



Definition

Prompt engineering is the process of designing and optimizing prompts to LLMs for a wide variety of applications and research topics. Prompts are short pieces of text that are used to guide the LLM's output. Prompt engineering skills help to better understand the capabilities and limitations of LLMs.

Prompt engineering involves selecting the right words, phrases, symbols, and formats that elicit the desired response from the LLM. Prompt engineering also involves using other controls, such as parameters, examples, or data sources, to influence the LLM's behavior. For example, if we want our LLM-powered application to generate responses for a 5-year-old child, we can specify this in a system message similar to “Act as a teacher who explains complex concepts to 5-year-old children.”

In fact, Andrej Karpathy, the previous Director of AI at Tesla, who returned to OpenAI in February 2023, tweeted that “English is the hottest new programming language.”

We will dive deeper into the concept of prompt engineering in *Chapter 4, Prompt Engineering*. In the next section, we are going to focus on the emerging AI orchestrators.

Introducing AI orchestrators to embed LLMs into applications

Earlier in this chapter, we saw that there are two main aspects to consider when incorporating LLMs within applications: a technical aspect and a conceptual aspect. While we can explain the conceptual aspect with the brand-new category of software called Copilot, in this section, we are going to further explore how to technically embed and orchestrate LLMs within our applications.

The main components of AI orchestrators

From one side, the paradigm shift of foundation models implies a great simplification in the domain of AI-powered applications: after producing models, now the trend is consuming models. On the other side, many roadblocks might arise in developing this new kind of AI, since there are LLM-related components that are brand new and have never been managed before within an application life cycle. For example, there might be malicious actors that could try to change the LLM instructions (the system message mentioned earlier) so that the application does not follow the correct instructions. This is an example of a new set of security threats that are typical to LLM-powered applications and need to be addressed with powerful counterattacks or preventive techniques.

The following is an illustration of the main components of such applications:

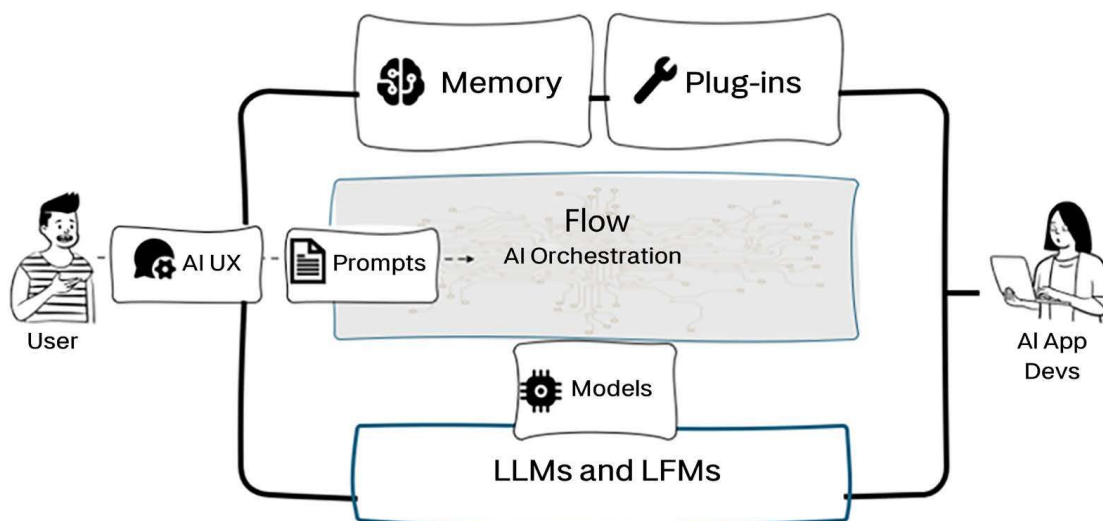


Figure 2.5: High-level architecture of LLM-powered applications


Let's inspect each of these components in detail:

- **Models:** The model is simply the type of LLM we decide to embed in our application. There are two main categories of models:
 - **Proprietary LLMs:** Models that are owned by specific companies or organizations. Examples include GPT-3 and GPT-4, developed by OpenAI, or Bard, developed by Google. As their source code and architecture are not available, those models cannot be re-trained from scratch on custom data, yet they can be fine-tuned if needed.
 - **Open-source:** Models with code and architecture freely available and distributed, hence they can also be trained from scratch on custom data. Examples include Falcon LLM, developed by Abu Dhabi's **Technology Innovation Institute (TII)**, or LLaMA, developed by Meta.

We will dive deeper into the main set of LLMs available today in *Chapter 3, Choosing an LLM for Your Application*.

- **Memory:** LLM applications commonly use a conversational interface, which requires the ability to refer back to earlier information within the conversation. This is achieved through a “memory” system that allows the application to store and retrieve past interactions. Note that past interactions could also constitute additional non-parametric knowledge to be added to the model. To achieve that, it is important to store all the past conversations – properly embedded – into VectorDB, which is at the core of the application's data.

Definition



VectorDB is a type of database that stores and retrieves information based on vectorized embeddings, the numerical representations that capture the meaning and context of text. By using VectorDB, you can perform semantic search and retrieval based on the similarity of meanings rather than keywords. VectorDB can also help LLMs generate more relevant and coherent text by providing contextual understanding and enriching generation results. Some examples of VectorDBs are Chroma, Elasticsearch, Milvus, Pinecone, Qdrant, Weaviate, and **Facebook AI Similarity Search (FAISS)**.

FAISS, developed by Facebook (now Meta) in 2017, was one of the pioneering vector databases. It was designed for efficient similarity search and clustering of dense vectors and is particularly useful for multimedia documents and dense embeddings. It was initially an internal research project at Facebook. Its primary goal was to better utilize GPUs for identifying similarities related to user preferences. Over time, it evolved into the fastest available library for similarity search and can handle billion-scale datasets. FAISS has opened up possibilities for recommendation engines and AI-based assistant systems.

- **Plug-ins:** They can be seen as additional modules or components that can be integrated into the LLM to extend its functionality or adapt it to specific tasks and applications. These plug-ins act as add-ons, enhancing the capabilities of the LLM beyond its core language generation or comprehension abilities.

The idea behind plug-ins is to make LLMs more versatile and adaptable, allowing developers and users to customize the behavior of the language model for their specific needs. Plug-ins can be created to perform various tasks, and they can be seamlessly incorporated into the LLM's architecture.

- **Prompts:** This is probably the most interesting and pivotal component of an LLM-powered application. We've already quoted, in the previous section, Andrej Karpathy's affirmation that "English is the hottest new programming language," and you will understand why in the upcoming chapters. Prompts can be defined at two different levels:
 - **"Frontend," or what the user sees:** A "prompt" refers to the input to the model. It is the way the user interacts with the application, asking things in natural language.
 - **"Backend," or what the user does not see:** Natural language is not only the way to interact, as a user, with the frontend; it is also the way we "program" the backend. In fact, on top of the user's prompt, there are many natural language instructions, or meta-prompts, that we give to the model so that it can properly address the user's query. Meta-prompts are meant to instruct the model to act as it is meant to. For example, if we want to limit our application to answer only questions related to the documentation we provided in VectorDB, we will specify the following in our meta-prompts to the model: *"Answer only if the question is related to the provided documentation."*

Finally, we get to the core of the high-level architecture shown in *Figure 2.5*, that is, the **AI orchestrator**. With the AI orchestrator, we refer to lightweight libraries that make it easier to embed and orchestrate LLMs within applications.

As LLMs went viral by the end of 2022, many libraries started arising in the market. In the next sections, we are going to focus on three of them: LangChain, Semantic Kernel, and Haystack.

LangChain

LangChain was launched as an open-source project by Harrison Chase in October 2022. It can be used both in Python and JS/TS. It is a framework for developing applications powered by language models, making them data-aware (with grounding) and agentic – which means they are able to interact with external environments.

Let's take a look at the key components of LangChain:

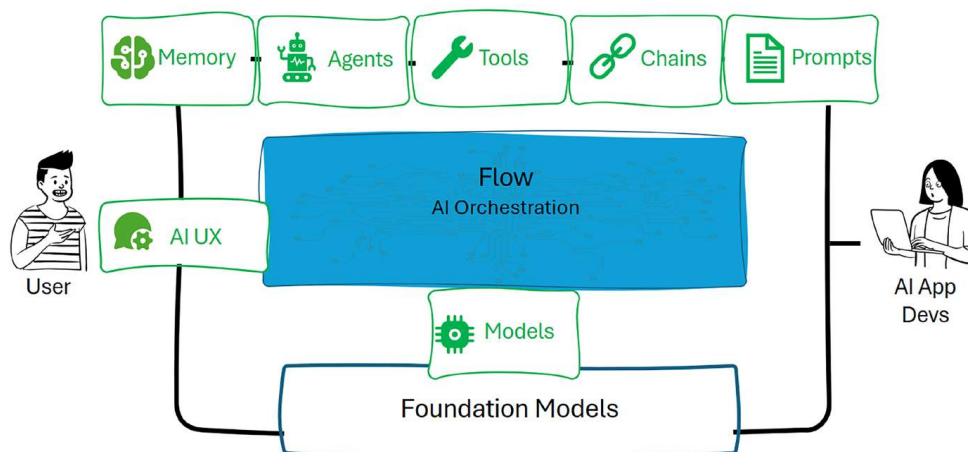


Figure 2.6: LangChain's components

Overall, LangChain has the following core modules:

- **Models:** These are the LLMs or LFMs that will be the engine of the application. LangChain supports proprietary models, such as those available in OpenAI and Azure OpenAI, and open-source models consumable from the **Hugging Face Hub**.

Definition



Hugging Face is a company and a community that builds and shares state-of-the-art models and tools for natural language processing and other machine learning domains. It developed the Hugging Face Hub, a platform where people can create, discover, and collaborate on machine learning models and LLMs, datasets, and demos. The Hugging Face Hub hosts over 120k models, 20k datasets, and 50k demos in various domains and tasks, such as audio, vision, and language.

Alongside models, LangChain also offers many prompt-related components that make it easier to manage the prompt flow.

- **Data connectors:** These refer to the building blocks needed to retrieve the additional external knowledge (for example, in RAG-based scenarios) we want to provide the model with. Examples of data connectors are document loaders or text embedding models.
- **Memory:** This allows the application to keep references to the user's interactions, in both the short and long term. It is typically based on vectorized embeddings stored in VectorDB.
- **Chains:** These are predetermined sequences of actions and calls to LLMs that make it easier to build complex applications that require chaining LLMs with each other or with other components. An example of a chain might be: take the user query, chunk it into smaller pieces, embed those chunks, search for similar embeddings in VectorDB, use the top three most similar chunks in VectorDB as context to provide the answer, and generate the answer.
- **Agents:** Agents are entities that drive decision-making within LLM-powered applications. They have access to a suite of tools and can decide which tool to call based on the user input and the context. Agents are dynamic and adaptive, meaning that they can change or adjust their actions based on the situation or the goal.

LangChain offers the following benefits:

- LangChain provides modular abstractions for the components we previously mentioned that are necessary to work with language models, such as prompts, memory, and plug-ins.
- Alongside those components, LangChain also offers pre-built **chains**, which are structured concatenations of components. Those chains can be pre-built for specific use cases or be customized.

In *Part 2* of this book, we will go through a series of hands-on applications, all LangChain based. So, starting from *Chapter 5, Embedding LLMs within Your Applications*, we will focus much deeper on LangChain components and overall frameworks.

Haystack

Haystack is a Python-based framework developed by Deepset, a startup founded in 2018 in Berlin by Milos Rusic, Malte Pietsch, and Timo Möller. Deepset provides developers with the tools to build **natural language processing (NLP)**-based applications, and with the introduction of Haystack, they are taking them to the next level.

The following illustration shows the core components of Haystack:

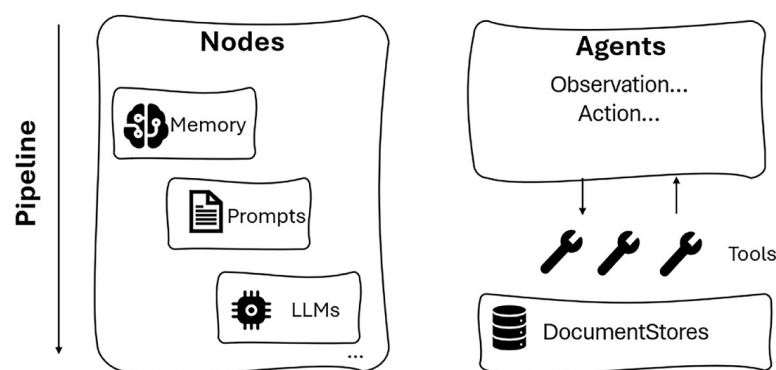


Figure 2.7: Haystack's components

Let's look at these components in detail:

- **Nodes:** These are components that perform a specific task or function, such as a retriever, a reader, a generator, a summarizer, etc. Nodes can be LLMs or other utilities that interact with LLMs or other resources. Among LLMs, Haystack supports proprietary models, such as those available in OpenAI and Azure OpenAI, and open-source models consumable from the Hugging Face Hub.
- **Pipelines:** These are sequences of calls to nodes that perform natural language tasks or interact with other resources. Pipelines can be querying pipelines or indexing pipelines, depending on whether they perform searches on a set of documents or prepare documents for search. Pipelines are predetermined and hardcoded, meaning that they do not change or adapt based on the user input or the context.
- **Agent:** This is an entity that uses LLMs to generate accurate responses to complex queries. An agent has access to a set of tools, which can be pipelines or nodes, and it can decide which tool to call based on the user input and the context. An agent is dynamic and adaptive, meaning that it can change or adjust its actions based on the situation or the goal.
- **Tools:** There are functions that an agent can call to perform natural language tasks or interact with other resources. Tools can be pipelines or nodes that are available to the agent and they can be grouped into toolkits, which are sets of tools that can accomplish specific objectives.
- **DocumentStores:** These are backends that store and retrieve documents for searches. DocumentStores can be based on different technologies, also including VectorDB (such as FAISS, Milvus, or Elasticsearch).

Some of the benefits offered by Haystack are:

- **Ease of use:** Haystack is user-friendly and straightforward. It's often chosen for lighter tasks and rapid prototypes.
- **Documentation quality:** Haystack's documentation is considered high-quality, aiding developers in building search systems, question-answering, summarization, and conversational AI.
- **End-to-end framework:** Haystack covers the entire LLM project life cycle, from data preprocessing to deployment. It's ideal for large-scale search systems and information retrieval.
- Another nice thing about Haystack is that you can deploy it as a REST API and it can be consumed directly.

Semantic Kernel

Semantic Kernel is the third open-source SDK we are going to explore in this chapter. It was developed by Microsoft, originally in C# and now also available in Python.

This framework takes its name from the concept of a “kernel,” which, generally speaking, refers to the core or essence of a system. In the context of this framework, a kernel is meant to act as the engine that addresses a user's input by chaining and concatenating a series of components into pipelines, encouraging **function composition**.