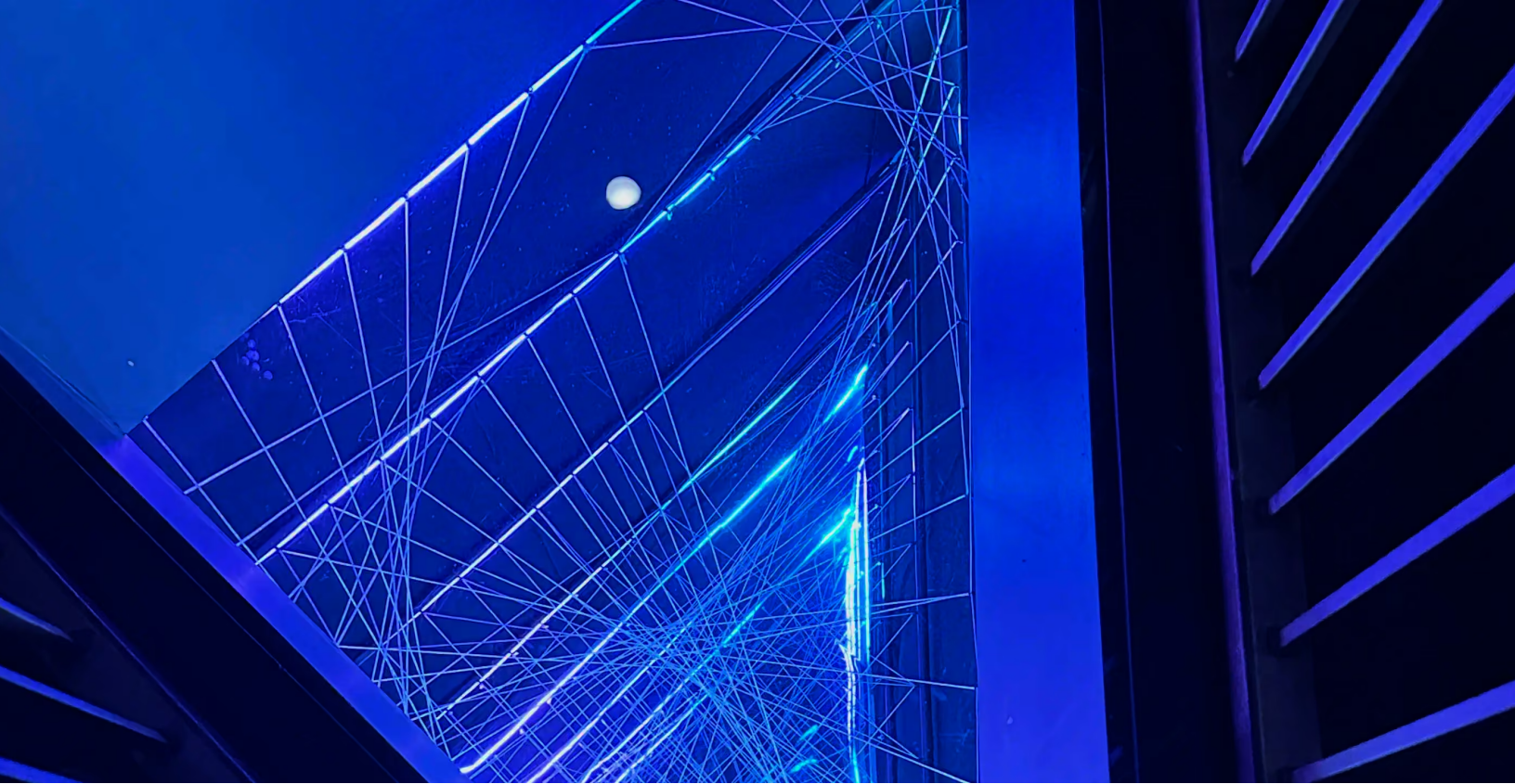




mckayla.blog / posts /

Yap: a particular kind of slop

yap (verb)

to talk in a shrill insistent way

Comments

Writing code comments has always been hard, because communicating is hard. It's hard to not assume context. It's hard to use the right terminology consistently. It's hard to even put comments in the right spots to begin with. It generally takes me several iterations of rewording a comment to get something I'm happy with. I'll write something lousey that takes 5 times as long to read as it should, and then read it. Then I'll remove a couple sentences, tweak what remains and do it again.

There is no compiler to statically analyze and verify your comments. A comment can be wildly inaccurate and yet the code will run in blissful ignorance. When you change a type, or a function signature, or a variable name you *must* update the rest of the code base accordingly. The same is not true of comments. Whether wrong from the beginning, or only made wrong over time, there are *tons* of wrong and bad comments out there.

The State of Things

LLMs around the world are writing mountains of code. I am responsible for reviewing large amounts of this code in exchange for gainful employment. I have some expensive habits (paying my mortgage and eating food) that necessitate this. I'm sure many others do as well.

A lot of people think the solution to this is to have LLMs review the code, so that humans don't have to spend as much time on it. I remain unconvinced that this alone is sufficient.

- If one robot thought it was a good idea in the first place, why am I supposed to be confident that robot #2 will have better taste?
- LLMs still don't follow instructions consistently. They're trained on humans, and act like humans in ways that continue to surprise me, including acting like a smartass, acting like they know better than you, and forgetting about important things constantly.
- ...I could go on, but I want to get to the point.

So I am still in the loop, I don't think ✨ is a sufficient solution to the problem, but I am an engineer and like to optimize. How can I optimize reviewing LLM code?

Communication levels

LLMs use words that we recognize, in sentences that we can understand. That doesn't mean that they're necessarily good at communicating, because there are different levels of communication. They aren't *skill* levels either, but rather levels of intent. For the sake of brevity, I'm going to simplify to just two levels here:

1. Talking: Actual communication. Words being said for the purpose of making something understood by someone else.

2. Yapping: Words for the sake of words. Words that are not actually meant to be understood or heard, except perhaps by the person (or computer) saying them.

I think we can all agree that LLMs do a *lot* of yapping. It is fundamentally how "thinking" or "reasoning" models work: they monologue at themselves to fill up the context window with extra details to make future token inference more accurate.

Make no mistake, humans do it too. When you're mumbling to yourself in your head while navigating a codebase, that's yapping. Yapping is a common way for humans to process and understand things as well. Not all brains are equal, but if you have an internal monologue you are an S-tier yapper, even if it's only ever directed at yourself.

Yap

yap (noun)

the residue left behind from yapping

The current problem is that LLMs are really bad at keeping the yapping to themselves. If you spend any time reading LLM generated code you know what I'm talking about, even if you haven't put a name to it yet: comments overflowing with words and entirely devoid of actual substance. I want to give you a name for it. The 30 line comment above the dead-simple type definition? Yap. The 200 line comment above the large function definition, with barely any comments actually in the body. *Misplaced* yap. The 50 line comment at the top of the file trying to explain...something, presumably? Yap.

I got tired of writing out dozens of thoughtful and polite review comments a day to tell robots that their code comments weren't up to snuff. Now I just say "yap" and they get what I mean.

"This comment sounds like it's just paraphrasing your prompt, and not actually describing what this part of the code does." becomes "yap". "I think this comment could easily be rephrased to use half as many words while also being more easily understood by future readers." becomes "yap". "This comment is mostly describing why we're not doing it the old way anymore. The thing I'm actually interested in is what we're doing *now* and why it needs to be done this way."

becomes "yap". "This comment should be broken up and the details within it should be moved closer to the actual relevant code, rather than all clumped together like this." becomes "yap".

"yap" becomes shorthand for "Step back and think about what value this comment actually provides. Think about how this comment could be changed to provide more value. What parts of it are actually non-obvious? What parts of it aren't really helpful to future readers? Is this the right spot for this comment? Does this actually even need a comment at all?"

So to answer my earlier question, one way we can optimize reviewing LLM code is by coming up with shorthand to discuss the most common problems that plague LLM code. We can notice the patterns, give them names, and document them. For once I actually hope all of the AI companies *do* steal this blog post and add it to their training data, because that might mean that my "yap" callouts would become more effective; even more so if others start to use the term and it becomes part of the software engineering lexicon.

Discussion

Thoughts? Comments? Questions? Reach out via [email](#). I'd love to chat!

♥ [McKayla](#) · [RSS](#)