Best practices for using third-party embeds

- What is an embed
- Performance impact of third-party embeds
- Loading best practices
  - Script ordering
  - Lazy-loading
    - Native lazy-loading for <iframe>
    - lazysizes library
    - Using data-lazy in Facebook
    - Lazy-loading Instagram feeds
  - Replace embeds with facades
    - Use static images as facades
    - Use dynamic images as facades
    - Use click-to-load to enhance facades
  - Remove or replace embeds with links
- Layout stability
  - Layout Shift Terminator
- Conclusion

Many sites use third-party embeds to create an engaging user experience by delegating some sections of a web page to another content provider. The most common examples of third-party content embeds are video players, social-media feeds, maps, and advertisements.

Third-party content can impact the performance of a page in many ways. It can be render-blocking, contend with other critical resources for network and bandwidth, or affect the Core Web Vitals metrics. Third-party embeds may also cause layout shifts as they load. This article discusses performance best practices that you can use when loading third-party embeds, efficient loading techniques, and the Layout Shift Terminator tool that helps reduce layout shifts for popular embeds.

It's best to use the techniques described in this post to load only offscreen or non-primary page content. This ensures that all the critical content gets indexed by search engines.

What is an embed

A third-party embed is any content displayed on your site that is:

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

Multiple offscreen embeds are shown, which could be lazy-loaded

Embeds are frequently used in the following:

- Websites related to sports, news, entertainment, and fashion use videos to augment textual content.
- Organizations with active Twitter or social media accounts embed feeds from these accounts to their web pages to engage and reach out to more people.
- Restaurant, park, and event venue pages often embed maps.

Third-party embeds are typically loaded in `<iframe>` elements on the page. Third-party providers offer HTML snippets often consisting of an `<iframe>` that pulls in a page composed of markup, scripts, and stylesheets. Some providers also use a script snippet that dynamically injects an `<iframe>` to pull other content in. This can make the third-party embeds heavy and affect the performance of the page by delaying its first-party content.

Performance impact of third-party embeds

Many popular embeds include over 100 KB of JavaScript, sometimes even going up to 2 MB. They take more time to load and keep the main thread busy when executing. Performance monitoring tools such as Lighthouse and Chrome DevTools help to measure the impact of third-party embeds on performance.

Reduce the impact of third-party code Lighthouse audit shows the list of third-party providers a page uses, with size and main-thread blocking time. The audit is available through Chrome DevTools under the Lighthouse tab.

It is a good practice to periodically audit the performance impact of your embeds and third-party code because embed source code may change. You can use this opportunity to remove any redundant code.

Reduce the impact of third-party code

Loading best practices

Third-party embeds can negatively impact performance, but they also offer important functionalities. To efficiently use third-party embeds and reduce their performance impact, follow the guidelines below.

Script ordering

In a well-designed page, the key first-party content will be the focus of the page, while the third-party embeds will occupy side-bars or appear after the first-party content.

For the best user experience, the main content should load quickly and before any other supporting content. For example, the news text on a news page should load before embeds for a Twitter feed or advertisements.

Requests for third-party embeds can get in the way of loading first-party content, so the position of a third-party script tag is important. Scripts can affect the loading sequence because the DOM construction pauses while scripts are executed. Place third-party script tags after the key first-party tags and use `async` or `defer` attributes to load them asynchronously.

```
<head>
   <title>Order of Things</title>
   <link rel="stylesheet" media="screen" href="/assets/application.css">
   <script src="index.js"></script>
   <script src="https://example.com/3p-library.js" async></script>
</head>
```

Lazy-loading

Since third-party content usually comes after the primary content, it may not be visible in the viewport when the page loads. In that case, downloading third-party resources may be deferred until the user scrolls down to that part of the page. This not only helps optimize the initial page load but also reduces the download costs for users on fixed data plans and slow network connections.

Delaying the download of content until it is actually needed is called lazy-loading. Depending on the requirements and the type of embed, you can use different lazy-loading techniques explained below.

Native lazy-loading for `<iframe>`

For third-party embeds loaded through `<iframe>` elements, you can use browser-level lazy-loading to defer loading offscreen iframes until users scroll near them. The loading attribute for `<iframe>` is available in Chrome 77 and above and has also been introduced to other Chromium-based browsers.

```
<iframe src="https://example.com"
        loading="lazy"
        width="600"
        height="400">
</iframe>
```

The loading attribute supports the following values:

- `lazy`: Indicates that the browser should defer loading the iframe. The browser will load the iframe when it is nearing the viewport. Use if the iframe is a good candidate for lazy-loading.
- `eager`: Loads the iframe immediately. Use if the iframe is not a good candidate for lazy-loading. If the `loading` attribute has not been specified, this is the default behavior—except in Lite mode.
- `auto`: The browser determines whether to lazy-load this frame.

Browsers that don't support the `loading` attribute ignore it, so you can apply native lazy-loading as a progressive enhancement. Browsers that support the attribute may have different implementations for the distance-from-viewport threshold (the distance at which the iframe starts loading).

Following are some ways in which you can lazy load iframes for different types of embeds.

- YouTube videos: To lazy-load a YouTube video player iframe, include the `loading` attribute to the embed code provided by YouTube. Lazy loading the YouTube embed can save approximately 500 KB on the initial page load.

```
<iframe src="https://www.youtube.com/embed/aKydtOXW8mI"
   width="560" height="315"
   loading="lazy"
   title="YouTube video player"
   frameborder="0"
```

```
   allow="accelerometer; autoplay; clipboard-write;
            encrypted-media; gyroscope; picture-in-picture"
   allowfullscreen>
</iframe>
```

- Google Maps: To lazy-load a Google Map iframe, include the `loading` attribute in the code for the iframe embed generated by the Google Maps Embed API. Following is an example of the code with a placeholder for the Google Cloud API key.

```
<iframe src="https://www.google.com/maps/embed/v1/place?key=API_KEY&q=PLACE_ID"
   width="600" height="450"
   style="border:0;"
   allowfullscreen=""
   loading="lazy">
</iframe>
```

lazysizes library

Because browsers use an embed's distance-from-viewport, in addition to signals like effective connection type and Lite-mode, to decide when an iframe should be loaded, native lazy-loading can be inconsistent. If you need better control on the distance thresholds or you want to provide a consistent lazy-loading experience across browsers, you can use the lazysizes library.

lazysizes is a fast, SEO-friendly lazy loader for both images and iframes. Once you have downloaded the component, it can be used with an iframe for a YouTube embed as follows.

```
<script src="lazysizes.min.js" async></script>

<iframe data-src="https://www.youtube.com/embed/aKydtOXW8mI"
   width="560" height="315"
   class="lazyload"
   title="YouTube video player"
   frameborder="0"
   allow="accelerometer; autoplay; clipboard-write;
        encrypted-media; gyroscope; picture-in-picture"
   allowfullscreen>
</iframe>
```

Similarly, lazysizes may be used with iframes for other third-party embeds.

Note that lazysizes uses the Intersection Observer API to detect when an element becomes visible.

Using data-lazy in Facebook

Facebook provides different types of social plugins that can be embedded. This includes posts, comments, videos, and the most popular *Like* button. All plugins include a setting for `data-lazy`. Setting it to `true` ensures that the plugin will use the browser's lazy-loading mechanism by setting the `loading="lazy"` iframe attribute.

Lazy-loading Instagram feeds

Instagram provides a block of markup and a script as part of the embed. The script injects an `<iframe>` into the page. Lazy-loading this `<iframe>` can improve performance as the embed can be over 100 KB gzipped in size. Many Instagram plugins for WordPress sites like WPZoom and Elfsight provide the lazy-loading option.

Replace embeds with facades

While interactive embeds add value to the page, many users may not interact with them. For example, not every user browsing a restaurant page will click, expand, scroll, and navigate the map embed. Similarly, not every user to a telecom service providers page will interact with the chatbot. In these cases, you can avoid loading or lazy-loading the embed altogether by displaying a facade in its place.

> A map embed with a zoom in and out feature.
>
> A map facade that is an image.

A facade is a static element that looks similar to the actual embedded third-party but is not functional and, therefore, much less taxing on the page load. Following are a few strategies to load such embeds optimally while still providing some value to the user.

Use static images as facades

Static images can be used instead of map embeds where you might not need to make the map interactive. You can zoom in on the area of interest on the map, capture an image, and use this instead of the interactive map embed. You can also use DevTools **Capture node screenshot** functionality to capture a screenshot of the embedded `iframe` element, as shown below.

Capture node screenshot

DevTools captures the image as a `png`, but you can also consider converting it to `WebP format for better performance`.

Use dynamic images as facades

This technique allows you to generate images corresponding to an interactive embed at run time. Following are some of the tools that allow you to generate static versions of embeds on your pages.

- **Maps Static API**: The Google Maps Static API service generates a map based on the URL parameters included in a standard HTTP request and returns the map as an image you can display on your web page. The URL needs to include the Google Maps API key and must be placed in the `<img>` tag on the page as the `src` attribute.

  The Static map maker tool helps to configure the parameters required for the URL and gives you the code for the image element in real-time.
  The following snippet shows code for an image with the source set to a Maps Static API URL. It has been included in a link tag that ensures that the actual map can be accessed by clicking on the image. (Note: API key attribute is not included in the url)

  ```
  <a href="https://www.google.com/maps/place/Albany,+NY/">
  <img
  src="https://maps.googleapis.com/maps/api/staticmap?center=Albany,+NY&zoom=13&scale=1
  &size=600x300&maptype=roadmap&format=png&visual_refresh=true" alt="Google Map of
  Albany, NY">
  </a>
  ```

- **Twitter screenshots**: Similar to map screenshots, this concept allows you to dynamically embed a Twitter screenshot instead of the live feed. Tweetpik is one of the tools that can be used to take screenshots of tweets. Tweetpik API accepts the URL of the tweet and returns an image with its contents. The API also accepts parameters to customize the background, colors, borders, and dimensions of the image.

Use click-to-load to enhance facades

The click-to-load concept combines lazy-loading and facades. The page initially loads with the facade. When the user interacts with the static placeholder by clicking on it, the third-party embed is loaded. This is also known as the import on interaction pattern and can be implemented using the following steps.

1. On page load: Facade or static element is included on the page.
2. On mouseover: Facade preconnects to the third-party embed provider.
3. On click: The facade is replaced by the third-party product.

Facades may be used with third-party embeds for video players, chat widgets, authentication services, and social media widgets. YouTube video embeds that are just images with a play button are facades that we come across frequently. The actual video loads only when you click on the image.

You can build a custom click-to-load facade using the *import on interaction* pattern or use one of the following open source facades available for different types of embeds.

- YouTube facade

  Lite-youtube-embed is a recommended facade for the YouTube player, which looks like the real player but is 224 times faster. It can be used by downloading the script and stylesheet and then using the `<lite-youtube>` tag in HTML or JavaScript. Custom player parameters supported by YouTube may be included through the `params` attribute.

  ```
  <lite-youtube videoid="ogfYd705cRs" playlabel="Play: Keynote (Google I/O
  '18)"></lite-youtube>
  ```

  Following is a comparison between the lite-youtube-embed and the actual embed.

  Lite YouTube embed

  ```
    <figcaption>
      A lite-YouTube embed
    </figcaption>
  </figure>
  <figure>
  ```

  Actual YouTube embed

  ```
    <figcaption>
      A YouTube embed
    </figcaption>
  </figure>
  ```

```
Other similar facades available for YouTube and Vimeo players are [lite-
youtube](https://github.com/justinribeiro/lite-youtube), [lite-vimeo-
embed](https://github.com/luwes/lite-vimeo-embed), and [lite-
vimeo](https://github.com/slightlyoff/lite-vimeo).
```

- Chat widget facade

  React-live-chat-loader loads a button that looks like a chat embed instead of the embed itself. It can be used with various chat provider platforms such as Intercom, Help Scout, Messenger, and so on. The look-alike widget is much lighter than the chat-widget and loads faster. It can be replaced by the actual chat widget when the user hovers or clicks on the button or if the page has been idle for a long time. The Postmark case study explains how they implemented react-live-chat-loader and performance improvements they achieved.

Postmark chat widget

Remove or replace embeds with links

If you find that some third-party embeds result in poor loading performance and using any of the techniques above is not an option, the simplest thing that you can do is remove the embed entirely. If you still want your users to be able to access the content in the embed, you can provide a link to it with `target="_blank"` so that the user can click and view it in another tab.

Layout stability

While dynamically loading embedded content can improve the loading performance of a page, it can sometimes cause unexpected movement of page content. This is known as layout shift.

Since visual stability is important to guarantee a smooth user experience, Cumulative Layout Shift (CLS) measures how often those shifts happen and how disruptive they are. Layout shifts can be avoided by reserving space during page load for elements that are going to be dynamically loaded later. The browser can determine the space to be reserved if it knows the width and height of the elements. You can ensure this by specifying the `width` and `height` attributes of iframes or by setting a fixed size for static elements where the third-party embed will be loaded. For example, an iframe for a YouTube embed should have width and height specified as follows.

```
<iframe src="https://www.youtube.com/embed/aKydtOXW8mI" width="560" height="315">
</iframe>
```

Popular embeds like YouTube, Google Maps, and Facebook provide the embed code with size attributes specified. However, there may be providers who do not include this. For example, this code snippet does not indicate the dimensions of the resulting embed.

```
<a class="twitter-timeline"
href="https://twitter.com/ChannelNewsAsia?ref_src=twsrc%5Etfw" data-tweet-
limit="1">Tweets by ChannelNewsAsia</a>
<script async src="https://platform.twitter.com/widgets.js" charset="utf-8"></script>
```

You can use DevTools to inspect the injected `iframe` after this page is rendered. As seen in the following snippet, the height of the injected iframe is fixed while the width is specified in percentage.

```
<iframe id="twitter-widget-0" scrolling="no" frameborder="0" allowtransparency="true"
allowfullscreen="true" class="twitter-timeline twitter-timeline-rendered"
style="position: static; visibility: visible; display: inline-block; width: 100%;
padding: 0px; border: none; max-width: 1000px; min-width: 180px; margin-top: 0px;
```

```
margin-bottom: 0px; min-height: 200px; height: 6238.31px;" data-widget-
id="profile:ChannelNewsAsia" title="Twitter Timeline">
</iframe>
```

This information can be used to set the size of the containing element to ensure that the container does not expand on loading the feed and there is no layout shift. Following snippet may be used to fix the size of the embed included previously.

```
<style>
    .twitterfeed { display: table-cell;  vertical-align: top; width: 100vw; }
    .twitter-timeline {height: 400px !important; }
</style>
<div class=twitterfeed>
       <a class="twitter-timeline"
href="https://twitter.com/ChannelNewsAsia?ref_src=twsrc%5Etfw" data-tweet-
limit="1">Tweets by ChannelNewsAsia</a>
       <script async src="https://platform.twitter.com/widgets.js" charset="utf-
8"></script>
</div>
```

Layout Shift Terminator

Since third-party embeds often omit the dimensions (width, height) for the final content they render, they can cause significant layout shifts on a page. This problem can be tricky to address without manually inspecting the final sizes using DevTools at a variety of different viewport sizes.

Now there's an automated tool, Layout Shift Terminator, that can help you reduce layout shifts from popular embeds, such as from Twitter, Facebook, and other providers.
Layout Shift Terminator:

- Loads the embed client-side in an iframe.
- Resizes the iframe to various popular viewport sizes.
- For each popular viewport, captures the dimensions of the embed to later generate media queries and container queries.
- Sizes a min-height wrapper around the embed markup using media queries (and container queries) until the embed initializes (after which the min-height styles are removed).
- Generates an optimized embed snippet that can be copy/pasted where you would otherwise be including the embed in your page.

Layour shift Terminal

Try out the Layout Shift Terminator, and feel free to leave any feedback on GitHub. The tool is in a beta state and aims to improve over time with further refinements.

Conclusion

Third-party embeds can provide a lot of value to users, but as the number and size of embeds on a page increases, performance can suffer. That’s why it is necessary to measure, judge, and use appropriate loading strategies for embeds based on their position, relevance, and potential users' needs.

□□□ #3
□ □□□ □□□ 2026-10-31 21:51:10 UTC
□ cocobook-sync □□□ 2026-11-04 17:37:06 UTC