**Minimize main thread work**

May 2, 2019 • Updated Oct 4, 2019

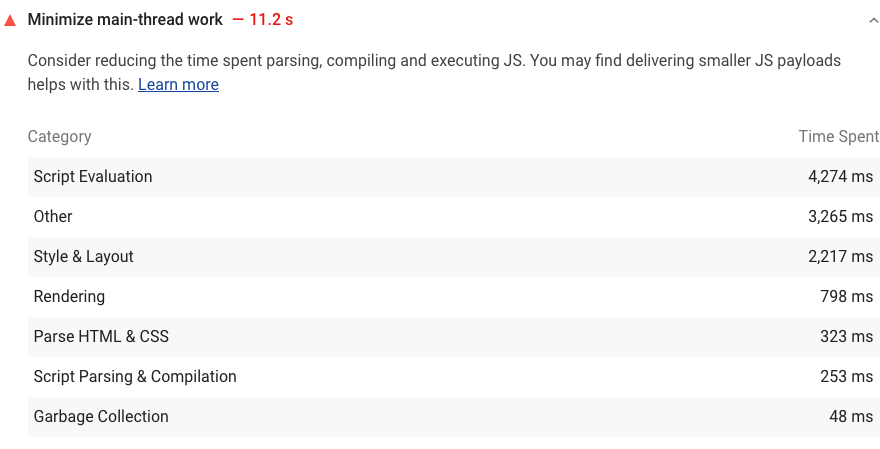
Appears in: [Performance audits](https://web.dev/lighthouse-performance)

The browser's [renderer process](https://developers.google.com/web/updates/2018/09/inside-browser-part3) is what turns your code into a web page that your users can interact with. By default, the [main thread](https://developer.mozilla.org/docs/Glossary/Main_thread) of the renderer process typically handles most code: it parses the HTML and builds the DOM, parses the CSS and applies the specified styles, and parses, evaluates, and executes the JavaScript.

The main thread also processes user events. So, any time the main thread is busy doing something else, your web page may not respond to user interactions, leading to a bad experience.

**How the Lighthouse main thread work audit fails** [**#**](https://web.dev/mainthread-work-breakdown/?utm_source=lighthouse&utm_medium=devtools#how-the-lighthouse-main-thread-work-audit-fails)

[Lighthouse](https://developers.google.com/web/tools/lighthouse/) flags pages that keep the main thread busy for longer than 4 seconds during load:



To help you identify the sources of main thread load, Lighthouse shows a breakdown of where CPU time was spent while the browser loaded your page.

See the [Lighthouse performance scoring](https://web.dev/performance-scoring) post to learn how your page's overall performance score is calculated.

**How to minimize main thread work** [**#**](https://web.dev/mainthread-work-breakdown/?utm_source=lighthouse&utm_medium=devtools#how-to-minimize-main-thread-work)

The sections below are organized based on the categories that Lighthouse reports. See [The anatomy of a frame](https://aerotwist.com/blog/the-anatomy-of-a-frame/) for an overview of how Chromium renders web pages.

See [Do less main thread work](https://developers.google.com/web/tools/chrome-devtools/speed/get-started#main) to learn how to use Chrome DevTools to investigate exactly what your main thread is doing as the page loads.

**Script evaluation** [**#**](https://web.dev/mainthread-work-breakdown/?utm_source=lighthouse&utm_medium=devtools#script-evaluation)

* [Optimize third-party JavaScript](https://web.dev/fast/#optimize-your-third-party-resources)
* [Debounce your input handlers](https://developers.google.com/web/fundamentals/performance/rendering/debounce-your-input-handlers)
* [Use web workers](https://web.dev/off-main-thread/)

**Style and layout** [**#**](https://web.dev/mainthread-work-breakdown/?utm_source=lighthouse&utm_medium=devtools#style-and-layout)

* [Reduce the scope and complexity of style calculations](https://developers.google.com/web/fundamentals/performance/rendering/reduce-the-scope-and-complexity-of-style-calculations)
* [Avoid large, complex layouts and layout thrashing](https://developers.google.com/web/fundamentals/performance/rendering/avoid-large-complex-layouts-and-layout-thrashing)

**Rendering** [**#**](https://web.dev/mainthread-work-breakdown/?utm_source=lighthouse&utm_medium=devtools#rendering)

* [Stick to compositor only properties and manage layer count](https://developers.google.com/web/fundamentals/performance/rendering/stick-to-compositor-only-properties-and-manage-layer-count)
* [Simplify paint complexity and reduce paint areas](https://developers.google.com/web/fundamentals/performance/rendering/simplify-paint-complexity-and-reduce-paint-areas)

**Parsing HTML and CSS** [**#**](https://web.dev/mainthread-work-breakdown/?utm_source=lighthouse&utm_medium=devtools#parsing-html-and-css)

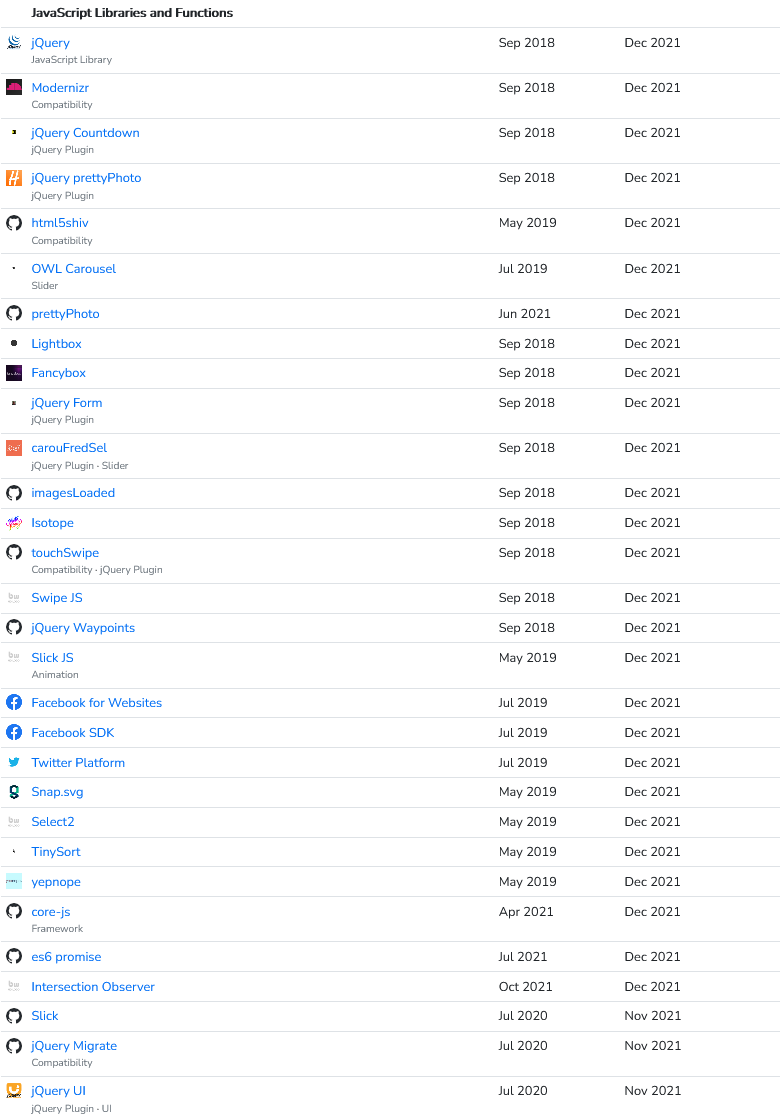
* [Extract critical CSS](https://web.dev/extract-critical-css/)
* [Minify CSS](https://web.dev/minify-css/)
* [Defer non-critical CSS](https://web.dev/defer-non-critical-css/)

**Script parsing and compilation** [**#**](https://web.dev/mainthread-work-breakdown/?utm_source=lighthouse&utm_medium=devtools#script-parsing-and-compilation)

* [Reduce JavaScript payloads with code splitting](https://web.dev/reduce-javascript-payloads-with-code-splitting/)
* [Remove unused code](https://web.dev/remove-unused-code/)

**Garbage collection** [**#**](https://web.dev/mainthread-work-breakdown/?utm_source=lighthouse&utm_medium=devtools#garbage-collection)

* [Monitor your web page's total memory usage with measureMemory()](https://web.dev/monitor-total-page-memory-usage/)



**Third-party JavaScript performance**

Find out how third-party JavaScript can affect performance and what you can do to keep it from slowing down your sites.

Aug 13, 2019

Appears in: [Fast load times](https://web.dev/fast)

[](https://web.dev/authors/mihajlija/)

[*Milica Mihajlija*](https://web.dev/authors/mihajlija/)

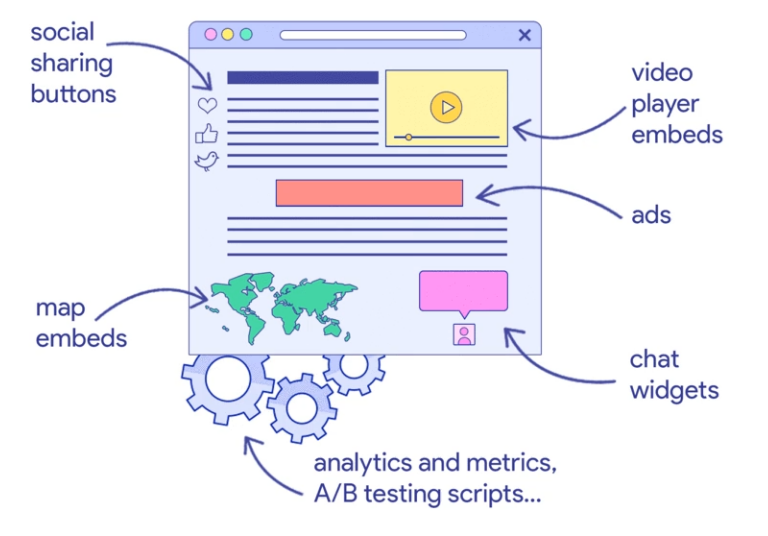
* [Twitter](https://twitter.com/bibydigital)
* [GitHub](https://github.com/mihajlija)
* [Blog](https://mihajlija.github.io/)

Third-party JavaScript generally refers to scripts embedded in your website that are:

* Not authored by you
* Served from third-party servers

Sites use these scripts for various purposes, including:

* Social sharing buttons
* Video player embeds
* Chat services
* Advertising iframes
* Analytics and metrics scripts
* A/B testing scripts for experiments
* Helper libraries (like date formatting, animation, and functional libraries)



Third-party scripts can provide powerful functionality, but that's not the whole story. They also affect privacy, security, and page behavior⁠—and they can be particularly problematic for performance.

**Performance** [**#**](https://web.dev/third-party-javascript/#performance)

Any significant amount of [JavaScript can slow down performance](https://web.dev/bootup-time). But because third-party JavaScript is usually outside your control, it can bring additional issues.

**Network** [**#**](https://web.dev/third-party-javascript/#network)

Setting up connections takes time, and sending too many requests to multiple servers causes slowdowns. That time is even longer for secure connections, which may involve DNS lookups, redirects, and several round trips to the final server that handles the user's request.

Third-party scripts often add to network overhead with things such as:

* Firing additional network requests
* Pulling in unoptimized images and videos
* Insufficient [HTTP caching](https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching), which forces frequent fetching of network resources
* Insufficient [server compression](https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/optimize-encoding-and-transfer) of resources
* Multiple instances of frameworks and libraries pulled in by different third-party embeds

**Rendering** [**#**](https://web.dev/third-party-javascript/#rendering)

The way third-party JavaScript is loaded matters a lot. If it's done synchronously in the critical rendering path it delays parsing of the rest of the document.

**Key Term**: The **critical rendering path** includes all resources that the browser needs to display the first screen's worth of content.

If a third party has server issues and fails to deliver a resource, rendering is blocked until the request times out, which can be anywhere from 10 to 80 seconds. You can test and simulate this problem with [WebPageTest Single-Point-of-Failure tests](https://css-tricks.com/use-webpagetest-api/#single-point-of-failure).

[A/B testing scripts](https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/loading-third-party-javascript/#ab_test_smaller_samples_of_users) can also often delay rendering. Most of them block content display until they complete processing—which can be true even for asynchronous A/B testing scripts.

**What to do about it** [**#**](https://web.dev/third-party-javascript/#what-to-do-about-it)

Using third-party JavaScript is often unavoidable, but there are things you can do to minimize adverse effects:

* When choosing third-party resources, favor those that send the least amount of code while still giving you the functionality you need.
* Use [performance budgets](https://web.dev/use-lighthouse-for-performance-budgets/) for third-party content to keep their cost in check.
* Don't use the same functionality from two different vendors. You probably don't need two tag managers or two analytics platforms.
* Routinely audit and clean out redundant third-party scripts.

To learn how to audit third-party content and load it efficiently for better performance and user experience, check out the other posts in the [Optimize your third-party resources](https://web.dev/fast/#optimize-your-third-party-resources) section.