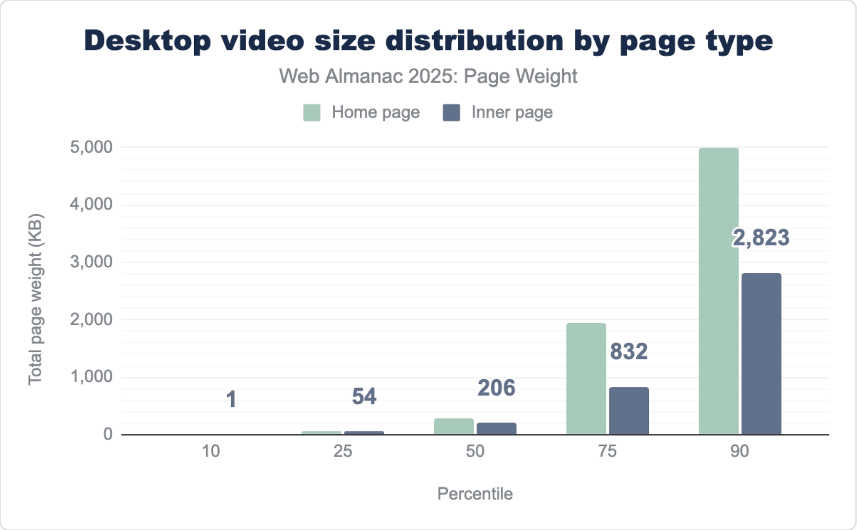


Figure 13.22. Distribution of desktop video response sizes by page type.

More video bytes were shipped on home than inner pages for desktop. The median home page with video saw 288 KB while inner pages with video came in at 206 KB. The gap between page types is largest in the 75th percentile where home pages come in at 1,934 KB, 232.6% larger than inner pages at 832 KB.

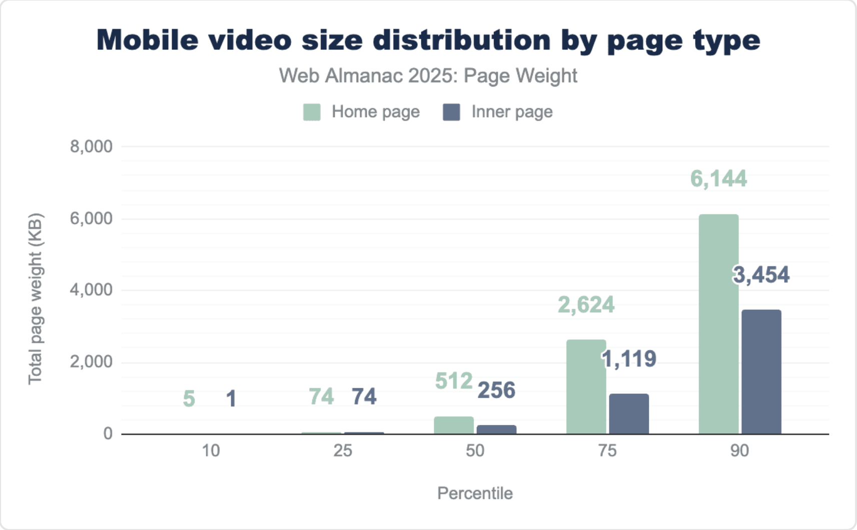


Figure 13.23. Mobile video response size distribution by page type.

The pattern of more video bytes on home pages continued for mobile devices as well. The median home page shipped double the bytes (512 KB) compared to inner pages at 256 KB. At the 90th percentile, mobile home pages loaded 6.1 MB of video.

Several technical layers impact a video files size:

- **File formats**: The "wrapper" (for example, MP4, WebM) that holds the video and audio streams. While the container itself is light, the choice of container often dictates which efficient codecs can be used.
- **Codecs (compression/decompression)**: The efficiency of the mathematical algorithm used to shrink the video. Legacy codecs like H.264 (AVC) are widely compatible but "heavy," whereas modern codecs like HEVC (H.265) and AV1 offer significantly better visual quality at a fraction of the byte count.
- **Chroma subsampling**: A method of reducing file size by implementing lower resolution for color information than for brightness (luma) information, taking advantage of the human eye's higher sensitivity to brightness.

File formats are the most easily observed factor on video bytes. WebM files are designed specifically for the web. Common knowledge expects WebM files to be lighter than MP4s because they are built for the constraints of a browser.

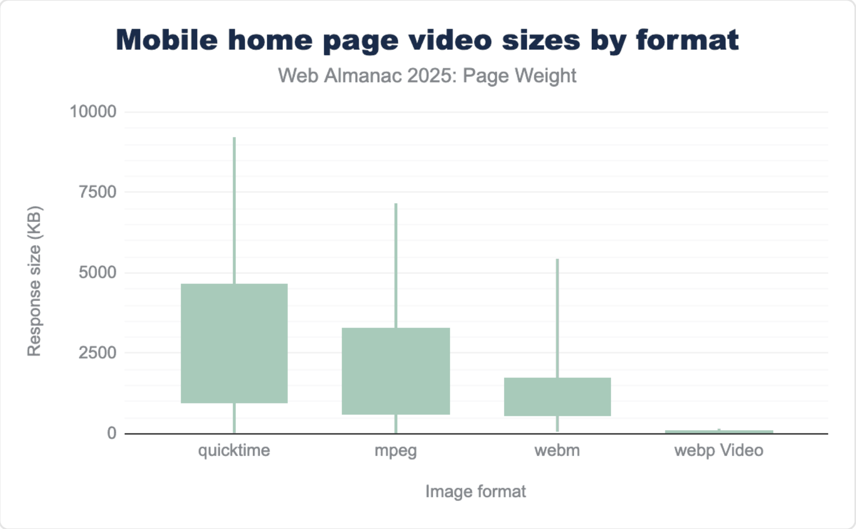


Figure 13.24. Distribution of mobile home page video sizes by format.

For the median percentile of mobile home page videos, QuickTime files have the highest median response size (976.0 KB), followed by mpeg (639.5 KB) and WebM (581.8 KB). webp Video has a significantly smaller median size of 21.3 KB. WebP Video files are consistently the smallest across all measured percentiles, with a maximum size of 5,743.5 KB and a 90th percentile of 155.6 KB.

The largest home page desktop video files encountered were mpeg, coming at 326 MB for the 100th percentile.

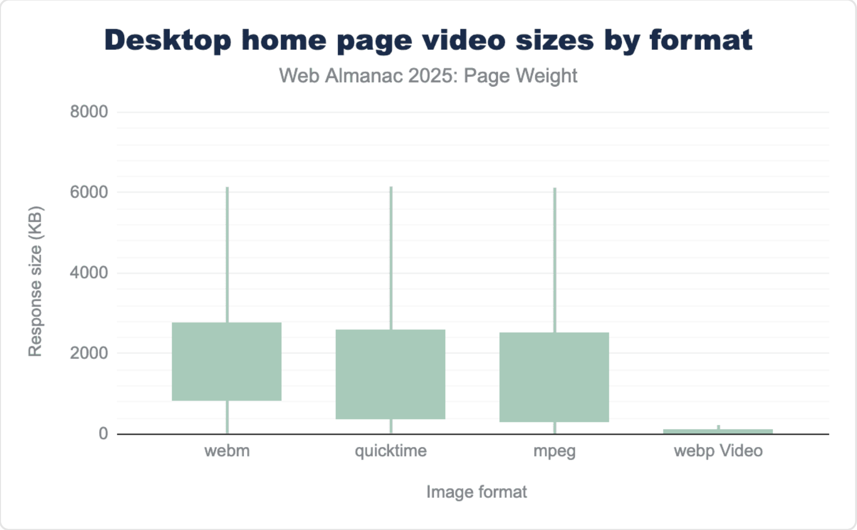


Figure 13.25. Distribution of desktop home page video sizes by format.

While the high-end sizes are similar, the median (50th percentile) sizes vary, with WebM having the largest median at 860.22, followed by QuickTime (399.97 KB) and mpeg (326.52 KB). This suggests that a 'typical' webm file is larger than its QuickTime or mpeg counterparts, despite the largest files being about the same size.

The median response size for webp Video is 32.58 KB, which is 10 to 26 times smaller than the median of the other formats. Even at the 90th percentile, webp Video's size of 228.33 is exceptionally small. This means 90% of webp Video files are smaller than the smallest 90% of the other three formats, where the 90th percentile is over 6,100.

The median size for WebM (860.22) is more than double that of QuickTime (399.97) and almost triple that of mpeg (326.52), suggesting that while all formats can handle very large files, the center of the WebM distribution is skewed toward larger sizes.

The largest home page desktop video files encountered were QuickTime, coming at 431 MB for the 100th percentile.

CSS bytes

Cascading style sheets or CSS have been used to stylise pages on the web for years, they are a relatively lightweight way to create the layout and control the visual elements of a page.

CSS works alongside HTML giving control of font styles, colors, spacing and even visibility of elements providing a more granular level of control but also a separation from other page aspects making it more maintainable. CSS is more lightweight than images and can help improve site performance by replacing large assets with CSS effects and animations.

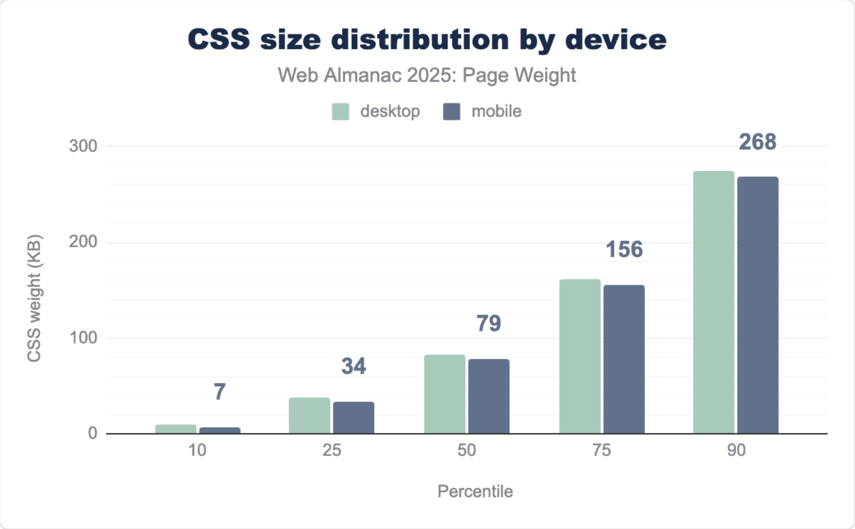


Figure 13.26. Distribution of CSS response sizes by device type.

CSS bytes for mobile have grown in all percentiles except the bottom 10 where we saw a contraction versus last year.

However this growth is mostly modest when compared to other aspects of a page. At the 90th percentile the total file size is just slightly over ¼ of a mb, growing by just 8 KB against last year,

At the 50th percentile it was a growth of 4 KB against last year which is roughly an additional 4000 characters added to CSS files pointing to continued heavy use.

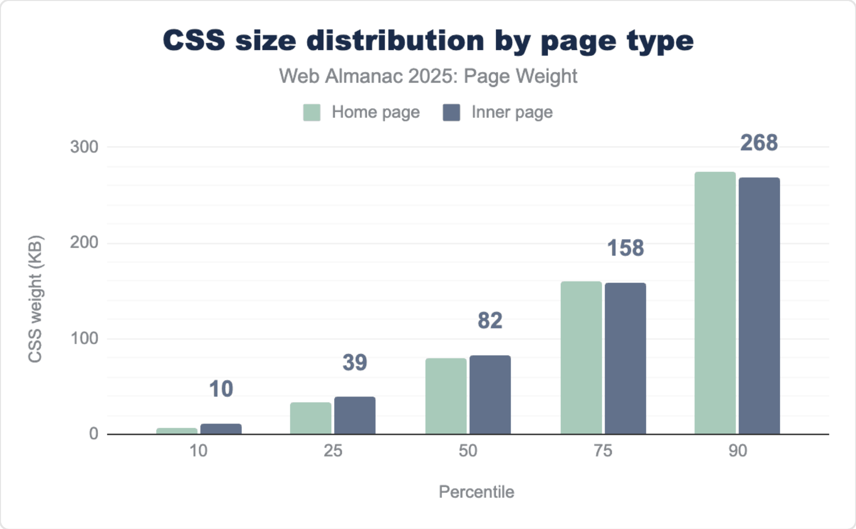


Figure 13.27. Distribution of CSS response sizes by page type.

Inner pages follow this same trend of growth in all percentiles except the lowest 10th which again saw a 1 KB contraction. Inner pages feature a slightly higher usage of CSS in the 25th, 50th and 75th percentile.

Notably when we get to the top end of the results, the 75th and 90th percentile, home pages actually have a greater CSS size. This could be due to inefficient use of CSS, i.e. loading the majority of files irrespective of use on the current page or potentially heavier reliance on CSS to provide visuals for all pages.

HTML bytes

HTML bytes refers to the pure textual weight of all the markup on the page. Typically it will include the document definition and commonly used on page tags such as `<div>` or `<span>`. However it also contains inline elements such as the contents of script tags or styling added to other tags. This can rapidly lead to bloating of the HTML doc.

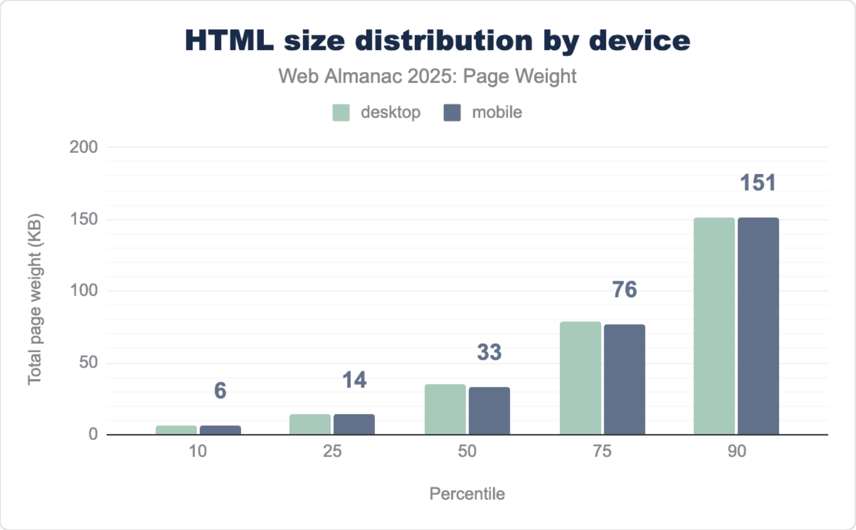


Figure 13.28. Distribution of HTML response sizes by device type.

HTML size remained uniform between device types for the 10th and 25th percentiles. Starting at the 50th percentile, desktop HTML was slightly larger. Not until the 100th percentile is a meaningful difference when desktop reached 401.6 MB and mobile came in at 389.2 MB.

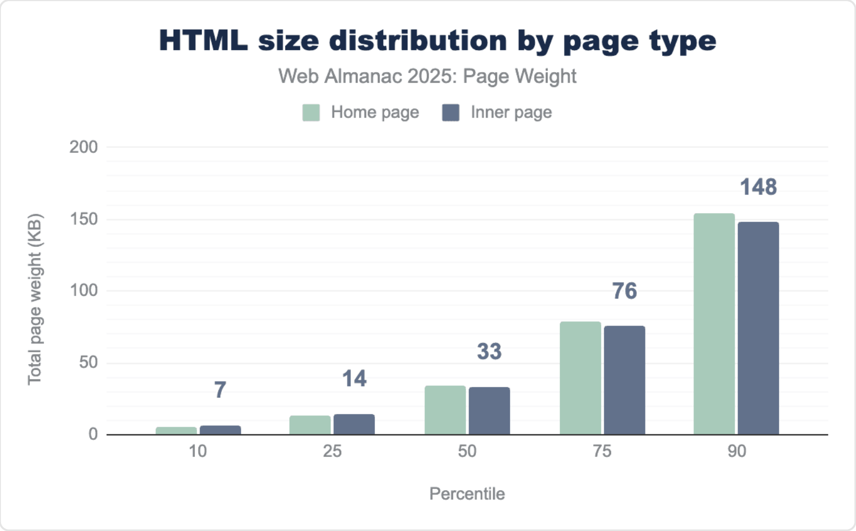


Figure 13.29. Distribution of HTML response sizes by page type.

There is little disparity between inner pages and the home page for HTML size, only really becoming apparent at the 75th and above percentile. At the 100th percentile, the disparity is significant. Inner page HTML reached an astounding 624.4 MB—375% larger than home page HTML at 166.5 MB.

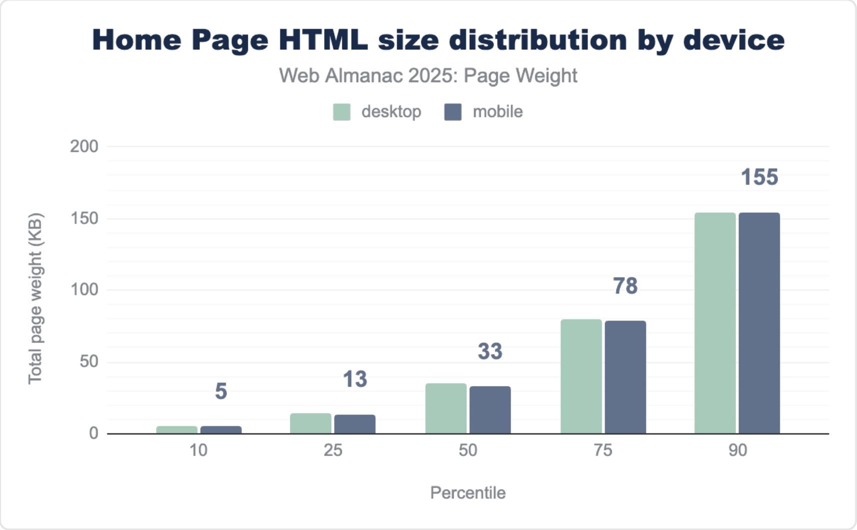


Figure 13.30. Distribution of home page HTML response sizes by device type.

The size difference between mobile and desktop is extremely minor, this implies that most websites are serving the same page to both mobile and desktop users.

This approach dramatically reduces the amount of maintenance for developers but does mean that overall page weight is likely to be higher as effectively two versions of the site are deployed into one page.

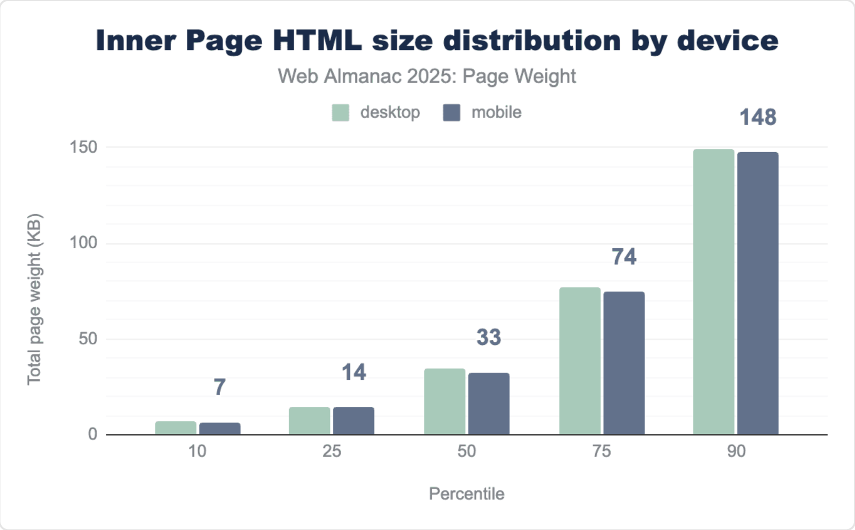


Figure 13.31. Distribution of inner page HTML response sizes by device type.

Inner pages behaved similar to their home page counter parties through the 90th percentile. At the 100th percentile, the largest inner page HTML response was the same for both device types: 624.4 MB. On a standard 10-25 Mbps connection, this would take 30 to 90 seconds to load.

Font bytes

While images and videos are the most obvious contributors to page bloat, web fonts represent a subtle but significant portion of modern page weight. Often overlooked during the design phase, the transition from system-standard fonts to custom “web fonts” has introduced a new layer of latency. Every unique weight (bold, light, thin) and style (italic) of a typeface requires a separate file download, meaning a single font family can quickly balloon into several hundred kilobytes of additional data.

The “weight” of these fonts is determined by three main factors:

1. **Character support**: A font that includes support for multiple languages (Latin, Cyrillic, Greek) contains thousands more “glyphs” (the shapes of the letters) than a basic English-only font.
2. **Styles and weights**: Each variation (for example, Roboto Regular, Roboto Bold, Roboto Bold Italic) is typically a distinct file. Loading a full “family” can be heavier than a large optimized image.
3. **Format efficiency**: Older formats like TTF (TrueType) and OTF (OpenType) are

uncompressed, whereas modern web-specific formats like WOFF2 use Brotli compression to reduce file sizes by up to 30% without losing quality.

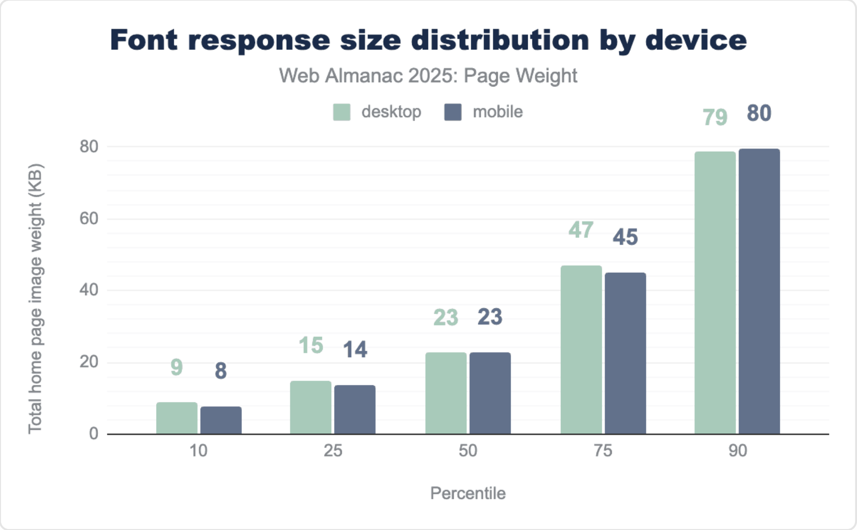


Figure 13.32. Distribution of font response sizes by device type.

The median page shipped 23 KB of fonts. Bytes shipped remained closely matched between devices through the 90th percentile. Home pages and inner pages shipped the exact same amount of bytes across percentiles with 23 KB representing the median. For a deeper dive, see the Fonts chapter.

Other file types

Ultimately the web is a diverse ecosystem with many file types. Some lesser encountered file types include 15 KB of WASM, 12 KB of audio, on the median page and across device types, audio. JSON, XML, and txt file types saw 0 KB of use on the median page. It's noteworthy that at least one site is shipping 117.6 MB of JSON on mobile pages, landing it the title of largest infrequent file type.

Page weight in request volume

While the total number of bytes measures the raw volume of data, the volume of requests measures the complexity of the conversation between the browser and the server. Studying page weight through both lenses is essential because a "light" page (in bytes) can still be "slow" if it requires too many individual requests to render. Bytes represent the bandwidth required to download a page. The volume of requests represents the latency overhead inherent in web architecture.

File type volume for the median page

To understand what file types contribute the most to page weight, we looked at the file type requests by the median page (50th percentile). This approach establishes a baseline for measuring the overall impact of each file type.

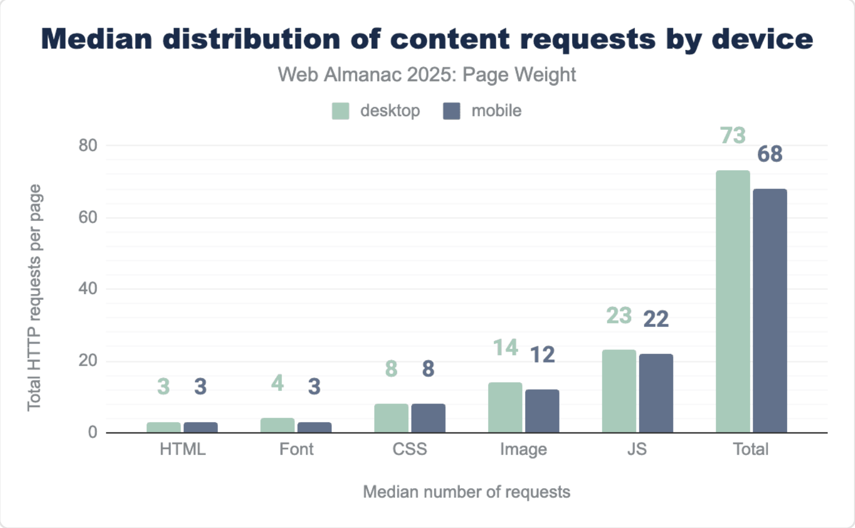


Figure 13.33. Median number of requests by content type and device type.

The median page makes 77 requests on desktop and 72 on mobile. Overall requests grew 9% year over year on mobile and 8% on desktop. Image file types continued their downward trend in 2025, decreasing 6% year-over-year.

Requests by file type remained largely consistent year over year. The median desktop calls 77 resources while its mobile counterpart requires only 72. The number of HTML files returned increased from 2 to 3 across both page types. Median home page CSS increased slightly to match the use on inner pages, which remained stable year-over-year. JavaScript remained the most requested file type.

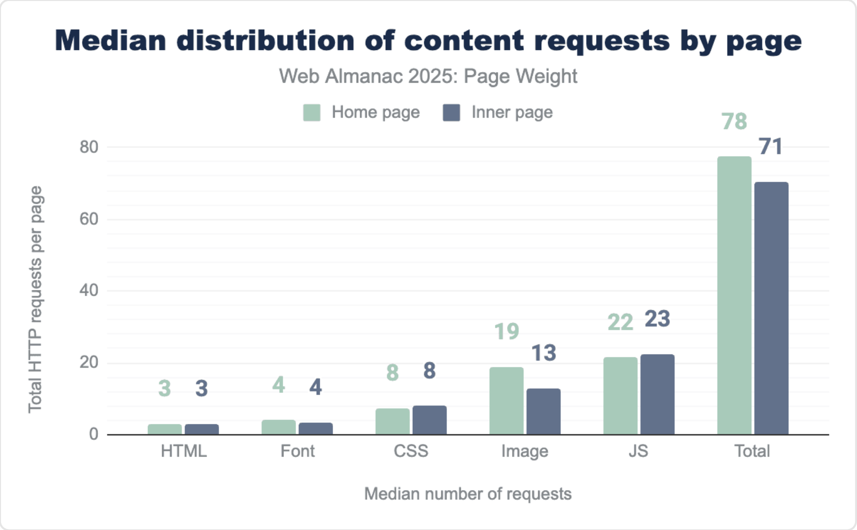


Figure 13.34. Median number of requests by content type and page type.

The median home page loads 3 HTML files, 4 font files, 8 CSS files, 19 images and 22 JavaScript files for 78 files total. The median inner page loads 3 HTML files, 4 font files, 8 CSS files, 13 images, 23 JavaScript files for 71 files total.

Request volume distribution

With our meridian behaviors established, we can step back to look at our data across percentiles. Percentiles provide a clear picture of how data points are spread out, making it easier to interpret patterns.

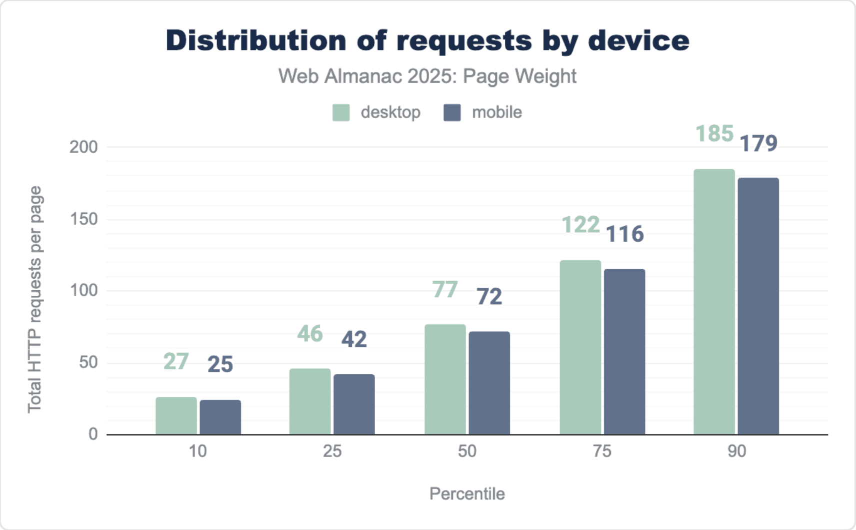


Figure 13.35. Distribution of request volume by device type.

Even the lightest of pages got heavier this year. At the 10th percentile, the most spartan pages added 3 requests this year. In contrast, the 90th percentile requested 9 additional files. This illustrates that mobile and desktop make the same number of requests, as demonstrated by the extremes.

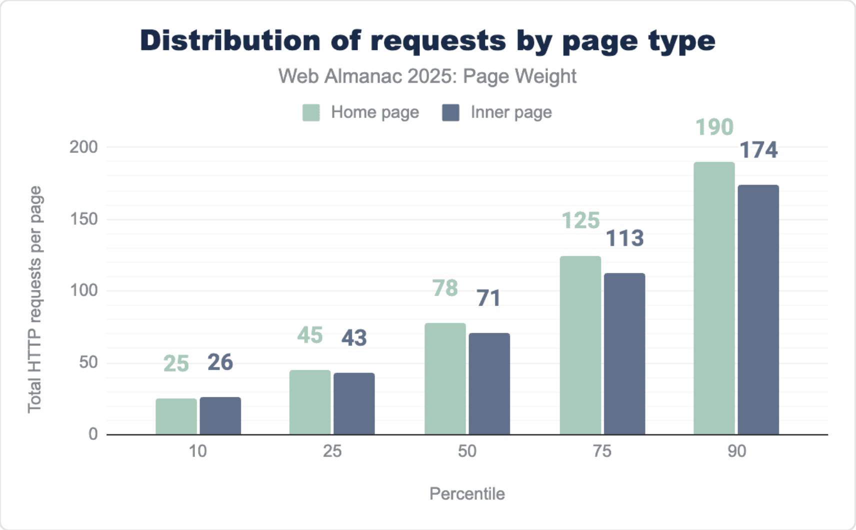


Figure 13.36. Distribution of request volume by page type.

When viewing the data grouped by home page versus inner page, increases in file requests were consistent across all percentiles. The median page makes 77 requests on desktop and 72 on mobile. The gap between home pages and inner pages increases as the percentile grows.

This is particularly interesting, as we see consistency in the number of requests when viewing the data by mobile versus desktop, indicating that home page requests are a key differentiator.

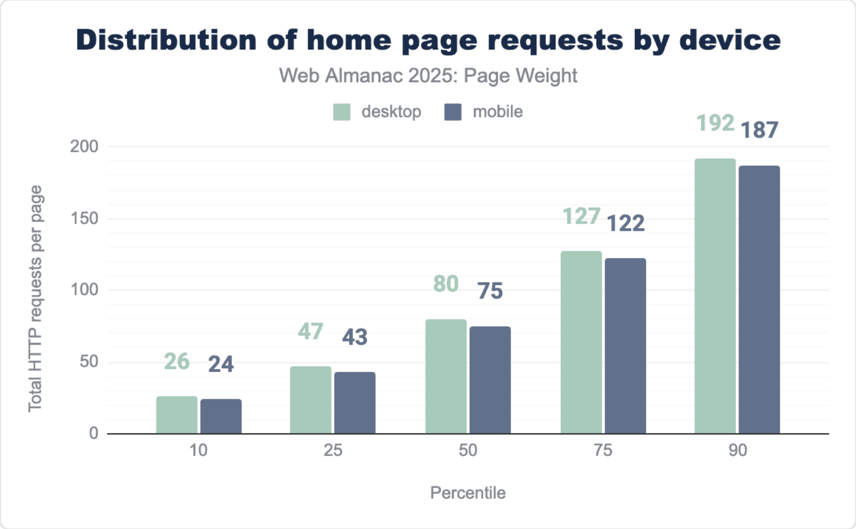


Figure 13.37. Distribution of home page request volume by device type.

Viewing home page requests by device type allows deep analysis. Each device requires slightly more requests than when home and inner pages data is blended. The median desktop home page makes 80 requests while its mobile counterpart makes 75. Behavior remains consistent across devices, with minor variance in percentiles.

Image request volume

Some say the internet was created for cat pictures. While that may be an exaggeration from devoted animal lovers, it highlights the fact that visual content dominates the web.

Images play a vital role in everything from product verification to credible news reporting. Yet, these static files are often among the heaviest digital resources, making them prime candidates for performance optimization and technological innovation.

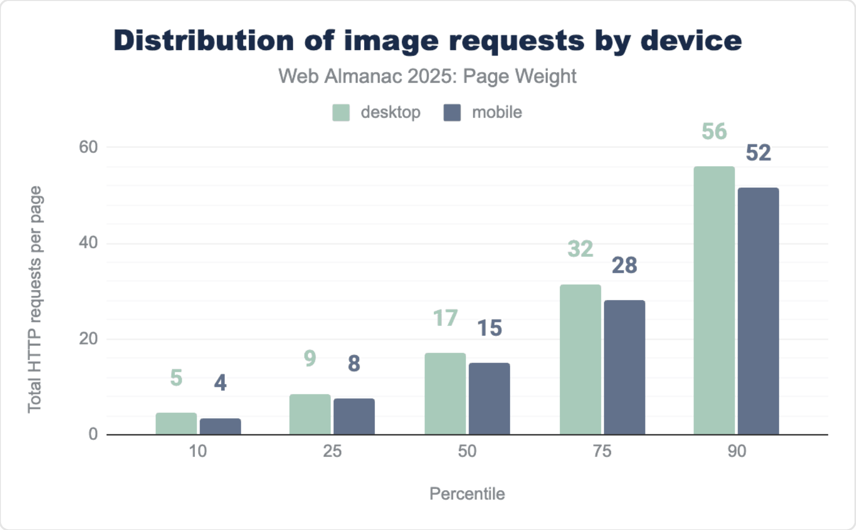


Figure 13.38. Distribution of image requests by device type.

At the 50th percentile, desktop made 17 and mobile made 15 images requests. Desktop used more image files consistently across percentiles. This is a slight decrease from 2024 where desktop pages called 18 images and mobile requested 16. At the 100th percentile, desktop pages requested 26,330 images. We can only imagine how magical that page must be.

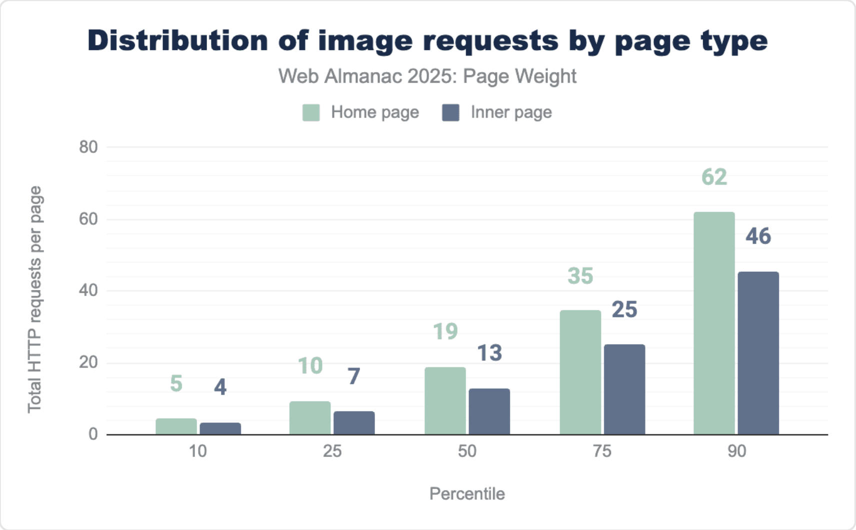


Figure 13.39. Distribution of image requests by page type.

When comparing home pages to inner pages, the difference remains consistent: the median home page loads 19 images versus 13 on an inner page. This pattern holds across the full distribution range and closely mirrors 2024"s data, where home pages averaged 20 image requests and inner pages 14.

Although there"s a slight decrease in image requests, a trend also reflected at the 75th and 90th percentiles, the minimal changes suggest no major shift in how site owners or publishers approach imagery. Instead, it appears to be business as usual in web image usage.

CSS request volume

Cascading Style Sheets (CSS[442]) act as the presentation layer of the web. It's a style language that allows developers to define the layout of a document written in a markup language that is visually presented. In the scope of the web almanac, then exclusively html. Occasionally, SVGs might also account for some.

CSS enables the developer to specify elements such as colors, fonts, layout and sizing ,separating presentation from the structural content defined in HTML. It continues to grow increasingly more powerful, offering interesting capabilities for responsive design, touches of animation or sophisticated layouts. Visual effects that once required JavaScript can now be done through CSS, making CSS a basis for modern web design.

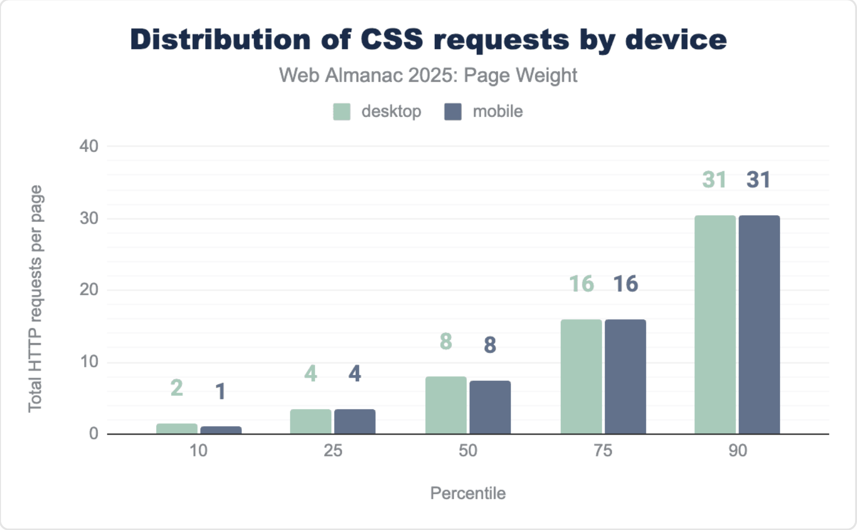


Figure 13.40. Distribution of CSS file requests by device type.

[442] https://web.dev/css

Across all both desktop and mobile devices, there was very little difference in the amount of CSS a page requested. At the 50th percentile, both mobile and desktop devices requested 8 CSS resources, there was no more than 1 resource request difference across the whole distribution. This very much matches 2024's pattern, and indeed up to the 90th percentile were more or less stable in the amount of requests, compared year over year

At the 90th percentile, 31 desktop and 30 mobile CSS requests were made this year, a slight increase over 2024's 26 on both desktop and mobile. This suggests that there's been little change to developer approaches to CSS over the preceding 12 months in terms of how many separate assets they are split into, nor the tools they use.

It also points to developers being far more likely serving the same CSS files, regardless of client type.

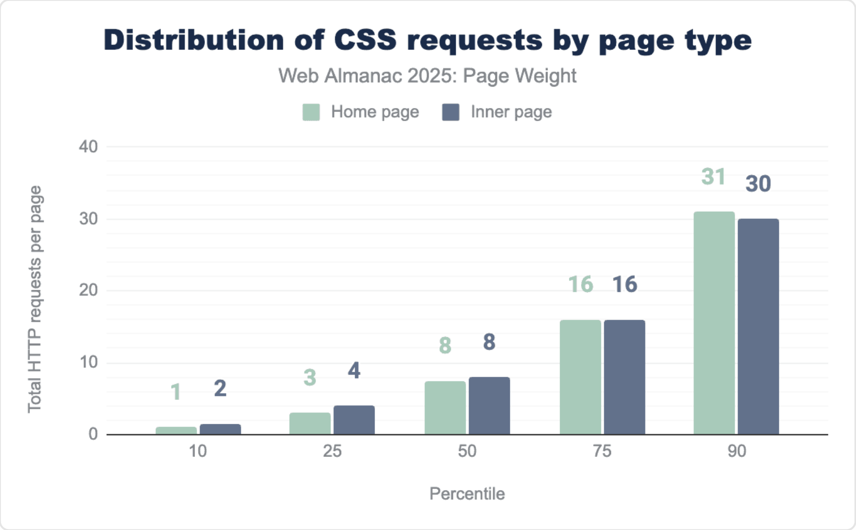


Figure 13.41. Distribution of CSS file requests by page type.

The median page made 8 CSS requests. Inner pages made more requests than home pages through the 75th percentile. At the 90th percentile, the home page made 1 more request than the inner pages.

There is very little variation in the number of files requested on home pages compared to inner pages. Whilst we can't tell from this analysis if it's the same files being served, it does point to the fact that developers are shipping one set of CSS resources across the whole site, which could be a missed opportunity to reduce the weight of CSS on individual pages by splitting these, and only sending the CSS needed for the actual page.

JavaScript request volume

While images and videos are static weight, JavaScript is active weight. Its impact is unique because it consumes resources twice: once when it is downloaded (network) and again when it is executed (CPU). When JavaScript is split into a high volume of requests—often due to modular coding practices or third-party plugins—it introduces a "double tax" on performance.

Even if the total byte count remains low, a high volume of script requests can paralyze a browser's main thread, leading to a site that looks loaded but remains unresponsive to user input.

98.1%

Figure 13.42. Pages using JavaScript.

98.1% of all pages make at least one request for a JavaScript file. This number doesn't include inlined JavaScript which can be found in HTML.

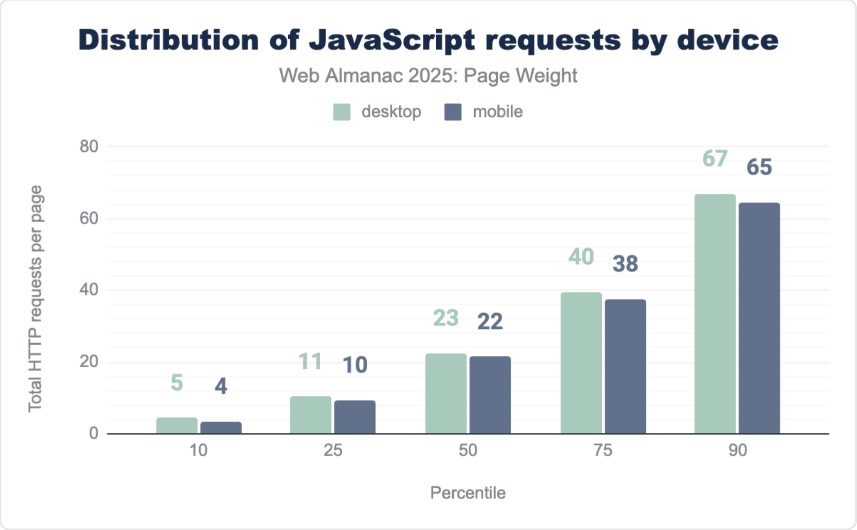


Figure 13.43. Distribution of JavaScript requests by device type.

The median desktop page made 23 requests for JavaScript on desktop and 22 on mobile. Desktop pages consistently called for more JavaScript files than mobile. Desktop home pages called for the most JavaScript files with 12,676 being requested at the 100th percentile. The Web Almanac crawler earned its annual hibernation this year.

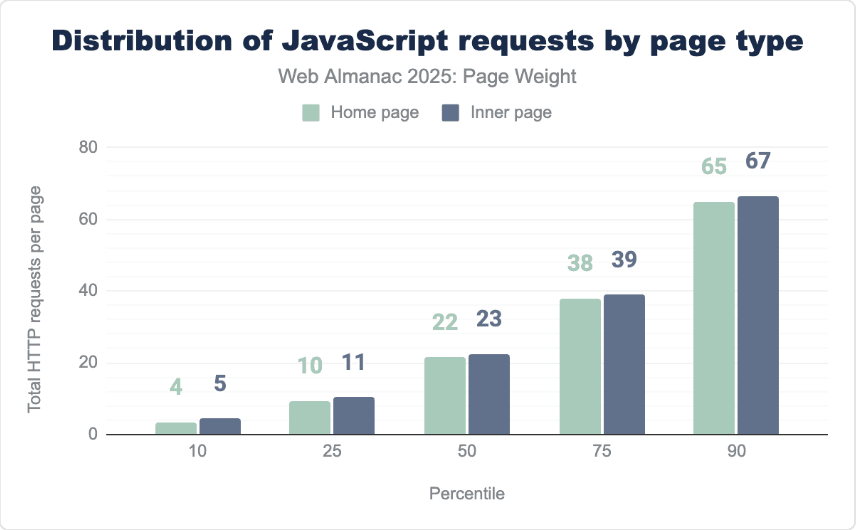


Figure 13.44. Distribution of JavaScript request volume by page type.

Inner pages consistently called slightly more JavaScript files than home pages. The median home page requested 22 JavaScript files while its small screened counterpart called 23.

Fonts request volume

Font files allow the website to specify styling for the text parts of the website, typically they consist of all letters from A-Z both upper and lowercase. This allows the browsers to transform text into other styles than just the system installed fonts.

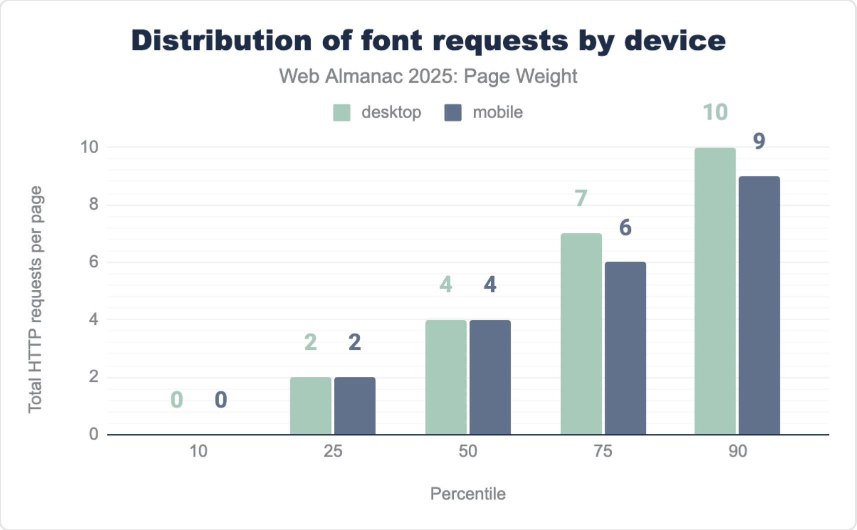


Figure 13.45. Distribution of font requests by device type.

The bottom 10th percentile do not use extra font requests, relying instead on the system installed fonts for both mobile and desktop. Mobile font requests are lower above the 75th percentile with one fewer request on average.

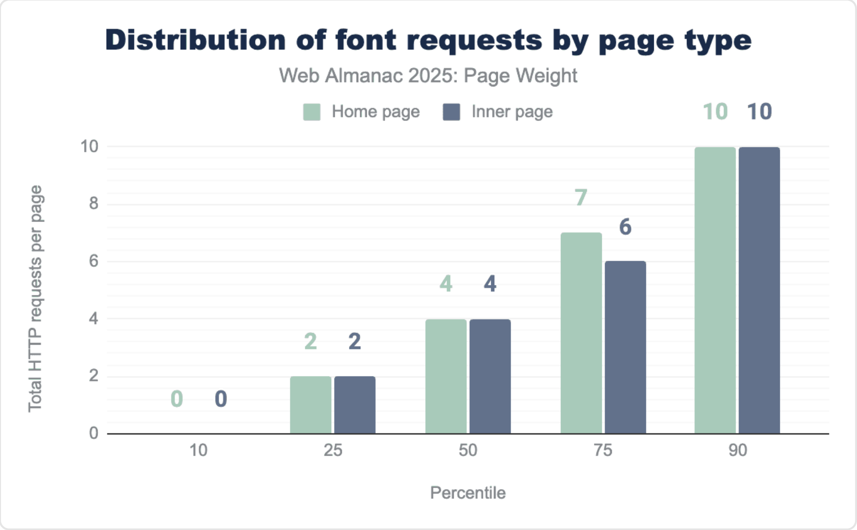


Figure 13.46. Distribution of font request by page type.

Home pages and inner pages are identical in average requests with the one exception at the 75th percentile. Desktop home pages loaded the most fonts with 3,038 files requested at the 100th percentile. We truly hope the site in question is a font repository with performance optimizations prioritized for Q1.

Broadly it would be expected that different fonts are not used on individual pages but rather as a style across the site. There are exceptions to this, such as for promotional material, but the average indicates that most of the time imported fonts are used across the whole site.

Other request volume

The request volume of "utility" files—including HTML fragments, JSON data, and third-party assets—creates a significant, cumulative "latency tax." In modern web architecture, particularly with the rise of microservices and modular design, a single page load can trigger dozens of requests for small, disparate files. Even if these files are lightweight in terms of bytes, the sheer quantity of requests can overwhelm the browser"s network stack and delay the rendering of the page.

The median page made 2 requests for HTML and 2 other file types. At the 90th percentile, this climbs to 12 HTML and 12 other file types. Still, there are some out there committed to setting new records for innocuous file types. Somewhere out there, there's a desktop inner page requesting 17,065 other file types for a page load. They have achieved the dream of being the 100th percentile. You did it, buddy. Put the JSON down.

Adoption rates of byte-saving technologies

The simplest way to reduce page weight is to not send the bytes in the first place. Being mindful of how you build web pages, and making optimizing resources a standard part of how you work, can have a big impact on the overall weight of your pages, before even considering specific technologies.

Along this that, there are also specific technologies and techniques that can be employed to reduce the amount of data sent over the network. In this section we look at three of these: facades for third-party embeds, minification, and compression.

Facades for videos & other embeds

Facades, also termed import on interaction, is a design pattern that replaces heavy items, such as video embeds, social media posts or chat widgets with a simple graphical representation that is only replaced by the full, interactive element when users interact.

Think of a simple thumbnail for a video that loads the youtube embed, and all the associated data and resources when users click on it.

This can provide performance improvements as users aren't trying to download these extra resources at page load time, and ultimately save page weight. If users don't want to see that video or open a chat bot session those bytes are never requested.

Detection of sites that are using facades is not really possible. However, Lighthouse, prior to the release of version 13, did have a test for common embeds that do have recognized facade solutions, called Lazy load third-party resources with facades[443]. The crawl this year was run with version 12, so we had access to this metric.

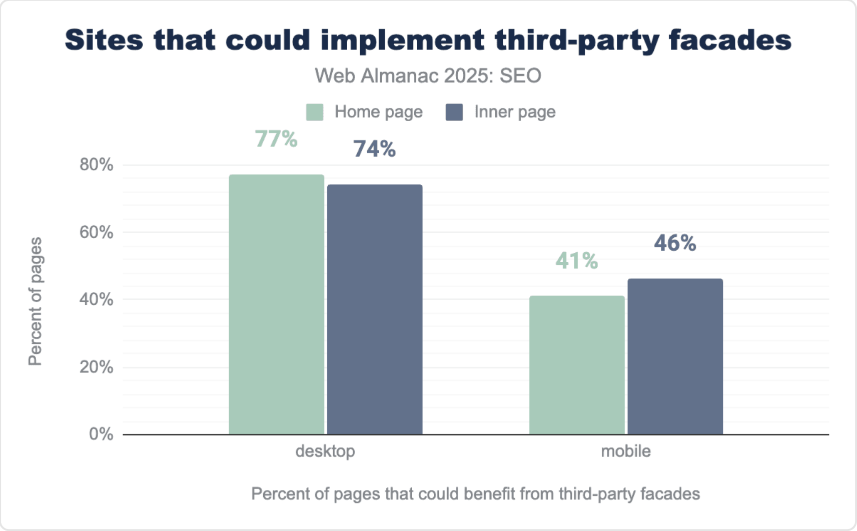


Figure 13.47. Sites that could implement Third-party facades.

41% of home pages on mobile loads and 77% of home pages loaded with the desktop client were detected as having an opportunity to implement lazy loading with facades. This represents a regression from 2024 for desktop, where 70% were detected, but mobile loads represent an improvement from 45%.

Like 2024, mobile pages seem more likely to implement a facade than desktop pages.

Interestingly inner pages scored better on desktop at 74% and yet worse on mobile at 46%. Perhaps developers are focusing more on home page performance for their mobile visitors?

The figures do show there could be significant opportunities to reduce page weight, but

[443] https://developer.chrome.com/docs/lighthouse/performance/third-party-facades

likewise, they aren't without their own issues, with sometimes perceptible delays once a user does interact with the element, and with videos in particular, double clicks or taps needed to start playing.

But it is worth assessing the embeds you have on your page, and see if any could benefit from implementing this approach.

Compression

Compression involves reducing the size of a resource before it's sent across the network to a device, where it's then decompressed. This process usually leads to faster page loads, especially when the network is the biggest bottleneck.

HTTP compression applied to text based files, like HTML, CSS, JavaScript, JSON, SVG, ico and ttf font files, can offer great over the wire page weight savings. It's worth noting the benefits don't extend to other file types, eg media, where the compression is part of their encoding.

Using GZip[444], Brotli[445], or the new kid on the compression block, Zstandard (zstd)[446] to compress HTTP requests can often significantly reduce the weight of these text-based resources, increasing performance.

It needs to be noted that compression doesn't entirely make page weight issues go away, they do need to be decompressed to their full size, and the process of compressing and decompressing could, if overdone, slow down the overall speed, although that would be a rare occurrence, certainly over the network bytes saved.

[444] *https://developer.mozilla.org/docs/Glossary/gzip_compression*
[445] *https://developer.mozilla.org/docs/Glossary/Brotli_compression*
[446] *https://developer.mozilla.org/docs/Glossary/Zstandard_compression*

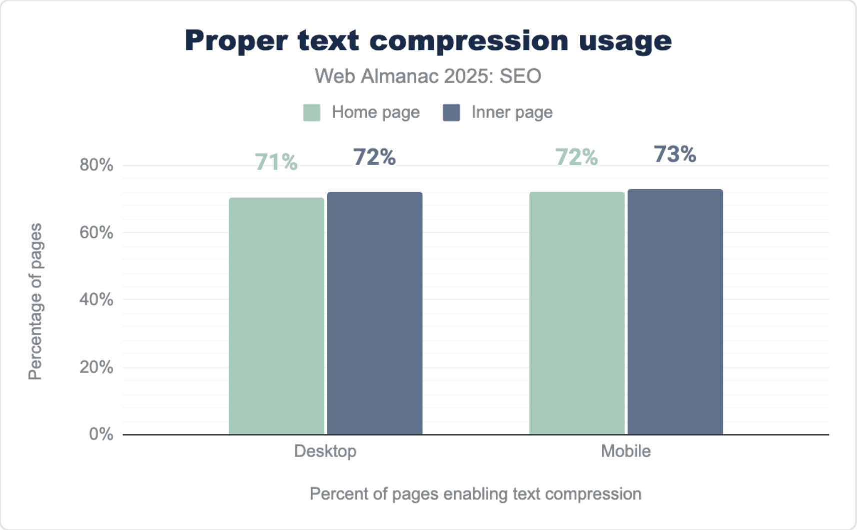


Figure 13.48. Proper text compression usage.

In 2025, 71% of desktop and 72% of mobile home page loads had proper text compression, as measured by Lighthouse's Enable text compression audit[447].

For inner pages, the pass rates were slightly higher at 72% on desktop and 73% on mobile.

This shows a small improvement compared to 2024, when home pages had a 70% desktop and 71% mobile pass rate, and inner pages reached a 71% desktop and 72% inner page pass rate. In other words, each segment improved by just 1 percentage point year over year.

While any increase is positive, it still is slightly disappointing that over 25% of pages across all devices and page types, still fail to use efficient compression.

Minification

Minifying[448] text resources removes unnecessary data, such as spaces, comments and can even shorten function and variable names to leave a smaller file.

There is no additional client side overhead in minifying, In other words, a file doesn't need to be *unminified* to work. Any overhead is purely server side if done on the fly.

However, for many resources, such as CSS or JavaScript, minification can be handled upfront as

[447] *https://developer.chrome.com/docs/lighthouse/performance/uses-text-compression/*
[448] *https://developer.mozilla.org/docs/Glossary/Minification*

part of the build process. It only needs to be done once.

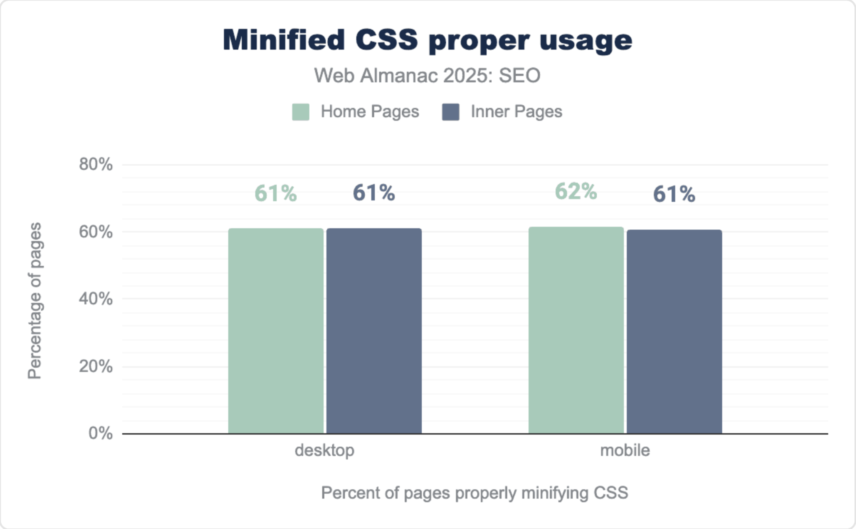


Figure 13.49. Minified CSS proper usage.

2025's analysis shows that 61% of desktop, 62% of mobile home page loads, and 61% of desktop and mobile inner page loads passed lighthouse's Minify CSS[449] audit.

This means a slight decline from 2024, with each metric dropping by 1%, a small but unfortunate trend

[449] https://developer.chrome.com/docs/lighthouse/performance/unminified-css/

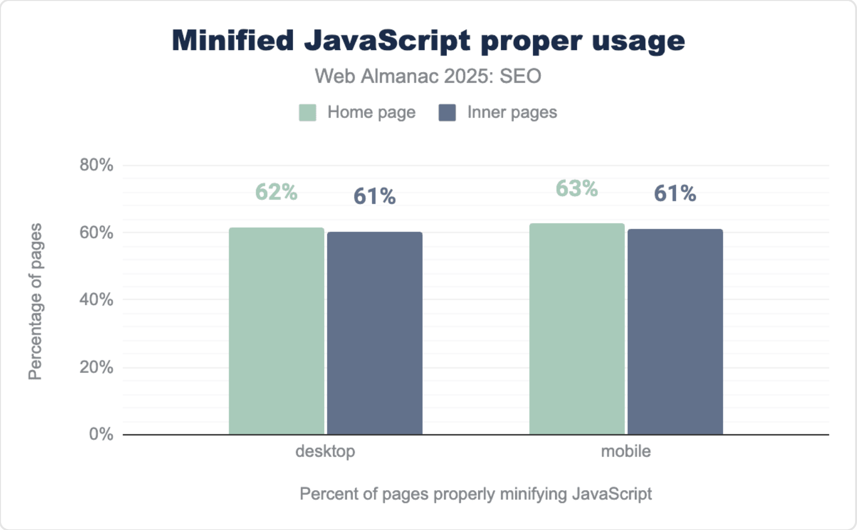


Figure 13.50. Minified JavaScript proper usage.

In 2025, 62% of desktop and 63% of mobile home page loads passed Lighthouse's Minify JavaScript[450] test.

For inner pages, the result was similar: 61% of desktop and 63% of mobile loads passed.

This represents an improvement from 2024, when 58% of desktop and 57& of mobile home pages passed and 59% of desktop and 58% of mobile inner pages did the same.

Although this year's slight decline in correctly minified CSS is somewhat, it is encouraging to see a bigger increase in JavaScript being minified, even when there still is plenty of room for further adoption.

Caching

The sensible use of caching rules can help mitigate excessive page weight. When a common resource is cached locally by the browser, it does not need to be downloaded again as visitors navigate that website or even when they return later. This results in fewer network requests and faster page loads.

Naturally, this doesn't entirely mitigate the issue of heavy resources, they still have to be requested and downloaded at least once.

[450] https://developer.chrome.com/docs/lighthouse/performance/unminified-javascript/

Caching is most effective for static resources, ones that are unlikely to change between requests.

Lighthouse offers a use efficient cache lifetimes[451] audit, which looks for static resources called by a page that have a cache lifetime of at least 30 days,unless explicitly set to not be cached.

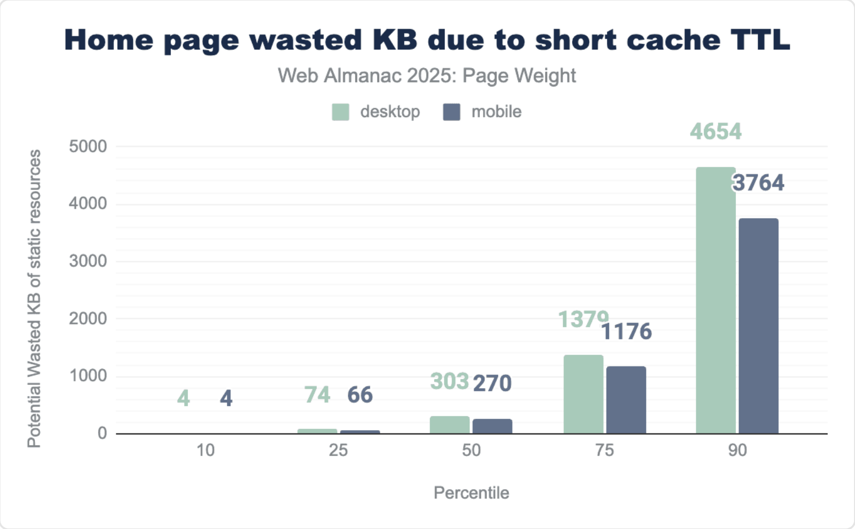


Figure 13.51. Distribution of wasted KB on home pages due to short cache TTL

[451] https://developer.chrome.com/docs/performance/insights/cache

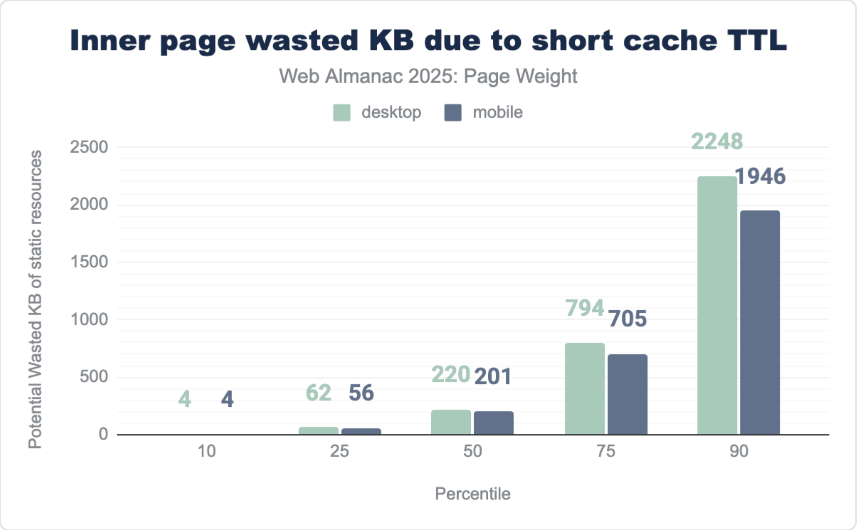


Figure 13.52. Distribution of wasted KB on inner pages due to short cache TTL

The data shows that many websites could reduce page weight by making better use of browser caching. For desktop home pages, the median site could save around 303 KB, while mobile home pages could save approximately 270 KB if resources were consistently loaded from cache.

Inner pages perform slightly better, but there"s still room for improvement. The median potential savings are 220 KB on desktop and 201 KB on mobile.

These numbers represent potential savings. They don"t apply to first-time visitors and there"s no guarantee that any given resource will be cached or subsequently retrieved from the browser cache.

However, given the potential savings, it may be worth reviewing your cache strategy. For example, consider looking at cache timings and review the guidance in Prevent unnecessary network requests with the HTTP Cache[452] article on eb.dev. Some adjustments may lead to faster experiences for returning users and reduce unnecessary network activity.

CDNs

Like caching, CDNs are not a definite solution for page weight, as they do not reduce file size by default. Instead, they serve as a mitigation .

[452] *https://web.dev/articles/http-cache*

CDNs can cache and serve content from a server geographically closer to the user than the origin server. This means that, while resources do not become lighter, the reduced physical distance means faster content delivery and improved perceived performance.

Many CDNs also offer additional features, such as handling compression, re-encoding and resizing of assets like images and video. These features are often automatic or near-automatic,ie, they require minimal development time, and can provide a simple way to reduce actual page weight..

If you are struggling to implement these page weight saving optimization techniques, it may be worth evaluating what your current CDN service offers, if you already use one, or exploring other CDN providers to see what their built-in tools might offer. CDNs grow increasingly prevalent and powerful, and we recommend getting more insights in the CDN chapter.

Page weight and Core Web Vitals

This year, we were able to include Core Web Vitals[453] data in our analysis. These new insights come with an important caveat: not all pages have URL-level CrUX data.

To be clear, CrUX or Chrome User Experience data is the data that is collected from opt-in Chrome users as they browse websites. This data reflects how actual users experience a web page, instead of how it performs in a controlled environment.

To be included in the CrUX dataset, a page must drive a minimum number of visitors[454]. As a result, the dataset will be weighted toward more popular sites and doesn't represent the entire crawl corpus.

Despite this limitation, we hope that you find these findings, as they offer a valuable perspective while we work to balance functionality and page weight on the ever-changing web.

45%

Figure 13.53. Mobile pages weighing 2-3 MB pass Core Web Vitals

Core Web Vitals are a set of human-centric metrics designed to measure the quality of the human experience when interacting with websites. The focus is on three key areas:

1. Largest Contentful Paint, which measures visual loading

[453] https://developers.google.com/search/docs/appearance/core-web-vitals
[454] https://developer.chrome.com/docs/crux/methodology#popularity-eligibility

2. Cumulative Layout Shift, which measures visual stability
3. Interaction to Next Time, which measures interactivity

CrUX data: Core Web Vitals assessment by weight

For a page to pass the Core Web Vitals assessment, it must have CrUX user data showing good LCP and CLS metrics at the 75th percentile, and its INP must either be good at the 75th percentile or be entirely absent. INP measures interactivity, and because not every page drives visits, INP dataset tends to be the most sparse. While page weight is not a part of their calculation, the data shows a clear correlation between total megabytes and pass rates.

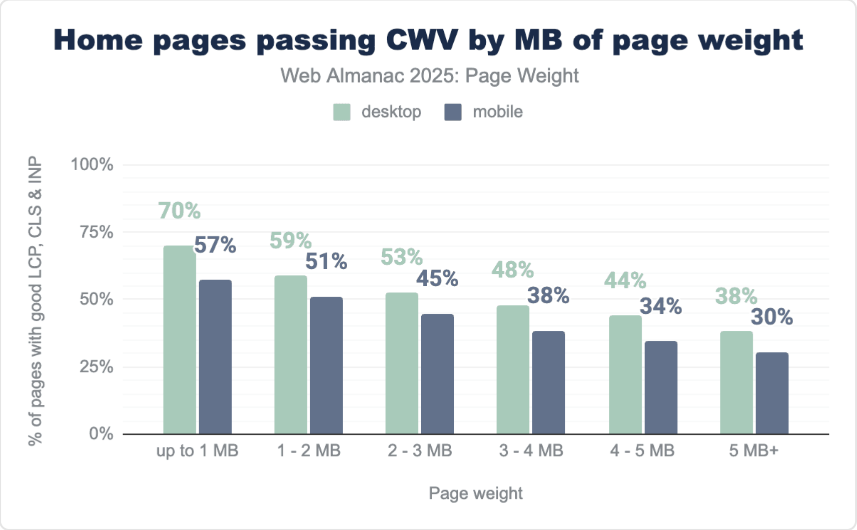


Figure 13.54. Percentage of home pages passing core web vitals, by page weight.

70% of desktop home pages under 1 MB passed this assessment. In contrast, only 38% of home pages 5 MB or larger passed, which is nearly half the success rate of their lighter counterparts. Mobile pages show a similar pattern: . 57% of pages under1 MB passed compared to only 30% of those 5 MB or larger. Passing rates decline steadily as home page weight increases with mobile consistently lagging behind desktop across all weight categories.

In 2025, the median home page was 2.9 MB for desktop and 2.6 MB on mobile. Within the 2-3 MB range, 63% of desktop pages passed the Core Web Vitals assessment. Because these figures rely on CrUX field data, they primarily reflect the performance of the most popular sites. For mobile in the same weight range, 53% of pages passed.

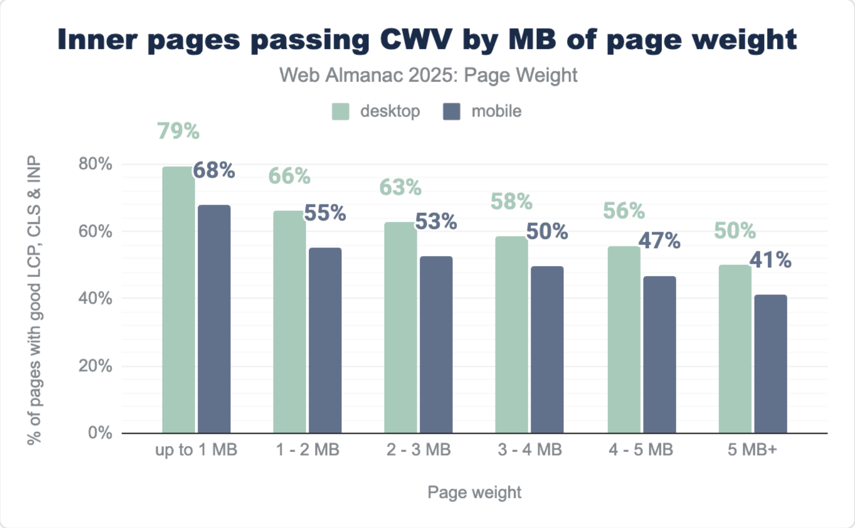


Figure 13.55. Percentage of inner pages passing core web vitals, by page weight.

As observed above, inner pages tend to be lighter than home pages. In 2025, the median home page weighted 2.9 MB for desktop and 2.6 MB for mobile. Inner pages were lighter in comparison, with median sizes of 1.8 MB for mobile, and 2 MB for desktop.

The data shows a softened pattern for inner pages than home pages. 79% of desktop inner pages weighing under 1 MB passed the Core Web Vitals assessment. 50% of inner pages 5 MB or larger passed the assessment. Mobile pages are less likely to pass, and passing rates decline gradually with each additional megabyte.

CrUX data: Largest Contentful Paint by weight

Largest Contentful Paint (LCP)[455] measures how long it takes for the largest image, text block or video in the viewport to render, starting when the user starts navigating to the page. A page achieves a passing score when the 75th percentile of user experiences fall under in 2.5 seconds.

[455] https://web.dev/articles/lcp

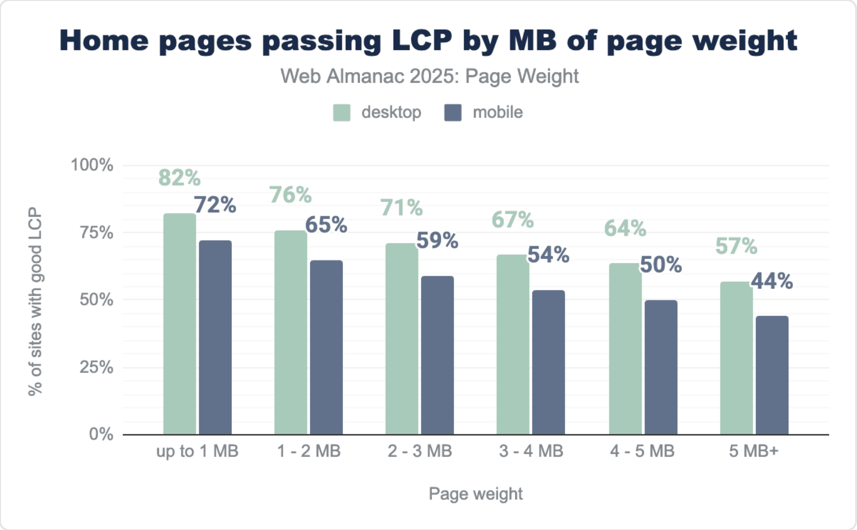


Figure 13.56. Percentage of home pages with good Largest Contentful Paint, by page weight.

The lightest pages passed LCP, with 82% of desktop and 72% of mobile pages achieving a passing score. Pages in the 2-3 MB range, which represents the median page weight, passed at rates of 71% for desktop and 59% for mobile. The heaviest of pages, ie, those 5 MB or larger, passed rates dropped to 57% for desktop and 44% for mobile.

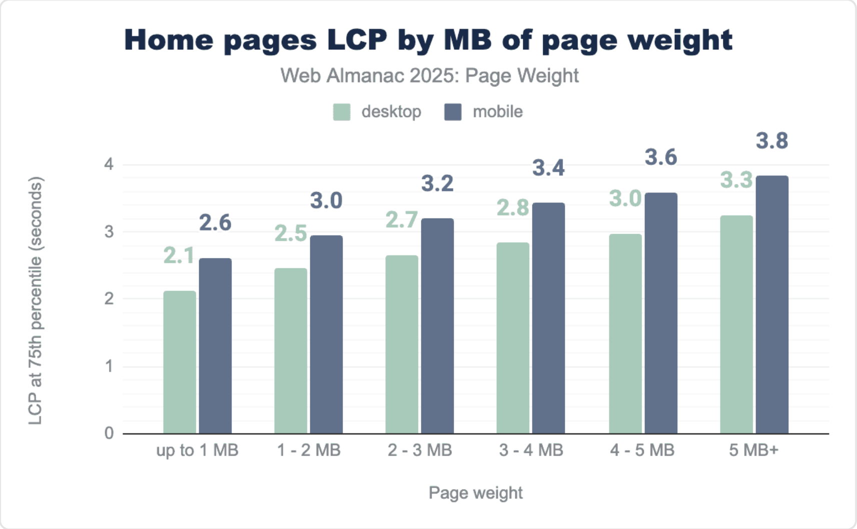


Figure 13.57. Largest Contentful Paint of home pages in seconds by page weight grouping.

For home pages in the 2-3 MB range, LCP was achieved in 2.7 seconds on desktop and 3.2 seconds on mobile. Mobile struggled to achieve sub-2.5-second times, even for pages under 1 MB, which recorded 2.6 seconds for LCP. As page weight increases, LCP times rise accordingly, with mobile taking roughly half a second longer to render the largest content element.

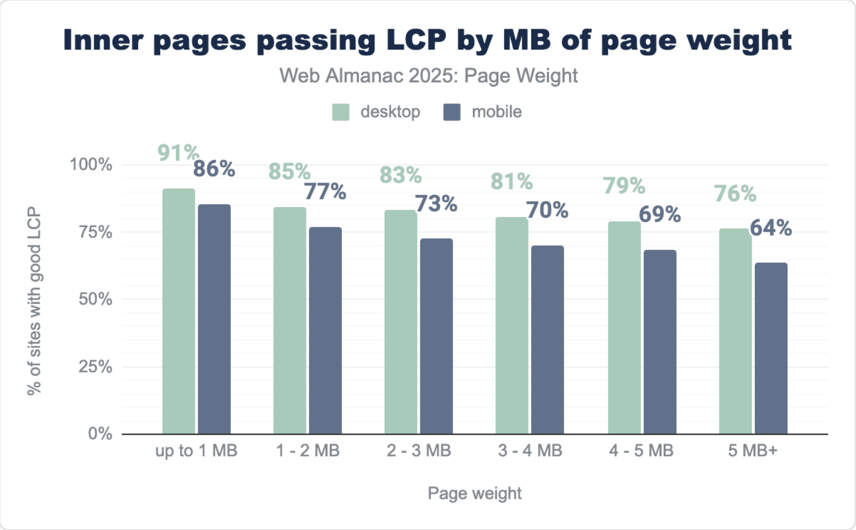


Figure 13.58. Percentage of inner pages with good Largest Contentful Paint, by page weight.

Inner pages show better Largest Contentful Paint performance than home pages. For pages under 1 MB, 91% of desktop and 86% of mobile inner pages passed the metric.

Pages in the 1-2 MB range represent the median mobile inner page size, with 77% of pages in this range achieving a score. At 2 MB, the desktop median inner page sits at the cutover for two ranges, both of which behave similarly. 83% of inner desktop pages passed in the 2-3 MB range.

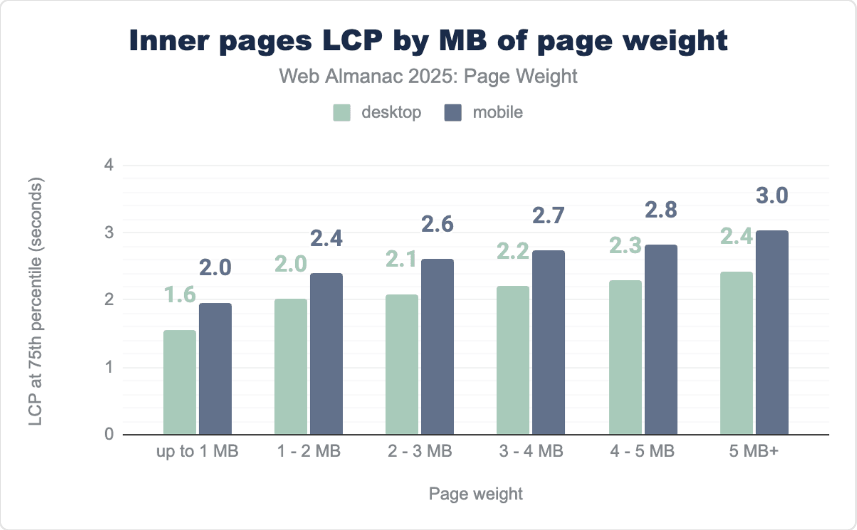


Figure 13.59. Largest Contentful Paint time of inner pages by page weight.

Inner pages show better LCP performance than home pages overall. Pages up to 2 MB met the LCP passing threshold for the 75th percentile. Desktop pages consistently passed the assessment for all size ranges.

CrUX data: Cumulative Layout Shift by weight

The second Core Web Vitals metric, Cumulative Layout Shift (CLS)[456], measures the visual stability of the page. It calculates how much visible elements unexpectedly move during the first five seconds of page load, factoring in both the size of the element that moves and the distance it travels. These shifts are averaged across the session window to produce the final score.

[456] *https://web.dev/articles/cls*

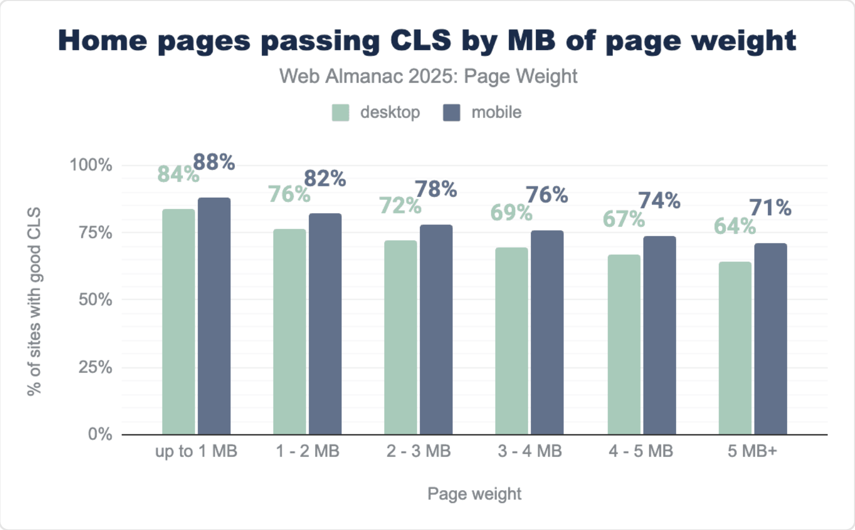


Figure 13.60. Percentage of inner pages with good Cumulative Layout Shift, by page weight.

At the 75th percentile, 72% of desktop and 78% of mobile home pages in the 2-3 MB range meet the CLS recommendation. These pages represent the median page in 2025. CLS saw less drop off between percentiles than LCP. 88% of mobile pages under 1 MB passed. 71% of mobile pages still passed when the page weight exceeded 5 MB.

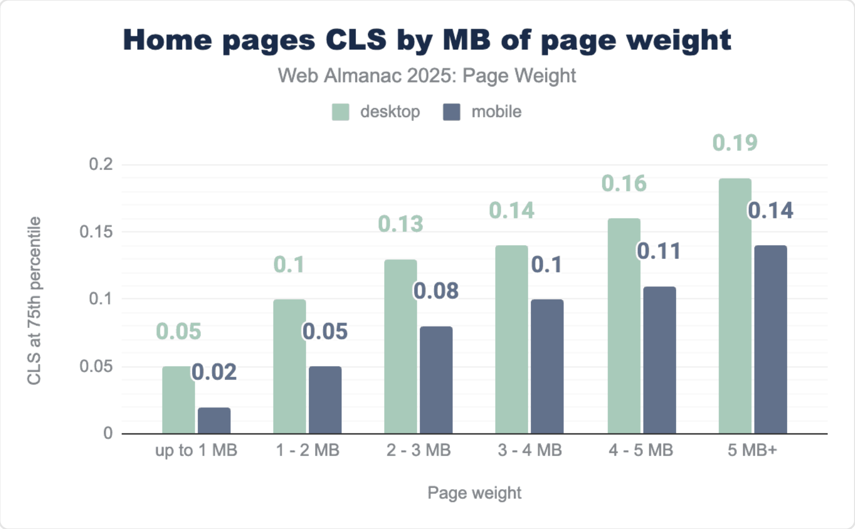


Figure 13.61. Cumulative Layout Shift of home pages by page weight.

A passing CLS score is ≤ 0.10. CLS scores were consistently higher on desktop, likely because the larger landscape provides more space for items to shift. Pages below 2 MB passed the assessment on desktop, while pages below 4 MB achieved the desired score on mobile.

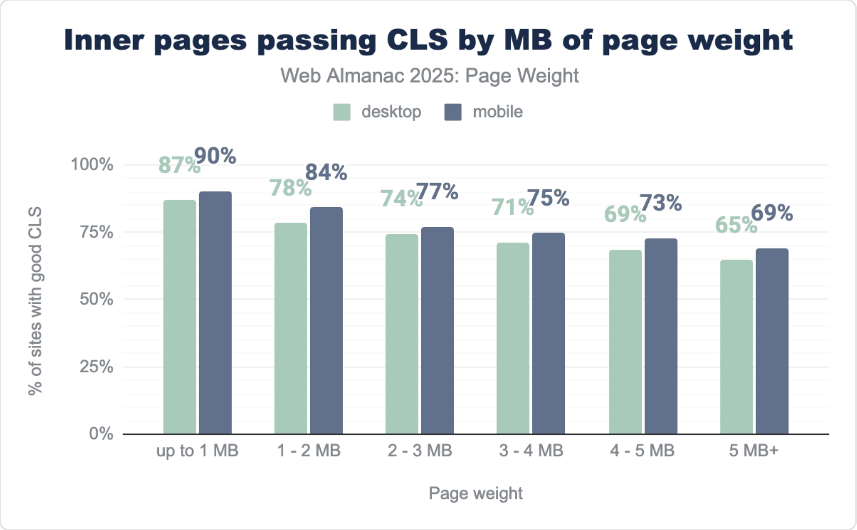


Figure 13.62. Percentage of inner pages with good Cumulative Layout Shift, by page weight.

The percentage of inner pages passing CLS was similar to the rates of home pages. While 72% of desktop mobile home pages represent the median 2-3 MB, 74% of their inner page counterparts passed. 77% of mobile inner pages passed compared to 78% of mobile desktop pages.

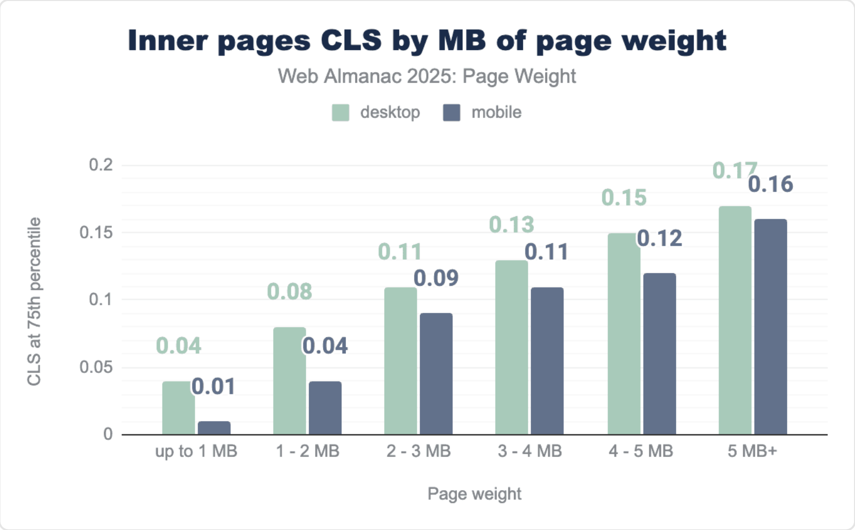


Figure 13.63. Cumulative Layout Shift of inner pages by page weight.

The impact of page weight on inner page CLS shows a consistent march across the ranges. Desktop pages under 2 MB and mobile pages under 3 MB were able to achieve a score of less than 0.10. Pages over 3 MB consistently scored over the desired threshold.

CrUX data: Interaction to Next Paint by weight

The third Core Web Vital Metric is Interaction to Next Paint (INP)[457]. Designed to represent interactivity throughout a page's lifecycle, INP measures the total latency of an interaction, from the moment a user clicks, taps or presses a key until the next visual update appears. An INP of 200 milliseconds or less indicates good responsiveness.

[457] *https://web.dev/articles/inp*

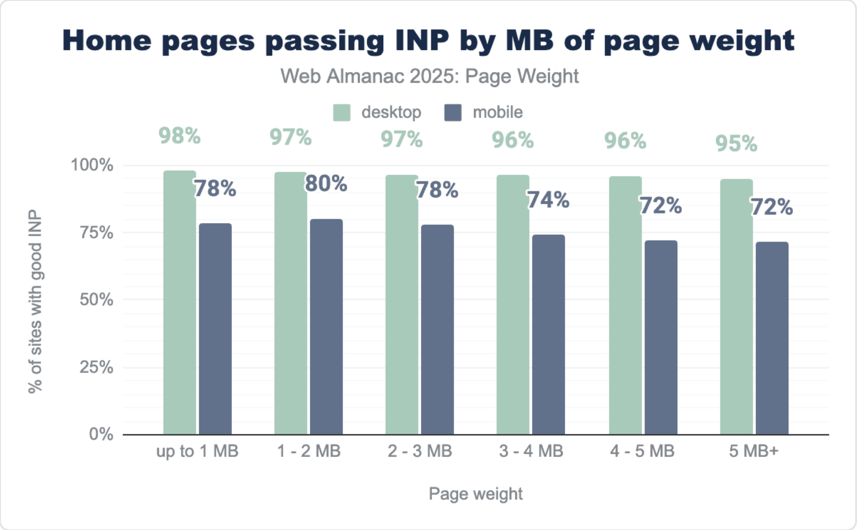


Figure 13.64. Percentage of home pages with good Interaction to Next Paint, by page weight.

The INP metric shows the highest desktop pass rates of all the Core Web Vitals metrics. For home pages under 1 MB, 98% passed the INP, and even at 5in the 5+ MB range, 95% of desktop home pages still passed the assessment.

Mobile home pages passed at lower score rates, with 78% of pages up to 1 MB passing, while those pages in the 1-2 MB range performed slightly better, with 80% of them achieving INP in under 200 ms.

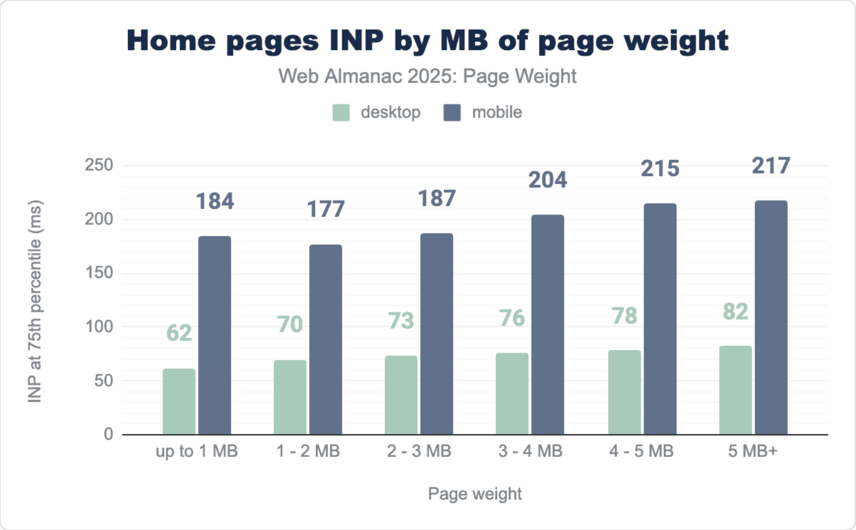


Figure 13.65. Interaction to Next Paint of home pages by page weight.

Desktop home pages consistently achieved INP times below the 200ms threshold across all weight categories even in the heaviest range, at 5 MB+, INP for the 75th percentile was just 82ms. Mobile home pages up to 3 MB achieved their INP goal. However, those in the 3-4 MB range, INP at the 75th percentile exceeded 200ms.

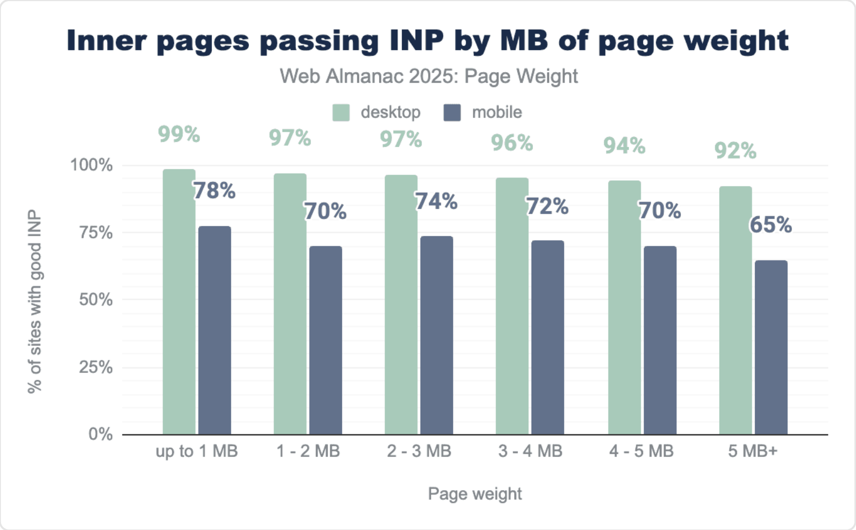


Figure 13.66. Percentage of inner pages with good Interaction to Next Paint, by page weight.

Inner pages show a similar gap between mobile and desktop. 99% of pages up to 1 MB achieved a sub-200ms INP compared to only 78% of mobile pages in the same weight range. At 5 MB+, 92% of desktop inner pages passed the interactivity assessment, while 65% of mobile pages achieved the goal.

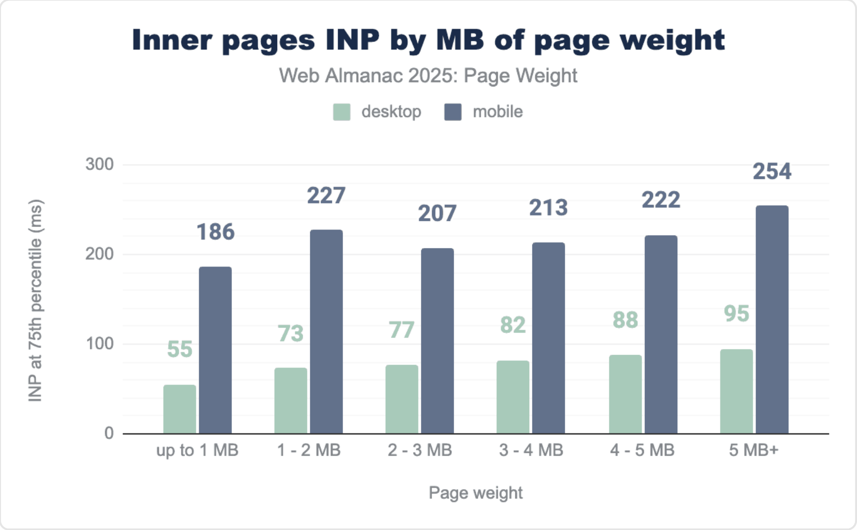


Figure 13.67. Interaction to Next Paint of inner pages by page weight.

Lighthouse data: Core Web Vitals assessment by weight

In addition to the newly available CrUX data analyzed above, we've continued our synthetic Core Web Vitals assessments, first introduced in 2024. This data is collected by running Lighthouse during the HTTP Archive crawl that powers this annual report. Lighthouse[458] is a free, open-source, automated tool developed by Google that evaluates web page quality across multiple areas, including performance, accessibility and SEO. It is frequently used by developers as part of their release process to monitor and mitigate issues in production.

It's essential to distinguish between Lighthouse data as lab or synthetic data. While designed to emulate a real experience, it can't account for all the factors that impact site performance for actual users. It also measures only the initial load of a page, limiting calculations for interactivity to ongoing interactivity and visual stability.

While CrUX is a more accurate representation of real human experience, it also has limitations. Web pages need to receive enough visits to meet data anonymization requirements before

[458] https://developer.chrome.com/docs/lighthouse

they can appear in the dataset, meaning that popular pages and websites are more likely to be represented.

When CrUX data is available for a URL, it should be considered the focus of efforts to create more performant experiences. Field data allows you to solve for real human problems. Lab data, like Lighthouse, provides more metrics to troubleshoot the issues as identified in real user metrics.

Lighthouse data: Largest Contentful Paint by weight

Largest Contentful Paint can be most accurately captured in synthetic testing of the three core web vitals. The output is only an estimate based on the specific test scenarios it runs (cold load, for its predefined screen size, network throttling, and CPU throttling). Whether those conditions are realistic for a site's particular users will determine how "accurate" the Lighthouse LCP is.

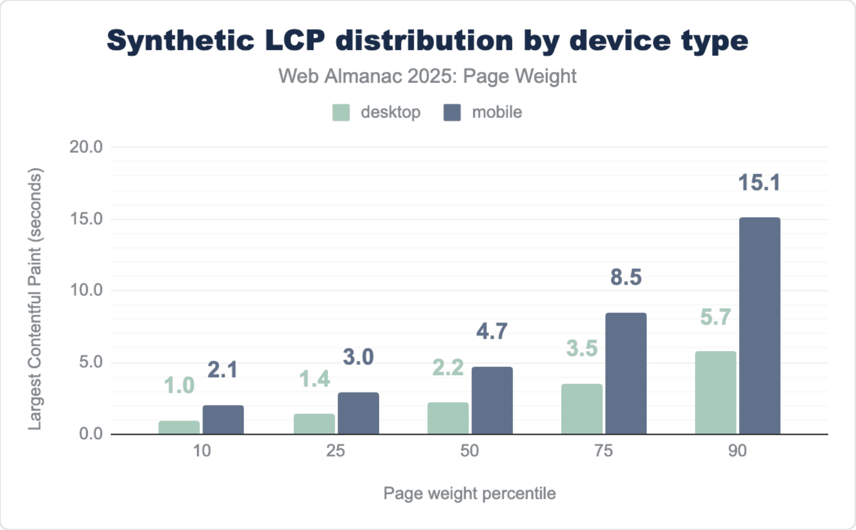


Figure 13.68. Synthetic Largest Contentful Paint in seconds, by device type.

A good Largest Contentful Paint score is less than 2.5 seconds. In Lighthouse tests, the median desktop page met this threshold, achieving 2.2 seconds. Mobile performance was significantly worse, as the same task took 4.7s to reach LCP. Only the fastest 10th percentile of mobile pages achieved a passing score, with an LCP of 2.1 seconds. In sharp contrast, mobile pages at the 90th percentile, achieved an LCP of 15.1 seconds, roughly six times slower than the recommended target. Desktop pages achieved LCP in under 2.5 seconds up to the 50th percentile.

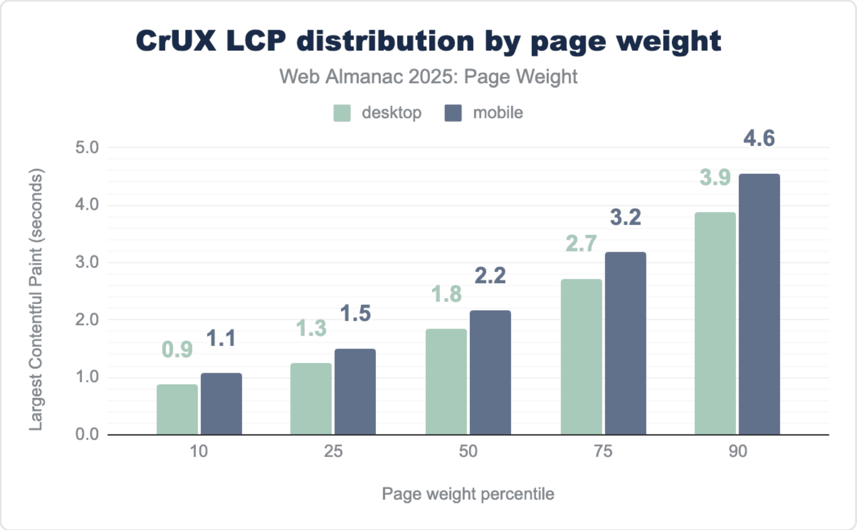


Figure 13.69. Real user metrics for Largest Contentful Paint in seconds, measured using CrUX.

Real user metrics provided by the Chrome User Experience report paint a much more performant picture of the web. Both mobile and desktop pages achieved LCP in under 2.5 seconds through the 50th percentile. At the 90th percentile, desktop field data was 1.8 seconds faster than lab data. mobile showed an even wider gap, as starting at the 25th percentile, Lighthouse reported LCP values more than double those seen in CrUX, and by the 90th percentile, synthetic results were 328% higher than real-user metrics.

Such significant differences in the data sets may serve to support the cause for the metric's existence. Pages with slower LCP are more likely to be abandoned as users perceive the lack of visual loading as unresponsive. More abandoned page loads reduce the likelihood of achieving enough page views to be eligible for the CrUX data set.

Lighthouse data: Cumulative Layout Shift by weight

Visual stability can only be measured during initial page load in synthetic testing. This does not accurately reflect the robust nature of the metric. CLS, as measured in the page loads of real users, is not a single measurement but an aggregation of all layout shifts within a "session window." A session window is a 5-second period where multiple layout shifts occur, separated by less than 1 second. The highest-scoring session window is reported as the page's CLS score.

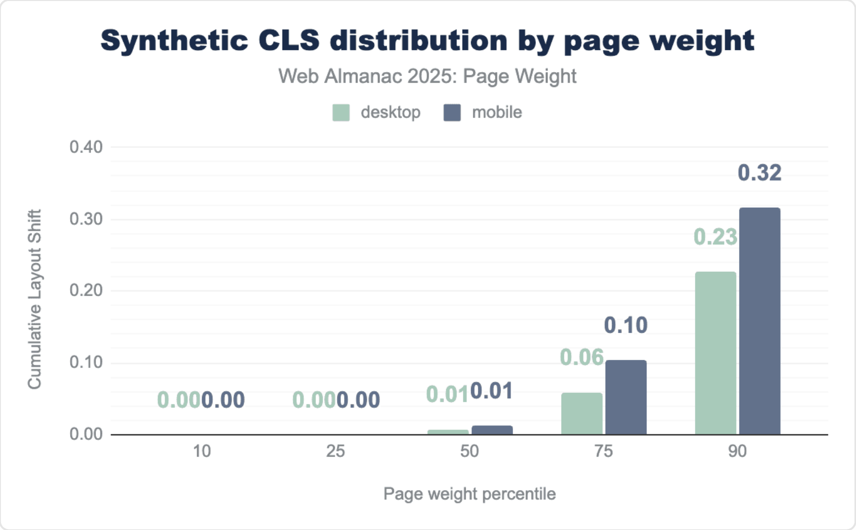


Figure 13.70. Synthetic Cumulative Layout Shift, by device type.

Synthetic data shows pages through the 75th percentile effectively passing CLS as judged by their initial pageload. Mobile and desktop scores were consistent through the 50th percentile. The highest scores were those of mobile pages in the 90th percentile, which scored 0.32.

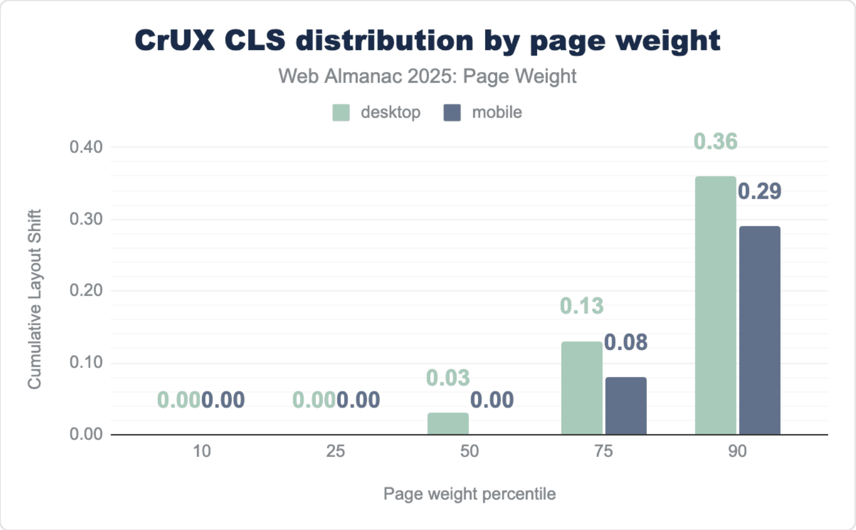


Figure 13.71. Real user metrics for Cumulative Layout Shift, measured using CrUX.

With CrUX data, we introduce an extended time frame and better accounting for the mechanics of single-page applications (SPAs) and dynamically loaded content. Real user metric calculation for CLS focuses on the largest "burst" of layout shifts that a user is likely to experience.

All users through the 50th percentile experienced CLS of less than 0.10. At the 75th percentile, desktop users saw good CLS while mobile inched over the threshold to a CLS of 0.13. At the 90th percentile, CrUX desktop users experienced higher visual instability (0.36) than mobile users (0.29). CrUX data for mobile users at the 90th percentile was lower than its synthetic counterpart (0.32 in Lighthouse; 0.29 in CrUX).

Lighthouse data: Total Blocking Time by weight

For CrUX data, interactivity is measured as Interaction to Next Paint (INP), in other words, this metric measures the time between user input and visual feedback throughout the page's lifecycle. Lighthouse can only measure the initial page load and, because of this, uses an alternative metric to estimate interactivity. Total Blocking Time (TBT)[459] is a lab metric that measures the total time during page load that the main thread was blocked. TBT serves as a proxy for INP because high TBT often leads to high INP, meaning that fixing TBT is a key way to improve INP.

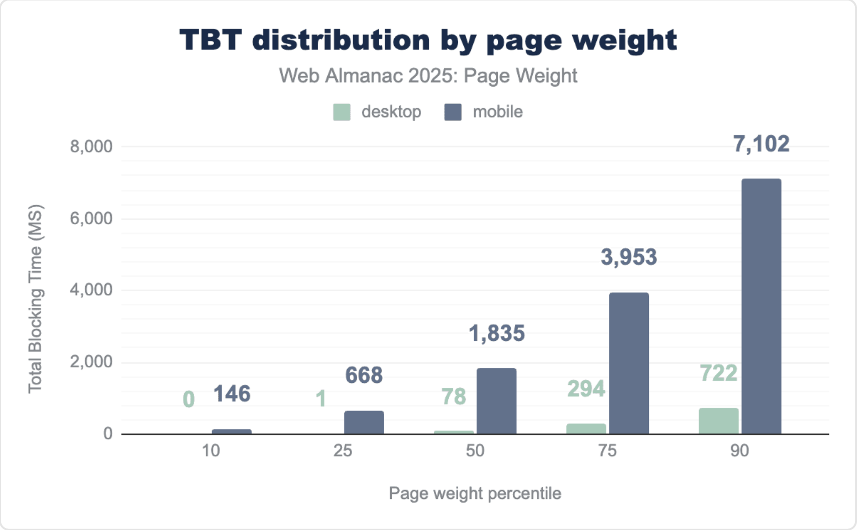


Figure 13.72. Total Blocking Time in milliseconds, measured using Lighthouse.

[459] https://web.dev/articles/tbt

A page is considered to have good TBT when the main thread is blocked for less than 200 milliseconds. Only the 10th percentile of mobile pages achieved this metric. Desktop pages achieved a TBT of under 200 ms through the 50th percentile. At the 90th percentile, the main thread of mobile devices was blocked for 7.1 seconds.

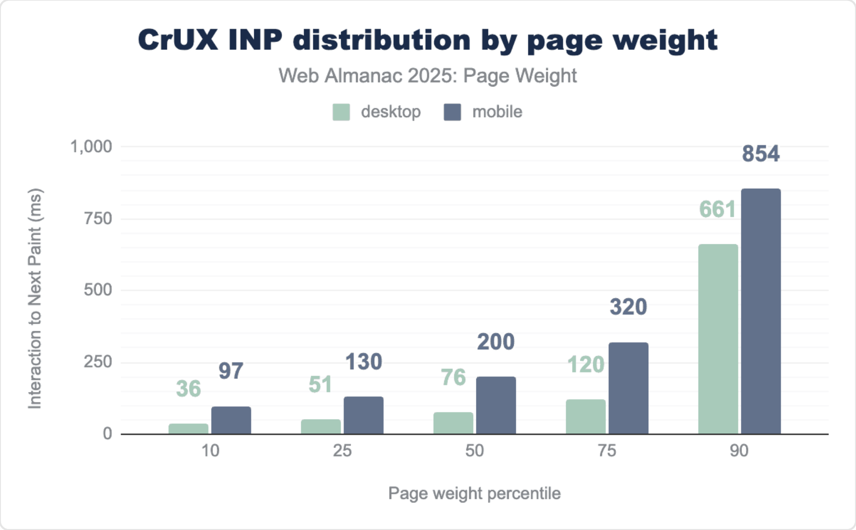


Figure 13.73. Real user metrics for Interaction to Next Paint in milliseconds, measured using CrUX.

INP, the field metric counterpart to TBT, shares the same timing goal of 200 ms. CrUX paints a more responsive picture of mobile experiences. Mobile and desktop pages through the 50th percentile achieved good responsiveness. At the 75th percentile, the differences between devices become more pronounced, with INP of 120 ms for desktop, still considered good responsiveness, and 320 on mobile, meaning users perceive more lag. The 90th percentile of mobile CrUX data calculated at INP at 4X the 200 ms goal. The 90th percentile of synthetic mobile data saw TBT 35X the same goal.

Conclusion

The simple conclusion is that all pages on the web continue to grow in size year on year. This growth is broadly accelerating increasing demand on user devices and internet connection.

The balance of functionality versus accessibility is continuing to tilt in favour of functionality meaning users on limited connections or devices suffer a worse and less inclusive experience.

Javascript is still a primary driver of this growth with 98.1% of pages making at least one request. Javascript is often inefficiently added to pages, adding weight to the pages but then not being actively used.

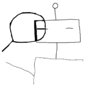
Images and their wide use on both home pages and inner pages contribute a significant amount of the weight to pages, the discrepancy between desktop and mobile has shrunk versus last year which implies that optimization is less of a focus which is troubling given the impact images have on the web experience.

The effect of all this weight can be seen in the high falling rates of Core Web Vitals as the weight increases, including in Chrome User Experience report data, showing the impact is real for the user, not theoretical.

Ultimately there is massive opportunity for improvements here without impacting the user experience at all, lossless compression and only using what is needed for javascript represent significant gains for users and reduced costs for web owners, but currently too many of these opportunities are being ignored.

Page weight continues to be a key issue but currently it seems we are getting further away from a solution.

Authors



Richard Barret

@barkseo.bsky.social rickb110 richardbarrettseo

Richard Barrett is director of professional services and a technical search consultant at Lumar[460]. They like helping web owners solve awkward problems and love a good puzzle game or two, or three.

Jamie Indigo

@not-a-robot.com fellowhuman1101 jamie-indigo https://not-a-robot.com

Jamie Indigo isn't a robot, but speaks bot. As director of technical SEO at Cox Automotive[461], they study how search engines crawl, render, and index the web. Jamie loves to tame wild JavaScript and optimize rendering strategies. When not working, they like horror movies, graphic novels, and terrorizing lawful good paladins in Dungeons & Dragons.

[460] https://www.lumar.io/
[461] https://www.coxautoinc.com/

Part IV Chapter 14

CDN

Written by Joe Viggiano, Alex Moening, and Nick McCord
Reviewed by Caroline Scholles
Analyzed by Alex Moening, Nick McCord, and Joe Viggiano
Edited by Caroline Scholles

Introduction

This chapter examines the rapidly evolving landscape of Content Delivery Networks (CDNs) in 2025, with an increased focus on their role in HTTP protocol optimization. CDNs fundamentally exist to solve HTTP delivery challenges at scale, from reducing TCP connection overhead through HTTP/2 multiplexing to eliminating head-of-line blocking via HTTP/3's QUIC transport. As HTTP protocols have evolved from HTTP/1.1's connection limitations to HTTP/3's advanced features, CDNs have served as the primary deployment vehicle, implementing these protocols years before origin servers adopt them.

Modern CDNs optimize delivery across the entire spectrum of web content. For highly cacheable resources (static assets, public API responses, shared content), CDNs provide traditional caching benefits enhanced by advanced compression and modern format delivery. For content with limited cacheability (user-specific data, frequently updated APIs, personalized experiences), CDNs still deliver performance improvements through connection optimization, intelligent routing, and edge processing, reducing latency even when content cannot be cached.

Building upon our comprehensive analysis from 2024, our 2025 analysis reveals shifts in protocol adoption and optimization strategies. A key focus of this chapter is the maturation of HTTP/3 adoption, the rise of modern optimization techniques like Server-Timing transparency, and the sophisticated multi-layered approaches CDNs are taking to performance, security, and user experience.

What is a CDN?

A *Content Delivery Network* (CDN) is a geographically distributed network of servers designed to provide high availability, enhanced performance, and improved security for web content and applications. The primary goal of a CDN is to minimize latency and optimize content delivery by serving data from locations closer to the end user.

CDNs serve as intermediary infrastructure between end users and origin servers, intercepting web requests and optimizing the complete delivery process. To understand how CDNs can enhance web performance, consider the traditional web interaction when a user types a hostname into a browser, and how different CDNs may improve each step:

- **DNS Resolution**
    - **Traditional**: Browser queries DNS for origin server IP, often with slow resolution times
    - **CDN Processed**: CDN DNS infrastructure may use various routing strategies (anycast or unicast) to direct users to optimal edge servers. Some CDNs support modern DNS records like HTTPS or SVCB (Service Binding) records that can advertise protocol capabilities directly in DNS responses, though adoption varies across providers
- **Connection Establishment**
    - **Traditional**: Browser establishes new TCP connection to distant origin server with full handshake overhead
    - **CDN Processed**: Connection to nearby edge server over TCP (for HTTP/1.1 and HTTP/2) or UDP with QUIC (for HTTP/3). CDNs may support HTTP/3's 0-RTT connection resumption for returning visitors, though not all CDNs have implemented these newer connection optimization features

- **Protocol Negotiation**
  - **Traditional**: Limited to origin server's protocol capabilities, often older HTTP versions
  - **CDN Processed**: Many CDNs can advertise modern protocol availability through `Alt-Svc` (Alternative Service) HTTP headers that inform browsers about alternative protocols. CDNs typically provide protocol translation benefits, accepting newer protocols from browsers while maintaining connections to origins, regardless of origin server capabilities
- **Request Processing & Optimization**
  - **Traditional**: Basic request forwarding with minimal processing
  - **CDN Processed**: Depending on the CDN, may include header normalization, intelligent routing decisions, addition of performance headers like Server-Timing which provides server-side performance metrics, security headers, and request optimization based on content type and user geographic location
- **Response Processing**
  - **Traditional**: Direct response from origin server, limited by origin's HTTP server capabilities
  - **CDN Processed**: CDNs may implement advanced caching strategies, cache validation, Content-Encoding optimization (such as Brotli or Gzip compression), conditional request support (like 304 Not Modified responses that save bandwidth), and response transformation, though specific features vary by provider
- **Connection Management**
  - **Traditional**: Single connection per request or basic keep-alive to origin
  - **CDN Processed**: Many CDNs implement dual-sided connection optimization, maintaining persistent connections to clients while using

intelligent connection pooling to origin servers, reducing overhead on both ends

CDNs serve as deployment platforms for emerging web standards, implementing new HTTP headers, compression algorithms, and security features at scale before they become widely adopted by origin servers. This positions CDNs as critical infrastructure for web technology evolution, though the specific features and optimizations available depend significantly on the CDN provider and their technology adoption timeline.

Caveats and disclaimers

Our 2025 analysis builds upon the methodology established in previous years while incorporating new metrics and deeper performance analysis. The statistics gathered focus on applicable technologies and optimization patterns rather than vendor-specific performance comparisons.

Important note on measurements: All TLS negotiation times, DNS resolution times, and performance metrics in this analysis are measured from HTTP Archive's simulated browser connections using Chrome on controlled infrastructure. These measurements represent:

- Consistent network conditions (controlled datacenter connectivity)
- Chrome browser's TLS implementation
- First-time connections without session resumption
- Standardized measurement methodology across all CDNs

Real-world performance may vary based on geographic proximity to CDN edge servers, network conditions, TLS session resumption capabilities, and client device characteristics.

Key limitations of our testing methodology:

- **Simulated network conditions:** Tests use controlled network environments
- **Single geographic perspective:** Analysis from limited datacenter locations
- **Cache effectiveness:** Each CDN uses proprietary technology and many, for security reasons, do not expose cache performance or depth of cache
- **Localization and internationalization:** Just like geographic distribution, the effects of language and geographic specific domains are also opaque to these tests

- **CDN detection:** This is primarily done through DNS resolution and HTTP headers. Most CDNs use a DNS CNAME to map a user to an optimal data center. However, some CDNs use Anycast IPs or direct A+AAAA responses from a delegated domain which hide the DNS chain. In other cases, websites use multiple CDNs to balance between vendors, which is hidden from the single-request pass of our crawler
- **Sampling bias:** HTTP Archive data reflects popular websites, potentially overrepresenting certain CDN providers
- **Market share interpretation:** Our data shows CDN usage patterns among crawled sites, not true market distribution

CDN adoption

A web page is composed of the following key components:

1. Base HTML page (for example, `www.example.com/index.html`—often available at a more friendly name like just `www.example.com`).
2. Embedded first-party content such as images, CSS, fonts and JavaScript files on the main domain (`www.example.com`) and the subdomains (for example, `images.example.com`, or `assets.example.com`).
3. Third-party content (for example, Google Analytics, advertisements) served from third-party domains.

The evolution of CDN adoption patterns reflects the changing nature of web architecture and the increasing complexity of modern web applications. CDNs continue to prove their value across different content types, with varying adoption rates that reflect their suitability for different use cases.

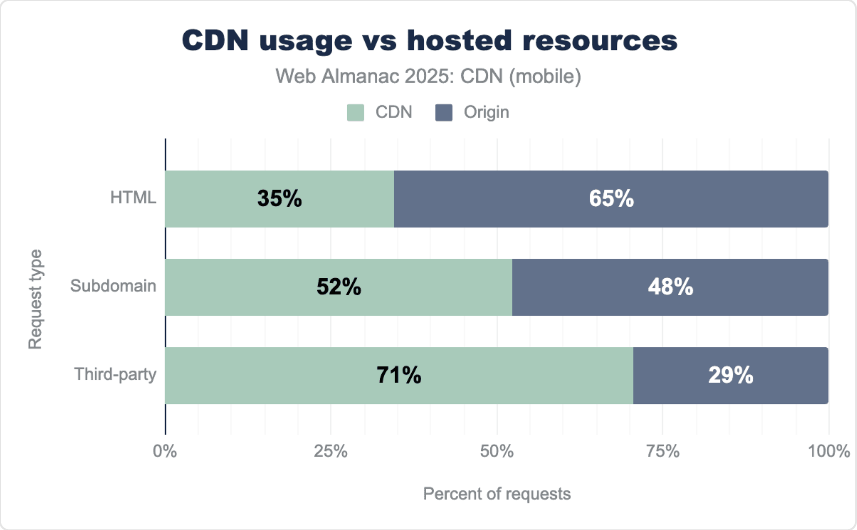


Figure 14.1. CDN usage vs hosted resources on mobile.

The chart shows the breakdown of requests for different types of content (HTML, Subdomain, and Third-party), showing the share of content served by CDN versus origin on mobile devices.

CDNs are often utilized for delivering static content such as fonts, image files, stylesheets, and JavaScript. This kind of content doesn't change frequently, making it a good candidate for caching on CDN proxy servers. We still see CDNs used more frequently for this type of resource, especially for third-party content, with 71% being served via CDN. Subdomain resources show moderate CDN adoption at 52%, while HTML content has the lowest CDN usage at 35%, as it's more commonly served directly from origin servers.

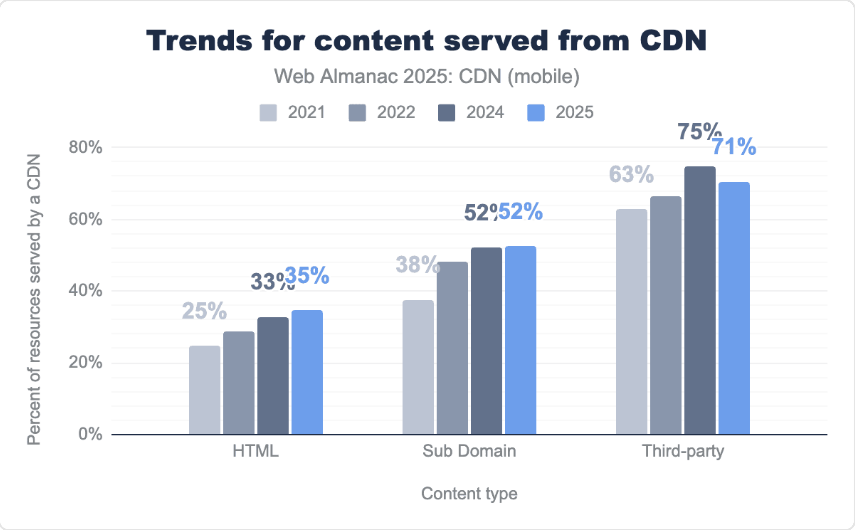


Figure 14.2. Trends for content served from CDN for mobile.

From 2024 to 2025, we see mixed trends across content types. HTML content continued its upward trajectory, increasing from 33% to 35%. Subdomain content remained stable at 52% in both years. However, third-party content experienced a notable decline from 75% in 2024 to 71% in 2025, representing a four percentage point decrease after years of consistent growth.

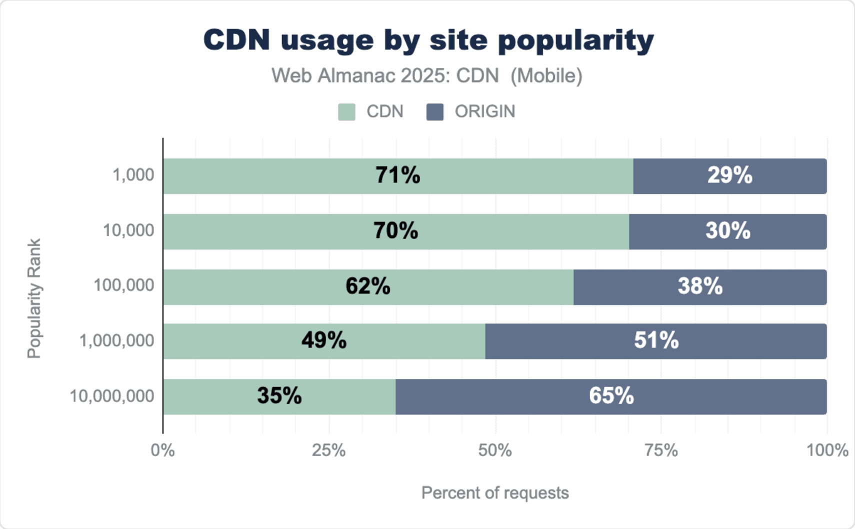


Figure 14.3. CDN usage by site popularity on mobile.

The share of CDN usage has increased over the years, particularly among the most popular websites according to Google Chrome's UX Report (CrUX) classification. As the graph shows, the top 1,000 websites have the highest CDN usage at 71%, followed by the top 10,000 at 70%, and the top 100,000 at 62%. Compared to 2024, CDN adoption has increased regardless of popularity rank.

As mentioned in previous editions, the increase in CDN usage of 33% in 2024 to 35% in 2025 among smaller sites can be attributed to the rise of free or flat-tiered or affordable CDN options. Additionally, many hosting solutions now bundle CDNs with their services, making it easier and more cost-effective for websites to leverage this technology.

CDN providers

CDN providers can generally be classified into two segments:

1. **Generic CDNs**: Providers that offer a wide range of content delivery services to suit various use cases, including Akamai, Cloudflare, Amazon CloudFront, and Fastly.
2. **Purpose-built CDNs**: Providers tailored to specific platforms or use cases, such as Netlify and WordPress.

Generic CDNs address broad market needs with offerings that include:

- Website delivery
- Web application API delivery
- Video streaming
- Edge Computing services
- Web security offerings

These capabilities appeal to a wide range of industries, which is reflected in the data.

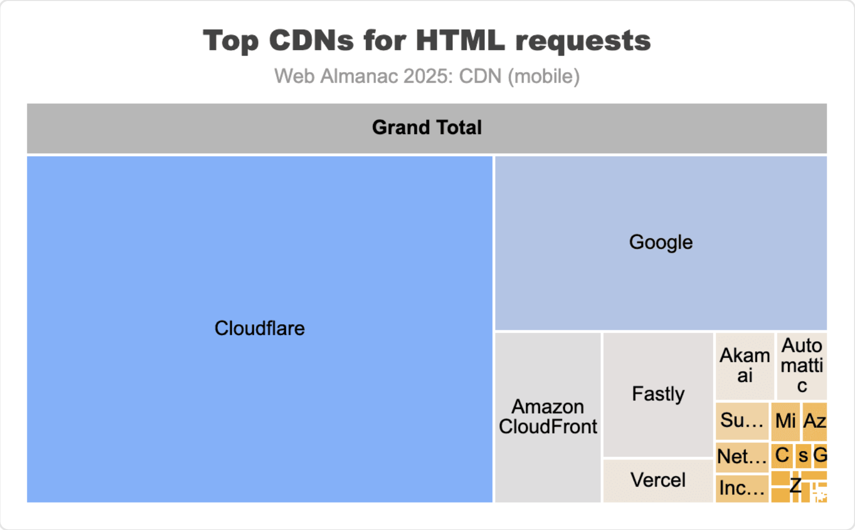


Figure 14.4. Top CDNs for HTML requests on mobile.

The leading vendors for serving base HTML requests are Cloudflare, with a 58% share, followed by Google (21%), Amazon CloudFront (7%), Fastly (5%), and Akamai and Vercel, each with a 2% share.

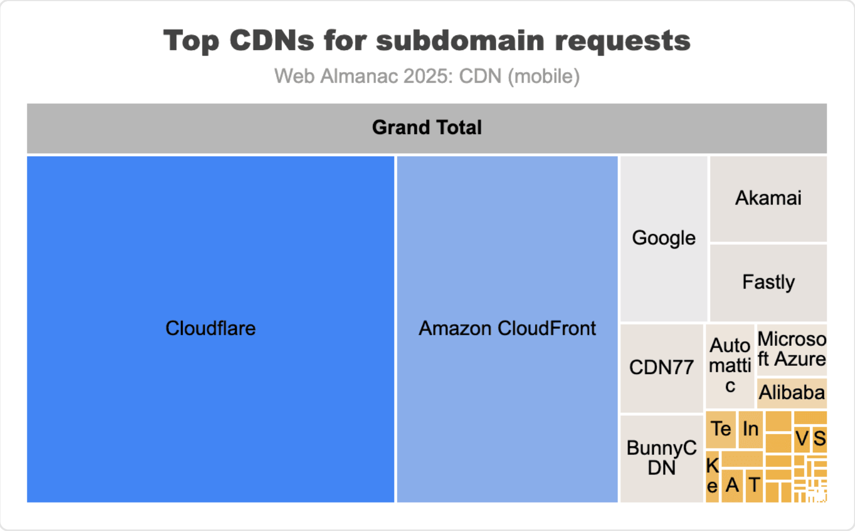


Figure 14.5. Top CDNs for subdomain requests on mobile.

Marginally changing from 2024, the leading vendors in this category are Cloudflare (46%), Amazon CloudFront (28%), Google (5%), and Akamai (4%).

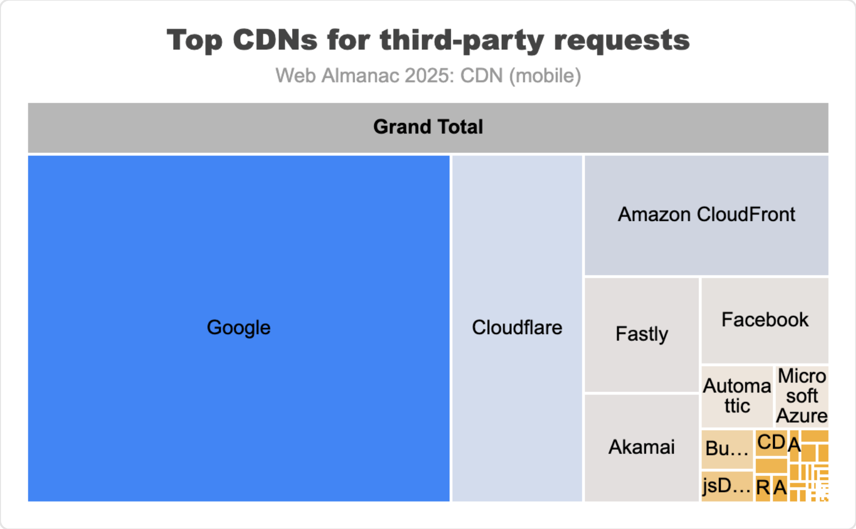


Figure 14.6. Top CDNs for third-party requests on mobile.

Google leads the list third-party domain usage at 53% market share, followed by well-known CDN providers such as Cloudflare (17%), Amazon CloudFront (11%), Fastly (5%), and Akamai and Facebook (4%).

While many CDNs offer purpose-built features optimized specifically for content delivery, they increasingly exist as part of larger service ecosystems. These CDNs are often tightly integrated with cloud infrastructure, security solutions, and edge computing platforms, with these adjacent services delivered through or alongside the CDN itself.

Different CDN providers take distinct approaches to optimization and specialization. Third-party platforms like Google and Facebook build highly specialized CDNs engineered specifically for their needs, handling massive throughput for ad delivery and capturing analytics beacons at scale. In contrast, general-purpose CDNs like Cloudflare and Amazon CloudFront optimize targeted feature sets while maintaining broader applicability. These platforms leverage their CDN capabilities as a foundation for managed services, enabling use cases such as globally distributed API gateways or real-time JavaScript injection for client-side device fingerprinting and security inspection.

HTTP/3 adoption

Published in June 2022 by IETF, HTTP/3[462] is a major revision of the HTTP network protocol, succeeding HTTP/2. The transition to HTTP/3 represents one of the most significant protocol upgrades in web history. Built on QUIC transport, HTTP/3 eliminates head-of-line blocking and reduces connection establishment overhead. Our 2025 analysis reveals dramatic shifts in adoption patterns, particularly among CDN providers.

HTTP/3's performance improvements stem from fundamental protocol design changes that CDNs are uniquely positioned to leverage at global scale. Unlike HTTP/2's reliance on TCP, HTTP/3 uses QUIC over UDP, eliminating TCP's head-of-line blocking. When one HTTP/2 stream encounters packet loss, all multiplexed streams on that TCP connection stall. CDNs implementing HTTP/3 can maintain performance for unaffected requests through QUIC's independent stream recovery. This is particularly valuable for CDNs serving mixed content where a slow loading image won't block critical CSS or JavaScript delivery.

HTTP/3 reduces connection establishment from HTTP/2's 3 RTTs (TCP handshake, TLS handshake, and HTTP negotiation) to 1 RTT through QUIC's integrated cryptographic handshake. CDNs amplify this benefit through geographic proximity, as users connecting to nearby CDN edge servers experience reduced connection time. For returning visitors, some CDNs implement 0-RTT connection resumption, eliminating handshake overhead entirely when reconnecting to the same edge server.

Modern CDNs are beginning to implement HTTPS DNS records, a type of Service Binding (SVCB) / HTTPS records that allows DNS responses to advertise HTTP/3 availability and connection parameters directly. When a browser queries DNS for a CDN enabled domain, HTTPS records can indicate HTTP/3 support on port 443 with specific QUIC parameters, enabling immediate HTTP/3 connections without the traditional HTTP/2 to HTTP/3 upgrade process. This DNS level optimization is particularly powerful for CDNs managing multiple domains and services, as it eliminates protocol upgrade negotiations and can reduce connection establishment time. However, HTTPS record implementation varies significantly across CDN providers, with some leading adoption while others have not yet implemented support.

Important methodological considerations

Our HTTP/3 adoption measurements reflect the specific characteristics of HTTP Archive's methodology and should not be interpreted as representative of overall internet traffic:

[462] *https://datatracker.ietf.org/doc/html/rfc9114*

- **Site Selection**: Analysis focuses on popular websites (top 10M) that are more likely to implement modern protocols
- **Resource Type Focus**: Measurements primarily reflect static resources (CSS, JS, images) served by CDNs rather than main document loads
- **Geographic Scope**: Testing from limited datacenter locations may not reflect global user experience
- **External Validation**: Cloudflare's own 2025 data reports ~21% HTTP/3 adoption globally, significantly lower than the 69% we observe for Cloudflare resources in our dataset

These measurements are valuable for understanding protocol adoption among leading web properties and CDN-served resources, but should not be extrapolated to represent general internet usage patterns.

HTTP/3 adoption among CDN providers

In 2025, using consistent methodology from previous years, we observed significant HTTP/3 adoption among CDN-served resources. While these numbers reflect the specific subset of sites and resources in our dataset, they demonstrate clear patterns in CDN protocol deployment:

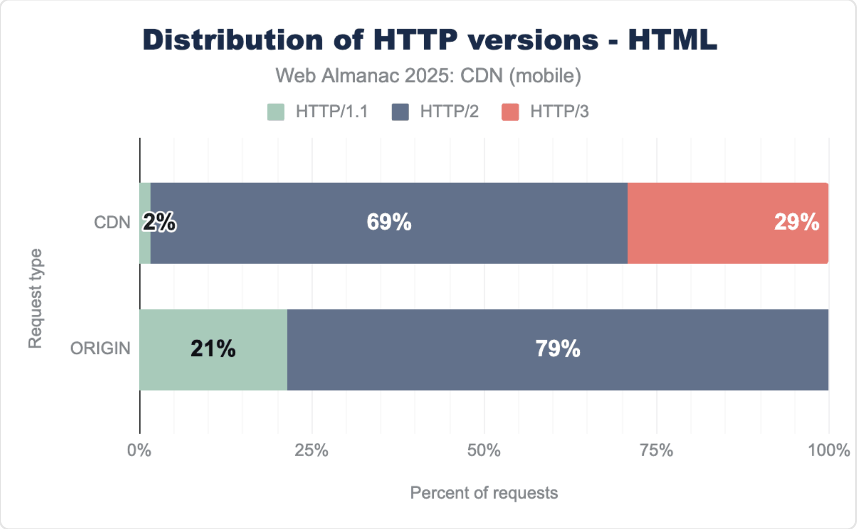


Figure 14.7. Distribution of HTTP versions for HTML (mobile).

Within our dataset, the data reinforces the role CDNs play in driving the adoption of new protocols among popular websites and static resources. CDNs saw 29% traffic from HTTP/3 with effectively 0% for origin traffic. Compared to 2024, we observed a sharp decrease in CDN usage for HTTP/1.1 in 2025.

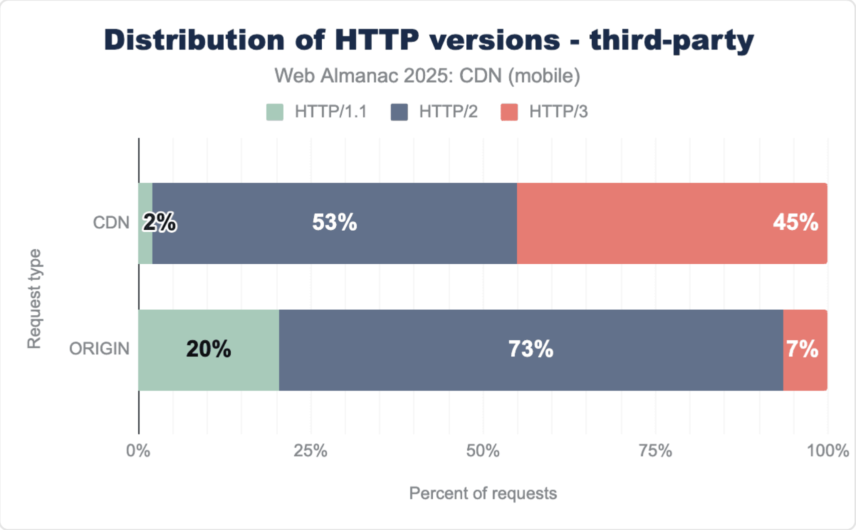


Figure 14.8. Distribution of HTTP versions for third-party requests (mobile).

While HTTP/1.1 usage has been on a continued decline with CDNs over the past several years, in 2025 we observed a sharp decrease in CDN usage for HTTP/1.1 going from 16% usage for HTML requests in 2024 to just 2% in 2025. This decrease was even more pronounced for origin requests with 56% HTTP/1.1 requests in 2024 down to 21% in 2025.

Looking at individual CDNs HTTP/3 support we see the following for documents (often the initial requests):

CDN	Desktop	Mobile
Cloudflare	*44.0%*	*49.9%*
CDN	*6.2%*	*5.0%*
Amazon CloudFront	*3.2%*	*3.8%*
Sucuri Firewall	*0.5%*	*0.6%*
Akamai	*0.2%*	*0.2%*
QUIC.cloud	*0.1%*	*0.1%*
Google	*0.1%*	*0.1%*

Figure 14.9. Top rates of document requests server by HTTP/3.

Special call out to Cloudflare who are leading the industry here.

While subresources requests have a much higher rate of adoption since they often are used for multiple requests where subsequent requests are more likely to be made over HTTP/3:

CDN	Desktop	Mobile
Facebook	*99.4%*	*99.9%*
Nexcess CDN	*97.7%*	*99.6%*
Sucuri Firewall	*71.2%*	*68.4%*
Cloudflare	*68.6%*	*69.5%*
Reapleaf	*59.5%*	*59.1%*
Erstream	*31.5%*	*77.4%*
QUIC.cloud	*48.8%*	*47.3%*
Google	*39.6%*	*42.2%*
Pressable CDN	*40.1%*	*40.8%*
Automattic	*25.6%*	*29.0%*

Figure 14.10. Top rates of document requests server by HTTP/3.

CDN Performance

CDN performance extends beyond simply caching content closer to users. CDNs actively optimize the underlying protocols and connection mechanisms that determine how quickly browsers can establish connections and receive data while providing transparent metrics to understand bottlenecks in modern web applications. Performance optimization techniques include connection pooling, protocol translation, and intelligent routing—all of which contribute to reduced latency and improved user experience.

HTTP/3 TTFB Performance

Below shows the time to first byte (TTFB) median percentile distribution for latency of HTTP/3, HTTP/2, and HTTP/1.1 across major CDNs. These measurements reflect simulated browser connections and may vary from real-world performance.

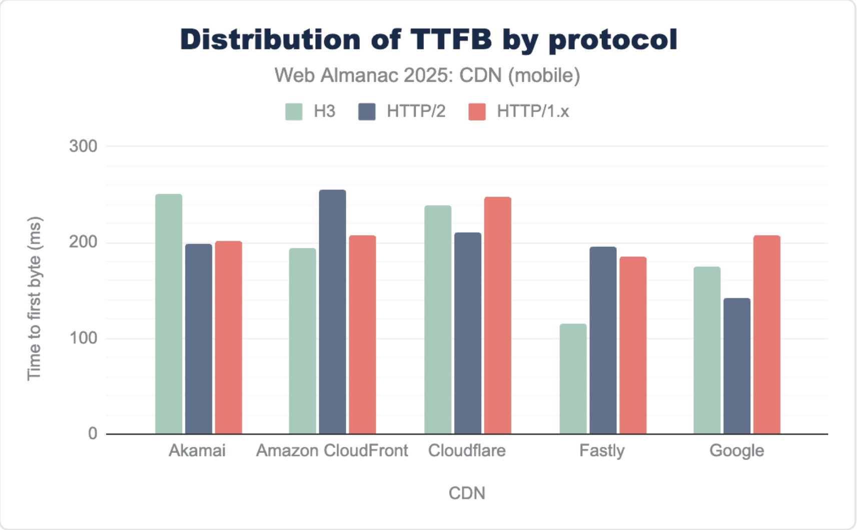


Figure 14.11. Distribution of Time to First Byte (TTFB) (mobile).

The TTFB performance for HTTP/3 compared to previous HTTP protocol versions varies by CDN provider, with Amazon CloudFront and Fastly both showing improved TTFB latency. Note that these are not uniform tests across CDNs and website owners should perform their own controlled performance testing.

DNS Performance

DNS resolution is the critical first step in any web request. Our 2025 data shows significant performance advantages for CDNs over origin servers:

DNS Resolution Performance (Median):

- **CDN (Mobile)**: 176ms
- **Origin (Mobile)**: 217ms
- **CDN advantage**: 19% faster
- **CDN (Desktop)**: 52ms
- **Origin (Desktop)**: 129ms
- **CDN advantage**: 60% faster

The desktop performance advantage is particularly striking, with CDNs delivering DNS responses 2.5x faster than origin servers. This improvement stems from CDNs' use of anycast routing, extensive edge networks, and optimized DNS infrastructure.

Alt-Svc

The Alt-Svc[463] (Alternative Services) HTTP response header tells browsers about alternative protocols or servers they can use to access the same resource. The most common use case is advertising HTTP/3 support. When a browser first connects to a server using HTTP/1.1 or HTTP/2, the server can include an Alt-Svc header like: `Alt-Svc: h3=":443"; ma=86400`

This informs the browser that it can use HTTP/3 on port 443 to reach the same service, and that this information remains valid for 86400 seconds (24 hours). Once the browser receives this header, it can establish future connections using HTTP/3 directly rather than beginning

[463] *https://datatracker.ietf.org/doc/html/rfc7838#section-3*

with HTTP/2 and negotiating an upgrade.

Our 2025 data shows:

- **Google HTTP/2 → H3 advertising**: 99.2% (44M HTTP/2 requests advertising HTTP/3)
- **Cloudflare HTTP/2 → H3 advertising**: 100% (44M HTTP/2 requests advertising HTTP/3)
- **Amazon CloudFront HTTP/2 → H3 advertising**: 100% (28M HTTP/2 requests advertising HTTP/3)
- **Origin servers HTTP/2 → H3 advertising**: 99.89% (54M HTTP/2 requests advertising HTTP/3)

The data demonstrates that CDNs using HTTP/2 nearly universally advertise HTTP/3 availability through Alt-Svc headers, enabling browsers to upgrade to HTTP/3 for subsequent requests. This widespread protocol advertising helps drive the broader adoption of HTTP/3 across the web.

Server-Timing

Defined in the W3C Server-Timing specification, the `Server-Timing` HTTP header allows servers to communicate performance metrics about request processing back to the browser. This header communicates performance metrics directly, transforming opaque server processing into observable and debuggable data. Specific to CDNs, `Server-Timing` headers can be useful for providing transparency into CDN edge processing time, origin fetch duration, or cache status without requiring additional monitoring infrastructure.

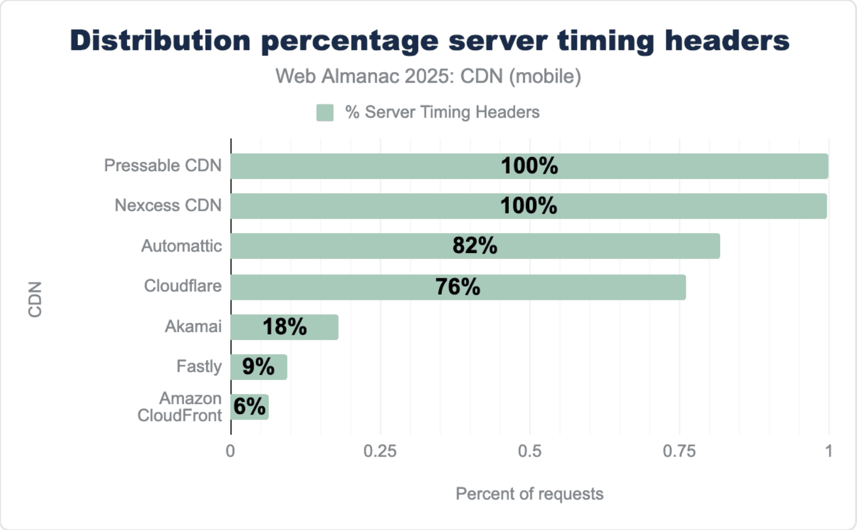


Figure 14.12. Distribution percentage server timing headers (mobile).

Adoption of the `Server-Timing` header varies across CDNs. We see Pressable and Nexcess CDNs had 100% adoption across their requests due to default configurations. CDNs like Akamai, Amazon CloudFront, and Fastly require non-default configuration likely leading to less adoption. However, enterprise concerns around security, privacy, and performance may drive this opt-in approach.

CDN Security Headers

CDNs play a critical role in web security by implementing and enforcing security headers that protect users from common attacks. Security headers like HTTP Strict Transport Security (HSTS), X-Frame-Options (XFO), and Content Security Policy (CSP) help prevent everything from man-in-the-middle attacks to clickjacking and cross-site scripting. Because CDNs sit between users and origin servers, they can insert or modify these headers regardless of what the origin provides, making it easier for site operators to deploy security best practices.

Security implementation varies significantly across CDN providers, reflecting different philosophies about defaults versus configurability. Some providers automatically inject security headers, while others require explicit configuration, balancing security with flexibility.

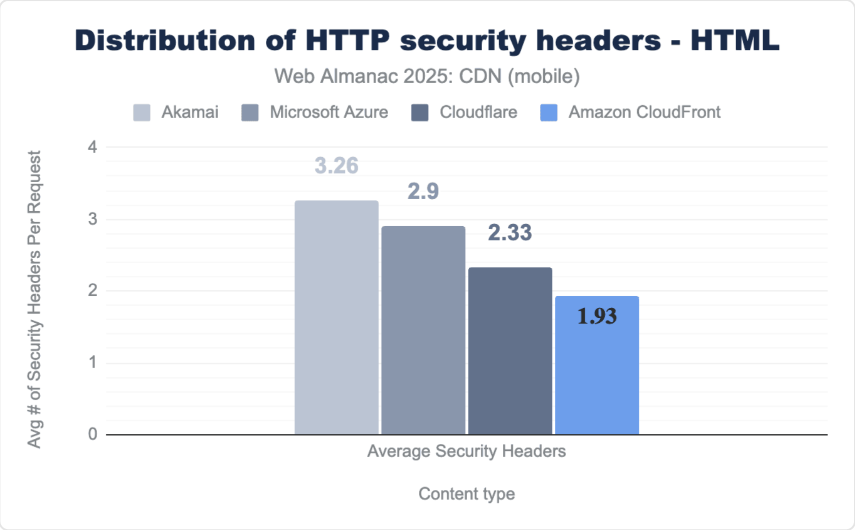


Figure 14.13. Distribution of HTTP security header count (mobile).

Both Cloudflare and Amazon CloudFront have a lower average number of security headers and this trend continues as we go more granular into specific headers as seen below. Header count is not a direct proxy for security posture; it often reflects how much a CDN auto-injects vs requires explicit configuration.

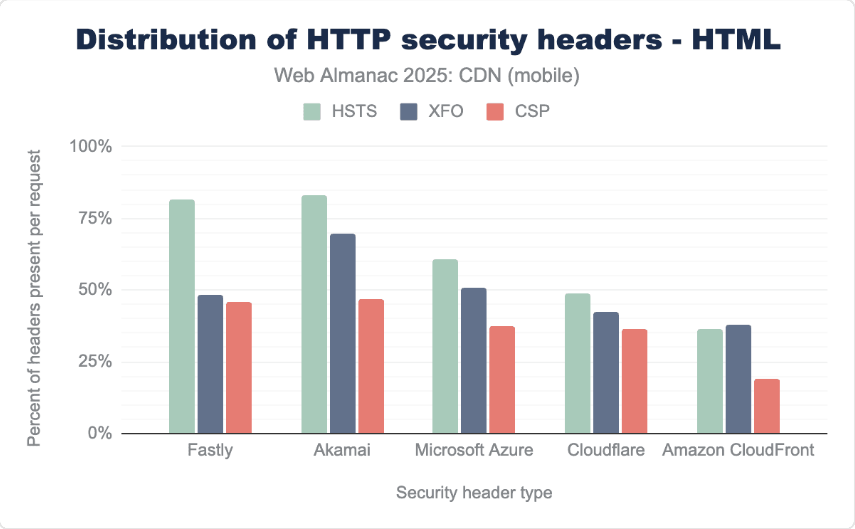


Figure 14.14. Distribution of HTTP security headers (mobile).

Fastly and Akamai have more secure defaults for security headers when basic features are enabled which drives higher rates of security headers. Amazon CloudFront and Cloudflare require more non-default configurations to inject and enforce security headers leading to a lower adoption.

Compression

Compression continues to be essential for web content delivery, representing one of the most accessible performance optimizations available for websites. Smaller file sizes mean faster page loads, lower bandwidth costs, and a better experience for users. Even as network speeds improve and connectivity options expand, compression remains important for optimizing performance across all types of internet connections.

Modern compression algorithms offer significant improvements over traditional methods, with CDNs leading the adoption of these advanced techniques. The choice of compression algorithm involves trade-offs between compression ratio, CPU usage, and compatibility requirements.

From the 2025 dataset we observed several commonly used compression algorithms:

- **Gzip**: The legacy standard, widely compatible but less efficient
- **Brotli**: Modern algorithm offering 20-26% better compression than Gzip
- **Zstandard (Zstd)**: Emerging standard with excellent compression ratios and speed

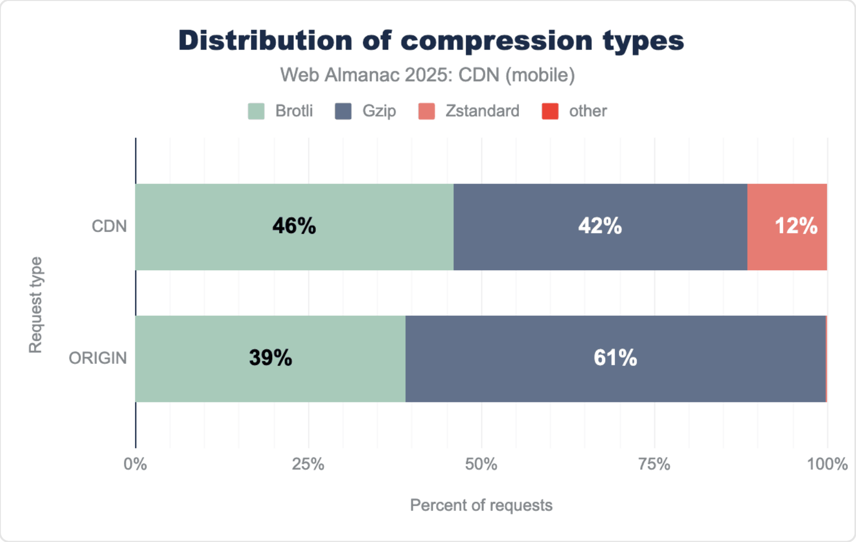


Figure 14.15. Distribution of compression types (mobile).

CDNs are leading the adoption of Brotli and Zstandard compression. Compared to 2024, Zstandard saw a significant increase from 3% to 12% in 2025. However, similar to 2024 Gzip remains the majority compression algorithm used by origin servers (61% Gzip vs 39% Brotli).

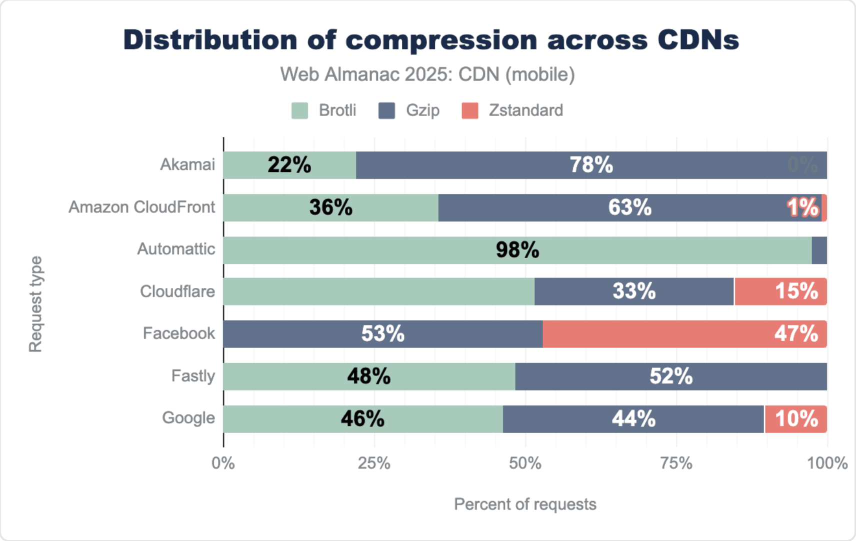


Figure 14.16. Distribution of compressions across CDNs (mobile).

Special credit to Automattic with 98% of their eligible resources being served using Brotli!

While still in 3rd place in terms of adoption, Zstandard has made gains in 2025 compared to 2024. In 2024, only Facebook's CDN had any statistically measurable usage of Zstandard, while in 2025 both Cloudflare (15%) and Google CDN (10%) showed measurable adoption. Cloudflare's newer compression defaults and configuration options may help explain the observed data. The outliers to the dataset were Amazon CloudFront and Automattic, which currently do not support Zstandard compression natively. However, the data observed showed CloudFront passing through already compressed data from origin using Zstandard. This demonstrates content owner's desire to use Zstandard even though not all major CDN providers support it.

Future compression trends

Looking ahead, the industry is exploring Dictionary Compression (Shared Brotli)[464] which promises:

- 20-40% better compression for repeated content patterns
- Shared dictionaries across related resources
- Chrome Origin Trial launched in 2024
- Expected CDN support rollout in 2025-2026

This represents the next frontier in compression optimization, building on Brotli's success and is one to watch in future editions of the Web Almanac.

TLS usage

Transport Layer Security (TLS) forms the foundation of secure web communications. CDNs have consistently led the adoption of newer TLS versions, providing enhanced security and performance benefits to websites without requiring infrastructure upgrades.

TLS 1.3 adoption

TLS 1.3 improves the overall security of web traffic compared to earlier versions that included weaker cryptographic algorithms that had known vulnerabilities. The protocol also reduces handshake latency through its optimized design.

[464] *https://developer.mozilla.org/docs/Web/HTTP/Guides/Compression_dictionary_transport*

Note on TLS 1.0 and 1.1 Absence: Our measurements show zero usage of TLS 1.0 and 1.1 across all requests. This is expected because HTTP Archive uses modern Chrome browsers for measurement, which completely removed support for these deprecated protocols in July 2020 (Chrome 84). Servers that only support TLS 1.0 or 1.1 would result in connection failures rather than successful requests, thus not appearing in our data. While these legacy protocols may still exist in some enterprise environments or IoT devices, they are no longer viable for public web services accessed by modern browsers.

Nearly all CDN traffic now uses TLS 1.3, with 99% of requests leveraging the latest protocol version. This benefits developers through faster connection establishment times, directly improving page load speeds. CDN providers continue to be early adopters of new web technologies, which means applications using CDNs get these performance and security enhancements with little to no additional effort.

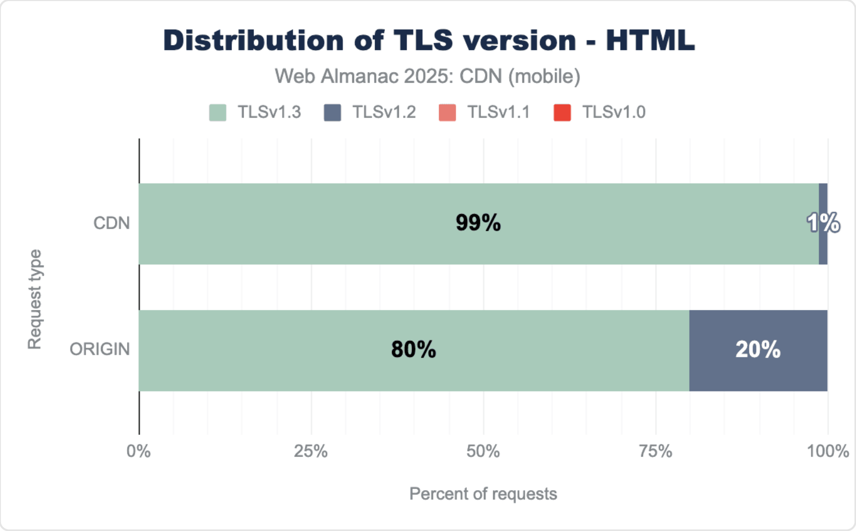


Figure 14.17. Distribution of TLS version for HTML (mobile).

Though origin server adoption of TLS 1.3 is improving, CDNs still demonstrate a clear advantage in rolling out new capabilities compared to organizations managing their own software and hardware upgrades.

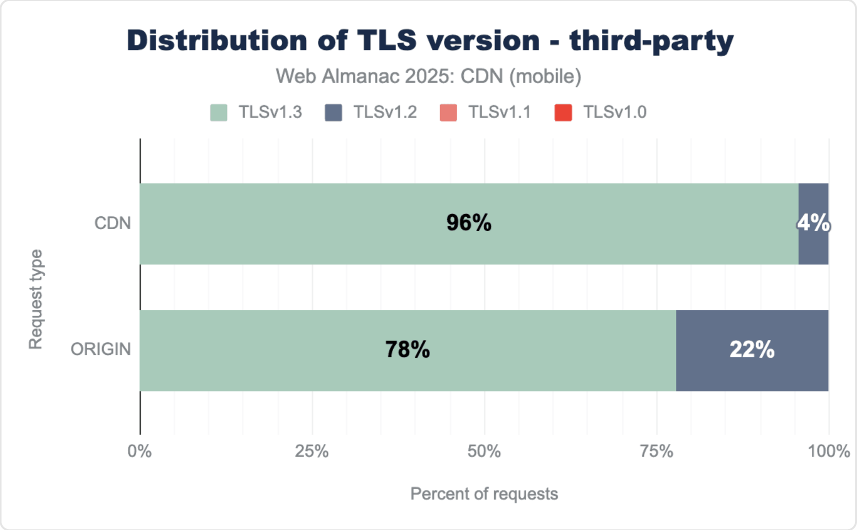


Figure 14.18. Distribution of TLS version for third-party requests (mobile).

Third-party content shows a similar pattern, with 96% of CDN requests using TLS 1.3 compared to 78% for origin servers. CDN adoption increased modestly from 93% in 2024 to 96% in 2025, while origin servers saw a larger jump from 68% to 78% over the same period.

TLS performance impact

Important Measurement Context: The following TLS negotiation times are measured from HTTP Archive's simulated browser connections using Chrome in controlled datacenter conditions. These measurements represent optimal network conditions and Chrome's TLS implementation, which may not reflect real-world variations in client capabilities, network quality, or geographic distribution. Additionally, since Chrome no longer supports TLS 1.0/1.1, these measurements only capture TLS 1.2 and 1.3 performance characteristics.

TLS negotiation times show clear performance differences between CDNs and origin servers, with additional variations between desktop and mobile connections.

Desktop users experience substantially faster TLS negotiations with CDNs compared to origin servers at every performance level. The median negotiation time for CDNs is 57 milliseconds versus 177 milliseconds for origins, over three times faster. This performance gap persists across the distribution. At the 90th percentile, CDNs complete negotiations in 89 milliseconds while origins require 277 milliseconds.

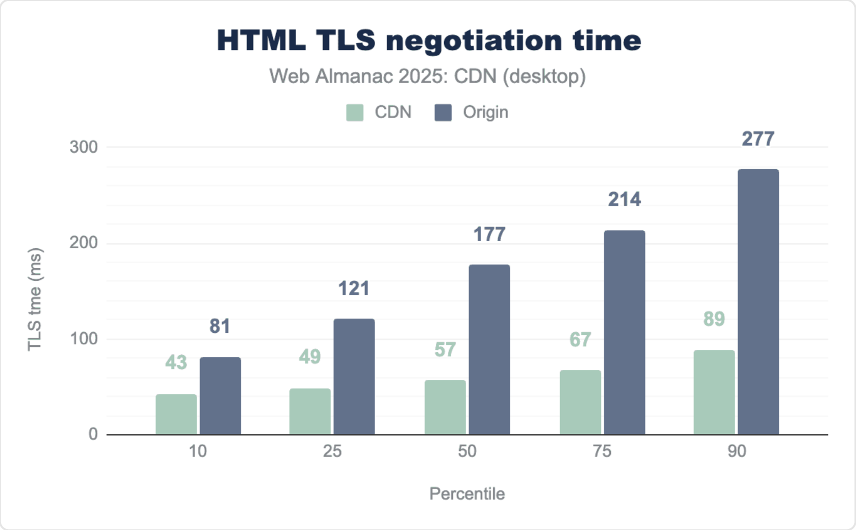


Figure 14.19. HTML TLS negotiation - CDN vs origin (desktop).

Mobile devices show a similar pattern, with CDN performing better than origin servers, but the overall negotiation times are higher compared to desktop. The median TLS negotiation time for mobile CDN is 183 milliseconds, while for origin servers it's 302 milliseconds. At the 90th percentile, mobile CDN takes 216 milliseconds, whereas origin servers require 416 milliseconds.

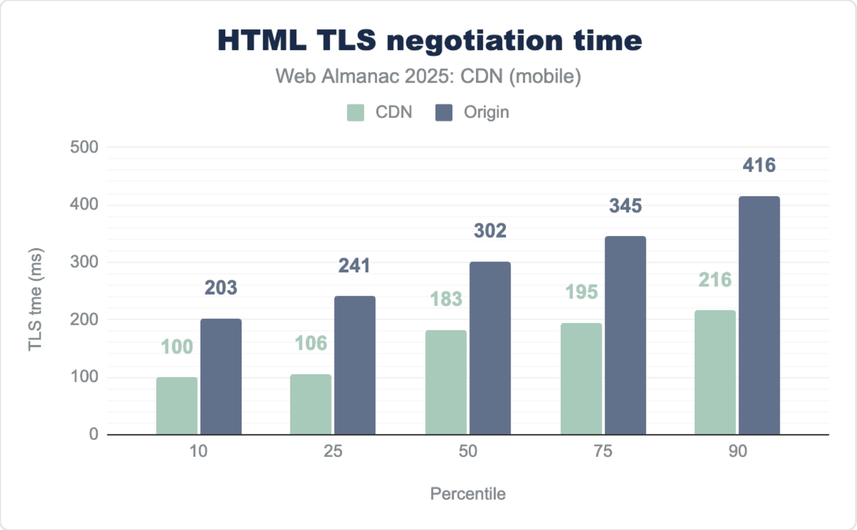


Figure 14.20. HTML TLS negotiation - CDN vs origin (mobile).

Mobile devices generally show slower performance than desktop due to their more limited processing power and often less reliable network conditions. CDNs outperform origin servers largely because of their distributed architecture, which positions content geographically closer to users and optimizes connection paths for reduced latency.

Image formats and optimization

Image formats continue to play a pivotal role in CDNs, directly influencing website performance, bandwidth costs, and overall user experience. Modern formats such as WebP, AVIF, and SVG remain the most efficient options, offering improved compression ratios and visual fidelity compared to legacy formats like JPEG and PNG. These efficiencies translate to faster page loads and lower data transfer, especially critical for mobile users and high-traffic sites.

Most CDNs now automatically detect browser capabilities and serve the most optimized format available—for example, AVIF for Chrome, WebP for Edge, and JPEG fallback for legacy browsers. Adaptive resizing, caching, and on-the-fly conversion have largely eliminated the need for maintaining static image variants across device types or resolutions.

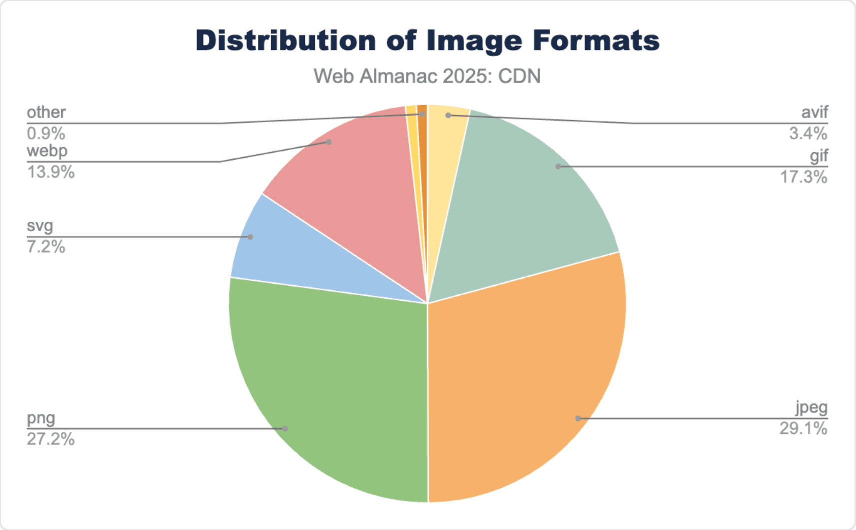


Figure 14.21. Distribution of Image Formats.

As of 2025, the data reflects a continued shift toward efficiency driven formats. While JPEG remains the most requested format, it declined by nearly 10% overall compared to 2022, with an even sharper 10.5% drop on mobile. Conversely, WebP (+5%), SVG (+2.5%), and AVIF (+3.1%) have all grown steadily since 2022. This may reflect shifts in site mix, image pipeline defaults, or fallback behavior rather than a reversal in industry support.

The GIF format showed a modest 2.3% increase, particularly on mobile (+0.5% higher than desktop), likely driven by short-loop animations and UI elements common in app-driven web traffic. Meanwhile, PNG usage saw a minor decline (-0.8%), suggesting designers and developers continue to prioritize lighter formats for both vector and raster assets.

Between 2024 and 2025, there was a slight regression in AVIF (-1.9%), SVG (-1.1%), and WebP (-1.8%), offset by small rebounds in GIF (+2.3%) and JPEG (+2.2%), a possible indicator of fallback scenarios or compatibility defaults on less optimized delivery stacks.

The 2025 data underscore a clear trajectory: legacy formats like JPEG still dominate total requests but are gradually ceding share to newer, more efficient formats. WebP, SVG, and AVIF are becoming the new baseline for high performance content delivery, especially in mobile-first ecosystems where latency and bandwidth efficiency are critical.

Client Hints

Client Hints were introduced as an alternative to the User-Agent string, letting web servers request specific information from browsers through HTTP headers. They fall into four main categories: device information, user agent preferences, user preference media features, and networking details. These hints are further split into high and low entropy types. High entropy hints can potentially be used for fingerprinting, so browsers usually require user permission or enforce other policies before sharing them. Low entropy hints are less useful for fingerprinting and may be sent by default based on browser or user settings.

In 2025, we continued measuring Client Hints using the same methodology established in 2024 - detecting the `Accept-CH` header in responses.

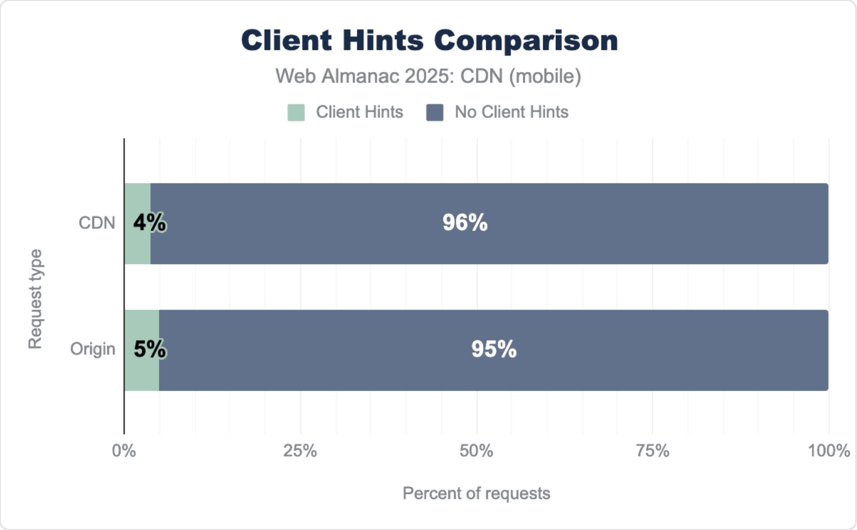


Figure 14.22. Client Hints Comparison (mobile).

Client Hints buck the usual pattern of CDNs leading adoption: CDN usage was flat year-over-year, while origin requests increased to ~5% in 2025. A likely contributor is cache-key explosion at the edge, which can reduce cache hit rates when Client Hints are incorporated without careful bucketing. This challenge may explain why adoption remains low despite the technology's potential benefits.

Early Hints

Early Hints uses the HTTP 103 status code[465] to let servers send initial headers to browsers while the main response is still being prepared. This is especially useful for preloading important resources like stylesheets, JavaScript files, and fonts before the full page is ready. Despite its potential for improving performance, adoption remains minimal.

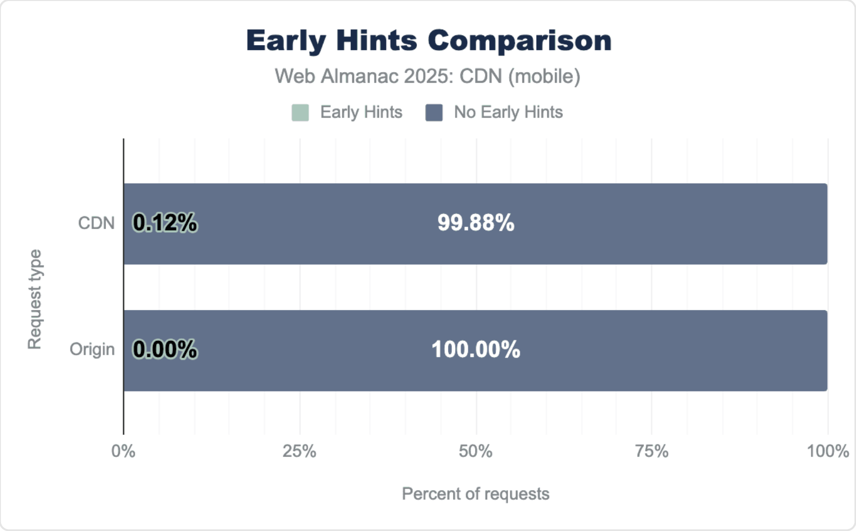


Figure 14.23. Early Hints Comparison (mobile).

Browser support for Early Hints is widespread, but we found minimal usage in the dataset with only 0.012% of CDN requests. This represents a marginal 0.002% increase from 2024 to 2025. Vercel was the only CDN to support over 1% adoption (2.84%) with Cloudflare and Fastly less than 1%.

We're interested to see how Early Hints affects performance as more sites start using it. Hopefully by next year's Web Almanac, we'll have more CDN providers implementing the feature and enough data to share detailed statistics on its impact.

Conclusion

In 2024, we saw CDNs leading the charge on adopting emerging technologies like HTTP/3, and that pattern has held steady into 2025. Looking at features like Brotli and Zstandard

[465] https://datatracker.ietf.org/doc/html/rfc8297#section-2

compression or TLS 1.3 encryption, CDNs make it easy for sites to implement these improvements through simple configuration changes instead of overhauling entire fleets of servers, load balancers, and networking equipment.

Our 2025 analysis reveals that CDNs have become the vanguard of web protocol evolution, with leading providers like Cloudflare achieving 69% HTTP/3 adoption compared to less than 5% for origin servers. This 10-15x difference demonstrates how edge infrastructure drives the adoption of next-generation web standards.

Key findings from our 2025 analysis include:

- **Protocol Leadership**: CDNs show 29-45% HTTP/3 adoption versus <7% for origins
- **Security Excellence**: 95.6% of CDN traffic uses TLS 1.3, compared to 77.7% for origin servers
- **Compression Innovation**: Leaders like Automattic achieve 97.5% Brotli adoption, with Zstandard growing from 3% to 12% year-over-year
- **Performance Advantages**: CDNs deliver DNS responses 19% faster on mobile and 60% faster on desktop

This year we took a deeper look at HTTP/3 and revisited Early Hints, which we first examined in 2024. For the first time, we broke out CDN performance and security and will dive deeper in 2026, specifically on trade-offs that exist between both topics. We initially planned to include IPv6 analysis, but the data wasn’t reliable enough to draw meaningful conclusions. We hope to address IPv6 adoption in the 2026 chapter once we have more robust measurements.

The CDN landscape in 2025 demonstrates that these platforms have evolved far beyond simple content delivery to become comprehensive optimization and security platforms that are essential infrastructure for the modern web.

We recommend readers visit the Security chapter of the 2025 Web Almanac where several topics in this chapter are expanded on and provide data through a different lens.

Join us again in 2026 as we collect and analyze more data to see what new insights we can share with our readers.

Authors

Joe Viggiano

joeviggiano

Joe Viggiano is a Principal Solutions Architect at Amazon Web Services helping Media & Entertainment customers deliver media content at scale.

Alex Moening

AlexMoening

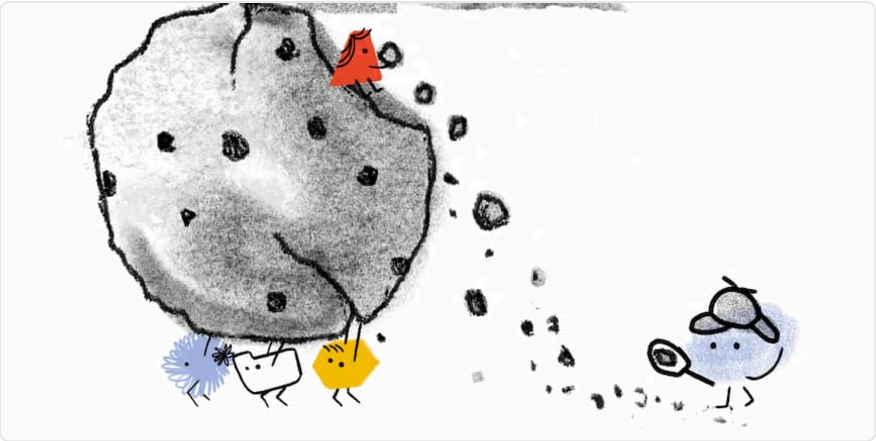
Alex Moening is a Senior Edge Solutions Architect at Amazon Web Services.

Nick McCord

nick-mccord nicholasmccord

Nick McCord is a Solutions Architect at Amazon Web Services focusing on startups and edge services.

Part IV Chapter 15
Cookies



Written by Yohan Beugin
Reviewed by Jannis Rautenstrauch and Martina Kraus
Analyzed by Chris Böttger
Edited by Brian Smith and Barry Pollard

Introduction

Cookies[466] allow websites to save data and maintain state information across HTTP requests, a stateless protocol. Web applications use cookies for several purposes, like authentication, fraud prevention and security, or remembering preferences and user choices. However, since their introduction in the mid-1990s, cookies have also played a dominant role in online tracking of web users.

Over the years, browser vendors such as Brave, Firefox, and Safari have imposed restrictions, partitioned, and removed third-party cookies[467]. While Chrome initially appeared to follow in these same steps by announcing plans to block all third-party cookies[468], several delays and postponements later, Google eventually decided to keep third-party cookies unrestricted and let users decide to disable them in Chrome[469].

[466] *https://developer.mozilla.org/docs/Web/HTTP/Cookies*
[467] *https://developer.mozilla.org/docs/Web/Privacy/Guides/Third-party_cookies#how_do_browsers_handle_third-party_cookies*
[468] *https://blog.chromium.org/2020/01/building-more-private-web-path-towards.html*
[469] *https://privacysandbox.com/news/update-on-plans-for-privacy-sandbox-technologies/*

In this chapter, we measure and report on the prevalence and structure of web cookies encountered on the web pages visited by the HTTP Archive crawl of July 2025. The majority of these results, except when mentioned otherwise, are for the top one million (top million) most popular websites according to their rank in the Chrome User Experience report (CrUX) rank recorded in the HTTP Archive dataset at the time of the crawl. Results are also shown for both desktop and mobile devices although, in practice for this chapter we rarely observe any significant difference between the two types of devices.

Background

First up let's get a common understanding of some of the terms used in this chapter.

HTTP cookie

When a user visits a website, they interact with a web server that can request the user's web browser to set and save an HTTP cookie[470]. This cookie corresponds to data saved in a text string on the user's device and is sent with subsequent HTTP requests to the web server. Cookies are used to persist stateful information about users across multiple HTTP requests—which can allow authentication, session management, and tracking. Cookies are also associated with privacy and security risks.

First and third-party cookies

Cookies are set by a web server and can be of two types: *first-party* and *third-party* cookies. First-party cookies are set by the same domain as the site the user is visiting, while third-party cookies are set from a different domain.

Third-party cookies may be from a third party, or from a different site or service belonging to the same "first party" as the top-level site. *Third-party cookies* are really *cross-site cookies*.

For example, imagine that the owner of the domain `example.com` also owns `example.net` and that the following cookies are set for a user visiting `https://www.example.com`:

[470] *https://developer.mozilla.org/docs/Web/HTTP/Cookies*

Cookie name	Set by	Type of cookie	Reason
`cookie_a`	`www.example.com`	*First-party*	*Same domain as visited website*
`cookie_b`	`cart.example.com`	*First-party*	*Same domain as visited website: subdomains do not matter*
`cookie_c`	`www.example.edu`	*Third-party*	*Different domain than visited website*
`cookie_d`	`tracking.example.org`	*Third-party*	*Different domain than visited website*
`cookie_e`	`login.example.net`	*Third-party*	*Different domain than visited website even if owned by the same owner in this example (cross-site cookie from the same "first party" at the top-level site)*

Figure 15.1. Cookie Context.

Privacy & security risks

Cookies, while fundamental to allow the web to work, do pose privacy & security risks:

- **Web tracking.** Cookies are used by third parties to track users across websites and record their browsing behavior and interests. In targeted advertising, this data is leveraged to show users advertisements aligned with their interest.

  This tracking usually takes place the following way: third-party code embedded on a site can set a cookie that identifies a user. Then, the same third-party can record user activity by obtaining that cookie back when the user visits other websites where it is embedded as well (see also the Privacy chapter).

  We note that first-party cookies can also be used for online tracking, methods such as cookie syncing allow bypassing the limitation of third-party cookies and track users across different websites[471].

[471] https://dl.acm.org/doi/abs/10.1145/3442381.3449837

- **Cookie theft and session hijacking.** Cookies are used to store session information such as credentials (for example, a session token) for authentication purposes across several HTTP requests. However, if these cookies were to be obtained by a malicious actor they could use them to authenticate to the corresponding web servers.

  If cookies are not properly set by web servers, they could be prone to cross-site vulnerabilities such as session hijacking[472], cross-site request forgery (CSRF)[473], cross-site script inclusion (XSS)[474], and others (see also the Security chapter).

First and third-party prevalence

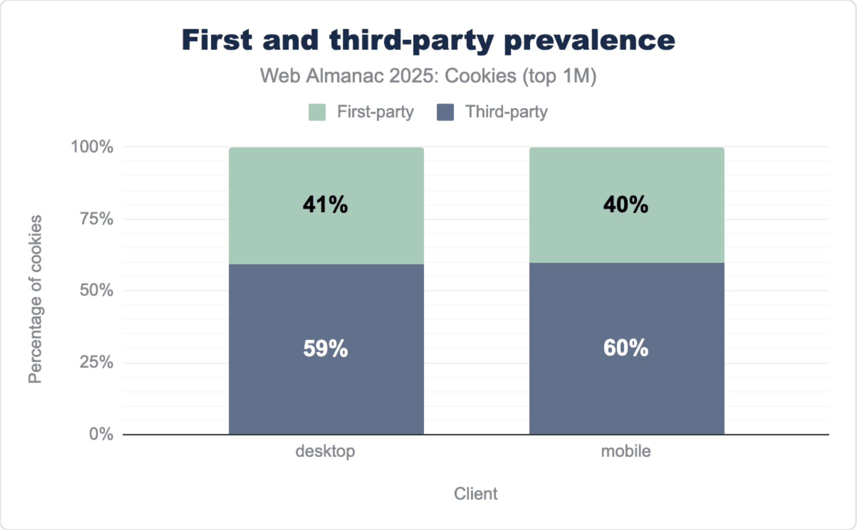


Figure 15.2. First- and third-party prevalence.

The overall prevalence of first- and third-party cookies on the top one million most popular websites from the HTTP Archive crawl of July 2025 is similar to last year's distribution[475].

On both desktop and mobile devices, roughly 40% of cookies are first-party and about 60% third-party.

[472] https://developer.mozilla.org/docs/Glossary/Session_Hijacking
[473] https://developer.mozilla.org/docs/Web/Security/Practical_implementation_guides/CSRF_prevention
[474] https://developer.mozilla.org/docs/Glossary/Cross-site_scripting
[475] https://almanac.httparchive.org/en/2024/cookies#first-and-third-party-prevalence

First and third-party prevalence by rank

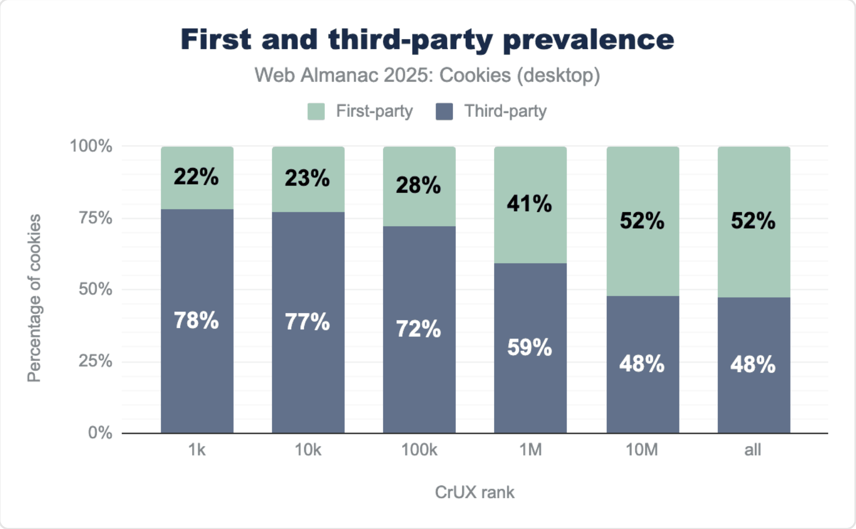


Figure 15.3. First- and third-party prevalence of cookies by rank on desktop clients.

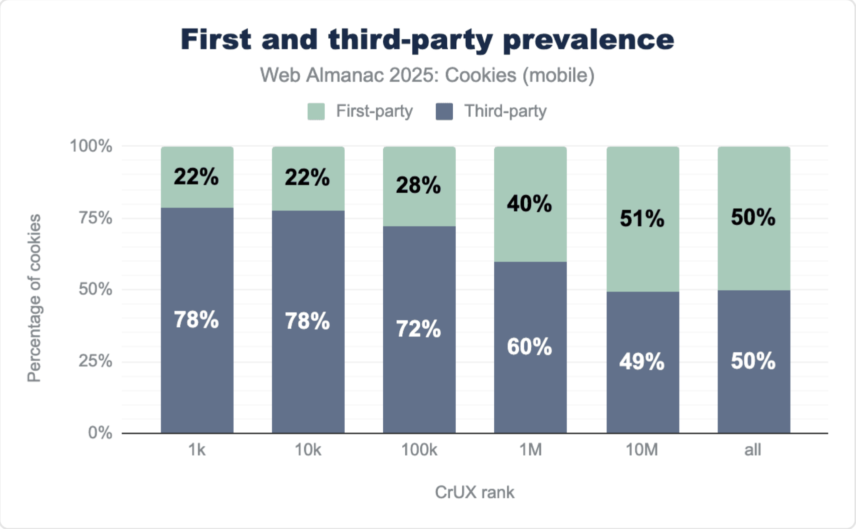


Figure 15.4. First- and third-party prevalence of cookies by rank on mobile clients.

We observe that the most popular websites set in proportion more third-party than first-party cookies: 78% of cookies are third-party on the top 1,000 most visited websites when it is just below 50% on the top 10 million. This may be explained by the fact that more popular websites also include more third-party content and scripts that in turn set third-party cookies to enable different functionalities.

Cookie attributes

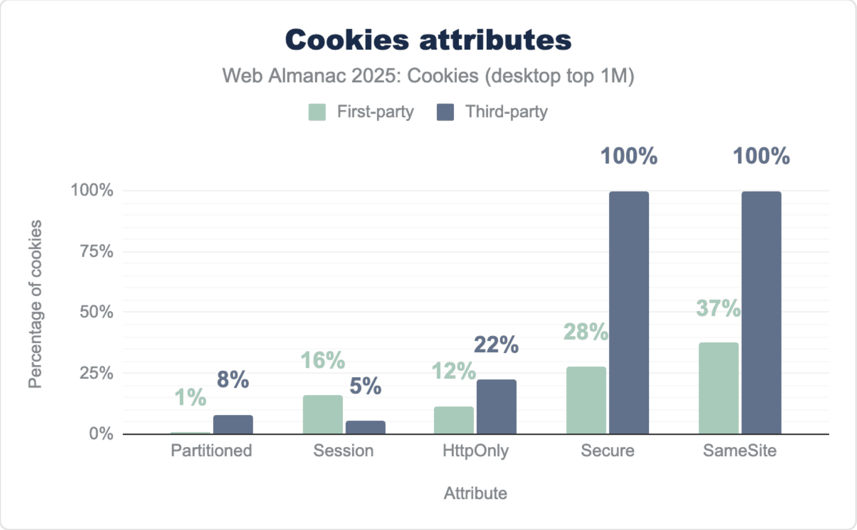


Figure 15.5. An overview of cookie attributes for desktop clients.

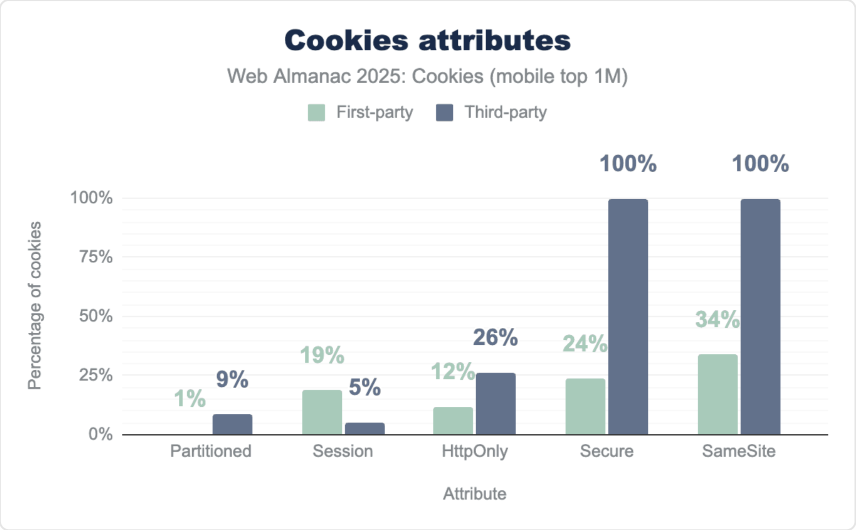


Figure 15.6. An overview of cookie attributes for mobile clients.

The data showcases the different cookie attributes[476] for each type of cookies observed. Let's delve into each of these.

`Partitioned` (CHIPS proposal)

On compatible browsers[477], partitioned cookies prevent third-party cookies to be used for cross-site tracking by placing them into a storage partitioned per top-level site.

8.6%

Figure 15.7. Percent of `Partition` cookies on mobile pages.

In July 2025, nearly 9% of third-party cookies on the top million are partitioned. We observe here a slight increase in adoption of this relatively new attribute in comparison to the 6% of last year's results[478].

[476] *https://developer.mozilla.org/docs/Web/HTTP/Headers/Set-Cookie*
[477] *https://developer.mozilla.org/docs/Web/Privacy/Privacy_sandbox/Partitioned_cookies#browser_compatibility*
[478] *https://almanac.httparchive.org/en/2024/cookies#partitioned*

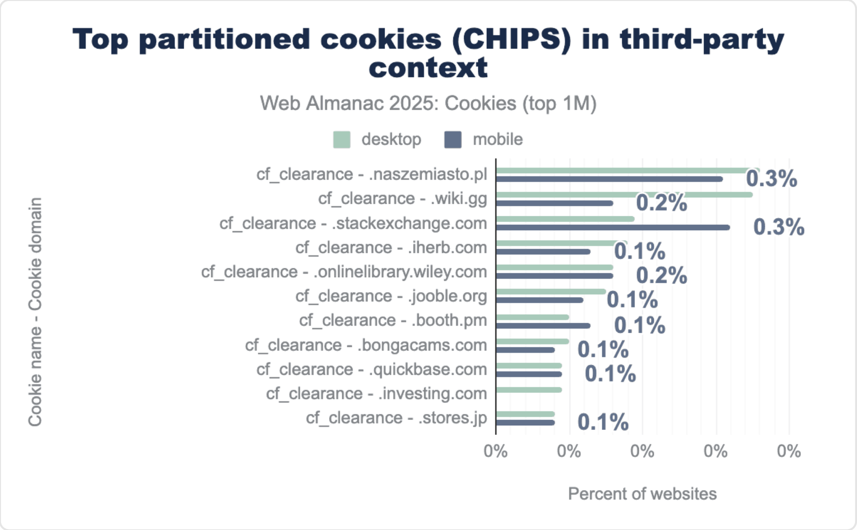


Figure 15.8. Top partitioned cookies (CHIPS) in third-party context.

The previous chart shows the 10 most common partitioned cookies (name and domain) found in third-party context on web pages in July 2025. Here, we observe a major change from last year's analysis, indeed the overall usage of third-party partitioned cookies in 2025 appears to have plummeted to very low levels.

Interestingly, partitioned cookies that were somewhat predominant in 2024 (on about 9% of websites with partitioned cookies) are not present anymore; two of these cookies were set by YouTube and another one was the `receive-cookie-deprecation` cookie set by domains that participated in the testing phase[479] of Chrome's Privacy Sandbox. Instead, Cloudflare's `cf_clearance` cookie accounts for the entirety of the top 10 most common partitioned third-party cookies in 2025.

So, in the past year YouTube appears to have altered how these cookies were set on `youtube.com` and on video iframes embedded on other websites. Potential reasons that could explain these changes include: incorrect setting, A/B testing, and more likely infrastructure or policy updates following Google's announcements on the pause and then deprecation of Privacy Sandbox APIs, despite support for partitioned cookies (CHIPS proposal) still being continued.

[479] *https://developers.google.com/privacy-sandbox/private-advertising/setup/web/chrome-facilitated-testing*

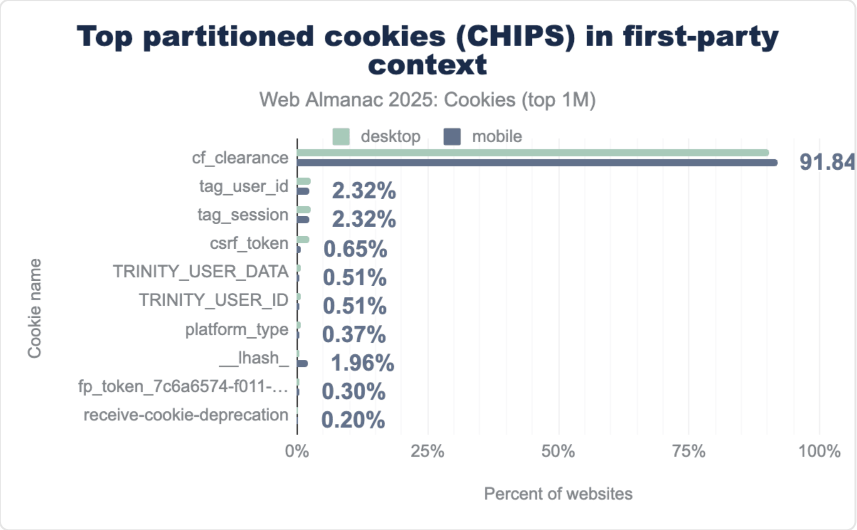


Figure 15.9. Top partitioned cookies (CHIPS) in first-party context.

In 2025, we continue to observe that 1% of first-party cookies are set as partitioned; this might be a bit surprising as the CHIPS proposal is mainly about partitioning third-party cookies, and even if it mentions a specific uncommon case[480] for partitioned first-party cookies, the behavior requirement[481] appears unclear in first-party context. One reason could be that some cookies are always set the same way—that is, the web server setting them is not distinguishing whether it is currently first-party or third-party.

In 2025, more than 90% of these first-party partitioned cookies are Cloudflare's `cf_clearance` cookie related to bot detection. Comparing to 2024's analysis, we remark that the first-party partitioned cookie `receive-cookie-deprecation`, set by domains participating in Privacy Sandbox APIs tests, is not as popular anymore. Perhaps, this observation can be explained by a pause or reduction of adoption of these APIs due to Google's corresponding announcements in this past year.

Session

19% of first-party and 7% of third-party cookies are session cookies; temporary cookies only valid for a single user session that expire once the user quits the corresponding website they were set on, or closes their web browser, whichever happens first.

[480] *https://github.com/privacycg/CHIPS?tab=readme-ov-file#first-party-chips*
[481] *https://github.com/privacycg/CHIPS/issues/51*

`HttpOnly`

`HttpOnly` cookies provide some mitigation against cross-site scripting (XSS)[482] as they can not be accessed by JavaScript code (but are still sent along `XMLHttpRequest` or `fetch` requests initiated from JavaScript).

12% and a little more than 26% of first- and third-party cookies have this attribute set, respectively.

`Secure`

`Secure` cookies are only sent to requests made through HTTPS, same trend as last year here[483]; while only 24% of first-party cookies set this attribute, all third-party cookies have to set it if they want to use `SameSite=None` (which they all do, see below).

`SameSite`

The `SameSite` cookie attribute allows sites to specify when cookies are included with cross-site requests:

- `SameSite=Strict`: a cookie is only sent in response to a request from the same site as the cookie's origin.
- `SameSite=Lax`: same as `SameSite=Strict` except that the browser also sends the cookie on navigation to the cookie's origin site. On Chrome, this is the default value of `SameSite` if no value is set.
- `SameSite=None`: cookies are sent on same-site or cross-site requests. This means that in order to make third-party tracking with cookies possible, the tracking cookies must have their `SameSite` attribute set to `None`.

To learn more about the `SameSite` attribute, see the following references:

- `SameSite` cookies explained
- "Same-site" and "same-origin"[484]
- What are the parts of a URL?[485]

[482] *https://developer.mozilla.org/docs/Glossary/Cross-site_scripting*
[483] *https://almanac.httparchive.org/en/2024/cookies#secure*
[484] *https://web.dev/articles/same-site-same-origin*
[485] *https://web.dev/articles/url-parts*

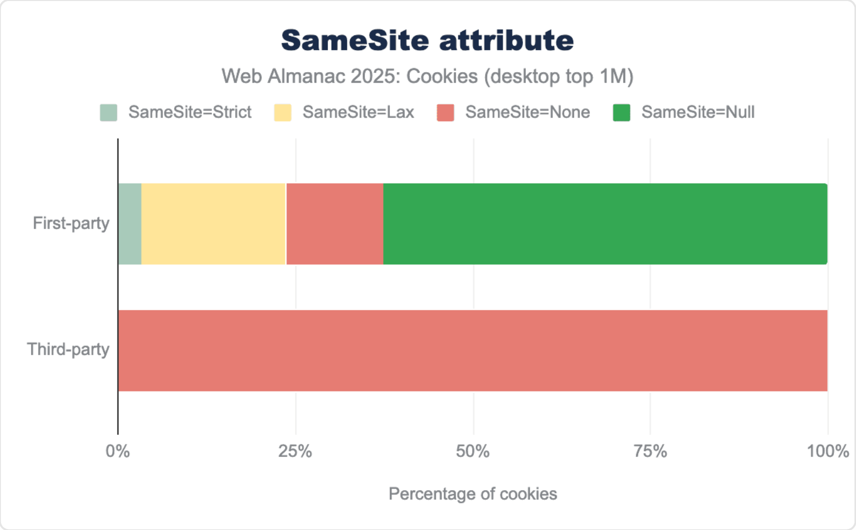


Figure 15.10. `SameSite` *attribute for cookies on desktop client.*

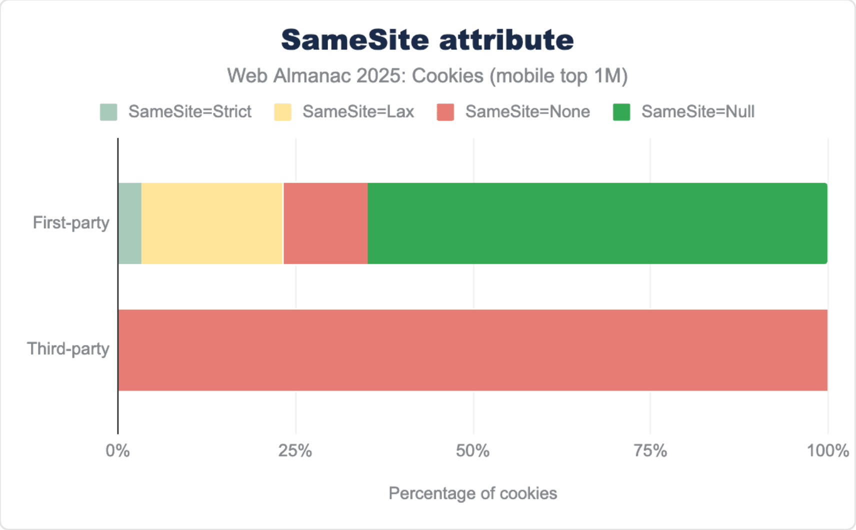


Figure 15.11. `SameSite` *attribute for cookies on mobile client.*

The overall distribution of this attribute for first- and third-party cookies across clients is similar to last year's[486]: nearly 100% of third-party cookies are sent on cross-site requests (`SameSite=None`) which can enable cross-site tracking.

[486] https://almanac.httparchive.org/en/2024/cookies#samesite

A majority of first-party cookies (66% on desktop, 62% on mobile) do not set this attribute and so are assigned by Chrome the default `Lax` behavior that 19% other first-party cookies explicitly pick, leaving only 3% setting it to the `Strict` setting, and the remaining 11% being sent on both same-site and cross-site requests (`SameSite=None`).

Cookie prefixes

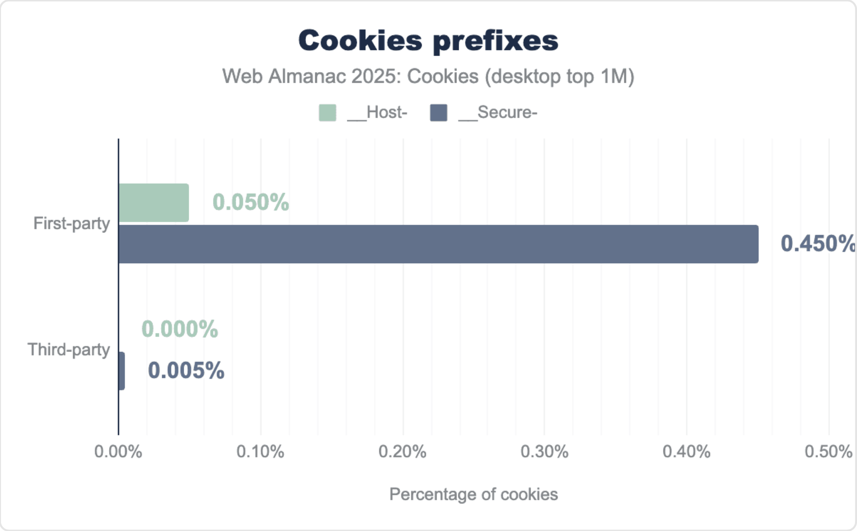


Figure 15.12. Cookie prefixes observed on desktop pages.

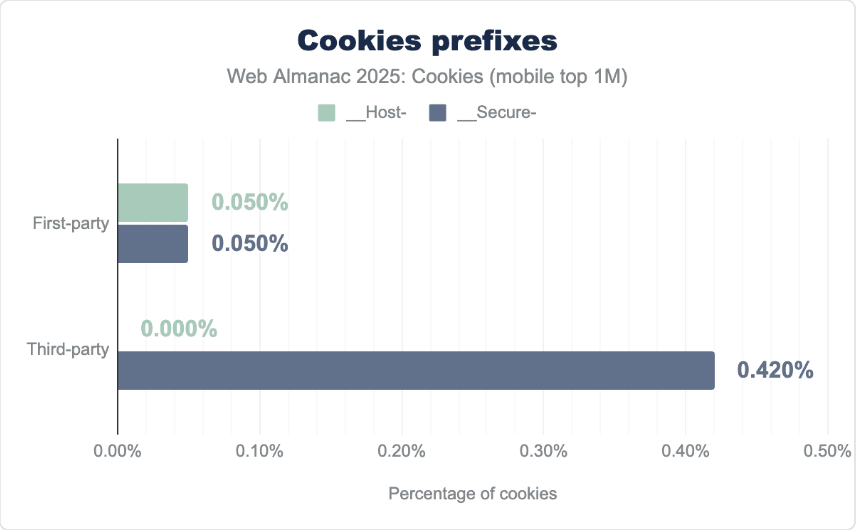


Figure 15.13. Cookie prefixes observed on mobile pages.

Cookie prefixes[487] `__Host-` and `__Secure-` can be used in the cookie name to indicate that they can only be set or modified by a secure HTTPS origin. This is to defend against session fixation[488] attacks.

Cookies with both prefixes must be set by a secure HTTPS origin and have the `Secure` attribute set. Additionally, `__Host-` cookies must not contain a `Domain` attribute and have their `Path` set to `/`, thus `__Host-` cookies are only sent back to the exact host they were set on, and so not to any parent domain.

Here, we draw the same conclusion as last year[489]: these prefixes have seen very low adoption on the web since their introduction 10 years ago, and so, in practice the defense-in-depth measure that they provide remains unused.

[487] *https://developer.mozilla.org/docs/Web/HTTP/Cookies#cookie_prefixes*
[488] *https://developer.mozilla.org/docs/Web/Security/Types_of_attacks#session_fixation*
[489] *https://almanac.httparchive.org/en/2024/cookies#cookie-prefixes*

Top cookies and domains setting them

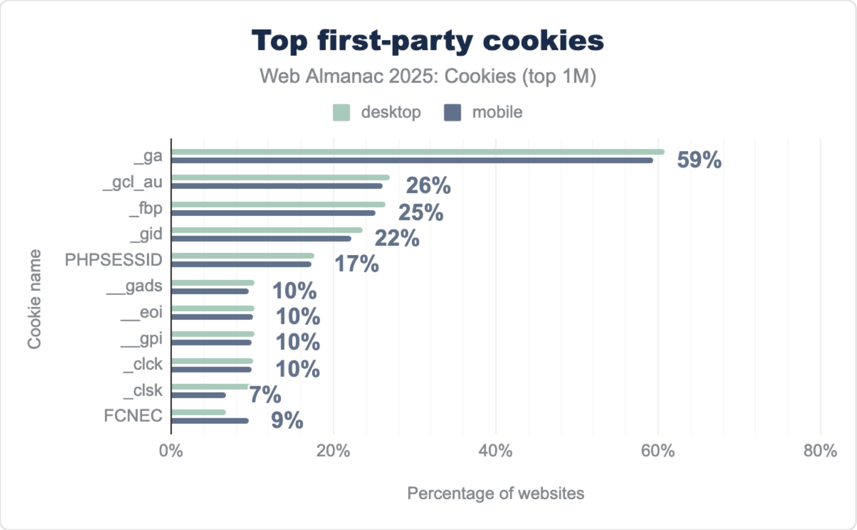


Figure 15.14. Top first-party cookies set.

The previous chart reports the top 10 most common first-party cookies names being set. Google Analytics sets the `_ga` and `_gcl_au` cookies, which are used for website statistics, analytics reports, and targeted advertising, on around 60% and 25% of websites. Other cookies present in this top 10 are related to online tracking, session cookies used to identify user's sessions, or performance.

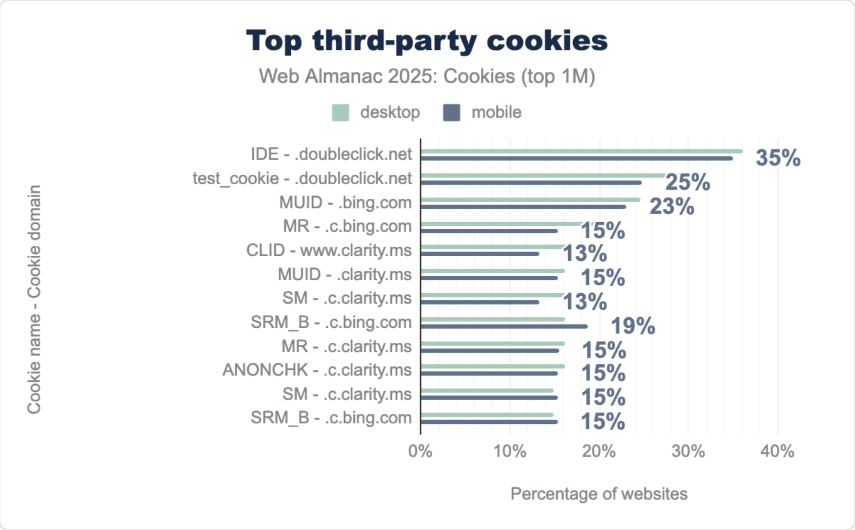


Figure 15.15. Top third-party cookies and domains that set them.

Similarly, this figure shows the top 10 most common third-party cookies being created on the top million websites.

The `IDE` and `test_cookie` cookies are set by `doubleclick.net` (owned by Google) and are present on more than 35% and 25% of websites respectively. DoubleClick checks if a user's web browser supports third-party cookies by trying to set `test_cookie`.

`MUID` from Microsoft comes next, present on more than 23% of websites, and is also used for targeted advertising and cross-site tracking.

As already pointed out in the `Partitioned` cookies section, this year we do not observe the `YSC` and `VISITOR_INFO1_LIVE` from YouTube among top third-party cookies anymore. This is likely due to changes from YouTube (perhaps linked to Google's announcements such as this one[490] on the Privacy Sandbox proposals), since the 2024 analysis. It appears that these cookies are not set anymore when the embedding page is just loaded and the video has not been played. Additionally, Google's Privacy & Terms[491] also document that `VISITOR_INFO1_LIVE` is being replaced by a `__Secure-YNID` cookie.

[490] *https://privacysandbox.google.com/blog/privacy-sandbox-next-steps*
[491] *https://policies.google.com/technologies/cookies?hl=en-US*

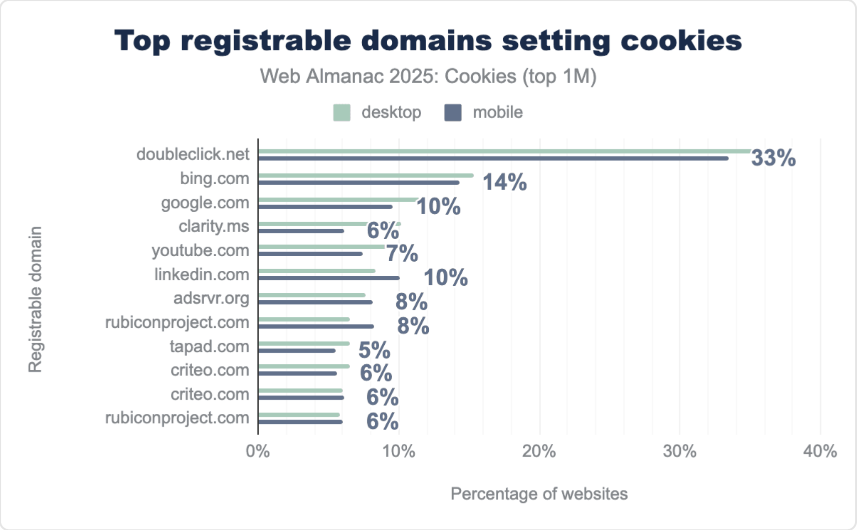


Figure 15.16. Top registrable domains setting cookies.

Perhaps, unsurprisingly from prior results, the 10 most common domains that set cookies on the web are all involved with search, targeting, and advertising services.

Google's coverage (`doubleclick.net` , `google.com` , and `youtube.com`) is reaching at least 33% of the websites, and Microsoft's (`bing.com` , `clarity.ms` , `linkedin.com`) at least 14%.

Number of cookies set by websites

Percentile	First-party	Third-party	All
min	1	1	1
p25	3	2	4
median	7	7	9
p75	13	16	23
p90	22	40	44
p99	45	399	395
max	178	885	915

Figure 15.17. Statistics for number of cookies set on the top one million desktop pages.

Percentile	First-party	Third-party	All
min	1	1	1
p25	3	2	4
median	6	4	9
p75	12	15	22
p90	21	39	43
p99	45	400	396
max	178	801	831

Figure 15.18. Statistics for number of cookies set on the top one million mobile pages.

Websites set a median of 9 cookies overall, with 7 first-party and 7 third-party cookies on desktop, and 6 first-party and 4 third-party cookies on mobile.

The tables report several other statistics about the number of cookies observed per website and the figures below display their cumulative distribution functions (cdf). For example: on desktop a maximum of 178 first-party and 885 third-party cookies are set per website:

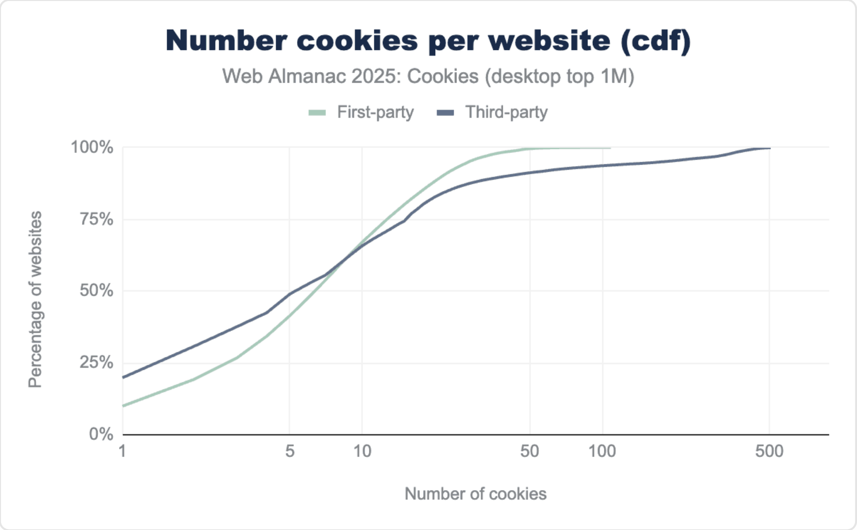


Figure 15.19. Number of cookies per website (cdf) for desktop pages.

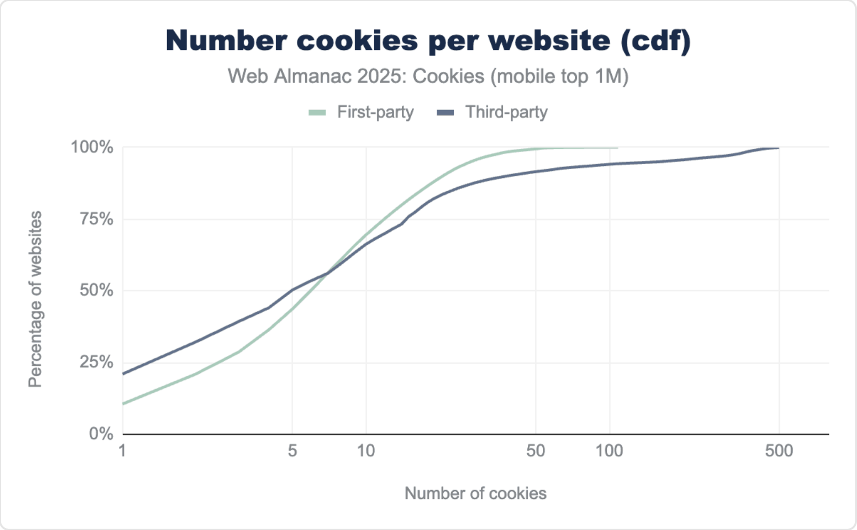


Figure 15.20. Number of cookies per website (cdf) for mobile pages.

Size of cookies

Percentile	*First-party*	*Third-party*	*All*
min	*1*	*1*	*1*
p25	*29*	*22*	*24*
median	*41*	*39*	*40*
p75	*67*	*59*	*64*
p90	*157*	*145*	*149*
p99	*414*	*321*	*338*
max	*4090*	*4096*	*4096*

Figure 15.21. Statistics for size of cookies set on the top one million desktop pages.

Percentile	*First-party*	*Third-party*	*All*
min	*1*	*1*	*1*
p25	*22*	*29*	*24*
median	*39*	*41*	*40*
p75	*62*	*67*	*65*
p90	*145*	*162*	*150*
p99	*326*	*414*	*388*
max	*4096*	*4081*	*4096*

Figure 15.22. Statistics for size of cookies set on the top one million mobile pages.

We find that the median size of cookies across all observed cookies is 40 bytes and with a maximum of 4 KB which is consistent with the limits defined in RFC 6265[492].

Similar to last year[493], we observe some cookies that are of a single byte in size, these are likely set by error by empty `Set-Cookie` headers.

We can chart the cumulative distribution function (cdf) of the size of all the cookies seen on the

[492] *https://datatracker.ietf.org/doc/html/rfc6265#section-6.1*
[493] *https://almanac.httparchive.org/en/2024/cookies#size-of-cookies*

top 1M websites for each client:

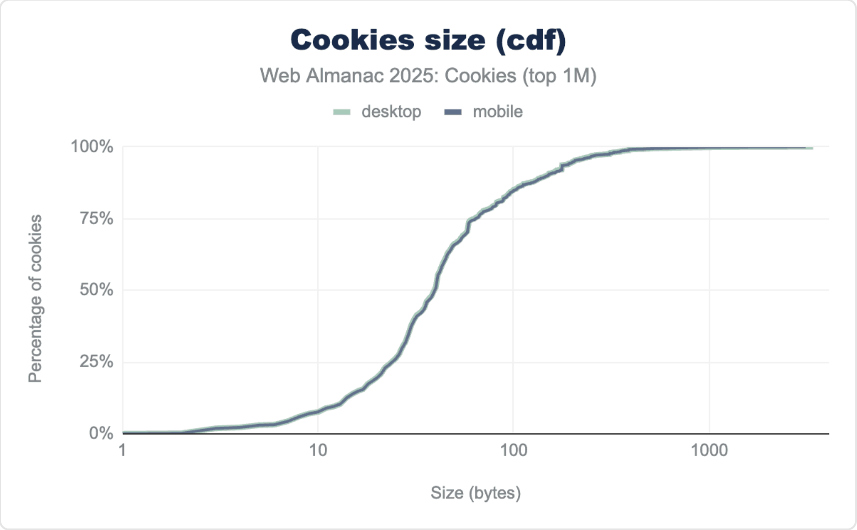


Figure 15.23. Size of cookies per website (cdf) for desktop and mobile pages.

Persistence (expiration)

Percentile	*First-party*	*Third-party*	*All*
min	*0*	*0*	*0*
p25	*1*	*30*	*21*
median	*365*	*360*	*364*
p75	*395*	*365*	*390*
p90	*400*	*400*	*400*
p99	*400*	*400*	*400*
max	*400*	*400*	*400*

Figure 15.24. Statistics for age of cookies set on the top one million desktop pages.

Percentile	First-party	Third-party	All
min	0	0	0
p25	1	30	30
median	365	270	360
p75	395	365	390
p90	400	400	400
p99	400	400	400
max	400	400	400

Figure 15.25. Statistics for age of cookies set on the top one million mobile pages.

Cookies are set to an expiration date when they are created. If session cookies expire immediately after the session is over (see previous section), most first- and third-party cookies do not and have a median age of a full year.

The longer cookies live, the longer they can be used for re-identification or cross-site tracking which is why most tracking cookies are typically set to be stored in the browser for a longer time.

The maximum age among the cookies that we can observe with the instrumentation and collection of the HTTP Archive Tools for this chapter is of 400 days, due to the hard limits[494] that Chrome imposes on cookie `Expires` and `Max-Age` attribute.

Conclusion

The observations from this chapter confirm the conclusions from last year's analysis[495]:

- A majority (60%) of cookies encountered on the web are third-party cookies and popular websites have significantly more third-party cookies than less popular sites.
- Most popular cookies can be linked to advertising, tracking, and analytics use cases.
- Cookies tend to be long-lived with a median average lifetime of 12 months. Ephemeral session cookies only represent 19% of first- and 7% of third-party

[494] https://developer.chrome.com/blog/cookie-max-age-expires

[495] https://almanac.httparchive.org/en/2024/cookies#conclusion

cookies.

- Other restrictions on cookies capabilities are used very little to not at all: if 10% of third-party cookies are partitioned (which represents a slight uptake from last year's 6%), 100% of third-party cookies have `SameSite=None` allowing them to be sent in cross-site requests. Additionally, cookies prefixes adoption is almost non-existent.

Finally, while several web browsers have deprecated or limited third-party cookies[496] due to privacy concerns, Google has decided to still support them in Chrome[497]. Google is also phasing out most technologies from its Privacy Sandbox initiative, initially designed to *"create a thriving web ecosystem that is respectful of users and private by default"*. As a result, whether trackers use third-party cookies or develop other techniques (first-party syncing, fingerprinting, etc.) to track users online, cookies remain a fundamental component of the web that continue to pose privacy and security risks for users.

Author

Yohan Beugin

yohhaan https://yohan.beugin.org

Yohan Beugin is a Ph.D. student in the Department of Computer Sciences at the University of Wisconsin–Madison where he is a member of the Security and Privacy Research Group and advised by Prof. Patrick McDaniel. He is interested in building more secure, privacy-preserving, and trustworthy systems. His current research so far has focused on tracking and privacy in online advertising as well as security of open-source software.

[496] https://developer.mozilla.org/docs/Web/Privacy/Guides/Third-party_cookies#how_do_browsers_handle_third-party_cookies
[497] https://privacysandbox.com/news/update-on-plans-for-privacy-sandbox-technologies/

Appendix A
Methodology

Overview

The Web Almanac is a project organized by HTTP Archive[498]. HTTP Archive was started in 2010 by Steve Souders with the mission to track how the web is built. It evaluates the composition of millions of web pages on a monthly basis and makes its terabytes of metadata available for analysis on BigQuery[499].

The Web Almanac's mission is to become an annual repository of public knowledge about the state of the web. Our goal is to make the data warehouse of HTTP Archive even more

[498] https://httparchive.org
[499] https://httparchive.org/faq#how-do-i-use-bigquery-to-write-custom-queries-over-the-data

accessible to the web community by having subject matter experts provide contextualized insights.

The 2025 edition of the Web Almanac is broken into four parts: content, experience, publishing, and distribution. Within each part, several chapters explore their overarching theme from different angles. For example, Part II explores different angles of the user experience in the Performance, Security, and Accessibility chapters, among others.

About the dataset

The HTTP Archive dataset is continuously updating with new data monthly. For the 2025 edition of the Web Almanac, unless otherwise noted in the chapter, all metrics were sourced from the July 2025 crawl. These results are publicly queryable[500] on BigQuery in tables in the `` `httparchive.crawl.*` `` tables for `date = '2025-07-01'`.

All of the metrics presented in the Web Almanac are publicly reproducible using the dataset on BigQuery. You can browse the queries used by all chapters in our GitHub repository[501].

Please note that some of these queries are quite large and can be expensive[502] to run yourself. For help controlling your spending, refer to our guide on Minimizing query costs[503].

For example, to understand the number of pages with accessible color contrast, see color_contrast.sql[504]:

```
SELECT
  client,
  is_root_page,
  COUNTIF(color_contrast_score IS NOT NULL) AS
total_applicable,
  COUNTIF(CAST(color_contrast_score AS NUMERIC) = 1) AS
total_good_contrast,
  COUNTIF(CAST(color_contrast_score AS NUMERIC) = 1) /
COUNTIF(color_contrast_score IS NOT NULL) AS
```

[500] https://har.fyi/guides/getting-started/
[501] https://github.com/HTTPArchive/almanac.httparchive.org/tree/main/sql/2025
[502] **https://cloud.google.com/bigquery/pricing**
[503] **https://har.fyi/guides/minimizing-costs/**
[504] https://github.com/HTTPArchive/almanac.httparchive.org/blob/main/sql/2025/accessibility/color_contrast.sql

```
    perc_good_contrast
    FROM (
      SELECT
        client,
        is_root_page,
        date,
        JSON_VALUE(lighthouse.audits.`color-contrast`.score) AS
    color_contrast_score
      FROM
        `httparchive.crawl.pages`
      WHERE
        date = '2025-07-01'
    )
    GROUP BY
      client,
      is_root_page,
      date
    ORDER BY
      client,
      is_root_page;
```

Results for each metric are publicly viewable in chapter-specific spreadsheets, for example Accessibility results[505]. Links to the raw results and queries are available at the bottom of each chapter. Metric-specific results and queries are also linked directly from each figure.

Websites

There are 16,213,084 websites in the dataset. Among those, 15,426,398 are mobile websites and 12,155,374 are desktop websites. Most websites are included in both the mobile and desktop subsets.

HTTP Archive sources the URLs for its websites from the Chrome UX Report. The Chrome UX Report is a public dataset from Google that aggregates user experiences across millions of websites actively visited by Chrome users. This gives us a list of websites that are up-to-date and a reflection of real-world web usage. The Chrome UX Report dataset includes a form factor

[505] *https://docs.google.com/spreadsheets/d/13sY_wmYODArxo-hH5cSuRAbvtLdGI3x5Xc9qUyqP8as*

dimension, which we use to get all of the websites accessed by desktop or mobile users.

The July 2025 HTTP Archive crawl used by the Web Almanac used the most recently available Chrome UX Report release for its list of websites. The 202507 dataset was released on July 11, 2025 and captures websites visited by Chrome users during the month of June.

Due to resource limitations, the HTTP Archive previously could only test two pages from each website in the Chrome UX report. Be aware that this will introduce some bias into the results because a home page is not necessarily representative of the entire website. In recent years, we included secondary pages[506], and many chapters use this new data. Some chapters, however, used just the home pages.

HTTP Archive is also considered a lab testing tool, meaning it tests websites from a datacenter and does not collect data from real-world user experiences. All pages are tested with an empty cache in a logged out state, which may not reflect how real users would access them.

Metrics

HTTP Archive collects thousands of metrics about how the web is built. It includes basic metrics like the number of bytes per page, whether the page was loaded over HTTPS, and individual request and response headers. The majority of these metrics are provided by WebPageTest, which acts as the test runner for each website.

Other testing tools are used to provide more advanced metrics about the page. For example, Lighthouse is used to run audits against the page to analyze its quality in areas like accessibility and SEO. The Tools section below goes into each of these tools in more detail.

To work around some of the inherent limitations of a lab dataset, the Web Almanac also makes use of the Chrome UX Report for metrics on user experiences, especially in the area of web performance.

Some metrics are completely out of reach. For example, we don't necessarily have the ability to detect the tools used to build a website. If a website is built using create-react-app, we could tell that it uses the React framework, but not necessarily that a particular build tool is used. Unless these tools leave detectible fingerprints in the website's code, we're unable to measure their usage.

Other metrics may not necessarily be impossible to measure but are challenging or unreliable. For example, aspects of web design are inherently visual and may be difficult to quantify, like whether a page has an intrusive modal dialog.

[506] *https://discuss.httparchive.org/t/improving-the-http-archive-pipeline-and-dataset-by-10x/2372*

Tools

The Web Almanac is made possible with the help of the following open source tools.

WebPageTest

WebPageTest[507] is a prominent web performance testing tool and the backbone of HTTP Archive. We use a private instance[508] of WebPageTest with private test agents, which are the actual browsers that test each web page. Desktop and mobile websites are tested under different configurations:

Config	Desktop	Mobile
Device	Linux VM	Emulated Moto G4
User Agent	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36 PTST/250604.160032	Mozilla/5.0 (Linux; Android 8.1.0; Moto G (4)) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Mobile Safari/537.36 PTST/250604.160032
Location	Google Cloud Locations, USA	Google Cloud Locations, USA
Connection	Cable (5/1 Mbps 25ms RTT)	4G (9 Mbps 170ms RTT)
CPU slowdown	*N/A*	8x
Viewport	1920 x 1080px	512 x 360px
DPR	1x	3x

Desktop websites are run from within a desktop Chrome environment on a Linux VM. The network speed is equivalent to a cable connection.

Mobile websites are run from within a mobile Chrome environment on an emulated Moto G4 device with a network speed equivalent to a 4G connection.

Test agents run from various Google Cloud Platform locations[509] based in the USA.

[507] *https://www.webpagetest.org/*
[508] *https://docs.webpagetest.org/private-instances/*
[509] *https://cloud.google.com/compute/docs/regions-zones/#locations*

HTTP Archive's private instance of WebPageTest is kept in sync with the latest public version and augmented with custom metrics[510], which are snippets of JavaScript that are evaluated on each website at the end of the test.

The results of each test are made available as a HAR file[511], a JSON-formatted archive file containing metadata about the web page, which we then store in BigQuery.

Lighthouse

Lighthouse[512] is an automated website quality assurance tool built by Google. It audits web pages to make sure they don't include user experience antipatterns like unoptimized images and inaccessible content.

HTTP Archive runs the latest version of Lighthouse for all pages. As of the July 2025 crawl, HTTP Archive used the 12.8.0[513] version of Lighthouse.

Lighthouse is run as its own distinct test from within WebPageTest, but it has its own configuration profile:

Config	Desktop	Mobile
CPU slowdown	*N/A*	1x/4x
Download throughput	10.0 Mbps	1.6 Mbps
Upload throughput	10.0 Mbps	0.75 Mbps
RTT	40 ms	150 ms

For more information about Lighthouse and the audits available in HTTP Archive, refer to the Lighthouse developer documentation[514].

Wappalyzer

Wappalyzer[515] is a tool for detecting technologies used by web pages. There are 108 categories[516] of technologies tested, ranging from JavaScript frameworks, to CMS platforms, and even cryptocurrency miners. There are over 3,984 supported technologies.

[510] *https://github.com/HTTPArchive/custom-metrics*
[511] *https://en.wikipedia.org/wiki/HAR_(file_format)*
[512] *https://developer.chrome.com/docs/lighthouse/overview*
[513] *https://github.com/GoogleChrome/lighthouse/releases/tag/12.8.0*
[514] *https://developer.chrome.com/docs/lighthouse/overview*
[515] *https://github.com/HTTPArchive/wappalyzer*
[516] *https://github.com/HTTPArchive/wappalyzer/blob/main/src/categories.json*

HTTP Archive runs it's fork of the last open source version of Wappalyzer (v6.10.65), with some extra detections added since.

Wappalyzer powers many chapters that analyze the popularity of developer tools like WordPress, Bootstrap, and jQuery. For example, the CMS chapter relies heavily on the respective CMS[517] category of technologies detected by Wappalyzer.

All detection tools, including Wappalyzer, have their limitations. The validity of their results will always depend on how accurate their detection mechanisms are. The Web Almanac will add a note in every chapter where Wappalyzer is used but its analysis may not be accurate due to a specific reason.

Chrome UX Report

The Chrome UX Report[518] is a public dataset of real-world Chrome user experiences. Experiences are grouped by websites' origin, for example `https://www.example.com`. The dataset includes distributions of UX metrics like paint, load, interaction, and layout stability. In addition to grouping by month, experiences may also be sliced by dimensions like country-level geography, form factor (desktop, phone, tablet), and effective connection type (4G, 3G, etc.).

The Chrome UX Report dataset includes relative website ranking data[519]. These are referred to as *rank magnitudes* because, as opposed to fine-grained ranks like the #1 or #116 most popular websites, websites are grouped into rank buckets from the top 1k, top 10k, up to the top 10M. Each website is ranked according to the number of eligible[520] page views on all of its pages combined. This year's Web Almanac makes extensive use of this new data as a way to explore variations in the way the web is built by site popularity.

For Web Almanac metrics that reference real-world user experience data from the Chrome UX Report, the July 2025 dataset (202507) is used.

You can learn more about the dataset in the Using the Chrome UX Report on BigQuery[521] guide on developer.chrome.com[522].

Blink Features

Blink Features[523] are indicators flagged by Chrome whenever a particular web platform feature is detected to be used.

[517] *https://www.wappalyzer.com/categories/cms*
[518] *https://developer.chrome.com/docs/crux*
[519] *https://developer.chrome.com/blog/crux-rank-magnitude*
[520] *https://developer.chrome.com/docs/crux/methodology#eligibility*
[521] *https://developer.chrome.com/docs/crux/guides/bigquery*
[522] *https://developer.chrome.com*
[523] *https://chromium.googlesource.com/chromium/src/+/HEAD/docs/use_counter_wiki.md*

We use Blink Features to get a different perspective on feature adoption. This data is especially useful to distinguish between features that are implemented on a page and features that are actually used.

Blink Features are reported by WebPageTest as part of our regular testing.

Third Party Web

Third Party Web[524] is a research project by Patrick Hulce, author of the 2019 Third Parties chapter, that uses HTTP Archive and Lighthouse data to identify and analyze the impact of third party resources on the web.

Domains are considered to be a third party provider if they appear on at least 50 unique pages. The project also groups providers by their respective services in categories like ads, analytics, and social.

Several chapters in the Web Almanac use the domains and categories from this dataset to understand the impact of third parties.

Rework CSS

Rework CSS[525] is a JavaScript-based CSS parser. It takes entire stylesheets and produces a JSON-encoded object distinguishing each individual style rule, selector, directive, and value. See this thread[526] for more information about how it was integrated with the HTTP Archive dataset on BigQuery.

Parsel

Parsel[527] is a CSS selector parser and specificity calculator, originally written by 2020 CSS chapter lead Lea Verou and open sourced as a separate library. It is used extensively in all CSS metrics that relate to selectors and specificity.

Analytical process

The Web Almanac took about a year to plan and execute with the coordination of more than a hundred contributors from the web community. This section describes why we chose the

[524] *https://www.thirdpartyweb.today/*
[525] *https://github.com/reworkcss/css*
[526] *https://discuss.httparchive.org/t/analyzing-stylesheets-with-a-js-based-parser/1683*
[527] *https://projects.verou.me/parsel/*

chapters you see in the Web Almanac, how their metrics were queried, and how they were interpreted.

Planning

The 2025 Web Almanac kicked off in April 2025 with a call for contributors[528]. We initialized the project with the same chapters from previous years and the community suggested additional topics, one of which become a new chapter this year: Generative AI.

As we stated in the inaugural year's Methodology:

> *One explicit goal for future editions of the Web Almanac is to encourage even more inclusion of underrepresented and heterogeneous voices as authors and peer reviewers.*

To that end, this year we've continued our author selection process with the following in mind[529]:

- New authors were specifically encouraged to make room for different perspectives.
- The project leads reviewed all of the author nominations and made an effort to select authors who will bring new perspectives and amplify the voices of underrepresented groups in the community.

We hope to iterate on this process in the future to ensure that the Web Almanac is a more diverse and inclusive project with contributors from all backgrounds.

Analysis

In May and June 2025, data analysts worked with authors and peer reviewers to come up with a list of metrics that would need to be queried for each chapter. In some cases, custom metrics[530] were created to fill gaps in our analytic capabilities.

Throughout July 2025, the HTTP Archive data pipeline crawled several million websites, gathering the metadata to be used in the Web Almanac. These results were post-processed and saved to BigQuery[531].

Being our sixth year, we were able to update and reuse the queries written by previous analysts, though they required rewriting to a new, more efficient dataset[532] that the HTTP

[528] https://github.com/HTTPArchive/almanac.httparchive.org/discussions/4062
[529] https://github.com/HTTPArchive/almanac.httparchive.org/discussions/2165
[530] https://github.com/HTTPArchive/custom-metrics
[531] https://console.cloud.google.com/bigquery?p=httparchive&d=almanac&page=dataset
[532] https://har.fyi/guides/migrating-to-crawl-dataset/

Archive moved to this year. Additionally, there were many new metrics that needed to be written from scratch. You can browse all of the queries by year and chapter in our open source query repository[533] on GitHub.

Interpretation

Authors worked with analysts to correctly interpret the results and draw appropriate conclusions. As authors wrote their respective chapters, they drew from these statistics to support their framing of the state of the web. Peer reviewers worked with authors to ensure the technical correctness of their analysis.

To make the results more easily understandable to readers, web developers and analysts created data visualizations to embed in the chapter. Some visualizations are simplified to make the points more clearly. For example, rather than showing a full distribution, only a handful of percentiles are shown. Unless otherwise noted, all distributions are summarized using percentiles, especially medians (the 50th percentile), and not averages.

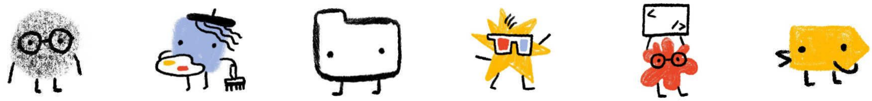
Finally, editors revised the chapters to fix simple grammatical errors and ensure consistency across the reading experience.

Looking ahead

The 2025 edition of the Web Almanac is the sixth in what is mostly an annual tradition (we took a break in 2023) in the web community of introspection and a commitment to positive change. Getting to this point has been a monumental effort thanks to many dedicated contributors and we hope to leverage as much of this work as possible to make future editions even more streamlined.

[533] *https://github.com/HTTPArchive/almanac.httparchive.org/tree/main/sql/2025*

Appendix B
Contributors



The Web Almanac has been made possible by the hard work of the web community. 70 people have volunteered countless hours in the planning, research, writing and production phases of the 2025 Web Almanac.

Aaron Gustafson
@AaronGustafson
aarongustafson
https://www.aaron-gustafson.com
Reviewer

Aaron T. Grogg
aarontgrogg.com
aarontgrogg
aarontgrogg
https://aarontgrogg.com/
Author

Aidan Tierney
aidantierney.bsky.social
AidanA11y
aidantierney
Reviewer

Alex Moening
AlexMoening
Analyst and Author

Alon Kochba
@alonkochba
alonkochba
alonkochba
Analyst

Amaka Chulwuma
Amaka-maxi
amaka-chukwuma-jubilee
Author

Amandeep Singh
@amanvirdi
amandeepsinghvirdi
amandeepsinghvirdi
https://byaman.com/
Analyst and Author

Anirudh Duggal
anirudhduggal
anirudh-duggal
Reviewer

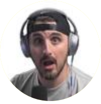
Augustin Delport
Analyst

Barry Pollard
@tunetheweb
https://webperf.social/@tunetheweb
tunetheweb.com
tunetheweb
tunetheweb
https://www.tunetheweb.com
Analyst, Organizing Committee, Developer, Editor, and Reviewer

Bogdan Lazar
bogdanlazar.com
tricinel
tricinel
https://bogdanlazar.com
Author and Editor

Bram Stein

https://typo.social/@bram
bramstein
bramstein
http://www.bramstein.com/
Reviewer



Brian Clark

@_clarkio
clarkio
https://www.clarkio.com
Reviewer

Brian Smith

https://mastodon.social/@bsmth
bsmth.de
bsmth
https://bsmth.de
Editor

Burak Güneli

@burakGuneli
burakguneli
Organizing Committee

Caroline Scholles

carolinescholles
Editor and Reviewer

Charles Berret

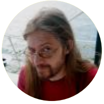
charlesberret
https://charlesberret.net/
Analyst and Author

Chris Böttger

ChrisBeeti
Analyst and Organizing Committee

Chris Green

@chrisgreenseo
chris-green.net
chr156r33n
chrisgreenseo
https://www.torquepartnership.com/
Analyst and Author

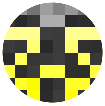


Chris Lilley

@svgeesus
svgeesus
https://svgees.us
Editor

Christian Liebel

@christianliebel
https://mastodon.cloud/@christianliebel
christianliebel
christianliebel
https://christianliebel.com
Analyst and Author



Daan V.

vsdaan
https://vsdaan.be
Analyst

Dan Knauss

dknauss
https://newlocalmedia.com
Reviewer

Dave Smart

tamethebots.com
dwsmart
davewsmart
https://tamethebots.com
Analyst and Reviewer

Diego Gonzalez

@diekus
diekus
https://diek.us
Author

Gertjan Franken

@GJFR_
GJFR
gertjan-franken
https://gjfr.dev
Author and Reviewer

Hidde de Vries

https://front-end.social/@hdv
hidde
https://hidde.blog/
Reviewer

Himanshu Jariyal

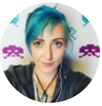
himanshujariyal
himanshujariyal
https://medium.com/@him_jar
Author

Humaira

hfhashmi
Author

Ivan Ukhov

IvanUkhov
Analyst



Jamie Indigo

not-a-robot.com
fellowhuman1101
jamie-indigo
https://not-a-robot.com
Author and Reviewer

Jannis Rautenstrauch

@jannis_r
JannisBush
jannis-rautenstrauch
https://jannisbush.github.io/
Reviewer

Joe Viggiano

joeviggiano
Analyst and Author

Jonathan Pagel

jcmpagel
jonathan-pagel
https://jonathanpagel.com
Analyst and Author

José Solé

@jmsoleb
jmsole
https://www.jmsole.cl/
Reviewer

Kai Hollberg

https://mastodon.social/@Schweinepriester
Schweinepriester
Reviewer

Martina Kraus

@MartinaKraus11
martinakraus
martina-kraus-398493108
Reviewer

Max Ostapenko

max-ostapenko
max-ostapenko
https://maxostapenko.com
Analyst and Editor

Maxim Salnikov

@webmaxru
webmaxru
https://medium.com/@webmaxru
Reviewer

Michael Lewittes

@MichaelLewittes
https://seocommunity.social/@MichaelLewittes
michaellewittes.bsky.social
MichaelLewittes
michael-lewittes-a22b831
https://www.ranktify.com/team
Editor

Michael Solati

@MichaelSolati
MichaelSolati
https://michaelsolati.com
Author

Mikael Araújo

@miknaraujo
mikaelaraujo.bsky.social
mikaelaraujo
mikael-araujo
https://www.mikaelaraujo.com
Organizing Committee

Mike Gifford

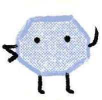
https://mastodon.social/@mgifford
mgifford.bsky.social
mgifford
mgifford
https://accessibility.civicactions.com/
Analyst and Author

Montserrat Cano

montsecano.bsky.social
montsec
montsecano-senior-digital-marketer
https://montserrat-cano.com/
Editor

Muhammad Abu Bakar Aziz

abubakaraziz
aziz313f
Author



Muhammad Jazlan

jazlan01
Analyst and Author

Nick McCord

nick-mccord
nicholasmccord
Analyst and Author

Nimesh Vadgama - VN

nimeshgit
ops-ml-architect
https://ops-ml-architect.blogspot.com/
Analyst and Author

Nurullah Demir

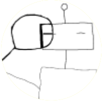
nrllh
nurullahdemir
https://ndemir.com
Author, Organizing Committee, Project Lead, and Reviewer

Onur Güler

onurglr
Analyst

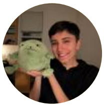
Prathamesh Rasam

25prathamesh
prathamesh-rasam
Author



Richard Barret

barkseo.bsky.social
rickb110
richardbarrettseo
Author



Rumaisa Habib

RumaisaHabib
rumaisahabib
https://rumaisahabib.com/
Author

Samuel Fatola

samuelfatola
Editor

Saptak Sengupta

@Saptak013
SaptakS
https://saptaks.website
Organizing Committee and Developer

Scott Davis

@scottdavis99
scottdavis99
http://thirstyhead.com
Reviewer

Selman Koral

krlslman
Organizing Committee, Developer, and Translator

Sharon McClintic

hello-sharon
sharonmcclintic
https://www.lumar.io/
Editor

Sophie Brannon

@SophieBrannon
SophieBrannon
Author

Sreemoyee Bhattacharya

sreemoyee-bhattacharya-17a4a61a6
Editor

Stoyan Stefanov

@stoyanstefanov
https://indieweb.social/@stoyan
stoyan.me
stoyan
stoyanstefanov
https://www.phpied.com
Reviewer

Tanner Hodges

tannerhodges.bsky.social
tannerhodges
https://tannerhodges.com/
Analyst and Reviewer

Thomas Steiner

@tomayac
tomayac
https://blog.tomayac.com/
Analyst and Reviewer

Tobias Urban

turban1988
https://internet-sicherheit.de/ueber-uns/team/alle-mitarbeiter/urban-tobias-2/
Organizing Committee

Umar Iqbal

@umaarr6
UmarIqbal
https://www.umariqbal.com
Reviewer

Vahe Arabian

VaheSODP
ahearabian
https://www.stateofdigitalpublishing.com/
Author

Vik Vanderlinden

@vikvanderlinden
vikvanderlinden
vikvanderlinden
https://vikvanderlinden.be/
Author

Vinod Tiwari

@@securient
securient
https://blog.securient.io
Author and Reviewer

Yash Vekaria

@vekariayash
Yash-Vekaria
https://www.yashvekaria.com/
Author and Organizing Committee

Yohan Beugin

yohhaan
https://yohan.beugin.org
Author