2

LLMs for AI-Powered Applications

In *Chapter 1*, *Introduction to Large Language Models*, we introduced **large language models** (**LLMs**) as powerful foundation models with generative capabilities as well as powerful common-sense reasoning. Now, the next question is: what should I do with those models?

In this chapter, we are going to see how LLMs are revolutionizing the world of software development, leading to a new era of AI-powered applications. By the end of this chapter, you will have a clearer picture of how LLMs can be embedded in different application scenarios, thanks to the new AI orchestrator frameworks that are populating the market of AI development.

In this chapter, we will cover the following topics:

- How LLMs are changing software development
- The copilot system
- Introducing AI orchestrators to embed LLMs into applications

How LLMs are changing software development

LLMs have proven to have extraordinary capabilities: from natural language understanding tasks (summarization, named entity recognition, and classification) to text generation, from common-sense reasoning to brainstorming skills. However, they are not just incredible by themselves. As discussed in *Chapter 1*, LLMs and, generally speaking, **large foundation models** (**LFMs**), are revolutionizing software development by serving as platforms for building powerful applications.

In fact, instead of starting from scratch, today developers can make API calls to a hosted version of an LLM, with the option of customizing it for their specific needs, as we saw in the previous chapter. This shift allows teams to incorporate the power of AI more easily and efficiently into their applications, similar to the transition from single-purpose computing to time-sharing in the past.

But what does it mean, concretely, to incorporate LLMs within applications? There are two main aspects to consider when incorporating LLMs within applications:

- **The technical aspect**, which covers the *how*. Integrating LLMs into applications involves embedding them through REST API calls and managing them with AI orchestrators. This means setting up architectural components that allow seamless communication with the LLMs via API calls. Additionally, using AI orchestrators helps to efficiently manage and coordinate the LLMs' functionality within the application, as we will discuss later in this chapter.
- **The conceptual aspect**, which covers the *what*. LLMs bring a plethora of new capabilities that can be harnessed within applications. These capabilities will be explored in detail later in this book. One way to view LLMs' impact is by considering them as a new category of software, often referred to as *copilot*. This categorization highlights the significant assistance and collaboration provided by LLMs in enhancing application functionalities.

We will delve into the technical aspect later on in this chapter, while the next section will cover a brand-new category of software – the copilot system.

The copilot system

The copilot system is a new category of software that serves as an expert helper to users trying to accomplish complex tasks. This concept was coined by Microsoft and has already been introduced into its applications, such as M365 Copilot and the new Bing, now powered by GPT-4. With the same framework that is used by these products, developers can now build their own copilots to embed within their applications.

But what exactly is a copilot?

As the name suggests, copilots are meant to be AI assistants that work side by side with users and support them in various activities, from information retrieval to blog writing and posting, from brainstorming ideas to code review and generation.

The following are some unique features of copilots:

- **A copilot is powered by LLMs**, or, more generally, LFMs, meaning that these are the reasoning engines that make the copilot "intelligent." This reasoning engine is one of its components, but not the only one. A copilot also relies on other technologies, such as apps, data sources, and user interfaces, to provide a useful and engaging experience for users. The following illustration shows how this works:

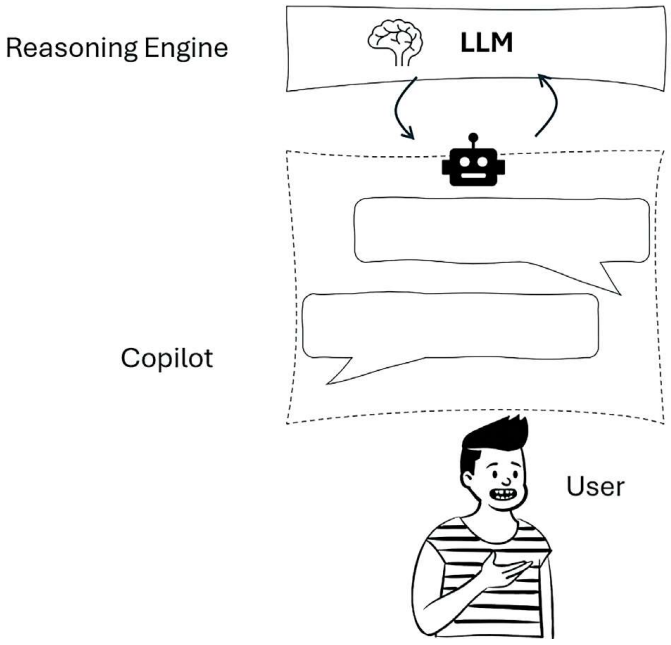

Figure 2.1: A copilot is powered by an LLM

- **A copilot is designed to have a conversational user interface,** allowing users to interact with it using natural language. This reduces or even eliminates the knowledge gap between complex systems that need domain-specific taxonomy (for example, querying tabular data needs the knowledge of programming languages such as T-SQL) and users. Let's look at an example of such a conversation:

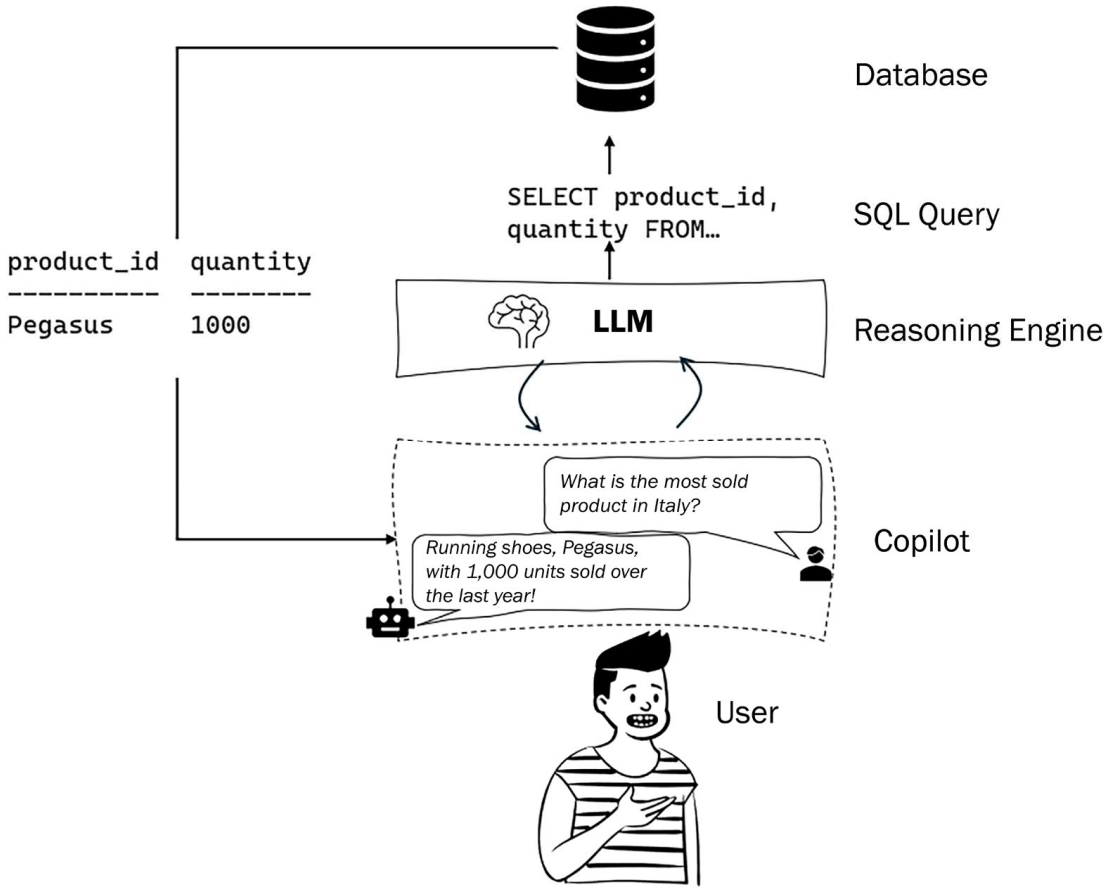

Figure 2.2: An example of a conversational UI to reduce the gap between the user and the database

- **A copilot has a scope.** This means that it is **grounded** to domain-specific data so that it is entitled to answer only within the perimeter of the application or domain.

Definition

Grounding is the process of using LLMs with information that is use case specific, relevant, and not available as part of the LLM's trained knowledge. It is crucial for ensuring the quality, accuracy, and relevance of the output. For example, let's say you want an LLM-powered application that assists you during your research on up-to-date papers (not included in the training dataset of your LLM). You also want your app to only respond if the answer is included in those papers. To do so, you will need to ground your LLM to the set of papers, so that your application will only respond within this perimeter.

Grounding is achieved through an architectural framework called retrieval-augmented generation (RAG), a technique that enhances the output of LLMs by incorporating information from an external, authoritative knowledge base before generating a response. This process helps to ensure that the generated content is relevant, accurate, and up to date.

What is the difference between a copilot and a RAG? RAG can be seen as one of the architectural patterns that feature a copilot. Whenever we want our copilot to be grounded to domain-specific data, we use a RAG framework. Note that RAG is not the only architectural pattern that can feature a copilot: there are further frameworks such as function calling or multi-agents that we will explore throughout the book.

For example, let's say we developed a copilot within our company that allows employees to chat with their enterprise knowledge base. As fun as it can be, we cannot provide users with a copilot they can use to plan their summer trip (it would be like providing users with a ChatGPT-like tool at our own hosting cost!); on the contrary, we want the copilot to be grounded only to our enterprise knowledge base so that it can respond only if the answer is pertinent to the domain-specific context.

The following figure shows an example of grounding a copilot system:

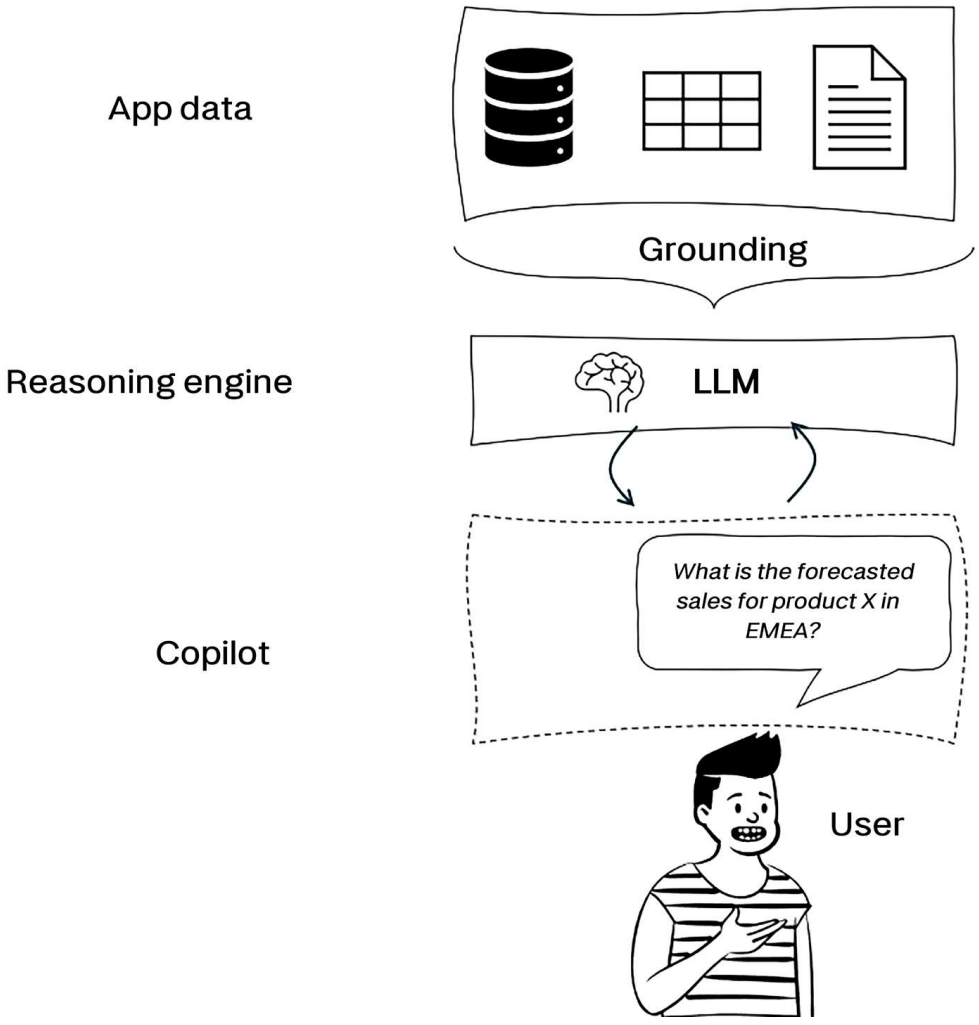

Figure 2.3: Example of grounding a copilot

- **The copilot's capabilities can be extended by skills**, which can be code or calls to other models. In fact, the LLM (our reasoning engine) might have two kinds of limitations:
    - **Limited parametric knowledge.** This is due to the knowledge base cutoff date, which is a physiological feature of LLMs. In fact, their training dataset will always be "outdated," not in line with the current trends. This can be overcome by adding non-parametric knowledge with grounding, as previously seen.
    - **Lack of executive power.** This means that LLMs by themselves are not empowered to carry out actions. Let's consider, for example, the well-known ChatGPT: if we ask it to generate a LinkedIn post about productivity tips, we will then need to copy and paste it onto our LinkedIn profile as ChatGPT is not able to do so by itself. That is the reason why we need plug-ins. Plug-ins are LLMs' connectors toward the external world that serve not only as input sources to extend LLMs' non-parametric knowledge (for example, to allow a web search) but also as output sources so that the copilot can actually execute actions. For example, with a LinkedIn plug-in, our copilot powered by an LLM will be able not only to generate the post but also to post it online.

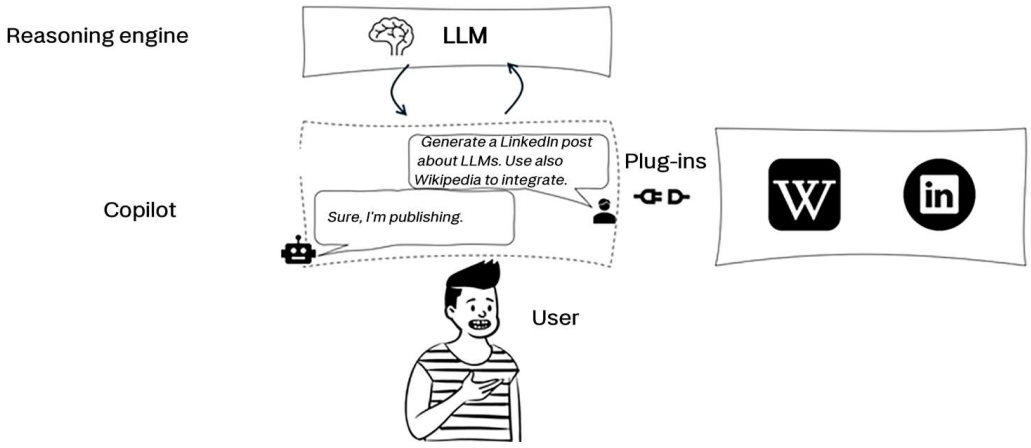

Figure 2.4: Example of Wikipedia and LinkedIn plug-ins

Note that the user's prompt in natural language is not the only input the model processes. In fact, it is a crucial component of the backend logic of our LLM-powered applications and the set of instructions we provide to the model. This *metaprompt* or system message is the object of a new discipline called **prompt engineering**.