List of JavaScript engines

The first engines for JavaScript were mere interpreters of the source code, but all relevant modern engines use just-in-time compilation for improved performance.[1] JavaScript engines are typically developed by web browser vendors, and every major browser has one. In a browser, the JavaScript engine runs in concert with the rendering engine via the Document Object Model and Web IDL bindings.[2] However, the use of JavaScript engines is not limited to browsers; for example, the V8 engine is a core component of the Node.js runtime system.[3] They are also called ECMAScript engines, after the official name of the specification. With the advent of WebAssembly, some engines can also execute this code in the same sandbox as regular JavaScript code.[4][3]

History

The first JavaScript engine was created by Brendan Eich in 1995 for the Netscape Navigator web browser.[5] It was a rudimentary interpreter for the nascent language Eich invented.[6] (This evolved into the SpiderMonkey engine, still used by the Firefox browser.[5])

Google debuted its Chrome browser in 2008, introducing the V8 JavaScript engine that was at the time much faster than its competition.[7][8] This sparked a race between browser vendors to deliver ever-faster JavaScript engines.[9] The key innovations around this era were switching from basic tree-walking interpreters to stack- and register-based bytecode VM interpreters, just-in-time compilation (JIT), inline caching (hidden classes) and generational GC. Apple released the JIT-enabled Nitro engine in June 2008 for its Safari browser, which had 30% better performance than its predecessor.[10][11] Mozilla followed suit in August 2008 with TraceMonkey, the first JIT compiler for SpiderMonkey engine, released in Firefox 3.1.[12] Opera joined the performance race with their register-bytecode based and JIT-enabled Carakan engine, announced in February 2009 [13] and released in April 2010.[14] Microsoft's first JIT-enabled Chakra engine, in development since 2008,[15] debuted as part of Internet Explorer 9 in 2011. Its major rewrite appeared in Microsoft Edge Legacy in 2015 and open-sourced as ChakraCore in 2016.

Further performance gains in major JavaScript engines were later achieved with the introduction of multi-tiered JIT architectures. Progressively advanced JIT compilers are used to optimize hotspots in user code, with each next tier delivering ever more performant native code at the cost of slower compile time. Chrome was the first to implement it in V8 in 2010 with the introduction of Crankshaft, a 2-tiered JIT compiler.[16] By 2023, architecture of V8 evolved into 4 tiers: Ignition – register-based bytecode interpreter, Sparkplug – a fast non-optimizing JIT compiler, Maglev and TurboFan – slower optimizing JIT compilers.[17] JavaScriptCore today has a similar 4-tier architecture,[18] while Firefox's SpiderMonkey and ChakraCore have 3 tiers (interpreter and 2-tiered JIT).[19][20] This ever-increasing complexity of JIT compilers, however, has been criticized as a rich source of browser bugs, prompting some browser vendors to disable JIT altogether, such as Microsoft Edge's "Super Duper Secure Mode", introduced in 2021 and reportedly with minor performance impact for daily browsing.[21]

V8's influence expanded beyond browsers with the release of Node.js in 2009 and its package manager npm in 2010. As their popularity exploded, V8 also became the engine powering vast amounts of server-side JavaScript code.[22][23] In 2013, Electron framework appeared that let developers create desktop apps with web technologies as well, using Chromium with V8 and Node.js under the hood.[24]

Taking advantage of performance improvements in JavaScript engines, Emscripten C/C++-to-JavaScript compiler appeared in 2010-2011 and allowed running existing complex C/C++ code, such as game engines and even whole virtual machines, directly in the browser. asm.js, a highly optimizable low-level subset of JavaScript for such compilers emerged in 2013, with Firefox being the first to implement specific optimizations for it with OdinMonkey module.[25] Eventually asm.js and NaCl (a competing Google's technology) evolved into WebAssembly standard in 2017, with all major engines adding support for it.[4]

The six-year gap between ECMAScript 5 (2009) and ES6 (2015) marked a major inflection point for JavaScript engine development. Whereas ES5 was a relatively compact language with straightforward basic implementation, ES6 introduced foundations of modern JavaScript, with substantially increased complexity, as well as an annual release cycle for new specification versions. This put the focus not only on performance optimization, but also on keeping pace with the rapidly evolving standards (of both ECMAScript and wider web ecosystem) and led to a consolidation around engines that had the resources to do it. Many smaller projects did not end up fully implementing ES6 spec, and Opera and Microsoft both ceased developing their engines and adopted V8 in 2013 and 2021.[26][27] Nashorn engine was dropped from OpenJDK over a similar concern about ECMAScript's rapid development.[28]

List

Engine	License	Standard	JIT	Wasm	Written in	Description
V8	BSD-3-Clause	ESnext	Yes	Yes	C++	JavaScript engine of Google Chrome and Chromium/Blink-based browsers, such as Microsoft Edge. Also used in Node.js and Deno runtimes, Electron framework and numerous other projects. Currently has a register-based bytecode interpreter (Ignition) and three tiers of JIT compilers (Sparkplug, Maglev and Turbofan).
SpiderMonkey	MPL-2.0	ESnext	Yes	Yes	C++	JavaScript engine of Firefox and other Mozilla Gecko applications. The engine currently includes interpreter, 2-tier JIT for JavaScript (Baseline Compiler and WarpMonkey), and a separate 2-tier JIT for WebAssembly.[19] Previously, the engine included components such as the TraceMonkey compiler (first JavaScript JIT), JägerMonkey, IonMonkey, as well as OdinMonkey optimization module notable for pioneering asm.js.[25]
JavaScriptCore	LGPL-2.0, parts under BSD	ESnext	Yes	Yes	C++	JavaScript engine of Safari and WebKit-based browsers. Also used in Bun runtime. Started out in 2001 as a fork of KJS engine. In 2008 the engine was rewritten to use a direct-threaded register-based bytecode interpreter, codenamed SquirrelFish. Shortly after, SquirrelFish Extreme (Nitro in Apple's marketing terms)[29] was released, achieving over 2x speedup over it by using polymorphic inline caching and context threading[30] – a basic form of JIT.[31] These components later became LLint bytecode interpreter and Baseline JIT.[32] In 2011, the first optimizing compiler was added – DFG ("Data Flow Graph") JIT.[32][33] In 2014, an advanced LLVM-based compiler was added, dubbed Fourth Tier LLVM (FTL) JIT.[34] In 2016, FTL was

						rewritten to use a newly built B3 backend instead of LLVM.[35][18]
ChakraCore	MIT	ES2019	Yes	Yes	C++	JavaScript engine of Microsoft Edge Legacy.[36] Open-sourced under the name ChakraCore in 2016. Microsoft discontinued maintenance in 2021, leaving it to the community, but it has received little attention since then. It features a register-based bytecode interpreter and a two-tier JIT compiler.[20]
LibJS	BSD-2-Clause	ESnext	No	Yes	C++	JavaScript engine of the SerenityOS and Ladybird browser.[37] Initially a basic AST-walking interpreter, later upgraded to a register-based bytecode VM interpreter.[38][39][40][41] Supports WebAssembly via LibWasm library.
Rhino	MPL-2.0	ES2023 (partial)	via JVM	No	Java	JavaScript engine from Mozilla for the Java platform. Development started at Netscape in 1997. Originally it compiled JavaScript to JVM bytecode, but the implementation was inefficient and it was mainly used in interpreter mode.[42] Bundled in JDK 6 (2006) as javax.script. In JDK 8 (2014) it was replaced by the faster Nashorn engine.[43] Despite this, the project continues development and implemented support for many parts of newer standards.[44] Used in RingoJS runtime.
Nashorn	GPL-2.0	ES6 (partial)	via JVM	No	Java	JavaScript engine from Oracle used in JDK 8–14.[45] Generates more efficient JVM bytecode than Rhino, taking advantage of the new invokedynamic instruction introduced in JDK 7.[46] Deprecated in JDK 11 (2018) due to maintenance challenges owing to fast evolution of ECMAScript standard,[28] and removed in JDK 15 (2020).[47] Development continues till today as a standalone library.[48]
GraalJS	GFTC, UPL-1.0	ESnext	via GraalVM	Yes	Java	JavaScript engine of GraalVM for JVM platform.

						Implemented with Truffle, a novel language framework for automatically deriving high-performance code from interpreters, based on the idea of partial evaluation of interpreters (Futamura projection).[49][50] Generally much faster than both Rhino and Nashorn, with performance comparable to major browser engines.[51] Supports WebAssembly via GraalWasm module.
Hermes	MIT	ES2023 (partial)	No	No	C++	Developed by Facebook for React Native mobile apps,[52] but can also be used independent from React Native. Precompiles JavaScript to optimized bytecode ahead-of-time to improve app start-up time.[53][54][41] Static Hermes project aims to add sound static typing and JavaScript-to-native code compilation – both currently experimental, as well as various improvements to VM – the latter released as Hermes V1 in React Native 0.82.[55]
Duktape	MIT	ES6 (partial)	No	No	C	A small footprint, easily embeddable ES5 engine with some features from ES6 and later.[56][57][58][59] Used by NetSurf browser.
XS	LGPL-3.0+, Apache-2.0	ES2025	No	No	C	JavaScript engine for microcontrollers with limited resources, compliant with latest ECMAScript versions.[60][61][59][41][62] Originally developed by Kinoma and first open-sourced in 2015 as a key component of their KinomaJS IoT framework. Now maintained by Moddable as part of the Moddable SDK.[63]
MuJS	ISC	ES5	No	No	C	A lightweight ECMAScript interpreter library, designed for embedding in other software to extend them with scripting capabilities. Originally developed for MuPDF.[64][57]
QuickJS	MIT	ES2025	No	No	C	A lightweight ECMAScript engine by Fabrice Bellard and Charlie Gordon, currently featuring almost complete support of ES2023.[65][57][53][66][41]

						Implements a stack-based bytecode VM interpreter and a single-pass AST-free direct-to-bytecode compiler. Can precompile .js to bytecode ahead-of-time and produce a binary executable with no external dependencies. Used in WinterJS and Amazon's LLRT runtimes, CouchDB, nginx (as alternative to njs), yt-dlp.
V4 (QJSEngine)	LGPL, GPL, Qt	ES2016	Yes	No	C++	JavaScript engine of Qt's QML framework, powering UI of modern Qt applications, Qt Quick, KDE Plasma.[67][68][69][70] Initially appeared in Qt 5.0 (2012) as a wrapper for V8,[71] later replaced by Qt's lightweight home-grown "V4" engine in Qt 5.2 (2013)[72][73] ES7/ES2016-compliant[74] and JIT-enabled, using macroassembler code borrowed from JavaScriptCore.[75][76] Only used for QML, not for the embedded browser engine modules: QtWebKit module used JavaScriptCore, later succeeded by Blink/V8-based QtWebEngine.
Espruino	MPL-2.0	ES5 (subset)	No	No	C	A very small footprint interpreter specifically for microcontrollers.[58] Can run in less than 8 kB of RAM by executing from source (rather than bytecode).
JerryScript	Apache-2.0	ES2022	No	No	C	Ultra-lightweight JavaScript engine by Samsung for microcontrollers with less than 64 KB RAM.[57][59] Supports ES5 and large parts of recent standards up to ES2022.
Escargot	LGPL-2.1	ESnext	No	Yes	C++	A newer lightweight JavaScript engine by Samsung for resource-constrained environments, targeting mid-range devices like phones and TVs. Under active development, fully supports latest ECMAScript standard except a few minor features, as well as WebAssembly.[77][57][78] Implements a register-based VM interpreter.
otto	MIT	ES5	No	No	Go	JavaScript engine in pure Go, implementing ES5 but with

						non-compliant regex engine from Go.
Goja	MIT	ES2023 (partial)	No	No	Go	JavaScript engine in pure Go, inspired by otto, fully implementing ES5 and parts of newer standards. Used by Ethereum's official Go implementation (Geth).[79]
Boa	MIT, Unlicense	ESnext	No	No	Rust	A JavaScript engine written in Rust.[80][81]
Kiesel	MIT	ESnext	No	No	Zig	JavaScript engine by Linus Groh written in Zig.[82][83][84][85]
engine262	MIT	ESnext	No	No	TypeScript	A JavaScript engine written in TypeScript for development and exploration, intended essentially as a reference implementation of the language.[41]

Historical engines

These engines have been discontinued and are mostly interesting for the historical perspective. Some may still be in use today for legacy purposes (especially Microsoft's old engines that continue to ship with Windows).

Engine	License	Standard	JIT	Written in	Description
Mocha	Proprietary	JS1.1 (≈ES1)	No	C++	The first JavaScript engine. Prototyped by Brendan Eich in a 10-day sprint in May 1995 and shipped in September 1995 in Netscape Navigator 2.0b1.[86] Due to feature freeze in August 1995, it was incomplete relative to the envisioned language design. Version 1.1 was released with Navigator 3.0 in 1996, completing the initial development of the language and served as the basis for the first edition of ECMAScript standard in 1997. The engine had a recursive descent parser that directly emitted bytecode, a stack-based virtual machine, used reference counting instead of GC and represented primitives as discriminated unions rather than tagged pointers. Source code as part of Navigator 3.0.2 is available from Internet Archive.[87] For Navigator 4.0, the engine was completely rewritten and renamed SpiderMonkey.[88]
KJS	LGPL-2.0+	ES5	No	C++	The engine originally used in Konqueror, and one component of KHTML. Predecessor to JavaScriptCore, which forked it in 2001 and significantly diverged since. Originally an AST-walking interpreter, upgraded to a bytecode interpreter (FrostByte) in 2008.[89] Development largely stopped as Konqueror transitioned from KHTML first to QtWebKit (JavaScriptCore) and then QtWebEngine (V8). KDE Plasma 6 finally removed KJS/KHTML engine altogether.[90]
JScript	Proprietary	ES3	No	C++[91]	JavaScript engine of Internet Explorer 3.0 - 8.0. One component of the MSHTML (Trident) browser engine.
JScript .NET	Proprietary	ES4	via CLR		JavaScript engine for .NET Framework 1.0 - 4.x developed by Microsoft, used in ASP.NET, based on Common Language Runtime and COM Interop. Originally introduced as JScript .NET in 2000,[92] with later versions called JScript 7.0, JScript 8.0[93] and, finally, JScript 10.0 in .NET Framework 4.0.[94] However, it is unrelated to both JScript (aka JScript 1.0 - 5.8) and JScript9 engines.[93] It supports ES3, many of the features proposed for ES4,[88] as well as many unique .NET-related dialectisms.[95][96] Dropped from .NET Core and the modern .NET platform.
Chakra (JScript9)	Proprietary	ES5	Yes	C++	JavaScript engine of Internet Explorer 9.0 - 11.0. It was first previewed in 2010 at MIX 10.[97] Still ships in Windows 11 as JScript9Legacy.dll, a drop-in compatible replacement for the original JScript engine (jscript.dll).[98][99]
Linear A, Linear B,	Proprietary	ES3	No	C++	A series of early JavaScript engines of Opera browser. Unnamed engine in Opera

Futhark					3 (1997–2000). Linear A in Opera 4–6 (2000–2002). Linear B in Opera 7–9.27 (2002–2007) – implemented complete ECMAScript support. Futhark in Opera 9.50-10.20 (2007–2010).[100]
Carakan	Proprietary	ES5	Yes	C++	JavaScript engine of Opera 10.50 - 12.x (2010–2013).[13][101] It featured register-based bytecode and JIT.[13] The next major release, Opera 15, switched from Presto to WebKit/Blink browser engine along with its V8 JavaScript engine.[26]
QtScript	LGPL, GPL, Qt	ES5	No	C++	Originally developed by Trolltech, now owned by The Qt Company. First appeared in Qt 4.3, deprecated in Qt 5.5 in favor of QML's V4 and then dropped in Qt 6.5. Based on JavaScriptCore engine and featured a few Qt-specific ECMAScript extensions, such as QObject-style signal and slot connections.
Epimetheus	MPL-1.1 GPL-2.0+ LGPL-2.1+	ES4	No	C++	Netscape's experimental "JavaScript 2.0" implementation towards ES4 proposal.[88][102]
Tamarin	MPL-2.0	ES4	Yes	C++	ActionScript and ECMAScript engine used in Adobe Flash Player 9. Implemented ES4 proposal.
Narcissus	MPL-1.1+ GPL-2.0+ LGPL-2.1+	ES5	No	JavaScript	JavaScript engine implemented in JavaScript (a meta-circular evaluator) by Brendan Eich, the creator of JavaScript. Used for experimentation with new language features for ES6 efforts.[88]
Links	GPL-2.0+	No	No	C	Links browser had a basic JavaScript implementation until version 2.1pre28. Removed in 2.1pre29 for being "very buggy".[103]
Voyager	GPL-3.0	ES3	No	C	Voyager, another web browser for Amiga in the late 1990s – early 2000s, had its own JavaScript engine.
ScriptEase	Proprietary	ES3[104]	No	C++	An old proprietary engine last updated in 2003. Only notable for its use in the James Webb Space Telescope.[105]

See also

- Comparison of browser engines
- Comparison of web browsers

References

1. Looper, Jen (21 September 2015). "A Guide to JavaScript Engines for Idiots" (https://web.archive.org/web/20181208123231/http://developer.telerik.com/featured/a-guide-to-javascript-engines-for-idiots/). Telerik Developer Network. Archived from the original (http://developer.tel

erik.com/featured/a-guide-to-javascript-engines-for-idiots/) on 8 December 2018. Retrieved 8 December 2018.

2. "How Blink Works" (https://docs.google.com/document/d/1aitSOucL0VHZa9Z2vbRJSyAIsAz24kX8LFByQ5xQnUg). Retrieved 12 March 2024.
3. "Documentation · V8" (https://v8.dev/docs). Google. Retrieved 3 March 2024.
4. Nelaturu, Keerthi (September 2020). "WebAssembly: What's the big deal?" (https://medium.com/coinmonks/webassembly-whats-the-big-deal-662396ff1cd6). *medium.com*. Retrieved 3 March 2024.
5. Eich, Brendan (21 June 2011). "New JavaScript Engine Module Owner" (http://brendaneich.com/2011/06/new-javascript-engine-module-owner/).
6. Fin JS (17 June 2016), "Brendan Eich – CEO of Brave" (https://www.youtube.com/watch?v=XOmhtfTrRxc), *YouTube*, retrieved 7 February 2018
7. "Big browser comparison test: Internet Explorer vs. Firefox, Opera, Safari and Chrome" (https://www.pcgameshardware.com/aid,687738/Big-browser-comparison-test-Internet-Explorer-vs-Firefox-Opera-Safari-and-Chrome-Update-Firefox-35-Final/Practice/). *PC Games Hardware*. Computec Media AG. 3 July 2009. Archived (https://web.archive.org/web/20120502043027/http://www.pcgameshardware.com/aid,687738/Big-browser-comparison-test-Internet-Explorer-vs-Firefox-Opera-Safari-and-Chrome-Update-Firefox-35-Final/Practice/) from the original on 2 May 2012. Retrieved 28 June 2010.
8. Purdy, Kevin (11 June 2009). "Lifehacker Speed Tests: Safari 4, Chrome 2" (https://lifehacker.com/lifehacker-speed-tests-safari-4-chrome-2-and-more-5286869). *Lifehacker*. Archived (https://web.archive.org/web/20210414095403/https://lifehacker.com/lifehacker-speed-tests-safari-4-chrome-2-and-more-5286869) from the original on 14 April 2021. Retrieved 8 May 2021.
9. "Mozilla asks, 'Are we fast yet?' " (https://www.wired.com/2010/09/mozilla-asks-are-we-fast-yet/). *Wired*. Archived (https://web.archive.org/web/20180622213244/https://www.wired.com/2010/09/mozilla-asks-are-we-fast-yet/) from the original on 22 June 2018. Retrieved 18 January 2019.
10. "Apple Releases Safari 5" (https://www.prnewswire.com/news-releases/apple-releases-safari-5-95817479.html). *www.prnewswire.com*.
11. Shankland, Stephen (2 March 2010). "Opera 10.5 brings new JavaScript engine" (https://web.archive.org/web/20131003051354/http://news.cnet.com/8301-30685_3-20000110-264.html). *CNET*. CBS Interactive. Archived from the original (https://news.cnet.com/8301-30685_3-20000110-264.html) on 3 October 2013. Retrieved 30 January 2012.
12. "TraceMonkey: JavaScript Lightspeed, Brendan Eich's Blog" (https://brendaneich.com/2008/08/tracemonkey-javascript-lightspeed/). Archived (https://web.archive.org/web/20151204091540/https://brendaneich.com/2008/08/tracemonkey-javascript-lightspeed/) from the original on 4 December 2015. Retrieved 22 July 2020.
13. "Carakan" (https://web.archive.org/web/20090531095136/http://labs.opera.com/news/2009/02/05/). Archived from the original (http://labs.opera.com/news/2009/02/05/) on 31 May 2009. Retrieved 9 July 2009.
14. "Opera 10.50 - A Browser Worth Switching To? - NYTimes.com" (https://archive.nytimes.com/www.nytimes.com/external/readwriteweb/2010/03/02/02readwriteweb-opera-1050-a-browser-worth-switching-to-88992.html). *archive.nytimes.com*. Retrieved 29 December 2025.
15. "Microsoft Edge's JavaScript engine to go open-source" (https://www.tenforums.com/windows-10-news/32685-microsoft-edge-s-javascript-engine-go-open-source.html). 7 December 2015.
16. "A New Crankshaft for V8" (https://blog.chromium.org/2010/12/new-crankshaft-for-v8.html). *Chromium Blog*. 7 December 2010. Retrieved 22 April 2011.
17. "Maglev - V8's Fastest Optimizing JIT · V8" (https://v8.dev/blog/maglev).

18. "Speculation in JavaScriptCore" (https://webkit.org/blog/10308/speculation-in-javascriptcore). 29 July 2020.
19. "SpiderMonkey — Firefox Source Docs documentation" (https://firefox-source-docs.mozilla.org/js/index.html).
20. "Architecture Overview" (https://github.com/chakra-core/ChakraCore/wiki/Architecture-Overview). *GitHub*.
21. "Super Duper Secure Mode" (https://microsoftedge.github.io/edgevr/posts/Super-Duper-Secure-Mode/). 4 August 2021.
22. "Deno 2 Arrives with Long-Term Support, Node.js Compatibility" (https://thenewstack.io/deno-2-arrives-with-long-term-support-node-js-compatibility/). *The New Stack*. 19 June 2024. Retrieved 25 August 2025.
23. Camden, Raymond K. (2015). *Apache Cordova in Action*. Manning. pp. 2–15. ISBN 9781617292392.
24. "Electron Internals: Using Node as a Library" (https://electronjs.org/blog/electron-internals-using-node-as-a-library). *electronjs.org*. 8 August 2016. Archived (https://web.archive.org/web/20171109205326/https://electronjs.org/blog/electron-internals-using-node-as-a-library) from the original on 9 November 2017. Retrieved 3 July 2020.
25. "Mozilla's Asm.js Gets Another Step Closer to Native Performance" (https://techcrunch.com/2013/12/21/mozillas-asm-js-gets-another-step-closer-to-native-performance/). 21 December 2013.
26. "Dev.Opera — 300 Million Users and Move to WebKit" (https://web.archive.org/web/20140628054037/https://dev.opera.com/blog/300-million-users-and-move-to-webkit/). Archived from the original (https://dev.opera.com/blog/300-million-users-and-move-to-webkit/) on 28 June 2014.
27. *New Microsoft Edge to replace Microsoft Edge Legacy with April's Windows 10 Update Tuesday release* (https://techcommunity.microsoft.com/t5/microsoft-365-blog/new-microsoft-edge-to-replace-microsoft-edge-legacy-with-april-s/ba-p/2114224), Microsoft, 12 February 2021, archived (https://web.archive.org/web/20210311213652/https://techcommunity.microsoft.com/t5/microsoft-365-blog/new-microsoft-edge-to-replace-microsoft-edge-legacy-with-april-s/ba-p/2114224) from the original on 11 March 2021, retrieved 12 March 2021
28. "JEP 335: Deprecate the Nashorn JavaScript Engine for Removal" (https://openjdk.org/jeps/335).
29. "JavaScriptCore – WebKit" (https://trac.webkit.org/wiki/JavaScriptCore). *trac.webkit.org*.
30. "Context Threading: A flexible and efficient dispatch technique for virtual machine interpreters" (https://web.archive.org/web/20250514082037/https://webdocs.cs.ualberta.ca/~amaral/cascon/CDP05/slides/CDP05-berndl.pdf) (PDF). Archived from the original (https://webdocs.cs.ualberta.ca/~amaral/cascon/CDP05/slides/CDP05-berndl.pdf) (PDF) on 14 May 2025.
31. "Introducing SquirrelFish Extreme" (https://webkit.org/blog/214/introducing-squirrelfish-extreme/). 18 September 2008.
32. "Speculation in JavaScriptCore" (https://webkit.org/blog/10308/speculation-in-javascriptcore/). 29 July 2020.
33. "Changeset 94559 – WebKit" (https://trac.webkit.org/changeset/94559/webkit).
34. "Introducing the WebKit FTL JIT" (https://webkit.org/blog/3362/introducing-the-webkit-ftl-jit/). 13 May 2014.
35. "Introducing the B3 JIT Compiler" (https://webkit.org/blog/5852/introducing-the-b3-jit-compiler/). 15 February 2016.
36. "Targeting Edge vs. Legacy Engines in JsRT APIs" (https://msdn.microsoft.com/en-us/library/dn903710(v=vs.94).aspx). Retrieved 10 September 2015.
37. "LibJS JavaScript engine" (https://libjs.dev/). *libjs.dev*. Retrieved 30 June 2023.

38. "LibJS: Start fleshing out a bytecode for the JavaScript engine :^) · LadybirdBrowser/Ladybird@69dddd4" (https://github.com/LadybirdBrowser/ladybird/commit/69dddd4ef58e51fb0db881b3b6801492b5f86285). *GitHub*.
39. "LibJS: Rip out the AST interpreter :^) · LadybirdBrowser/ladybird@2eaa528" (https://github.com/LadybirdBrowser/ladybird/commit/2eaa528a0ea606b2be9a1868ba711b1987409d70). *GitHub*. Retrieved 10 July 2024.
40. "Meet Ladybird browser, a promising cross-platform web browser - gHacks Tech News" (https://www.ghacks.net/2023/06/20/meet-ladybird-browser-a-promising-cross-platform-web-browser/). 20 June 2023.
41. "JavaScript technologies overview - JavaScript | MDN" (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/JavaScript_technologies_overview). 8 July 2025.
42. "Rhino History" (https://web.archive.org/web/20201022021252/https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino/History). Mozilla Foundation. Archived from the original (https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino/History) on 22 October 2020. Retrieved 20 March 2008.
43. "JEP 174: Nashorn JavaScript Engine" (https://openjdk.org/jeps/174).
44. "GitHub · Where software is built" (https://github.com/mozilla/rhino/milestones). *github.com*. Retrieved 29 December 2025.
45. "Oracle Nashorn: A Next-Generation JavaScript Engine for the JVM" (https://www.oracle.com/technetwork/articles/java/jf14-nashorn-2126515.html). *www.oracle.com*.
46. Lagergren, Marcus. "Nashorn War Stories (from a battle scarred veteran of invokedynamic)" (https://www.oracle.com/technetwork/java/jvmls2013lager-2014150.pdf) (PDF). *Oracle Corporation*.
47. "JEP 372: Remove the Nashorn JavaScript Engine" (https://openjdk.org/jeps/372).
48. "Using Nashorn with different Java versions" (https://github.com/szegedi/nashorn/wiki/Using-Nashorn-with-different-Java-versions). *GitHub*. Retrieved 23 January 2025.
49. "Truffle: A Self-Optimizing Runtime System" (http://wiki.jvmlangsummit.com/images/b/b7/2012-07-31_TruffleJVMSummit2012_Presentation.pdf) (PDF). Oracle Corporation. Retrieved 23 August 2013.
50. Wimmer, Christian; Würthinger, Thomas (2012). "Truffle: A self-optimizing runtime system" (https://dl.acm.org/doi/abs/10.1145/2384716.2384723). *Proceedings of the 3rd annual conference on Systems, programming, and applications: Software for humanity*. pp. 13–14. doi:10.1145/2384716.2384723 (https://doi.org/10.1145%2F2384716.2384723). ISBN 978-1-4503-1563-0.
51. Seaton, Chris (20 September 2016). "Faster Ruby and JavaScript with GraalVM" (https://chrisseaton.com/truffleruby/javaone16/faster-ruby-javascript-graalvm.pdf) (PDF). *chrisseaton.com*.
52. "Using Hermes" (https://reactnative.dev/docs/hermes). Facebook. Retrieved 9 April 2021.
53. Wen, Elliott; Zhou, Jiaxiang; Luo, Xiapu; Russello, Giovanni; Dietrich, Jens (2024). "Keep Me Updated: An Empirical Study on Embedded JavaScript Engines in Android Apps" (https://doi.org/10.1145/3643991.3644901). *MSR '24: Proceedings of the 21st International Conference on Mining Software Repositories*. Lisbon, Portugal: ACM. pp. 361–372. doi:10.1145/3643991.3644901 (https://doi.org/10.1145%2F3643991.3644901).
54. Xie, Juanying; Kaiser, M. Shamim; Rathore, Vijay Singh (2023). *Information and Communication Technology for Competitive Strategies (ICTCS 2022): Intelligent Strategies for ICT* (https://books.google.com/books?id=5hW_EAAAQBAJ&dq=hermes+javascript+engine&pg=PA562). Springer Nature Singapore. p. 562. ISBN 9789811993046.
55. "React Native 0.82 - A New Era · React Native" (https://reactnative.dev/blog/2025/10/08/react-native-0.82). *reactnative.dev*. 8 October 2025. Retrieved 29 December 2025.
56. "Duktape" (http://www.duktape.org/). Retrieved 21 September 2013.

57. Meng Wu; Weixin Dong; Qiang Zhao; Zhizhong Pan; Baojian Hua (2023). "An Empirical Study of Lightweight JavaScript Engines". *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security Companion (QRS-C)*. IEEE. pp. 413–422. doi:10.1109/QRS-C60940.2023.00103 (https://doi.org/10.1109%2FQRS-C60940.2023.00103). ISBN 979-8-3503-5939-8.
58. Minsu Kim; Hyuk-Jin Jeong; Soo-Mook Moon (2016). *Small Footprint JavaScript Engine* (https://link.springer.com/chapter/10.1007/978-3-319-42304-3_6). Springer. pp. 103–116. doi:10.1007/978-3-319-42304-3_6 (https://doi.org/10.1007%2F978-3-319-42304-3_6).
59. Grunert, Kai (2020). "Overview of JavaScript Engines for Resource-Constrained Microcontrollers". *2020 5th International Conference on Smart and Sustainable Technologies (SpliTech)*. pp. 1–7. doi:10.23919/SpliTech49282.2020.9243749 (https://doi.org/10.23919%2FSpliTech49282.2020.9243749). ISBN 978-953-290-105-4.
60. "moddable/documentation/xs/XS Conformance.md at public" (https://github.com/Moddable-OpenSource/moddable/blob/public/documentation/xs/XS%20Conformance.md#results). *GitHub*.
61. "Apps for IoT" (https://web.archive.org/web/20181228063939/http://www.moddable.com/faq.php). Archived from the original (https://www.moddable.com/faq.php#what-is-xs) on 28 December 2018.
62. Hoddie, Peter; Prader, Lizzie (2020). *IoT Development for ESP32 and ESP8266 with JavaScript: A Practical Guide to XS and the Moddable SDK*. Apress. p. 593. ISBN 9781484250709.
63. "Moddable | JavaScript for Embedded" (https://www.moddable.com/XS7-TC-39). *www.moddable.com*.
64. "MuJS" (http://mujs.com/). Retrieved 22 September 2014.
65. "QuickJS Javascript Engine" (https://bellard.org/quickjs/).
66. Tadashi Saito; Hideya Iwasaki (2025). "Integrating Static Optimization and Dynamic Nature in JavaScript". *GPCE '25: Proceedings of the 24th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*. ACM. pp. 41–53. doi:10.1145/3742876.3742877 (https://doi.org/10.1145%2F3742876.3742877).
67. "V4 - Qt Wiki" (https://wiki.qt.io/V4).
68. Tandel, Mrunali; Joshi, Uday; Golhani, Amit (2016). "Literature Review: Scripting in Qt for Scripting Engine Development" (https://www.ijcsit.com/docs/Volume%207/vol7issue3/ijcsit2016070347.pdf) (PDF). *International Journal of Computer Science and Information Technologies*. **7** (3): 1347–1350. Retrieved 20 September 2025.
69. Strakhov, Pavel Vladimirovich (2018). *Game Programming using Qt 5 Beginner's Guide: Create amazing games with Qt 5, C++, and Qt Quick* (https://books.google.com/books?id=kPBZDwAAQBAJ) (2nd ed.). Packt Publishing Ltd. ISBN 9781788390651. Retrieved 21 September 2025.
70. "Plasma Desktop scripting | Developer" (https://develop.kde.org/docs/plasma/scripting/). *develop.kde.org*. Retrieved 29 December 2025.
71. "Qt-5-Beta - Qt Wiki" (https://wiki.qt.io/Qt-5-Beta).
72. "New Features in Qt 5.2 - Qt Wiki" (https://wiki.qt.io/New_Features_in_Qt_5.2).
73. Pannu, Mandeep; Kay, Iain; Gill, Bod (17–19 October 2019). "Inherited Vulnerabilities: JavaScript Engine V4 Secure Coding Compared to Google's V8". *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. Vancouver, BC, Canada: IEEE. doi:10.1109/IEMCON.2019.8936228 (https://doi.org/10.1109%2FIEMCON.2019.8936228).
74. "JavaScript Host Environment | Qt QML | Qt 6.10.0" (https://doc.qt.io/qt-6/qtqml-javascript-hostenvironment.html).
75. "V4 - Qt Wiki" (https://wiki.qt.io/V4). *wiki.qt.io*. Retrieved 24 April 2021.

76. "Qt-contributors-summit-2013-QML-engine - Qt Wiki" (https://wiki.qt.io/Qt-contributors-summit-2013-QML-engine).
77. "Releases · Samsung/Escargot" (https://github.com/Samsung/escargot/releases). *GitHub*.
78. Hyukwoo Park; Seonghyun Kim (2024). "Tail Call Optimization Tailored for Native Stack Utilization in JavaScript Runtimes" (https://doi.org/10.1109%2FACCESS.2024.3441750). *IEEE Access*. **12**: 111801–111817. Bibcode:2024IEEEA..12k1801P (https://ui.adsabs.harvard.edu/abs/2024IEEEA..12k1801P). doi:10.1109/ACCESS.2024.3441750 (https://doi.org/10.1109%2FACCESS.2024.3441750).
79. "JavaScript Console" (https://geth.ethereum.org/docs/interacting-with-geth/javascript-console).
80. "Let's build a JavaScript Engine" (https://2019.jsconf.eu/jason-williams/lets-build-a-javascript-engine.html). *2019.jsconf.eu*. Retrieved 23 March 2022.
81. "GitHub - boa-dev/boa: Boa is an embeddable and experimental Javascript engine written in Rust" (https://github.com/boa-dev/boa). *github.com*. Retrieved 8 April 2022.
82. "Kiesel JavaScript Engine" (https://kiesel.dev/). *kiesel.dev*. Retrieved 30 September 2025.
83. kiesel-js. "kiesel" (https://codeberg.org/kiesel-js/kiesel). *Codeberg.org*. Retrieved 30 September 2025.
84. "Test262: ECMAScript Test Suite" (https://test262.fyi/). Retrieved 30 September 2025.
85. "Temporal_rs is here! The datetime library powering Temporal in Boa, Kiesel, and V8" (https://boajs.dev/blog/2025/09/24/temporal-release). 24 September 2025. Retrieved 30 September 2025.
86. "Netscape Navigator 2.0b1 for Windows" (https://web.archive.org/web/19970614001024/http://home.netscape.com/eng/mozilla/2.0/relnotes/windows-2.0b1.html). Archived from the original (http://home.netscape.com/eng/mozilla/2.0/relnotes/windows-2.0b1.html) on 14 June 1997.
87. "Netscape Communicator 3.0.2 Source Tree" (https://archive.org/details/netscape-communicator-3-0-2-source). *Archive.org*. 28 October 2011. Retrieved 23 September 2025.
88. Wirfs-Brock, Allen; Eich, Brendan (2020). "JavaScript: the first 20 years" (https://doi.org/10.1145%2F3386327). *Proceedings of the ACM on Programming Languages*. **4** (HOPL): 1–189. doi:10.1145/3386327 (https://doi.org/10.1145%2F3386327).
89. "News from the land of Konquerors - KDE Blogs" (https://blogs.kde.org/2008/05/22/news-land-konquerors/). *blogs.kde.org*. Retrieved 29 December 2025.
90. Brockmeier, Joe (28 February 2024). "The KDE desktop gets an overhaul with Plasma 6" (https://lwn.net/Articles/963851/).
91. Ratanaworabhan, Paruj; Livshits, Benjamin; Zorn, Benjamin G. "JSMeter: Comparing the Behavior of JavaScript Benchmarks with Real Web Applications" (https://www.usenix.org/legacy/event/webapps10/tech/full_papers/Ratanaworabhan.pdf) (PDF). *USENIX*.
92. "Introducing JScript .NET | Microsoft Learn" (http://msdn.microsoft.com/ms974588.aspx). *msdn.microsoft.com*. Retrieved 29 December 2025.
93. "JScript Version Information" (https://learn.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2010/2z6exc9e(v=vs.100)). *learn.microsoft.com*.
94. "JScript Reference" (https://learn.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2010/x85xxsf4(v=vs.100)). *learn.microsoft.com*.
95. "Microsoft JScript Features - Non-ECMA (Visual Studio - JScript)" (https://learn.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2010/894hfyb4(v=vs.100)). *learn.microsoft.com*.
96. "JScript Language Tour | Microsoft Learn" (https://learn.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2010/t895bwkh(v=vs.100)). *learn.microsoft.com*. Retrieved 29 December 2025.

97. *Frequently Asked Questions* (https://web.archive.org/web/20100322193213/http://ie.microsoft.com/testdrive/info/FrequentlyAskedQuestions/Default.html), Microsoft, 13 March 2010, archived from the original (http://ie.microsoft.com/testdrive/info/FrequentlyAskedQuestions/Default.html) on 22 March 2010, retrieved 18 March 2010
98. "JScript9Legacy scripting engine now enabled by default - Windows IT Pro Blog" (https://techcommunity.microsoft.com/blog/windows-itpro-blog/jscript9legacy-scripting-engine-now-enabled-by-default/4431326). *techcommunity.microsoft.com*. Retrieved 29 December 2025.
99. "Windows 11 22H2 has a different way of using the JS engine? - Microsoft Q&A" (https://learn.microsoft.com/en-us/answers/questions/1071647/windows-11-22h2-has-a-different-way-of-using-the-j). *learn.microsoft.com*.
100. "Opera's archived version history 1-12" (https://help.opera.com/en/operas-archived-history/).
101. "Opera Desktop Team's Blog | Opera" (https://web.archive.org/web/20060303160759/http://my.opera.com/desktopteam/blog/). Archived from the original (http://my.opera.com/desktopteam/blog/) on 3 March 2006.
102. "Epimetheus" (https://www-archive.mozilla.org/js/language/epimetheus).
103. "Release 2.1pre17" (https://web.archive.org/web/20050311202407/http://links.twibright.com:80/download/ChangeLog). Archived from the original (http://links.twibright.com/download/ChangeLog) on 11 March 2005.
104. "JavaScript Engines and Products" (https://joehni.github.io/JsUnit/products.html).
105. Clark, Mitchell (18 August 2022). "The James Webb Space Telescope runs JavaScript, apparently" (https://www.theverge.com/2022/8/18/23206110/james-webb-space-telescope-javascript-jwst-instrument-control). *The Verge*. Retrieved 2 September 2022.

Retrieved from "https://en.wikipedia.org/w/index.php?title=List_of_JavaScript_engines&oldid=1346949836"