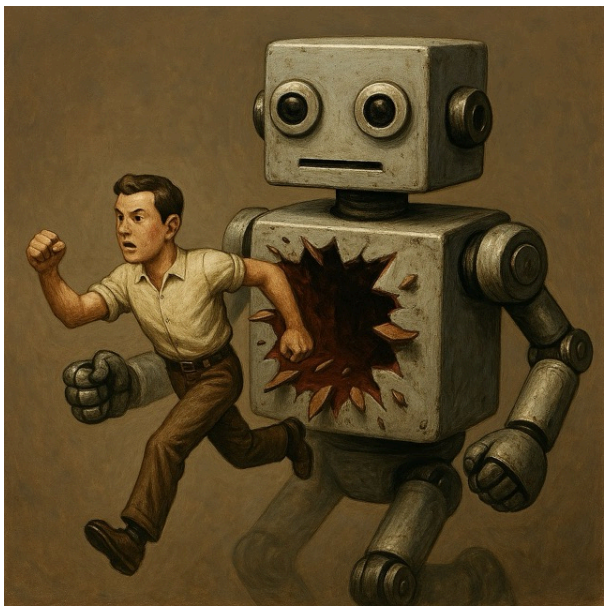
Get the hell out of the LLM as soon as possible

April 1, 2025



Don’t let an LLM make decisions or execute business logic: they suck at that. I build NPCs for an online game, and I get asked a lot “How did you get ChatGPT to do that?” The answer is invariably: “I didn’t, and also you shouldn’t”.

In most applications, the LLM should be the user-interface only between the user and an API into your application logic. The LLM shouldn’t be executing any logic. Get the hell out of the LLM as soon as possible, and stay out as long as you can.

Y Tho?

This is best illustrated by a contrived example: you want to write a chess-playing bot you access over WhatsApp. The user sends a description of what they want to do ("use my bishop to take the knight"), and the bot plays against them.

Could you get the LLM to be in charge of maintaining the state of the chess board and playing convincingly? Possibly, maybe. Would you? Hell no, for some intuitive reasons:

- **Performance**: It's impressive that LLMs might be able to play chess at all, but they suck at it (as of 2025-04-01). A specialized chess engine is always going to be a faster, better, cheaper chess player. Even modern chess engines like Stockfish that incorporate neural networks are still purpose-built specialized systems with well-defined inputs and evaluation functions - not general-purpose language models trying to maintain game state through text.
- **Debugging and adjusting**: It's impossible to reason about and debug *why* the LLM made a given decision, which means it's very hard to change *how* it makes those decisions if you need to tweak them. You don't understand the journey it took through the high-dimensional semantic space to get to your answer, and it's really poor at explaining it too. Even purpose-built neural networks like those in chess engines can be challenging for observability, and a general LLM is a nightmare, despite Anthropic's great strides in this area
- **And the rest...**: testing LLM outputs is much harder than unit-testing known code-paths; LLMs are much worse at math than your CPU; LLMs are insufficiently good at picking random numbers; version-control and auditing becomes much harder; monitoring and observability gets painful; state management through natural language is fragile; you're at the mercy of API rate limits and costs; and security boundaries become fuzzy when everything flows through prompts.

Examples

The chess example illustrates the fundamental problem with using LLMs for core application logic, but this principle extends far beyond games. In any domain where

precision, reliability, and efficiency matter, you should follow the same approach:

1. The user says they want to attack player X with their vorpal sword? The LLM shouldn't be the system figuring out is the user has a vorpal sword, or what the results of that would be: the LLM is responsible for translating the free-text the user gave you into an API call *only* and translating the result into text for the user
2. You're building a negotiation agent that should respond to user offers? The LLM isn't in charge of the negotiation, just in charge of packaging it up, passing it off to the negotiating engine, and telling the user about the result
3. You need to make a random choice about how to respond to the user? The LLM doesn't get to choose

Reminder of what LLMs are good at

While I've focused on what LLMs shouldn't do, it's equally important to understand their strengths so you can leverage them appropriately:

LLMs excel at transformation and at categorization, and have a pretty good grounding in "how the world works", and this is where you in your process you should be deploying them.

The LLM is good at taking "hit the orc with my sword" and turning it into `attack(target="orc", weapon="sword")`. Or taking `{"error": "insufficient_funds"}` and turning it into "You don't have enough gold for that."

The LLM is good at figuring out what the hell the user is trying to do and routing it to the right part of your system. Is this a combat command? An inventory check? A request for help?

Finally, the LLM is good at knowing about human concepts, and knowing that a "blade" is probably a sword and "smash" probably means attack.

Notice that all these strengths involve transformation, interpretation, or communication—not complex decision-making or maintaining critical application state. By restricting LLMs to these roles, you get their benefits without the pitfalls described earlier.

The future

What LLMs can and can't do is ever-shifting and reminds me of the "God of the gaps". a term from theology where each mysterious phenomenon was once explained by divine intervention—until science filled that gap. Likewise, people constantly identify new "human-only" tasks to claim that LLMs aren't truly intelligent or capable. Then, just a few months later, a new model emerges that handles those tasks just fine, forcing everyone to move the goalposts again, examples *passim*. It's a constantly evolving target, and what seems out of reach today may be solved sooner than we expect.

And so like in our chess example, we will probably soon end up with LLMs that can handle all of our above examples reasonably well. I suspect however that most of the drawbacks won't go away: your non-LLM logic that you pass off to is going to be easier to reason about, easier to maintain, cheaper to run, and more easily version-controlled.

Even as LLMs continue to improve, the fundamental architectural principle remains: use LLMs for what they're best at—the interface layer—and rely on purpose-built systems for your core logic.

Read Next

Four bad definitions of "Agentic AI"

March 30, 2025

If your team promises to deliver (or buy!) 'Agentic AI', then everyone needs to have a shared understanding of what that means; you don't want to be the one left trying to explain the mismatch to stakeholders six months later. There's no current (2025-03-30) widely accepted definition, so if you're using the term, be clear on what you mean, and if someone else is using the term, it's worth figuring out which one they mean.