ChatGPT

Emacs has a storied legacy as one of the oldest and most powerful text editing platforms. First released in 1976, GNU Emacs has continuously evolved while preserving backward compatibility [1]. It is far more than a simple editor: at its core it is an extensible Lisp interpreter that users can script to perform virtually any task. As one observer notes, people "use Emacs for the power of Elisp," enabling Emacs to control processes, manage files, configure the system, and filter command output in arbitrary ways [2]. Over the decades Emacs has grown into a comprehensive environment: it supports hundreds of major modes (for programming languages, markup, etc.), integrated tools (compilers, debuggers, project management), note-taking (Org mode), email/calendar, and an enormous package ecosystem. In short, Emacs has long been described as an "operating system around text," offering an all-in-one IDE and workflow platform for developers and writers alike [1] [2].

However, Emacs's age and architecture also bring limitations in modern contexts. Users frequently observe that Emacs's core C implementation – much of which dates from the 1980s–90s – exhibits performance lags on today's hardware. Even on fast machines the UI can "lock up for a number of seconds" under load, and Emacs is often slower on macOS/Windows than on Linux [3]. Its design is essentially single-threaded (with only cooperative Lisp threads added in recent versions), so long-running tasks can stall the editor. These issues are compounded by Emacs's steep learning curve. A recent analysis notes that raw Emacs "seems much more complex than solutions like VSCode, Sublime Text, [or] IntelliJ," and that newer editors emphasize ease of entry for beginners [4]. In sum, critics point to persistent slowness, a dated, monolithic core, and the high barrier to entry as serious obstacles to broader adoption [3] [4].

These concerns have fueled community discussion about Emacs's future. Some commentators argue that Emacs must modernize or risk stagnation. For example, one user bluntly observes that the fact Emacs "isn't modern" suggests it "hasn't kept up with the times" – persistent bugs and performance issues linger – implying that "serious changes" to the core may be needed [3]. At the same time, the success of community-driven variants (such as Doom Emacs and Spacemacs) shows there is demand for more modern, user-friendly Emacs configurations. As another advocate notes, these projects "brought forth a ton of new users" and indicate that "visionary thinking" is needed to grow Emacs's user base [5]. In short, both users and developers have voiced a desire to rethink parts of Emacs (its defaults, UI, or even core implementation) to improve performance and ease of use, while still preserving its extensible nature.

This situation motivates exploring a complete reimplementation of the Emacs core using a modern language like Rust. Proponents point out that Rust's features align well with Emacs's ideals. Rust supports incremental porting of existing C code – one can replace Emacs's regular-expression engine or individual functions piece by piece – so the editor remains fully functional at each step [6]. More importantly, Rust's ecosystem and tooling offer concrete benefits: its compiler and type system catch many bugs early, built-in formatting and testing tools improve code quality, and a rich "crate" (library) ecosystem (e.g. a high-performance regex crate) can boost functionality and speed [7]. Crucially, Rust provides memory safety without a garbage-collection pause and "fearless" concurrency: its ownership model eliminates data races at compile time while delivering C-like performance [8]. These modern paradigms would let an Emacs reimplementation offer true multi-threading and robust stability without sacrificing the traditional Elisp-based extensibility. In summary, a Rust-based Emacs core promises to uphold the editor's extensibility and ecosystem while bringing the advantages of safe, concurrent, and maintainable systems programming [7] [8].

Sources: Authoritative blog posts and community discussions on Emacs and its alternatives were consulted. These include technical essays, EmacsConf talk summaries, and user/developer commentary [2] [5] [4] [6], which document Emacs's capabilities, limitations, and the rationale for exploring new implementations.

---

[1] [4] What can Emacs give me in 2021? A response to the Mother of All Emacs Papers | Tech.ToryAnderson.com
https://tech.toryanderson.com/2021/05/29/what-can-emacs-give-me-in-2021-a-response-to-the-mother-of-all-emacs-papers/

[2] LWN.net Weekly Edition for January 9, 2025 [LWN.net]
https://lwn.net/Articles/1003479/

[3] [5] Modern Emacs: Redux - (think)
https://batsov.com/articles/2022/06/09/modern-emacs-redux/

[6] [7] Announcing Remacs: Porting Emacs to Rust – Wilfred Hughes::Blog
http://www.wilfred.me.uk/blog/2017/01/11/announcing-remacs-porting-emacs-to-rust/

[8] A vision of a multi-threaded Emacs • Core Dumped
https://coredumped.dev/2022/05/19/a-vision-of-a-multi-threaded-emacs/