Architectures for LLM Applications

Purpose of the document

Large language models (LLM) are a powerful new primitive for building software. But since they are so new and emerging technologies to industries and behave so differently from normal computing resources.

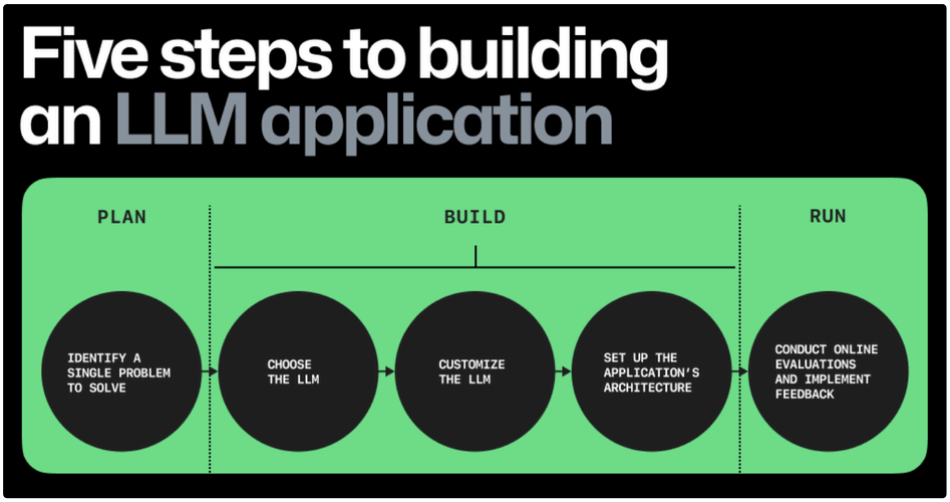
In this document we're sharing a reference architecture for the emerging LLM application stack (GitHub - a16z-infra/llm-app-stack). It shows the most common systems, tools, and design patterns .This stack is still very early stage and may change substantially as the underlying technology advances, but we hope it will be a useful reference for developers working with LLMs now.

Essential Steps to Building Language Models Apps

Building software with LLMs, or any machine learning (ML) model, is fundamentally different from building software without them. For one, rather than compiling source code into binary to run a series of commands, developers need to navigate *datasets, embeddings, and parameter weights* to generate consistent and accurate outputs. After all, LLM outputs are probabilistic and don't produce the same predictable outcomes.

Let's dive right in. Here are the steps for building language Models application:

1. **Choose the Right LLM**: Select from proprietary models (like GPT4), open-source LLMs (like Llama, Mistral), or develop your own. Consider performance, knowledge cutoff, and infrastructure costs.
2. **Customize the Model**: Adapt the LLM to your needs using techniques like fine-tuning, RLHF, RLAIF, or RAG, each suitable for different tasks.
3. **Evaluate Performance**: Use benchmarks and A/B testing to assess the model's effectiveness, considering the subjective nature of text outputs (specific examples in the video).
4. **Deployment**: Tackle challenges like computational power, memory, and cost. Use methods like model distillation and quantization for efficiency, keeping in mind ethical and privacy considerations.
5. **Monitor and Refine Post-Deployment**: Continuously oversee the model to manage bugs and unexpected behaviors. Tools like Weights and Biases can assist in this ongoing process.



Let's break down, at a high level, the steps to build an LLM applications in detail,

Focus on a single use case

Find a use case that's the right size: one that's focused enough so you can quickly iterate and make progress, but also big enough so that the right solution will provide best fit for users.

Choose the right LLM:

We can saving costs by building an LLM app with a pre-trained model, but how do we pick the right one? Here are some factors to consider:

- **Licensing**. If you hope to eventually sell your LLM app, you'll need to use a model that has an API licensed for commercial use.To get you started on your search, here's a community-sourced list of open LLMs that are licensed for commercial use (GitHub - eugeneyan/open-llms: 📋 A list of open LLMs available for commercial use.)
- **Model size.** The size of LLMs can range from 7 to 175 billion parameters—and some, like Ada , are even as small as 350 million parameters. Most LLMs (as of now) range in size from 7-13 billion parameters.if a model has more parameters (variables that can be adjusted to improve a model's output), the better the model is at learning new information and providing predictions.
- **Model performance**. Before customize LLM using techniques like fine-tuning and in-context learning , evaluate how well and fast—and how consistently—the model generates desired output. To measure model performance, can be use offline evaluations.

Customize the LLM

When we customize a pre-trained LLM, need to adapting the LLM to specific tasks, such as generating text around a specific topic or in a particular style. To customize a pre-trained LLM to our specific use cases, we can try in-context learning, reinforcement learning from human feedback (RLHF), or fine-tuning.

- **In-context learning** (Prompt Engineering)

when we provide the model with specific instructions or examples at the time of inference (or) the time you're querying the model (or) asking it to infer what is our expectation and generate a contextually relevant output.(*In-context learning can be done in a variety of ways, like providing examples, rephrasing queries, and adding a sentence that states goal at a high-level)*

- **RLHF(**Reinforcement learning from human feedback**)** is a ML techniques which comprises a reward model for the pre-trained LLM. The reward model is trained to predict if a user will accept or reject the output from the pre-trained LLM. The learnings from the reward model are passed to the pre-trained LLM, which will adjust its outputs based on user acceptance rate.
- **Fine-tuning** is when the model's generated output is evaluated against an intended or known output. For example, you know that the sentiment behind a statement like this is negative: "The soup is too salty." To evaluate the LLM, you'd feed this sentence to the model and query it to label the sentiment as positive or negative. If the model labels it as positive, then you'd adjust the model's parameters and try prompting it again to see if it can classify the sentiment as negative.

Set up the app's architecture

The different components needs to be set up optimized LLM application can be roughly grouped into three categories:

- **User input** which requires a UI, an LLM, and an app hosting platform.
- **Input enrichment and prompt construction tools.** This includes data source, embedding model, a vector database, prompt construction and optimization tools, and a data filter.
- **Efficient and responsible AI tooling,** which includes an LLM cache, LLM content classifier or filter, and a telemetry service to evaluate the output of LLM application.

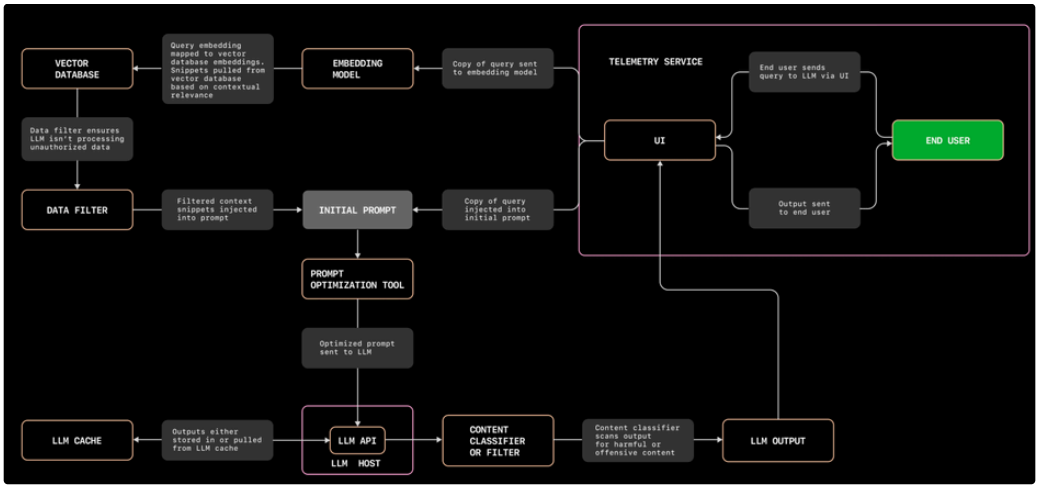
Conduct online evaluations of LLM Application

These evaluations are considered "online" because they assess the LLM's performance during user interactions.

The below diagram represents the high level architecture of LLM application workflow ,the different components can be roughly grouped into FOUR categories

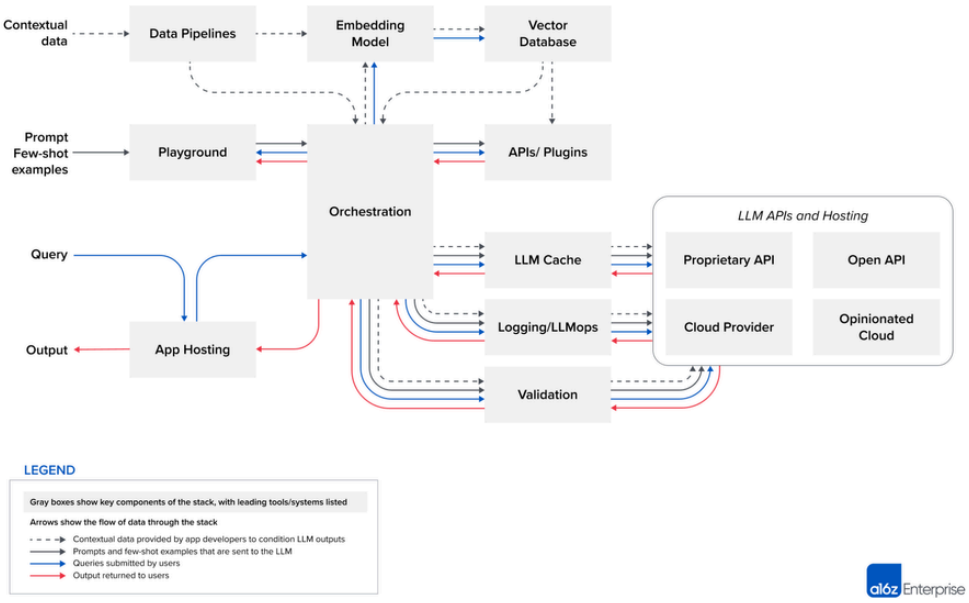
- User Inputs

- Input Enrichment Tools and Prompt Constructions
- Efficient and Responsible AI Tooling
- LLM Outputs



(Ref - The architecture of today's LLM applications)

Here is the another pictorial representation of LLM application Stack



Ref - llm-app-stack/README.md at main · a16z-infra/llm-app-stack

There are many different ways to build with LLMs, including training models from scratch, fine-tuning open-source models, or using hosted APIs. The stack we're showing above is based on **in-context learning**, which is the design pattern we've seen the majority of AI developers start with (and is only possible now with foundation models).

Looking ahead

Pre-trained AI models represent the most important architectural change in software since the internet. They make it possible for individual developers to build incredible AI apps, in a matter of days, that surpass supervised machine learning projects that took big teams months to build.

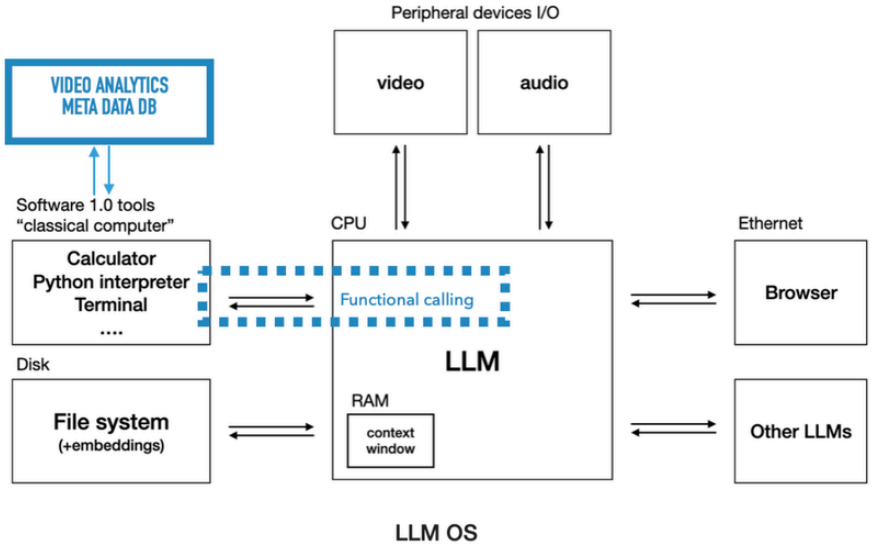
The tools and patterns have laid out here are likely the starting point, not the end state, for integrating LLMs. We'll update this as major changes take place (e.g., a shift toward model training) and release new reference architectures where it makes sense.

Function Calling

While much attention has been given to the text-generation capabilities of language models, the ability to generate structured output is very important if we want to interface with computer systems. This is achieved by function calling.

We build a function-calling model based on intel-neural-chat-7b-v3-1 replacing OpenAI's functional calling capabilities, for better performance and faster inference times.

Functional calling is a critical component for any AI search/query application.



Example for function -calling

Consider a user query for search: **"I want to see a Rajnikanth movie scene where he is fighting with guns."** Or a query for a health app: **"Get my glucose readings for the last 10 days."** Or for a calorie-tracking app: **"I ate 1 pizza for breakfast and 1 biryani for dinner."** In all these scenarios, the query text needs to be converted to a structured form with predefined functions and parameters, to enable the system to understand user intents and take appropriate actions.