BLOG >

React Compiler v1.0

Oct 7, 2025 by Lauren Tan, Joe Savona, and Mofei Zhang.

The React team is excited to share new updates:

1. React Compiler 1.0 is available today.
2. Compiler-powered lint rules ship in `eslint-plugin-react-hooks`'s `recommended` and `recommended-latest` preset.
3. We've published an incremental adoption guide, and partnered with Expo, Vite, and Next.js so new apps can start with the compiler enabled.

We are releasing the compiler's first stable release today. React Compiler works on both React and React Native, and automatically optimizes components and hooks without requiring rewrites. The compiler has been battle tested on major apps at Meta and is fully production-ready.

React Compiler is a build-time tool that optimizes your React app through automatic memoization. Last year, we published React Compiler's first beta and received lots of great feedback and contributions. We're excited about the wins we've seen from folks adopting the compiler (see case studies from Sanity Studio and Wakelet) and are excited to bring the compiler to more users in the React community.

This release is the culmination of a huge and complex engineering effort spanning almost a decade. The React team's first exploration into compilers started with Prepack in 2017. While this project was eventually shut down, there were many learnings that informed the team on the design of Hooks, which were designed with a future compiler in mind. In 2021, Xuan Huang demoed the first iteration of a new take on React Compiler.

Although this first version of the new React Compiler was eventually rewritten, the first prototype gave us increased confidence that this was a tractable problem, and the learnings that an alternative compiler architecture could precisely give us the

memoization characteristics we wanted. Joe Savona, Sathya Gunasekaran, Mofei Zhang, and Lauren Tan worked through our first rewrite, moving the compiler's architecture into a Control Flow Graph (CFG) based High-Level Intermediate Representation (HIR). This paved the way for much more precise analysis and even type inference within React Compiler. Since then, many significant portions of the compiler have been rewritten, with each rewrite informed by our learnings from the previous attempt. And we have received significant help and contributions from many members of the React team along the way.

This stable release is our first of many. The compiler will continue to evolve and improve, and we expect to see it become a new foundation and era for the next decade and more of React.

You can jump straight to the quickstart, or read on for the highlights from React Conf 2025.

DEEP DIVE

How does React Compiler work?

Show Details

Use React Compiler Today

To install the compiler:

npm

Terminal Copy

```
npm install --save-dev --save-exact babel-plugin-react-compiler@latest
```

pnpm

Terminal Copy

```
pnpm add --save-dev --save-exact babel-plugin-react-compiler@latest
```

yarn

```
Terminal                                                    Copy
yarn add --dev --exact babel-plugin-react-compiler@latest
```

As part of the stable release, we've been making React Compiler easier to add to your projects and added optimizations to how the compiler generates memoization. React Compiler now supports optional chains and array indices as dependencies. These improvements ultimately result in fewer re-renders and more responsive UIs, while letting you keep writing idiomatic declarative code.

You can find more details on using the Compiler in our docs.

What we're seeing in production

The compiler has already shipped in apps like Meta Quest Store. We've seen initial loads and cross-page navigations improve by up to 12%, while certain interactions are more than 2.5× faster. Memory usage stays neutral even with these wins. Although your mileage may vary, we recommend experimenting with the compiler in your app to see similar performance gains.

Backwards Compatibility

As noted in the Beta announcement, React Compiler is compatible with React 17 and up. If you are not yet on React 19, you can use React Compiler by specifying a minimum target in your compiler config, and adding `react-compiler-runtime` as a dependency. You can find docs on this here.

Enforce the Rules of React with compiler-powered linting

React Compiler includes an ESLint rule that helps identify code that breaks the Rules of React. The linter does not require the compiler to be installed, so there's no risk in upgrading eslint-plugin-react-hooks. We recommend everyone upgrade today.

If you have already installed `eslint-plugin-react-compiler`, you can now remove it and use `eslint-plugin-react-hooks@latest`. Many thanks to @michaelfaith for contributing to this improvement!

To install:

npm

Terminal Copy

```
npm install --save-dev eslint-plugin-react-hooks@latest
```

pnpm

Terminal Copy

```
pnpm add --save-dev eslint-plugin-react-hooks@latest
```

yarn

Terminal Copy

```
yarn add --dev eslint-plugin-react-hooks@latest
```

```
// eslint.config.js (Flat Config)
import reactHooks from 'eslint-plugin-react-hooks';
import { defineConfig } from 'eslint/config';

export default defineConfig([
  reactHooks.configs.flat.recommended,
]);
```

```
// eslintrc.json (Legacy Config)
{
  "extends": ["plugin:react-hooks/recommended"],
  // ...
}
```

To enable React Compiler rules, we recommend using the `recommended` preset. You can also check out the README for more instructions. Here are a few examples we featured at React Conf:

- Catching `setState` patterns that cause render loops with `set-state-in-render`.
- Flagging expensive work inside effects via `set-state-in-effect`.
- Preventing unsafe ref access during render with `refs`.

What should I do about useMemo, useCallback, and React.memo?

By default, React Compiler will memoize your code based on its analysis and heuristics. In most cases, this memoization will be as precise, or moreso, than what you may have written — and as noted above, the compiler can memoize even in cases where `useMemo`/`useCallback` cannot be used, such as after an early return.

However, in some cases developers may need more control over memoization. The `useMemo` and `useCallback` hooks can continue to be used with React Compiler as an escape hatch to provide control over which values are memoized. A common use-case for this is if a memoized value is used as an effect dependency, in order to ensure that an effect does not fire repeatedly even when its dependencies do not meaningfully change.

For new code, we recommend relying on the compiler for memoization and using `useMemo`/`useCallback` where needed to achieve precise control.

For existing code, we recommend either leaving existing memoization in place (removing it can change compilation output) or carefully testing before removing the memoization.

New apps should use React Compiler

We have partnered with the Expo, Vite, and Next.js teams to add the compiler to the new app experience.

Expo SDK 54 and up has the compiler enabled by default, so new apps will automatically be able to take advantage of the compiler from the start.

Terminal Copy

```
npx create-expo-app@latest
```

Vite and Next.js users can choose the compiler enabled templates in `create-vite` and `create-next-app`.

Terminal Copy

```
npm create vite@latest
```

Terminal Copy

```
npx create-next-app@latest
```

Adopt React Compiler incrementally

If you're maintaining an existing application, you can roll out the compiler at your own pace. We published a step-by-step incremental adoption guide that covers gating strategies, compatibility checks, and rollout tooling so you can enable the compiler with confidence.

swc support (experimental)

React Compiler can be installed across several build tools such as Babel, Vite, and Rsbuild.

In addition to those tools, we have been collaborating with Kang Dongyoon (@kdy1dev) from the swc team on adding additional support for React Compiler as an swc plugin. While this work isn't done, Next.js build performance should now be considerably faster when the React Compiler is enabled in your Next.js app.

We recommend using Next.js 15.3.1 or greater to get the best build performance.

Vite users can continue to use vite-plugin-react to enable the compiler, by adding it as a Babel plugin. We are also working with the oxc team to add support for the compiler. Once rolldown is officially released and supported in Vite and oxc support is added for React Compiler, we'll update the docs with information on how to migrate.

Upgrading React Compiler

React Compiler works best when the auto-memoization applied is strictly for performance. Future versions of the compiler may change how memoization is applied, for example it could become more granular and precise.

However, because product code may sometimes break the rules of React in ways that aren't always statically detectable in JavaScript, changing memoization can occasionally have unexpected results. For example, a previously memoized value might be used as a dependency for a `useEffect` somewhere in the component tree. Changing how or whether this value is memoized can cause over or under-firing of that `useEffect`. While we encourage useEffect only for synchronization, your codebase may have `useEffect`s that cover other use cases, such as effects that needs to only run in response to specific values changing.

In other words, changing memoization may under rare circumstances cause unexpected behavior. For this reason, we recommend following the Rules of React and employing continuous end-to-end testing of your app so you can upgrade the compiler with confidence and identify any rules of React violations that might cause issues.

If you don't have good test coverage, we recommend pinning the compiler to an exact version (eg `1.0.0`) rather than a SemVer range (eg `^1.0.0`). You can do this by passing the `--save-exact` (npm/pnpm) or `--exact` flags (yarn) when upgrading the compiler. You should then do any upgrades of the compiler manually, taking care to check that your app still works as expected.

Thanks to Jason Bonta, Jimmy Lai, Kang Dongyoon (@kdy1dev), and Dan Abramov for reviewing and editing this post.

PREVIOUS
React Conf 2025 Recap

NEXT
Introducing the React Foundation

Meta Open Source

uwu?

Learn React

- Quick Start
- Installation
- Describing the UI
- Adding Interactivity
- Managing State
- Escape Hatches

API Reference

- React APIs
- React DOM APIs

Community

- Code of Conduct
- Meet the Team
- Docs Contributors
- Acknowledgements

More

- Blog
- React Native
- Privacy
- Terms