

Let Them Write RFCs

August 11, 2026

vibecoding

thoughts

Or: why I spent hundreds of millions of tokens to write four versions of the same markdown file.

I'm working on a complex project that's *just* beyond what I know how to build, which is pretty fun! But also hard, especially when I try to speed things up by using LLMs.¹ As a result:

1. Sometimes I don't understand the problem well enough.
2. Sometimes I don't *know that*.
3. Sometimes I'm easily fooled by ideas that sound good due to 1 & 2.

Some would argue that one shouldn't be going around working on things they don't understand armed with nothing but good vibes.



That's BS. I *never* had a good understanding of something when I *started* working on it. Working on the thing is what makes you understand the thing!

But these problems *are* amplified by LLM usage: friction is removed, so you can *continue* working on the thing without understanding it, or even noticing that you don't, which of course makes it easier to be fooled - resulting in a terrible cycle of LLM psychosis.

Combine this with the tendency of the current generation of LLMs to tenaciously tunnel toward a target, and you get why people are losing their minds.



For this project, there were two things that helped. The first was throwing away two “complete” implementations of the codebase. Nothing will teach you more about a problem than a bad first draft, and building bad drafts has become (too?) economical these days. But even with these rough drafts it was clear that things could go wrong in both expected and unexpected ways.

So while I knew *more* about the problem, I also had a bad feeling there where *many more* unknowns ahead.

Optimizing for Discussing a Problem, Not a Solution

When you ask an LLM² to solve a problem, even in plan mode, it will optimize for “how to solve this problem” (d’oh), and even if there are a few follow-up questions, they serve the ultimate goal of getting the problem solved - usually by making a beeline for the most immediate solution possible. Do it enough times, and you’ll be drowning in slop.

You can navigate towards the right solution *if* you know you’re looking for.

But do you do when you don’t? Especially when you *should* suspect that you don’t know enough about the problem? We can’t keep throwing away *all* the code!

I started asking, sometimes after intensive rambling, “... this is the vague problem we have right now. Write a markdown file called \$PROBLEM_RFC.md about this problem and how to solve it.

Keep the prose³ short, have good code examples.”

By asking a model to produce an RFC, I’m optimizing for understanding and explanation, which makes problems more visible. An RFC is unlikely to be good enough on the first try, so I’ll iterate and revise (V1.md, V2.md, etc.), consulting documentation and looking things up, until I understand the problem well enough and think the ideas are sound, which is where most of my learning happens.

I found it both more useful in that it produces better solutions and code, and more enjoyable for me to work on: instead of reading slop code, iterating on an RFC forces me to explain what I don’t understand and to focus on grasping the hardest parts of the problem.

Why RFCs

An RFC⁴ needs to get a few things into my brain (💬➡️🧠):

1. What *exactly* is the problem?
2. How bad or important is it?
3. Will this solution fix the problem?
4. How costly will the solution be, vs. the alternatives?

If I'm not sold on any of these, especially by the definition of the problem, I'll just ask. "Explain this part of the problem", "Why is it this and not that?", "What about ...", "Didn't we already do this in ..." - again and again until I'm actually convinced.

RFCs have a high standard for both prose and explanatory code, forcing more work on the LLM than on the reader, and they make problems-with-the-problem easier to spot than they are in an implementation.

Throw Away Code Anyway

The real test of an RFC, and of my understanding, is the final code: sometimes an RFC will sound great on paper but the resulting code is still, unfortunately, garbage. Worse, I might not understand the code well enough to tell whether or not it's garbage.

When that happens, I can say "this and that went wrong, read the diff, read the RFC, write a new version that ...", or "this code is garbage, here's the RFC, what went wrong? Write a markdown file with a review". Then, I'll throw away the changes and start again.

vs. Plans

Usually, an RFC is not an actual implementation plan (like a plan in Cursor or plan mode in Claude Code): the focus is on the problem, the general shape of the solution, and the alternatives.

For a plan, the details are everything: I mostly care that all the things I asked for are accounted for, and I should already know where and how the new code will fit into the existing codebase, so it's a way to spend more tokens on high-level work before kicking off the grunt work. How many times have you asked to improve the explanation of something that's mentioned in a plan?

Summary

So this is why I spent hundreds of millions of tokens to write four versions of the same markdown file: I wanted the LLM to produce an artifact that makes it easier for *me* to both understand the problem and identify what I don't (yet) understand about it, similar - I think - to what I would get from working on the problem "by hand".

Give it a try! I found it both more useful and more fun: less time reading slop, more thinking and learning about hard problems.

And, finally:

Read the Code

Of course I read the f*cking code.

```
2306 fn column_context(side: &str) -> &str {
2307     if side == "input" { "input" } else { side }
```

I don't read every single line, but I go over each file and most functions and structs.

Yes, it's hard in a 5k-line PR. No, there is no alternative, at least for now.

1. This project is an example of when talking to the tool is *draining* but totally worth it. 
2. Even a very smart LLM. Even a person! 
3. I know LLM prose can be tiring—believe me. But LLM code can be much more tiring, so there's a tradeoff. However, when asking for *short* prose, you'll get less cruft and fewer Claude-isms (I guess Sol also has Claude-isms? Sol-isms? Gemini-isms?). 
4. I sometimes ask for the *style* of Rust's RFC format (for example, Rust's Match Ergonomics RFC) but the actual structure should be flexible enough to fit the problem. 

© 2026

Made with ❤️ using Gokarna

Teapot icon by Icons8