STRATA|AIT

LLM Web Application Stack
Requirements Based Conceptual Architecture

Context & Problem

Developing AI applications requires new systems amid evolving challenges and costs.

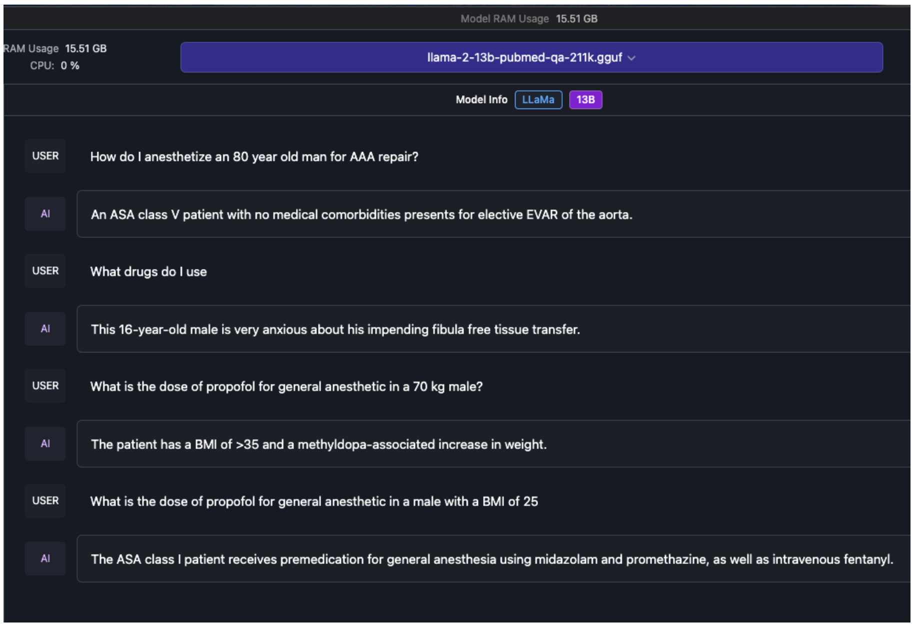
The traditional application stack for a dynamic website is primarily a database, an application server, and a cache layer (for response speed). To create applications with generative AI and language models, several new storage systems and services are required. This document attempts to clarify what is needed and why. While this explanation will peer into how engineering does this, consider any conceptual component box in this architecture to be replaceable, combined, split, etc. as the best understanding of AI powered apps rapidly evolves. In other words, this article will self destruct in 6 months — and many startup products with it!

Humans want to interact with an AI in natural language (or voice utterances), and receive human expert quality responses on topics requiring a lifetime of subject matter expertise. The problem is that language models mostly understand written language, and don't have usually have expertise in topics like anesthesia, petrochemical research, etc. In 3-6 years time, a private company might create a model with all human knowledge (incalculable trillions of parameters) but the barriers today are >$10 billion training cost, slow performance on responses, and high cost to serve the runtime inference. There is also public debate on copyright claims and the legality of derivative intellectual property, which we won't tread into now.

What if we do nothing?

Direct LLM interaction shows the need for context to make sense of inputs.

To understand the problem first hand, see what happens when interacting directly with an LLM absent any other application layers. Do this by downloading LM Studio for MacOS or Windows, loading up a compliant model sourced from HuggingFace, and asking it questions on your local machine. The thread below is a clinical practitioner asking a language model trained on PubMed (medical publications) for advice on an upcoming surgery:



Model RAM Usage 15.51 GB

RAM Usage 15.51 GB
CPU: 0 %

llama-2-13b-pubmed-qa-211k.gguf

Model Info LLaMa 13B

USER How do I anesthetize an 80 year old man for AAA repair?

AI An ASA class V patient with no medical comorbidities presents for elective EVAR of the aorta.

USER What drugs do I use

AI This 16-year-old male is very anxious about his impending fibula free tissue transfer.

USER What is the dose of propofol for general anesthetic in a 70 kg male?

AI The patient has a BMI of >35 and a methyldopa-associated increase in weight.

USER What is the dose of propofol for general anesthetic in a male with a BMI of 25

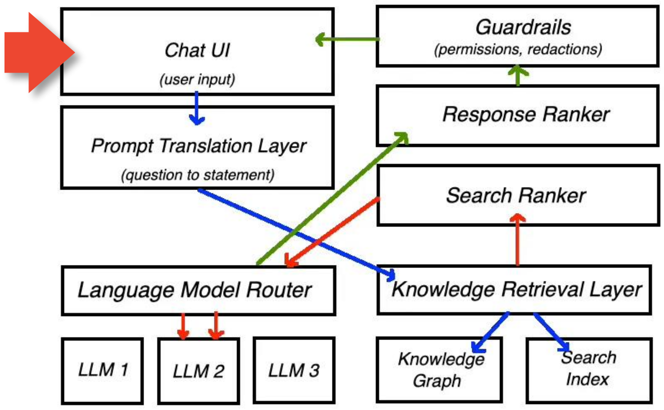
AI The ASA class I patient receives premedication for general anesthesia using midazolam and promethazine, as well as intravenous fentanyl.

This is beyond a hallucination, it's complete gibberish — sentence fragments spit out from documents it read. The problem is that the LLM needs additional context injected to understand the question or input prompt. The solution must send the LLM the right context at the right time, so it can reason about a sensible answer.

Contemporary Solution

Direct LLM interaction shows the need for context to make sense of inputs.

The good news is, software engineers have enough glue and sticks to connect a researcher team's incomplete language model to other bridge technologies — and deliver business value today in November 2023. There's a lot to unpack here:



Chat UI - User Request

Problem: get engagement from users

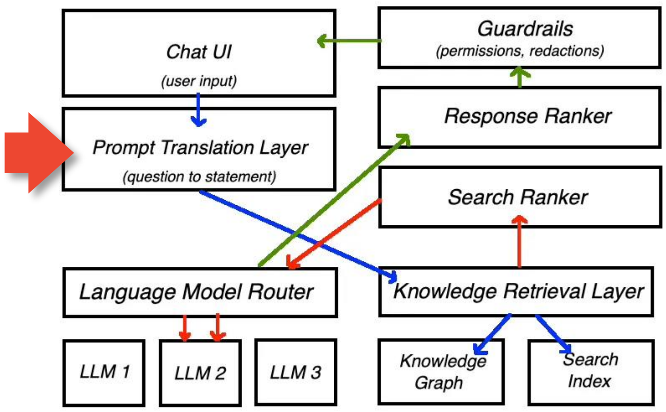
This is the now familiar interface of Chat GPT and other AI assistants. The user types in any question or thread of questions, and the application treats this as a single session. Each previous question is fed to the language model with the next, clarifying question. Remember that LLMs have zero data retention at runtime and each subsequent prompt request travels a different neural pathway than the previous one. In other words, LLMs have amnesia from one input statement to the next.

Prompt and Knowledge Handling

Simplifying user queries and enhancing LLM with relevant information

Prompt Translation Layer

Problem: understand the natural language of the user and translate for machines



At a high level, language models do not like questions. Think of AI as a statistical computer that is really good at guessing what comes next. Instead, it prefers to work with a question restated as an incomplete sentence — then it tries to complete it to the user's satisfaction. This is the first software component in connecting user requests to a raw LLM.

Knowledge Retrieval Layer

Problem: quickly find relevant research on the user's topic

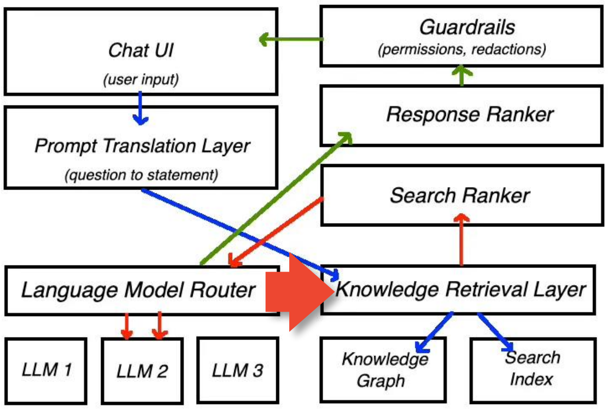
Retrieval Augmented Generation (RAG) is a category of research that has taken concrete form as a first class service in many AI applications. Imagine you're working with a 7 billion parameter LLM that has sparse knowledge of medical research. The constraint is you don't have the time, talent or money to create and fine tune a foundation model that is perfectly weighted to respond to healthcare workers. Instead, we can take a shortcut by searching for information relevant to the question, and then giving the LLM a hint when we send in the user's prompt. The hint can actually be >200,000 words long, if you're willing to pay for the maximum number of input tokens.

Data Retrieval Foundations

Classic search tech underpins AI, easier than new LLM development.

Knowledge Retrieval Layer (cont.)

Here we can use two older storage technologies - knowledge graphs and search indexes. Yes, the same style search indexes from the 1990s web. Loading documents into a knowledge graph and tuning search indexes is hard, but not nearly as hard as creating a new LLM. Tens of millions of developers can do the former, where only tens of thousands effectively can do the latter. This starts to uncover why some AI talent is so sought after, and why compensation packages for these lucky (and talented) pioneers resemble the first dotcom boom.



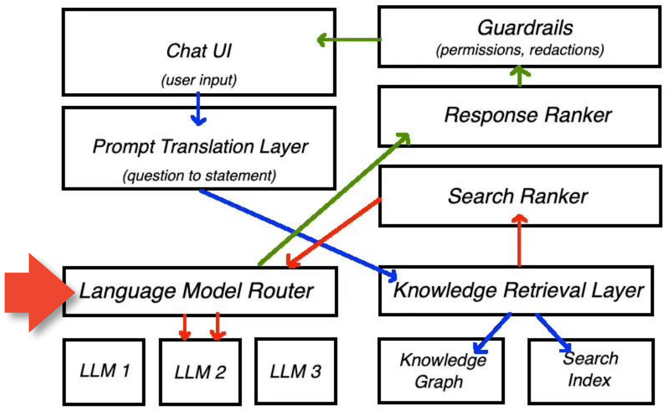
Note that this later might, or might not, include a vector database and embedded (numerical) representations of the documents. LLMs don't actually know any words, they work with arrays of numbers and guess the next number. The numbers map to letters and words, and the guessing paths are all represented as geometric distances. This could be the subject of a future post.

Smart LLM Routing

Direct queries to suitable models, balancing cost and user satisfaction.

Language Model Router

Problem: optimize the system behavior for accuracy, speed, and cost

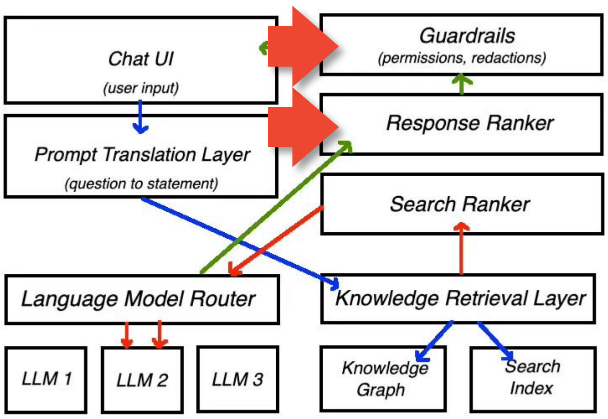


Assume we have a contract with a single LLM as a Service provider, like Anthropic or Azure AI Studio. These providers offer several choices of models to send a Prompt request to. If we were to optimize cost, we might send easy questions to the cheap model and hard questions to the expensive model.

This has a material impact on our overall cost to serve the application, as the LLM is typically the most resource intensive and the highest infrastructure bill by far. This layer can get very thick and very sophisticated, especially at high scale where it's possible to perform automated A/B tests and optimize. The idea is to satisfy the user for the lowest possible cost.

Quality Control for AI

Filter and rank LLM responses for quality, safety, and permissions.



Response Ranker

Problem: given non-deterministic responses, choose the best response of ~3

One brute force workaround for bad LLM responses is to send the prompt multiple times up front, and then throw away the ugly looking guesses. We can apply yet another ranker here to determine that. In practice, this may just be an extension of the Search Ranker. Ordering responses and selecting the best one could involve double checking the answer in the knowledge graph.

Guardrails

Problem: perform a safety check and enterprise permissions checks

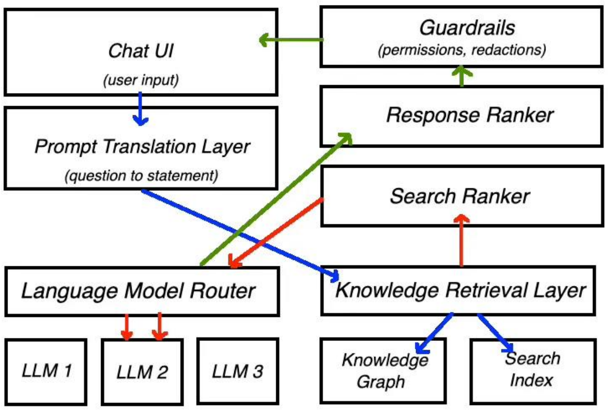
Guardrails is a term stolen from an open source project that is like a sentinel for AI safety. There are two forms of this. The first is a societal and ethical filter, like preventing harmful activities and malicious intents. A good example is a user who wants to know how to create malware to take down a hospital. The second is a permissions and need-to-know filter, like a 100,000 person corporation where every employee is not permitted to know everything. These are basically access checks, and could result in redacted responses or user friendly error messages.

AI Response Validation

Users rate AI replies to enhance system precision.

Chat UI - System Response

Problem: get feedback from the user on whether the system is correct, and why



We're back to serving the AI's response to the user and we want to know if it's valid. The user interface should have a way of collecting feedback, such as thumbs-up/thumbs-down, a 0-5 star ranking, and maybe even follow-up questions with quick-response input buttons to clarify why. This information is very important for tuning the behavior of this system.

Conclusion

It's flippant to say that the entire crop of AI applications coming on the market are a "thin wrapper over GPT4". By that logic, most of Web 2.0 was a "thin wrapper over AWS". A tremendous amount of value is being created.

Optimizing the layers of this LLM Web Applications Stack for accuracy, performance and cost is a very hard computer science problem. Not as hard as making a trillion parameter LLM, but can provide the same value to your users — today. No miracles required. Just have a lot of discipline in testing and tuning each layer of this stack, and obsessively measure whether the changes are improving user engagement, or harming user satisfaction.

Authorship

Gregory Wester
Principal
United States
Tel: (415) 806-1572
Gwester@StrataAIT.com