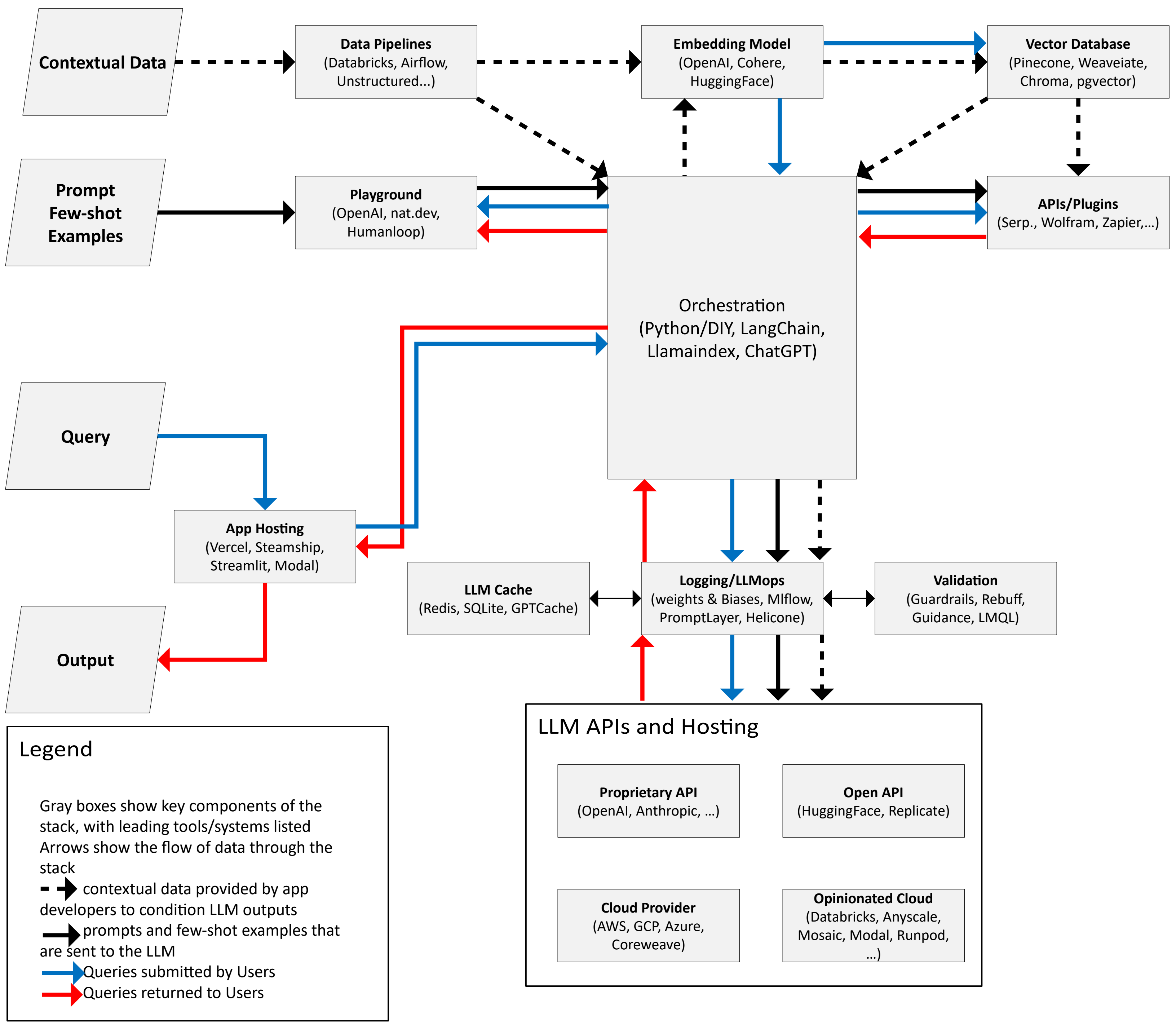
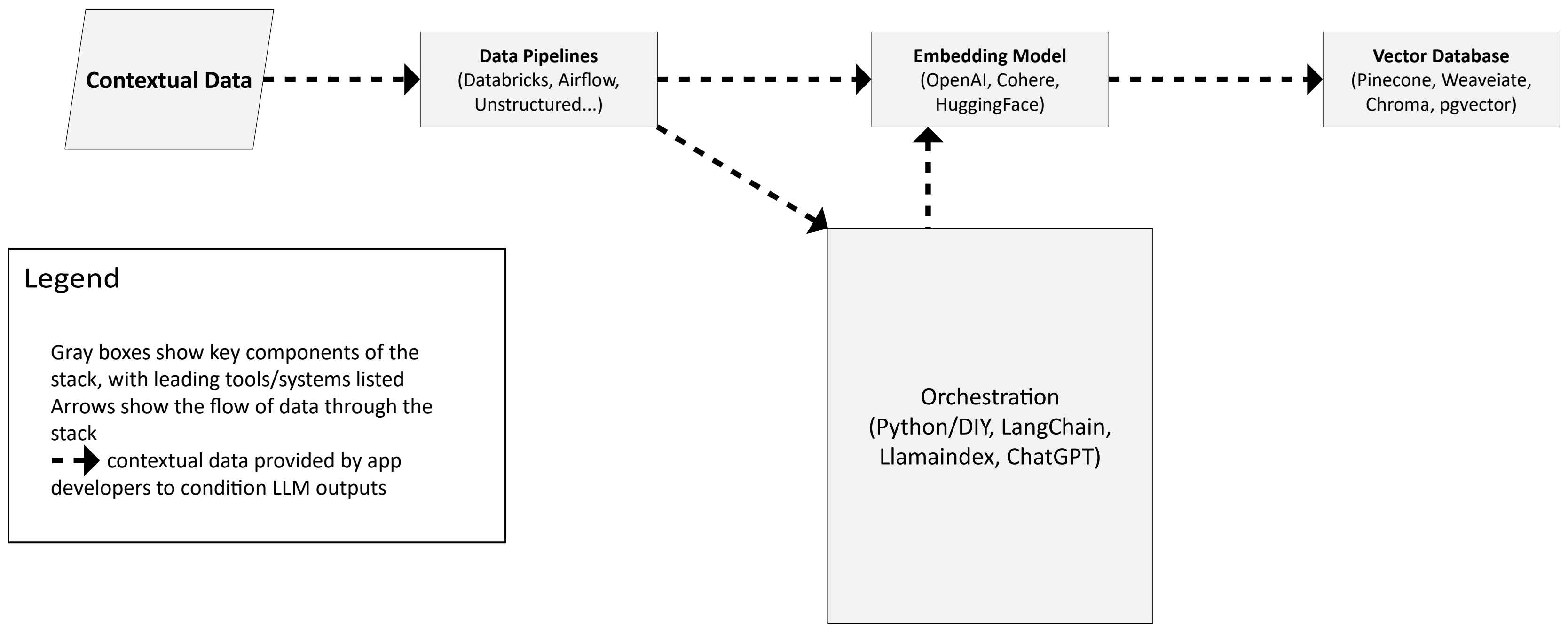
Emerging LLM App Stack


Contextual Data
Data Pipelines (Databricks, Airflow, Unstructured...)
Embedding Model (OpenAI, Cohere, HuggingFace)
Vector Database (Pinecone, Weaveiate, Chroma, pgvector)
Prompt Few-shot Examples
Playground (OpenAI, nat.dev, Humanloop)
APIs/Plugins (Serp., Wolfram, Zapier,...)
Orchestration (Python/DIY, LangChain, Llamaindex, ChatGPT)
Query
App Hosting (Vercel, Steamship, Streamlit, Modal)
Output
LLM Cache (Redis, SQLite, GPTCache)
Logging/LLMops (weights & Biases, Mlflow, PromptLayer, Helicone)
Validation (Guardrails, Rebuff, Guidance, LMQL)
LLM APIs and Hosting
Proprietary API (OpenAI, Anthropic, ...)
Open API (HuggingFace, Replicate)
Cloud Provider (AWS, GCP, Azure, Coreweave)
Opinionated Cloud (Databricks, Anyscale, Mosaic, Modal, Runpod, ...)

Legend

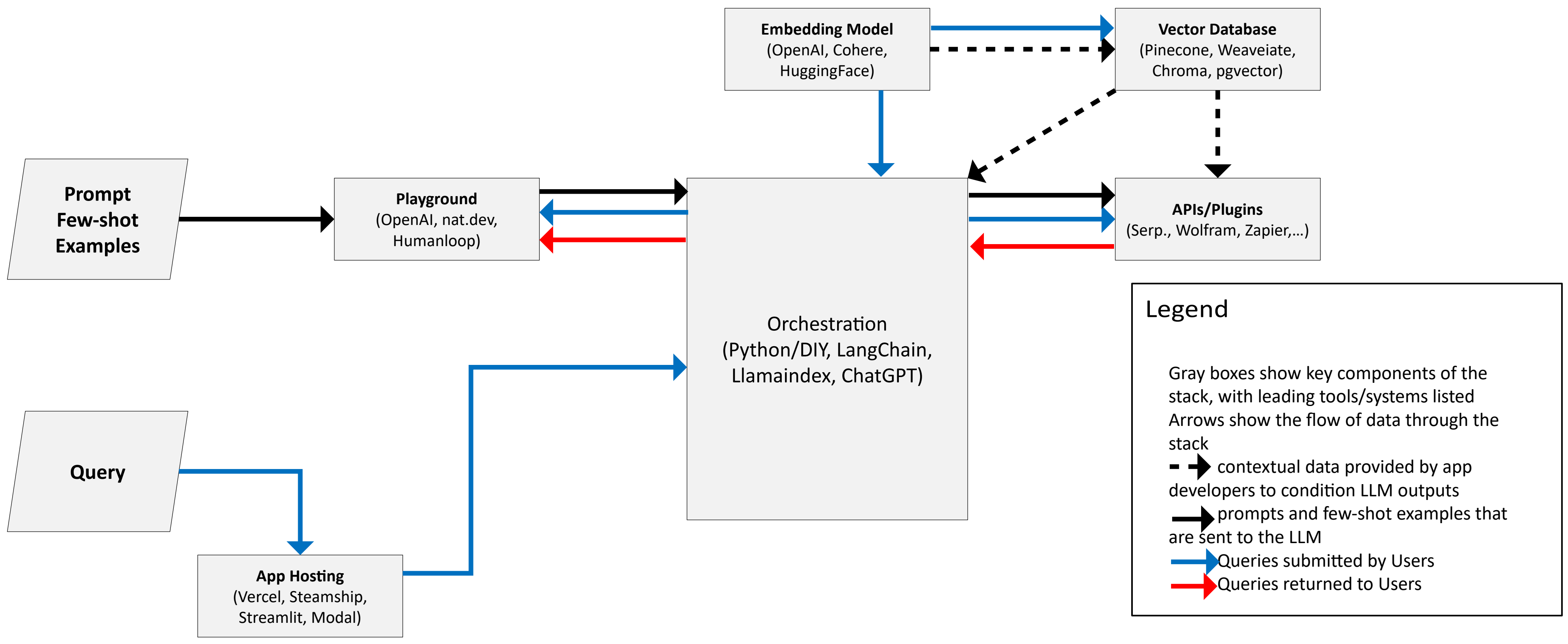
Gray boxes show key components of the stack, with leading tools/systems listed
Arrows show the flow of data through the stack
- - → contextual data provided by app developers to condition LLM outputs
→ prompts and few-shot examples that are sent to the LLM
→ Queries submitted by Users
→ Queries returned to Users


Contextual Data
Data Pipelines (Databricks, Airflow, Unstructured...)
Embedding Model (OpenAI, Cohere, HuggingFace)
Vector Database (Pinecone, Weaveiate, Chroma, pgvector)
Orchestration (Python/DIY, LangChain, Llamaindex, ChatGPT)
Legend
Gray boxes show key components of the stack, with leading tools/systems listed
Arrows show the flow of data through the stack
contextual data provided by app developers to condition LLM outputs

Contextual data for LLM apps includes text documents, PDFs, and even structured formats like CSV or SQL tables. Data-loading and transformation solutions for this data vary widely across developers. Most use traditional ETL tools like Databricks or Airflow. Some also use document loaders built into orchestration frameworks like LangChain (powered by Unstructured) and LlamaIndex (powered by Llama Hub). We believe this piece of the stack is relatively underdeveloped, though, and there's an opportunity for data-replication solutions purpose-built for LLM apps.

For embedding's, most developers use the OpenAI API, specifically with the text-embedding-ada-002 model. It's easy to use (especially if you're already using other OpenAI APIs), gives reasonably good results, and is becoming increasingly cheap. Some larger enterprises are also exploring Cohere, which focuses their product efforts more narrowly on embedding's and has better performance in certain scenarios. For developers who prefer open-source, the Sentence Transformers library from Hugging Face is a standard. It's also possible to create different types of embedding's tailored to different use cases; this is a niche practice today but a promising area of research.

The most important piece of the preprocessing pipeline, from a systems standpoint, is the vector database. It's responsible for efficiently storing, comparing, and retrieving up to billions of embeddings (i.e., vectors). The most common choice we've seen in the market is Pinecone. It's the default because it's fully cloud-hosted—so it's easy to get started with—and has many of the features larger enterprises need in production (e.g., good performance at scale, SSO, and uptime SLAs).


Embedding Model
(OpenAI, Cohere, HuggingFace)
Vector Database
(Pinecone, Weaveiate, Chroma, pgvector)
Prompt Few-shot Examples
Playground
(OpenAI, nat.dev, Humanloop)
Orchestration
(Python/DIY, LangChain, Llamaindex, ChatGPT)
APIs/Plugins
(Serp., Wolfram, Zapier,...)
Query
App Hosting
(Vercel, Steamship, Streamlit, Modal)
Legend
Gray boxes show key components of the stack, with leading tools/systems listed
Arrows show the flow of data through the stack
contextual data provided by app developers to condition LLM outputs
prompts and few-shot examples that are sent to the LLM
Queries submitted by Users
Queries returned to Users

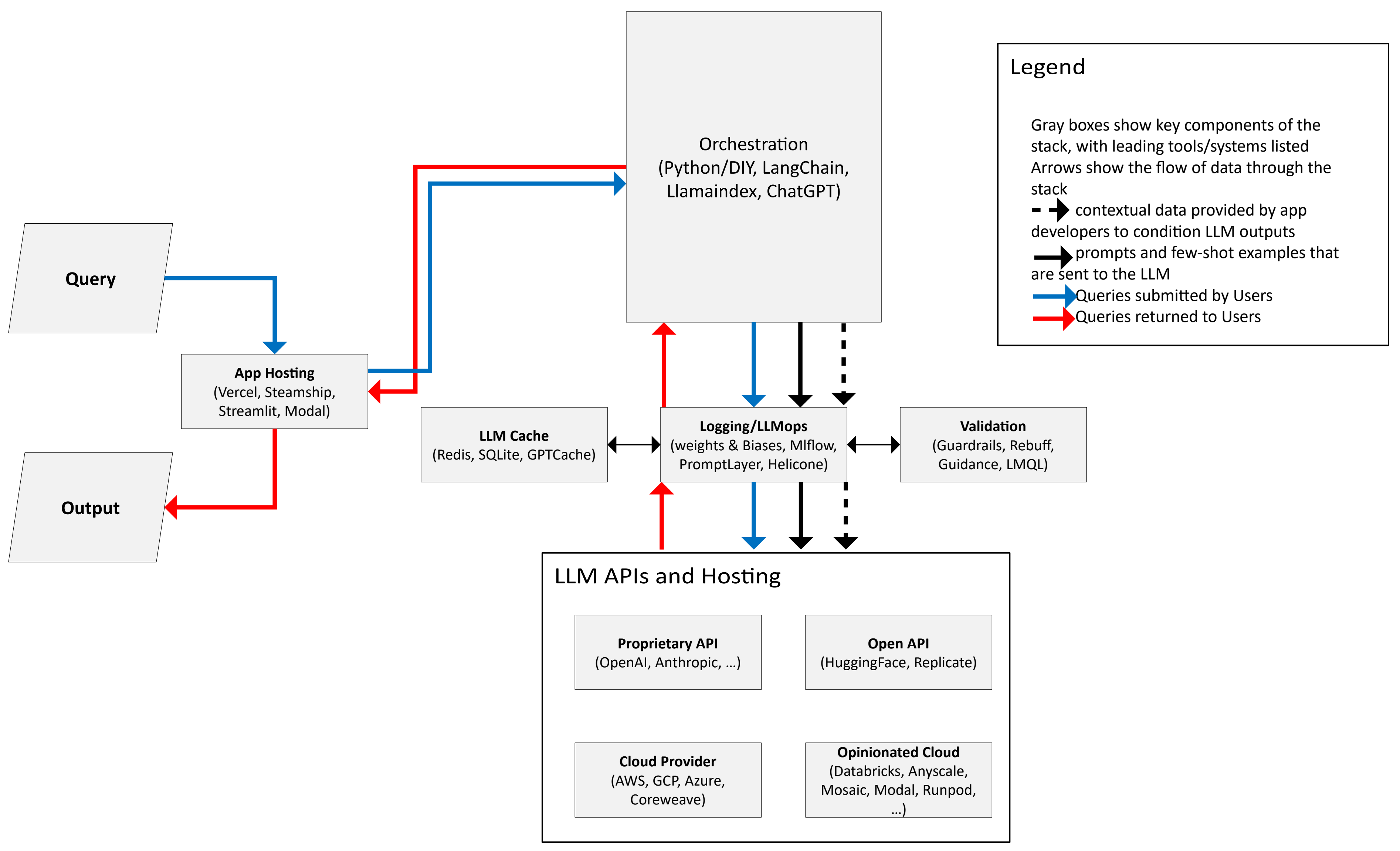
Strategies for prompting LLMs and incorporating contextual data are becoming increasingly complex—and increasingly important as a source of product differentiation. Most developers start new projects by experimenting with simple prompts, consisting of direct instructions (zero-shot prompting) or possibly some example outputs (few-shot prompting). These prompts often give good results but fall short of accuracy levels required for production deployments.

The next level of prompting jiu jitsu is designed to ground model responses in some source of truth and provide external context the model wasn't trained on. The Prompt Engineering Guide catalogs no fewer than 12 (!) more advanced prompting strategies, including chain-of-thought, self-consistency, generated knowledge, tree of thoughts, directional stimulus, and many others. These strategies can also be used in conjunction to support different LLM use cases like document question answering, chatbots, etc.

This is where orchestration frameworks like LangChain and LlamaIndex shine. They abstract away many of the details of prompt chaining; interfacing with external APIs (including determining when an API call is needed); retrieving contextual data from vector databases; and maintaining memory across multiple LLM calls. They also provide templates for many of the common applications mentioned above. Their output is a prompt, or series of prompts, to submit to a language model. These frameworks are widely used among hobbyists and startups looking to get an app off the ground, with LangChain the leader.

LangChain is still a relatively new project (currently on version 0.0.201), but we're already starting to see apps built with it moving into production. Some developers, especially early adopters of LLMs, prefer to switch to raw Python in production to eliminate an added dependency. But we expect this DIY approach to decline over time for most use cases, in a similar way to the traditional web app stack.

Sharp-eyed readers will notice a seemingly weird entry in the orchestration box: ChatGPT. In its normal incarnation, ChatGPT is an app, not a developer tool. But it can also be accessed as an API. And, if you squint, it performs some of the same functions as other orchestration frameworks, such as: abstracting away the need for bespoke prompts; maintaining state; and retrieving contextual data via plugins, APIs, or other sources. While not a direct competitor to the other tools listed here, ChatGPT can be considered a substitute solution, and it may eventually become a viable, simple alternative to prompt construction.


Orchestration (Python/DIY, LangChain, Llamaindex, ChatGPT)
Legend
Gray boxes show key components of the stack, with leading tools/systems listed
Arrows show the flow of data through the stack
contextual data provided by app developers to condition LLM outputs
prompts and few-shot examples that are sent to the LLM
Queries submitted by Users
Queries returned to Users
Query
App Hosting (Vercel, Steamship, Streamlit, Modal)
Output
LLM Cache (Redis, SQLite, GPTCache)
Logging/LLMops (weights & Biases, Mlflow, PromptLayer, Helicone)
Validation (Guardrails, Rebuff, Guidance, LMQL)
LLM APIs and Hosting
Proprietary API (OpenAI, Anthropic, ...)
Open API (HuggingFace, Replicate)
Cloud Provider (AWS, GCP, Azure, Coreweave)
Opinionated Cloud (Databricks, Anyscale, Mosaic, Modal, Runpod, ...)

Today, OpenAI is the leader among language models. Nearly every developer we spoke with starts new LLM apps using the OpenAI API, usually with the gpt-4 or gpt-4-32k model. This gives a best-case scenario for app performance and is easy to use, in that it operates on a wide range of input domains and usually requires no fine-tuning or self-hosting.

When projects go into production and start to scale, a broader set of options come into play. Some of the common ones we heard include:

Switching to gpt-3.5-turbo: It's ~50x cheaper and significantly faster than GPT-4. Many apps don't need GPT-4-level accuracy, but do require low latency inference and cost effective support for free users.
Experimenting with other proprietary vendors (especially Anthropic's Claude models): Claude offers fast inference, GPT-3.5-level accuracy, more customization options for large customers, and up to a 100k context window (though we've found accuracy degrades with the length of input).
Triaging some requests to open source models: This can be especially effective in high-volume B2C use cases like search or chat, where there's wide variance in query complexity and a need to serve free users cheaply.
This usually makes the most sense in conjunction with fine-tuning open source base models. We don't go deep on that tooling stack in this article, but platforms like Databricks, Anyscale, Mosaic, Modal, and RunPod are used by a growing number of engineering teams.
A variety of inference options are available for open source models, including simple API interfaces from Hugging Face and Replicate; raw compute resources from the major cloud providers; and more opinionated cloud offerings like those listed above.

Open-source models trail proprietary offerings right now, but the gap is starting to close. The LLaMa models from Meta set a new bar for open source accuracy and kicked off a flurry of variants. Since LLaMa was licensed for research use only, a number of new providers have stepped in to train alternative base models (e.g., Together, Mosaic, Falcon, Mistral). Meta is also debating a truly open source release of LLaMa 2.

When (not if) open source LLMs reach accuracy levels comparable to GPT-3.5, we expect to see a Stable Diffusion-like moment for text—including massive experimentation, sharing, and productionizing of fine-tuned models. Hosting companies like Replicate are already adding tooling to make these models easier for software developers to consume. There's a growing belief among developers that smaller, fine-tuned models can reach state-of-the-art accuracy in narrow use cases.

Most developers we spoke with haven't gone deep on operational tooling for LLMs yet. Caching is relatively common—usually based on Redis—because it improves application response times and cost. Tools like Weights & Biases and MLflow (ported from traditional machine learning) or PromptLayer and Helicone (purpose-built for LLMs) are also fairly widely used. They can log, track, and evaluate LLM outputs, usually for the purpose of improving prompt construction, tuning pipelines, or selecting models. There are also a number of new tools being developed to validate LLM outputs (e.g., Guardrails) or detect prompt injection attacks (e.g., Rebuff). Most of these operational tools encourage use of their own Python clients to make LLM calls, so it will be interesting to see how these solutions coexist over time.

Finally, the static portions of LLM apps (i.e. everything other than the model) also need to be hosted somewhere. The most common solutions we've seen so far are standard options like Vercel or the major cloud providers. However, two new categories are emerging. Startups like Steamship provide end-to-end hosting for LLM apps, including orchestration (LangChain), multi-tenant data contexts, async tasks, vector storage, and key management. And companies like Anyscale and Modal allow developers to host models and Python code in one place.