Why Your LLM App Feels Slow (And What You Can Do About It)

From the outside, a large language model can look like magic: you send a prompt, and a fluent answer appears. But once teams move past the first demo, they hit the same complaint from users and stakeholders: “Why does this thing feel so slow?”

Latency is not just an annoyance. In customer-facing workflows, a few extra seconds can mean higher abandonment rates, lower satisfaction scores, and less trust in your product. In internal tools, slow responses push people back to old habits and manual work.

This article breaks down the main reasons LLM-powered apps feel slow and the practical levers you can pull to improve perceived and actual performance.

Where latency really comes from

When a user sends a prompt to your application, several things happen before they see a single token on the screen:

1. Network and request overhead
   Your frontend has to send a request through your backend, which then calls the model API or your own inference server. Every hop adds a few milliseconds. If you are not careful with retries, timeouts, and region selection, these small delays stack up quickly.

2. Tokenization and context window size
   Models do not see words; they see tokens. A long prompt plus a long conversation history can easily add up to several thousand tokens. The more tokens the model has to read and generate, the more compute time you pay for. This is especially painful if you append entire chat histories on every request.

3. Model size and hardware
   Larger models usually produce better answers, but they also need more memory and more compute to run. If you are hosting a big model on underpowered hardware, or you are sharing GPUs across many tenants without good scheduling, responses will stall when the system is saturated.

4. Generation settings
   Your max_tokens setting, sampling strategy, and temperature all affect how long generation takes. If you always ask for long, detailed answers, the model will happily generate them – one token at a time.

Five levers to improve performance

The good news is that you do not need to "solve AI" to make your application feel faster. You can make targeted changes in a few areas.

1. Trim your prompts and histories

Treat your prompt like a scarce resource. Remove boilerplate text that does not change. Summarize long histories instead of resending every message. Use system prompts to set behavior once, rather than repeating instructions in user messages.

2. Use retrieval instead of dumping documents into the prompt

Instead of pasting entire documents into a chat, store them in a vector database and use retrieval to pull in only the most relevant chunks. This reduces token count and keeps responses focused while still grounding answers in your data.

3. Choose the right model for the job

You do not need your heaviest, most expensive model for every task. Route simple classification or extraction jobs to smaller models, and reserve your largest models for tasks that truly benefit from them. This can significantly reduce both latency and cost.

4. Stream tokens to improve perceived speed

If the API and your UI support it, stream tokens as they are generated. Users perceive a system as faster when they can see an answer forming, even if the total generation time is unchanged. A simple streaming UI often produces a big boost in satisfaction.

5. Cache what you can

Common prompts, system messages, and even entire responses can be cached. If you know that certain workflows generate repeatable queries, you can avoid recomputing the same answer. This is especially useful for documentation-style content and frequently asked questions.

Measure, then optimize

Before you change anything, measure your current latency end to end: from click to first token on the screen. Look at average, p95, and p99 values. Then make small, focused changes and watch how they affect both your metrics and your users' behavior.

LLM-powered apps will always involve some amount of waiting, but they do not have to feel sluggish. By understanding where the time actually goes – and by treating tokens, models, and histories as resources you can manage – you can deliver AI features that feel responsive enough to earn a permanent place in your users' workflows.