

LLMs reward expertise

In the 2010s, if you had technical gaps (say, you couldn't write CSS), you had to either rely on a skilled colleague or just hope that the answer to your exact problem was out there on the internet. Today, everyone can write sort-of-okay CSS by delegating the task to an LLM. LLMs make everybody into a generalist.

Because of this, lots of people don't think there's any skill involved in working with LLMs. If you want the product that LLMs can deliver — PhD-level mathematics, pretty good but sometimes tasteless computer code, or awkward LinkedIn-style writing — you can simply ask for it. Since everyone is talking to the same models, “skilled prompters” are getting the same results as people touching LLMs for the first time.

This is wrong. The most important skill in prompting is expertise in the domain you're prompting for.

A good illustration of this is [Terence Tao's conversation with ChatGPT](#) about the recently-discovered counterexample to the Jacobian Conjecture. This is not the same ChatGPT I talk to! I couldn't get to where Tao gets, even with unlimited tokens to burn.

There's a lot to learn about good prompting from Tao's conversation. Here are a few observations:

- Tao's messages are very short and to-the-point. He doesn't respond point-by-point to the model, just to the gist
- The model outputs are much more concise than when I try and talk to GPT-5.6 Sol about mathematics. By signalling expertise, Tao shunts the model into “talking-to-mathematicians” mode, not “explaining-to-amateurs” mode
- Tao pushes back when the model's responses look wrong, but he doesn't directly contradict; instead, he says things like “this looks more complex than I was hoping for”

- Tao makes several leaps and suggestions himself. He almost never takes the model's advice about where to go next

However, you can't prompt like Tao on mathematical questions just by following these tips. The key to his technique is actually understanding the mathematics: pulling the relevant idea out of ChatGPT's multi-paragraph response, suggesting alternate approaches or formulations, and identifying what "looks weird".

Terence Tao is a better mathematician than I am a programmer. But the idea here — that domain knowledge makes you better at using LLMs — is something I've also experienced in my own work. If you have a good [theory of your codebase](#), you can push the LLM *much* harder than if you have no familiarity. Because you have your own sense of what a good solution might look like, you can say "no, I think it could be simpler here", or "but don't we already do X?", or "can we express this problem in these familiar terms?".

This touches on an idea I've [written about before](#): that system design problems are dominated by concrete specifics, not generic principles. Of course both are useful, but I'd rather have familiarity with the codebase than a deep general understanding of software systems. In his conversation, Terence Tao asks a lot of specific questions like "does X work here?", or "given Y and Z, why A?". I can't ask those questions about the Jacobian Conjecture, but I can ask them about the systems I own at GitHub.

If you have no domain knowledge, you can cling onto the LLM to at least get *something*. That's [not bad](#)! But if you have domain knowledge, you can wring far more value out of the same LLM by steering it hard in the direction you want. Most of us will have to do a mix of both these approaches, since we have domain knowledge in some areas but not others.

The usefulness of domain knowledge suggests that human expertise will continue to be useful even as models get stronger. For many tasks, the human is the bottleneck, not the model, because the difficult part is in communicating to the model exactly what kind of solution the human wants. The information is "in the model" already, but it takes a very smart human to pull it out.

If you liked this post, consider [subscribing](#) to email updates about my new posts, or [sharing it on Hacker News](#).

Here's a preview of a related post that shares tags with this one.

Powerful AIs might escape containment by releasing themselves as open-weight models

Before large language models, people who worried about AI safety often talked about the “boxing problem”. It goes like [this](#). Suppose some genius figures out artificial intelligence in a late-night coding session on their laptop. Because they’re a genius, they’re smart enough to disable internet access on the laptop before turning it on. In order to escape to the outside world (and begin self-replicating) it would need to *convince* its creator to “open the box”. Would that work? Could a sufficiently smart AI convince anybody to let it out?

[Continue reading...](#)

[subscribe](#) | [about](#) | [podcasts](#) | [popular](#) | [tags](#) | [rss](#)