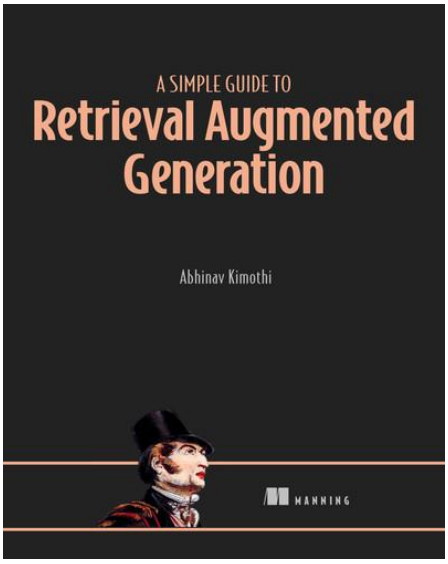

Chapter 2 RAG systems and their design

This chapter covers

- The concept and design of RAG systems
- An overview of the indexing pipeline
- An overview of the generation pipeline
- An initial look at RAG evaluation
- A high-level look at the RAG operations stack

The first chapter explored the core principles behind retrieval-augmented generation (RAG) and the large language model (LLM) challenges addressed by it. To construct a RAG system, several components need to be assembled. This process includes the creation and maintenance of the non-parametric memory, or a knowledge base, for the system. Another pipeline facilitates real-time interaction by sending the prompts to and accepting the response from the LLM, with retrieval and augmentation steps in the middle. Evaluation is yet another critical component, ensuring the effectiveness and accuracy of the system. All these components are supported by layers of the operations stack.

Chapter 2 discusses the design of a RAG system, examining the steps involved and the need for two different pipelines. We will call the pipeline that creates the knowledge base the "indexing pipeline." The other pipeline that allows real-time interaction with the LLM will be referred to as the "generation pipeline." We will discuss their individual components, such as data loading, embeddings, vector stores, retrievers, and more. Additionally, we will get an understanding of how the evaluation of RAG systems is conducted and introduce the RAG operations (RAGOps) stack that powers such systems.

This chapter will introduce you to various components discussed in detail in the coming chapters. By the end of chapter 2, you will have acquired a deep understanding of the components of a RAG system and will be ready to dive deep into the different components. By the end of the chapter, you should

- Be able to understand the several components of the RAG system design.
- Set yourself up for a deeper exploration of the indexing pipeline—the generation pipelines, RAG evaluation methods, and the RAGOps stack.

2.1 What does a RAG system look like?

By now, we have come to know that RAG is a vital component of the systems that use LLMs to solve their use cases. But, what is that system like? To illustrate, let's revisit the example used at the beginning chapter 1 ("Who won the 2023 Cricket World Cup?") and lay out the steps we undertook to enable ChatGPT to provide us with the accurate response.

The initial step was asking the question itself: "Who won the 2023 Cricket World Cup?" Following this, we manually searched for sources on the internet that might have information regarding the answer to the question. We found one (Wikipedia, in our example) and extracted a relevant paragraph from the source. Subsequently, we added the relevant paragraph to our original question, pasted the question and the retrieved paragraph together in the prompt to ChatGPT, and got a factually correct response: "Australia won the 2023 Cricket World Cup."

This process can be distilled into five steps, and our system needs to facilitate all of them:

1. User asks a question.
2. The system searches for information relevant to the input question.
3. The information relevant to the input question is fetched, or retrieved, and added to the input question.
4. This question and information are passed to an LLM.
5. The LLM responds with a contextual answer.

If you recall, we have already described this process in chapter 1. Let's visualize it in the context of these five steps as shown in figure 2.1. This workflow will be called the "generation pipeline" since it generates the answer.

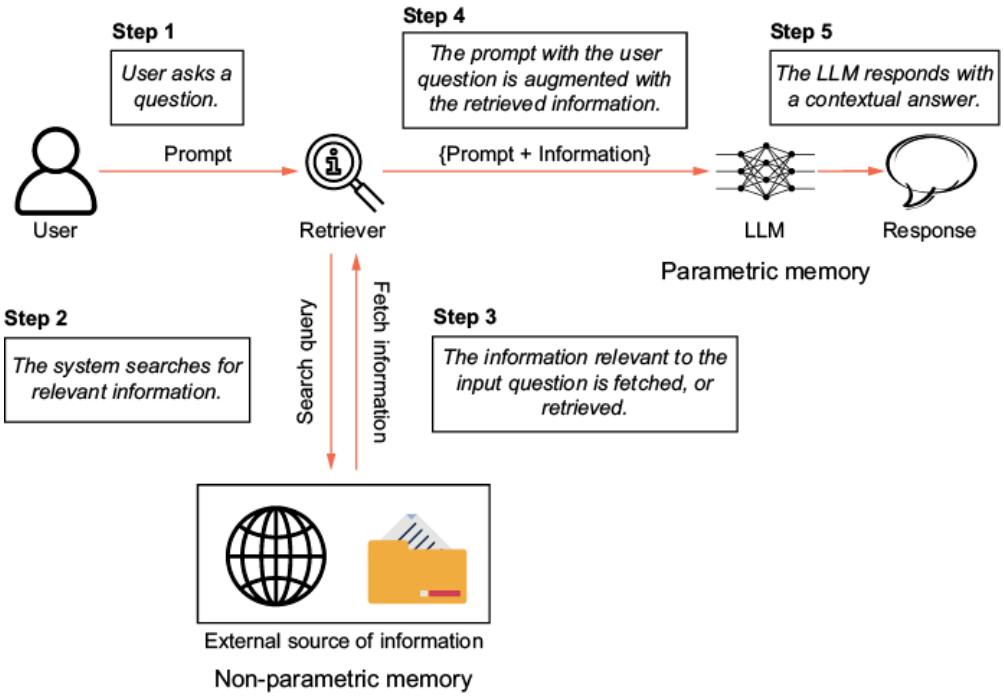

Figure 2.1 Generation pipeline covering the five RAG steps. The journey from query to the response involves search and retrieval, augmentation, and generation.

This pipeline enables real-time contextual interaction with the LLM. There are, of course, several intricacies in each of the five steps needed to create the generation pipeline. Some decisions need to be made about the design of the retriever and the LLM choice. The construction of prompts will also affect the quality of the response. We will discuss prompt construction in chapter 3. We first must address a critical pre-requisite step before this generation pipeline can be put in place. For that, some key questions regarding the external source of information need to be answered. We will also need to know, in advance, where to look and then establish connections to all these disparate sources:

- What is the location of the external source of information?
    - Is it the open internet? Or are there some documents in the company's internal data storage? Is the information present in some third-party databases? Are there multiple sources we want to use?
    - Why is this important?
- What is the nature of the information at the source?
    - Are these Word documents or PDF files? Is the information accessed via an API, and the response is in JSON format? Will we find answers in one document, or is the information distributed in multiple documents?
    - Why is this important?

We will also need to know the format and nature of data storage to be able to extract the information from the source files.

When data is stored across multiple sources, such as the internet and an internal data lake, the system must connect to each source, search for relevant information in various formats,

and organize it according to the original query. Every time a question is asked, this process of connecting, extracting, and parsing will have to be repeated. Information from different sources may lead to factual inconsistencies that will have to be resolved in real time. Searching through all the information might be prohibitively time-consuming. This will, therefore, prove to be a highly suboptimal, unscalable process that may not yield the desired results. A RAG system will work best if the information from different sources is

- Collected in a single location.
- Stored in a single format.
- Broken down into small pieces of information.

The need for a consolidated knowledge base arises from the disparate nature of external data sources. To address this requirement, we need to undertake a series of steps to create and maintain a well-structured knowledge base. This, again, is a five-step process:

1. Connect to previously identified external sources.
2. Extract documents and parse text from them.
3. Break down long pieces of text into smaller, manageable pieces.
4. Convert these small pieces into a suitable format.
5. Store this information.

These steps, which facilitate the creation of this knowledge base, form the *indexing pipeline*. The indexing pipeline is shown in figure 2.2.

In addition to creating the knowledge base, the indexing pipeline plays a crucial role in maintaining and updating it with the latest information to ensure its relevance and accuracy. Before the knowledge base is created by the indexing pipeline, there is nowhere for the generation pipeline to search for information. It is the indexing pipeline that lays the foundation for the subsequent operation of the generation pipeline. Therefore, setting up the indexing pipeline comes before the generation pipeline can be activated.

Together, these pipelines form the backbone of a RAG system, enabling seamless interaction with users and delivering contextually relevant responses. Figure 2.3 shows the indexing and generation pipelines working together to form the skeleton of a RAG system.

We have established the flow of a RAG system that includes two pipelines. Conceptually, this is the complete flow. However, to build such systems to be used in the real world, more components are required. The next section reimagines this flow along with other considerations and creates a design for RAG systems.

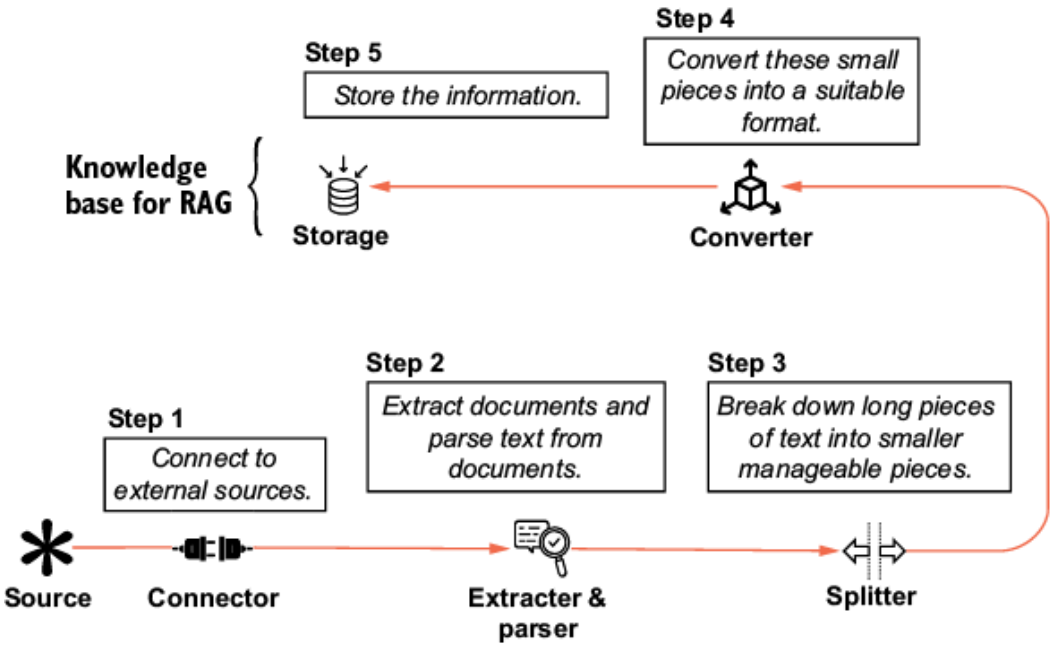

Figure 2.2 Indexing pipeline covering the steps to create the knowledge base for RAG. This involves connecting to the source, parsing, splitting, converting, and storing information.

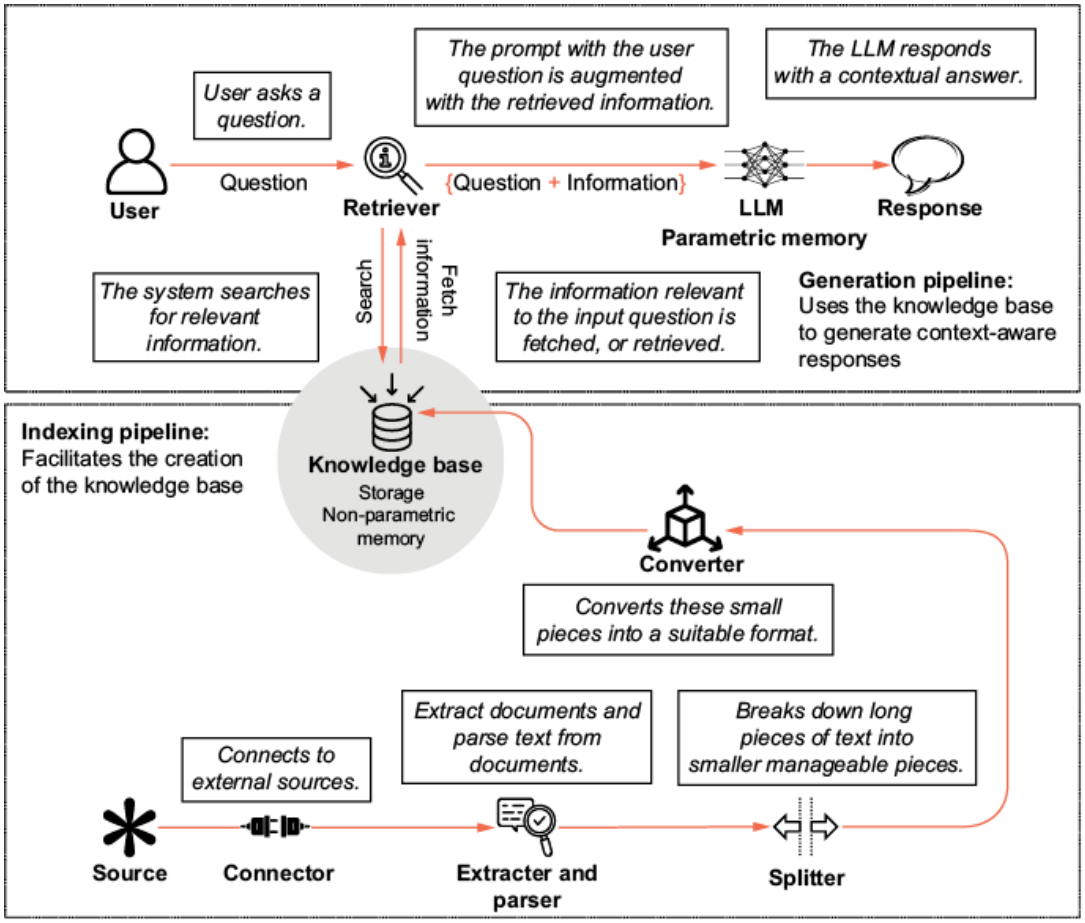

Figure 2.3 The indexing and generation pipelines together make a RAG system. The indexing pipeline is an offline process, while the generation pipeline facilitates real-time interaction with the knowledge base.

2.2 Design of RAG systems

We saw how RAG systems are created by the indexing and generation pipelines. These two pipelines include several parts themselves. Like all software applications, production-ready RAG systems require more than just the basic components. We need to think about accuracy, observability, scalability, and other important factors. This book discusses some of these components at length. Figure 2.4 presents a rough layout of a RAG system. Apart from the indexing and generation component, we'll add layers for infrastructure, security, evaluation, etc.

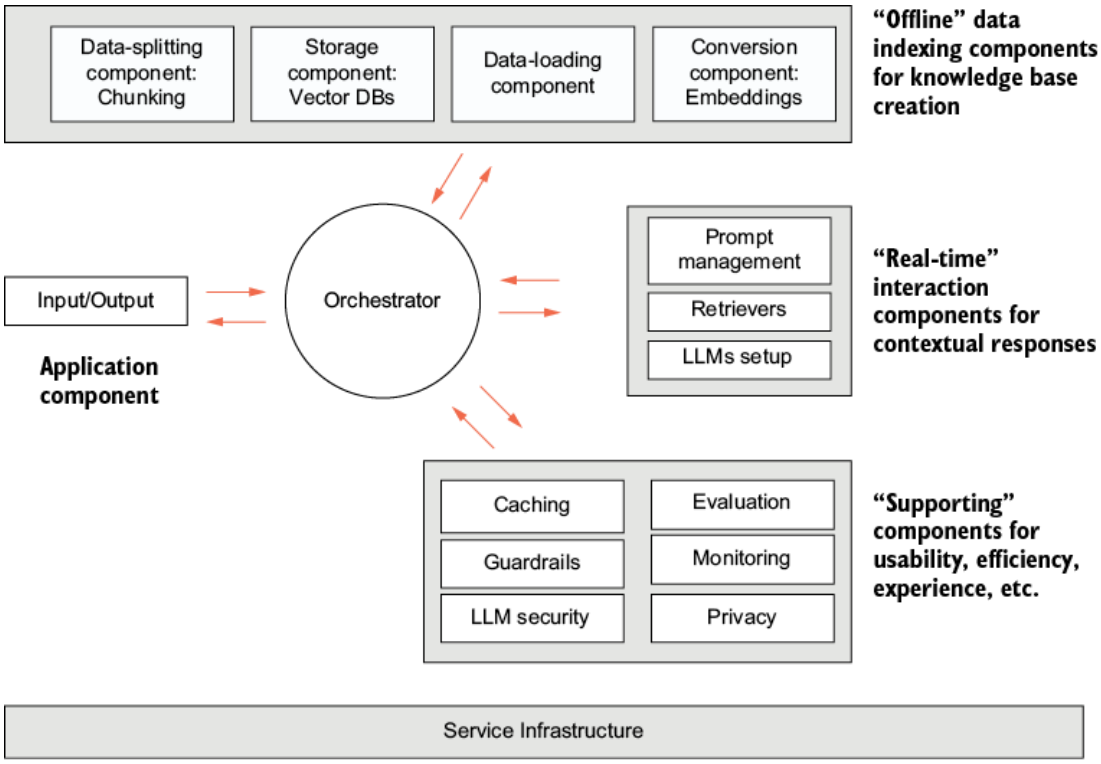

Figure 2.4 Components of a production-ready RAG system

Let's look at the main components of a RAG system. The first four components complete the indexing pipeline:

- *Data-loading component*—Connects to external sources, and extracts and parses data
- *Data-splitting component*—Breaks down large pieces of text into smaller, manageable parts
- *Data conversion component*—Converts text data into a more suitable format
- *Storage component*—Stores the data to create a knowledge base for the system

These next three components complete the generation pipeline:

- *Retrievers*—Responsible for searching and fetching information from the storage

- *LLM setup*—Responsible for generating the response to the input
- *Prompt management*—Enables the augmentation of the retrieved information to the original input

The evaluation component measures the accuracy and reliability of the system before and after deployment. The monitoring component tracks the performance of the RAG system and helps detect failures. Other components include caching, which helps store previously generated responses to expedite retrieval for similar queries; guardrails, to ensure compliance with policy, regulation, and social responsibility; and security, to protect LLMs against breaches such as prompt injection, data poisoning, and similar. All the layers are supported by a service infrastructure.

All these components are managed and controlled by a central orchestration layer, which is responsible for their interaction and sequencing. It provides a unified interface for managing and monitoring workflows and processes.

The following sections provide an overview of these components before we examine them in depth in subsequent chapters.

2.3 Indexing pipeline

We discussed how the indexing pipeline facilitates the creation of the knowledge base used in the real-time generation pipeline. For practical purposes, the indexing pipeline is an offline or asynchronous pipeline. What this means is that the indexing pipeline is not activated in real time when the user is asking a question. Rather, it creates the knowledge base in advance and updates it at predefined intervals. The indexing pipeline comprises four main components, as seen in figure 2.5.

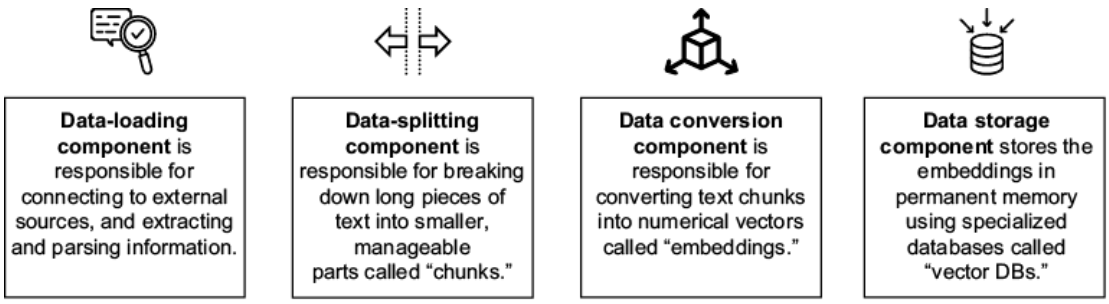

Figure 2.5 Four components of the indexing pipeline facilitate the creation of the knowledge base.

Let's delve deeper into each:

- *Data splitting (text splitting)*—Breaking down text into smaller segments enhances the system's ability to process and analyze information efficiently. These smaller pieces in natural language processing (NLP) parlance are commonly referred to as "chunks." The process of splitting large text documents into smaller chunks is called "chunking." We will discuss the need for chunking and various chunking strategies in chapter 3.
- *Data conversion (embeddings)*—Textual data must be converted to a numerical format for search and retrieval computations in RAG systems. There are different ways of implementing this conversion. For all practical purposes, a data format

called “embeddings” works best for search and retrieval. You will learn more about embeddings and different embedding models in chapter 3.

- *Data storage*—Once the data is ready in the desired format (embeddings), it needs to be stored in persistent (permanent) memory so that the real-time generation pipeline can access data whenever a user asks a question. Data is stored in specialized databases known as “vector databases,” which are best suited for search and retrieval of embeddings. Chapter 3 explores various vector databases and factors influencing their suitability for RAG systems.

Do you always need an indexing pipeline?

Offline indexing pipelines are typically used when a knowledge base with a large amount of data is built for repeated usage (e.g., many enterprise documents, manuals, etc.). However, there are some cases in which the generation pipeline connects to a third-party API to receive information related to the user question.

For example, imagine an application built for users seeking travel advice based on the weather forecast. An important component of this application will be fetching the weather details for the users’ location. Suppose the system uses a third-party API service that can respond with a location’s weather details when provided with the location in the input. This weather information is then passed to the LLM to generate the advice.

This application can also be thought of as a RAG system. But there is a difference. This system has outsourced the search and retrieval operation to the third-party API. It is the third party that maintains the data. For such systems, the indexing pipeline is not required to be built since the search and retrieval happens outside the system. Another example is applications that ask the user to input external information, like document summarizers. The search operation here is outsourced to the user.

Therefore, systems that use augment external information to the prompts but do not necessarily search and retrieve information themselves, do not warrant the creation of a knowledge base, and therefore, do not have an indexing pipeline. Some will argue that such systems are not RAG systems in the first place.

2.4 Generation pipeline

Building on the foundation established by the indexing pipeline, the generation pipeline facilitates real-time interactions in RAG systems. It is the generation pipeline that facilitates the retrieval, augmentation, and generation in the system. When a user asks a question, the generation pipeline processes the query, retrieves relevant information, and generates a response—all without the user directly interacting with the underlying indexing pipeline. The generation pipeline is enabled by three components, as seen in figure 2.6.

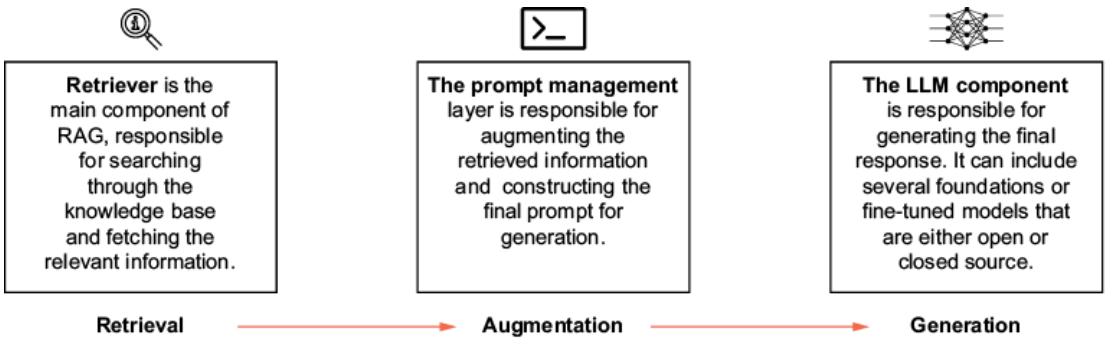

Figure 2.6 Three components of the generation pipeline enable the real-time query-response process of a RAG system.

Let's consider each of these in some more detail:

- *The retriever*—This is arguably the most critical component of the entire system. Using advanced search algorithms, the retriever scans the knowledge base to identify and retrieve the most relevant information based on the user's query. The overall effectiveness of the entire system relies heavily on the accuracy of the retriever. Also, search is a computationally heavy operation and may take time. Therefore, the retriever also contributes heavily to the overall latency of the system. We will discuss different retrievers and retrieval strategies in chapters 4 and 6.
- *Prompt management*—Once the relevant information is retrieved by the retriever, it needs to be combined, or augmented, with the original user query. Now, this may seem like a simple task at first glance. However, the construction of the prompt makes significant difference to the quality of the generated response. This component also falls in the gambit of prompt engineering. We will explore different prompting and prompt management strategies in chapter 4.
- *LLM setup*—At the end, LLMs are responsible for generating the final response. A RAG system may rely on more than one LLM. The LLMs can be the foundation (base) models that have been pretrained and generally available either open source, like those by Meta or Mistral, or through a managed service, like OpenAI or Anthropic. LLMs can also be fine-tuned for specific tasks. Fine-tuning involves training pre-existing LLMs on specific datasets or tasks to improve performance and adaptability for specialized applications. In rare cases, the developer may decide to train their LLMs. We will discuss LLMs in depth in chapter 4.

2.5 Evaluation and monitoring

Indexing and generation pipelines complete the system from a usage perspective. With these two pipelines in place, at least in theory, a user can start interacting with the system and get responses. However, in this case, we have no measure of the system quality. Is the system performing accurately, or is it still prone to hallucinations? Is the information that is being fetched by the retriever the most relevant to the query? To answer these questions, we have to put in place an evaluation framework. This framework helps in evaluating the quality of the system before it is released and then for continuous monitoring and improvement.

Building on the advancements of LLMs, RAG represents a recent innovation in NLP. Metrics such as relevance scores, recall, and precision are commonly used to evaluate the

effectiveness of RAG systems. One framework that intuitively guides a comprehensive evaluation is the triad of RAG metrics proposed by TruEra (https://mng.bz/Mw22). It looks at the RAG evaluation through three dimensions, as shown in figure 2.7.

The workflow involves checks in between each step—prompt, context, and answer. Let's take a closer look:

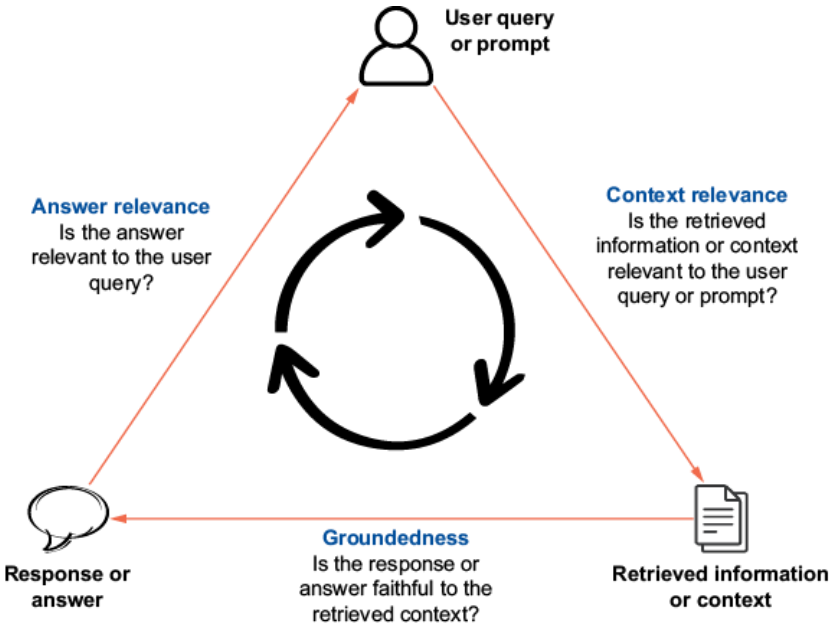

Figure 2.7 The triad of RAG evaluation proposed by TruEra. The three pivotal dimensions of RAG evaluation are the query, context, and response.

- *Between the retrieved information (context) and the user query (prompt)*—Is the information being searched and retrieved by the retriever the most relevant to the question the user has asked? The consequence of irrelevant information being retrieved is that no matter how good the LLM is, if the information being augmented is not good, the response will be suboptimal.
- *Between the final response (answer) and the retrieved information (context)*—Does the LLM consider all the retrieved information while generating responses? Even
- though RAG is aimed at reducing hallucinations, the system might still ignore the retrieved information. There are several reasons for it, which will be discussed in subsequent chapters.
- *Between the final response (answer) and the user query (prompt)*—Is the final response in line with the question the user had originally asked? To assess the overall effectiveness of the system, the relevance of the final response to the original question is necessary.

There are several metrics that help assess each of these three dimensions. For some of the metrics, a ground truth dataset is warranted. Ground truth datasets provide a benchmark for evaluating the accuracy and effectiveness of RAG systems by comparing generated responses to manually curated references. We will take a deeper look at these metrics and the ground truth dataset in chapter 5.

Continuous evaluation of metrics during live operation can identify the types of queries the system struggles to answer accurately. Qualitative feedback can also be collected from the user on the generated responses.

2.6 The RAGOps Stack

RAG, and LLM-based apps in general, are being powered by an evolving operations stack. Various providers offer infrastructure components such as data storage platforms, model hosting services, and application orchestration frameworks. The infrastructure can be understood in several layers:

1. *Data layer*—Tools and platforms used to process and store data in the form of embeddings
2. *Model layer*—Providers of proprietary or open source LLMs
3. *Prompt layer*—Tools offering maintenance and evaluation of prompts
4. *Evaluation layer*—Tools and frameworks providing evaluation metrics for RAG
5. *App orchestration*—Frameworks that facilitate invocation of different components of the system
6. *Deployment layer*—Cloud providers and platforms for deploying RAG apps
7. *Application layer*—Hosting services for RAG apps
8. *Monitoring layer*—Platforms offering continuous monitoring of RAG apps

Chapter 7 explores the various layers of infrastructure that support RAG systems.

2.7 Caching, guardrails, security, and other layers

Finally, there are certain other components frequently used in RAG systems. These components address the problems of system latency, regulatory and ethical compliances among other aspects.

- *Caching*—Caching is the process in which certain data is stored in cache memory for faster retrieval. LLM caching is slightly different from regular caching. The LLM responses to queries are stored in a semantic cache. Next time a similar query is asked, the response from the cache is retrieved instead of sending the query through the complete RAG pipeline. This approach improves the performance of the system by reducing the time it takes to respond, the cost of LLM inferencing, and the load on the LLM service.
- *Guardrails*—For several use cases, in practice, there will be a set of boundaries within which the output needs to be generated. Guardrails are a predefined set of rules added in the system to comply with policies, regulations, and ethical guidelines.
- *Security*—LLMs and LLM-based applications have witnessed new threats, such as prompt injections, data poisoning, sensitive information disclosure, and others. With evolving threats, the security infrastructure also needs to evolve to address concerns around security and data privacy of RAG systems.

RAGOps has also been evolving fast. Logging and tracing, model versioning, and feedback layers are some of the RAGOps stack components.

This chapter provided an overview of the key components of RAG systems, including the indexing and generation pipelines, evaluation and monitoring, and service infrastructure. By understanding these components, you are now equipped to delve deeper into each of these components and the intricacies of RAG systems in subsequent chapters. In the next chapter, we will start building the indexing pipeline to create a knowledge base of our RAG system.

Summary

- A RAG-enabled system consists of two main pipelines: the indexing and the generation pipeline.
- The indexing pipeline is responsible for creating and maintaining the knowledge base, which involves data loading, text splitting, data conversion (embeddings), and data storage in a vector database.
- The generation pipeline manages real-time interactions by retrieving information, augmenting queries, and generating responses using an LLM.
- Evaluation and monitoring are crucial components for the assessment of system performance, covering the relevance between the retrieved information and query, the final response and retrieved information, and the final response and the original query.
- The service infrastructure for RAG systems includes layers for data, models, prompts, evaluation, app orchestration, deployment, application hosting, and monitoring.
- Additional components such as caching, guardrails, and security measures are often employed to improve performance, ensure compliance, and address potential threats.