# Retrieval-Augmented Generation for Large Language Models: A Survey

**Yunfan Gao** [1], **Yun Xiong** [2], **Xinyu Gao** [2], **Kangxiang Jia** [2], **Jinliu Pan** [2], **Yuxi Bi** [3], **Yi Dai**[1], **Jiawei Sun**[1] and **Haofen Wang** [1,3] *

[1] Shanghai Research Institute for Intelligent Autonomous Systems, Tongji University
[2] Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
[3] College of Design and Innovation,Tongji University
gaoyunfan1602@gmail.com



## Abstract

Large language models (LLMs) demonstrate powerful capabilities, but they still face challenges in practical applications, such as hallucinations, slow knowledge updates, and lack of transparency in answers. Retrieval-Augmented Generation (RAG) refers to the retrieval of relevant information from external knowledge bases before answering questions with LLMs. RAG has been demonstrated to significantly enhance answer accuracy, reduce model hallucination, particularly for knowledge-intensive tasks. By citing sources, users can verify the accuracy of answers and increase trust in model outputs. It also facilitates knowledge updates and the introduction of domain-specific knowledge. RAG effectively combines the parameterized knowledge of LLMs with non-parameterized external knowledge bases, making it one of the most important methods for implementing large language models. This paper outlines the development paradigms of RAG in the era of LLMs, summarizing three paradigms: Naive RAG, Advanced RAG, and Modular RAG. It then provides a summary and organization of the three main components of RAG: retriever, generator, and augmentation methods, along with key technologies in each component. Furthermore, it discusses how to evaluate the effectiveness of RAG models, introducing two evaluation methods for RAG, emphasizing key metrics and abilities for evaluation, and presenting the latest automatic evaluation framework. Finally, potential future research directions are introduced from three aspects: vertical optimization, horizontal scalability, and the technical stack and ecosystem of RAG.[1]

## 1 Introduction

The large language models (LLMs) are more powerful than anything we have seen in Natural Language Processing (NLP) before. The GPT series models[Brown *et al.*, 2020, OpenAI, 2023], the LLama series models[Touvron *et al.*, 2023], Gemini[Google, 2023], and other large language models demonstrate impressive language and knowledge mastery, surpassing human benchmark levels in multiple evaluation benchmarks[Wang *et al.*, 2019, Hendrycks *et al.*, 2020, Srivastava *et al.*, 2022].

However, large language models also exhibit numerous shortcomings. They often fabricate facts[Zhang *et al.*, 2023b] and lack knowledge when dealing with specific domains or highly specialized queries[Kandpal *et al.*, 2023]. For instance, when the information sought extends beyond the model's training data or requires the latest data, LLM may fail to provide accurate answers. This limitation poses challenges when deploying generative artificial intelligence in real-world production environments, as blindly using a black-box LLM may not suffice.

Traditionally, neural networks adapt to specific domains or proprietary information by fine-tuning models to parameterize knowledge. While this technique yields significant results, it demands substantial computational resources, incurs high costs, and requires specialized technical expertise, making it less adaptable to the evolving information landscape. Parametric knowledge and non-parametric knowledge play distinct roles. Parametric knowledge is acquired through training LLMs and stored in the neural network weights, representing the model's understanding and generalization of the training data, forming the foundation for generated responses. Non-parametric knowledge, on the other hand, resides in external knowledge sources such as vector databases, not encoded directly into the model but treated as updatable supplementary information. Non-parametric knowledge empowers LLMs to access and leverage the latest or domain-specific information, enhancing the accuracy and relevance of responses.

Purely parameterized language models (LLMs) store their world knowledge, which is acquired from vast corpora, in the parameters of the model. Nevertheless, such models have their limitations. Firstly, it is difficult to retain all the knowledge from the training corpus, especially for less common and more specific knowledge. Secondly, since the model parameters cannot be updated dynamically, the parametric knowledge is susceptible to becoming outdated over time. Lastly, an expansion in parameters leads to increased com-

*Corresponding Author

[1]Resources are available at: https://github.com/Tongji-KGLLM/RAG-Survey

putational expenses for both training and inference. To address the limitations of purely parameterized models, language models can adopt a semi-parameterized approach by integrating a non-parameterized corpus database with parameterized models. This approach is known as Retrieval-Augmented Generation (RAG).

The term Retrieval-Augmented Generation (RAG) was first introduced by [Lewis *et al.*, 2020]. It combines a pre-trained retriever with a pre-trained seq2seq model (generator) and undergoes end-to-end fine-tuning to capture knowledge in a more interpretable and modular way. Before the advent of large models, RAG primarily focused on direct optimization of end-to-end models. Dense retrievals on the retrieval side, such as the use of vector-based Dense Passage Retrieval (DPR)[Karpukhin *et al.*, 2020], and training smaller models on the generation side are common practices. Due to the overall smaller parameter size, both the retriever and generator often undergo synchronized end-to-end training or fine-tuning[Izacard *et al.*, 2022].

After the emergence of LLM like ChatGPT, generative language models became predominant, showcasing impressive performance across various language tasks[Bai *et al.*, 2022, OpenAI, 2023, Touvron *et al.*, 2023, Google, 2023]. However, LLMs still face challenges such as hallucinations [Yao *et al.*, 2023, Bang *et al.*, 2023], knowledge updates, and data-related issues. This affects the reliability of LLMs, making them struggle in certain serious task scenarios, especially in knowledge-intensive tasks requiring access to a vast amount of knowledge, such as open-domain question answering[Chen and Yih, 2020, Reddy *et al.*, 2019, Kwiatkowski *et al.*, 2019] and commonsense reasoning[Clark *et al.*, 2019, Bisk *et al.*, 2020]. Implicit knowledge within parameters may be incomplete and insufficient.

Subsequent research found that introducing RAG into large models' In-Context Learning (ICL) can alleviate the aforementioned issues, with significant and easily implementable effects. During the inference process, RAG dynamically retrieves information from external knowledge sources, using the retrieved data as references to organize answers. This substantially improves the accuracy and relevance of responses, effectively addressing the hallucination issues present in LLMs. This technique quickly gained traction after the advent of LLM and has become one of the hottest technologies for improving chatbots and making LLM more practical. By separating factual knowledge from the training parameters of LLMs, RAG cleverly combines the powerful capabilities of generative models with the flexibility of retrieval modules, providing an effective solution to the incomplete and insufficient knowledge problem inherent in purely parameterized models.

The paper systematically reviews and analyzes the current research approaches and future development paths of RAG, summarizing them into three main paradigms: Naive RAG, Advanced RAG, and Modular RAG. Subsequently, the paper provides a consolidated summary of the three core components: Retrieval, Augmented, and Generation, highlighting the improvement directions and current technological characteristics of RAG. In the section on augmentation methods, the current work is organized into three aspects: the augmentation stages of RAG, augmentation data sources, and augmentation process. Furthermore, the paper summarizes the evaluation system, applicable scenarios, and other relevant content related to RAG. Through this article, readers gain a more comprehensive and systematic understanding of large models and retrieval-Augmented generation. They become familiar with the evolutionary path and key technologies of knowledge retrieval augment, enabling them to discern the advantages and disadvantages of different techniques, identify applicable scenarios, and explore current typical application cases in practice.It is noteworthy that in previous work, Feng el al.[2023b] systematically reviewed the methods, applications, and future trends of combining large models with knowledge, with a primary focus on knowledge editing and retrieval augmentation methods. Zhu et al.[2023] introduced the latest advancements in augmenting retrieval systems for Large Language Models, with a specific focus on the retrieval system. Meanwhile, Asai et al.[2023a] focusing on questions such as "What", "When", "How", analyzed and elucidated the key processes in Retrieval-based Language Models. In comparison with them, this paper aims to systematically outline the entire process of Retrieval-Augmented Generation (RAG) and focuses specifically on research related to augmenting the generation of large language models through knowledge retrieval.

The development of RAG algorithms and models is illustrated in Fig 1. On a timeline, most of the research related to RAG emerged after 2020, with a significant turning point in December 2022 when ChatGPT was released. Since the release of ChatGPT, research in the field of natural language processing has entered the era of large models. Naive RAG techniques quickly gained prominence, leading to a rapid increase in the number of related studies.In terms of enhancement strategies, research on reinforcement during the pre-training and supervised fine-tuning stages has been ongoing since the concept of RAG was introduced. However, most of the research on reinforcement during the inference stage emerged during the era of LLMs. This is primarily due to the high training costs associated with high-performance large models. Researchers have attempted to enhance model generation by incorporating external knowledge in a cost-effective manner through the inclusion of RAG modules during the inference stage. Regarding the use of augmented data, early RAG primarily focused on the application of unstructured data, particularly in the context of open-domain question answering. Subsequently, the range of knowledge sources for retrieval expanded, with the use of high-quality data as knowledge sources effectively addressing issues such as internalization of incorrect knowledge and hallucinations in large models. This includes structured knowledge, with knowledge graphs being a representative example. Recently, there has been increased attention on self-retrieval, which involves mining the knowledge of LLMs themselves to enhance their performance.

The subsequent chapters of this paper are structured as follows: Chapter 2 provides an introduction to the background of RAG.Chapter 3 introduces the mainstream paradigms of RAG.Chapter 4 analyzes the retriever in RAG.Chapter 5 fo-

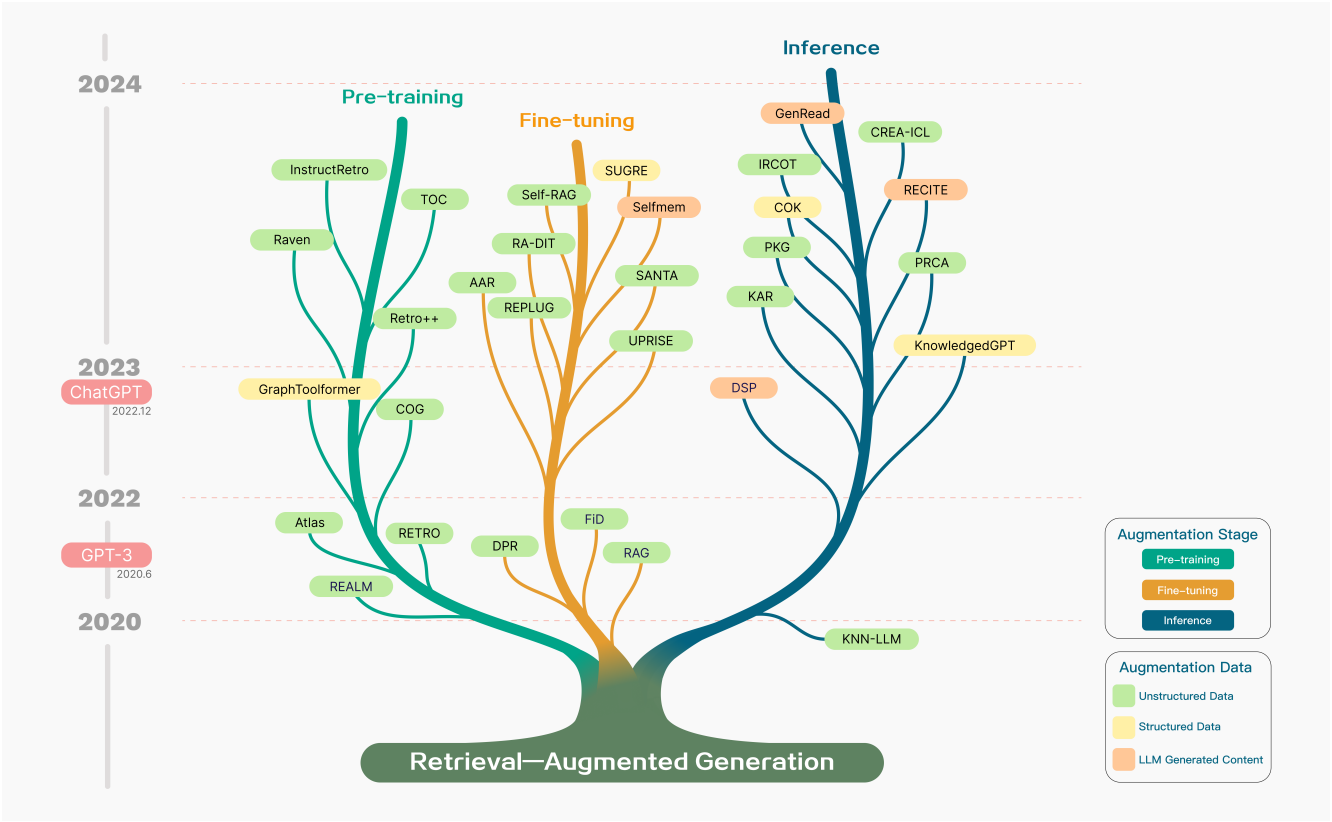


Figure 1: A timeline of existing RAG research. The timeline was established mainly according to the release date.

cuses on introducing the generator in RAG.Chapter 6 emphasizes the introduction of the augmentation methods in RAG.Chapter 7 introduces the evaluation system of RAG. Chapter 8 provides an outlook on the future development trends of RAG. Finally, in Chapter 9, we summarize the main contents of the survey.

## 2 Background

In this chapter, we will introduce the definition of RAG, as well as the comparison between RAG and other model optimization techniques, such as fine-tuning.

### 2.1 Definition

The meaning of RAG has expanded in tandem with technological developments. In the era of Large Language Models, the specific definition of RAG refers to the model, when answering questions or generating text, first retrieving relevant information from a vast corpus of documents. Subsequently, it utilizes this retrieved information to generate responses or text, thereby enhancing the quality of predictions. The RAG method allows developers to avoid the need for retraining the entire large model for each specific task. Instead, they can attach a knowledge base, providing additional information input to the model and improving the accuracy of its responses. RAG methods are particularly well-suited for knowledge-intensive tasks. In summary, the RAG system consists of two key stages:

1. Utilizing encoding models to retrieve relevant documents based on questions, such as BM25, DPR, ColBERT, and similar approaches[Robertson *et al.*, 2009, Karpukhin *et al.*, 2020, Khattab and Zaharia, 2020].
2. Generation Phase: Using the retrieved context as a condition, the system generates text.

### 2.2 RAG vs Fine-tuning

In the optimization of Large Language Models (LLMs), in addition to RAG, another important optimization technique is fine-tuning.

RAG is akin to providing a textbook to the model, allowing it to retrieve information based on specific queries. This approach is suitable for scenarios where the model needs to answer specific inquiries or address particular information retrieval tasks. However, RAG is not suitable for teaching the model to understand broad domains or learn new languages, formats, or styles.

Fine-tuning is similar to enabling students to internalize knowledge through extensive learning. This approach is useful when the model needs to replicate specific structures, styles, or formats. Fine-tuning can enhance the performance of non-fine-tuned models and make interactions more efficient. It is particularly suitable for emphasizing existing knowledge in the base model, modifying or customizing the model's output, and providing complex directives to the model. However, fine-tuning is not suitable for incorporating

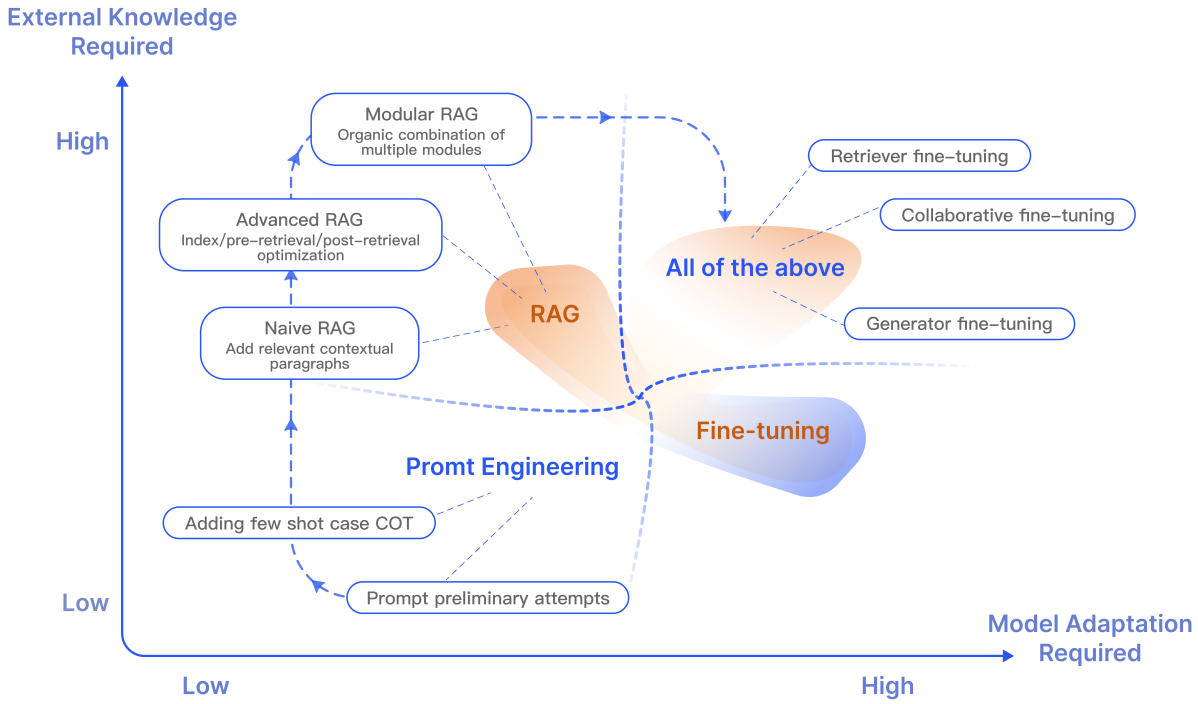


Figure 2: RAG compared with other model optimization methods

new knowledge into the model or for situations that demand quick iteration for new use cases.

Fine-tuning is similar to having students internalize knowledge through prolonged learning. This method is applicable when the model needs to replicate specific structures, styles, or formats. Fine-tuning can achieve performance superior to non-fine-tuned models, and interactions are more efficient. Fine-tuning is particularly suitable for emphasizing existing knowledge in the base model, modifying or customizing the model's output, and instructing the model with complex directives. However, fine-tuning is not suitable for adding new knowledge to the model or for scenarios that require rapid iteration for new use cases. The specific comparison between RAG and Fine-tuning (FT) can be elucidated in Table 1.

RAG and fine-tuning are not mutually exclusive but can complement each other, enhancing the model's capabilities at different levels. In certain situations, combining these two techniques can achieve optimal model performance. The entire process of optimizing with RAG and fine-tuning may require multiple iterations to achieve satisfactory results.

Existing research has demonstrated significant advantages of Retrieval-Augmented Generation (RAG) compared to other methods for optimizing large language models[Shuster *et al.*, 2021, Yasunaga *et al.*, 2022, Wang *et al.*, 2023c, Borgeaud *et al.*, 2022]:

- RAG improves accuracy by associating answers with external knowledge, reducing hallucination issues in language models and making generated responses more accurate and reliable.
- The use of retrieval techniques allows the identification of the latest information. Compared to traditional language models relying solely on training data, RAG maintains the timeliness and accuracy of responses.
- Transparency is an advantage of RAG. By citing sources, users can verify the accuracy of the answers, increasing trust in the model's output.
- RAG has customization capabilities. Models can be tailored to different domains by indexing relevant textual corpora, providing knowledge support for specific fields.
- In terms of security and privacy management, RAG, with its built-in roles and security controls in the database, can better control data usage. In contrast, fine-tuned models may lack clear management of who can access which data.
- RAG is more scalable. It can handle large-scale datasets without the need to update all parameters and create training sets, making it more economically efficient.
- Lastly, results produced by RAG are more trustworthy. RAG selects deterministic results from the latest data, while fine-tuned models may exhibit hallucinations and inaccuracies when dealing with dynamic data, lacking transparency and credibility.

## 3 RAG Framework

The research paradigm of RAG is constantly evolving. This chapter primarily introduces the evolution of the RAG research paradigm. We categorize it into three types: Naive RAG, Advanced RAG, and Modular RAG. Although the early RAG was cost-effective and performed better than the native LLM, it still faced many shortcomings. The emergence

| Feature Comparison | RAG | Fine-tuning |
|---|---|---|
| Knowledge Updates | Directly updates the retrieval knowledge base, ensuring information remains current without the need for frequent retraining, suitable for dynamic data environments. | Stores static data, requiring retraining for knowledge and data updates. |
| External Knowledge | Proficient in utilizing external resources, particularly suitable for documents or other structured/unstructured databases. | Can be applied to align the externally learned knowledge from pretraining with large language models, but may be less practical for frequently changing data sources. |
| Data Processing | Requires minimal data processing and handling. | Relies on constructing high-quality datasets, and limited datasets may not yield significant performance improvements. |
| Model Customization | Focuses on information retrieval and integrating external knowledge but may not fully customize model behavior or writing style. | Allows adjustments of LLM behavior, writing style, or specific domain knowledge based on specific tones or terms. |
| Interpretability | Answers can be traced back to specific data sources, providing higher interpretability and traceability. | Like a black box, not always clear why the model reacts a certain way, with relatively lower interpretability. |
| Computational Resources | Requires computational resources to support retrieval strategies and technologies related to databases. External data source integration and updates need to be maintained. | Preparation and curation of high-quality training datasets, definition of fine-tuning objectives, and provision of corresponding computational resources are necessary. |
| Latency Requirements | Involves data retrieval, potentially leading to higher latency. | LLM after fine-tuning can respond without retrieval, resulting in lower latency. |
| Reducing Hallucinations | Inherently less prone to hallucinations as each answer is grounded in retrieved evidence. | Can help reduce hallucinations by training the model based on specific domain data but may still exhibit hallucinations when faced with unfamiliar input. |
| Ethical and Privacy Issues | Ethical and privacy concerns arise from storing and retrieving text from external databases. | Ethical and privacy concerns may arise due to sensitive content in the training data. |

Table 1: Comparison between RAG and Fine-tuning

of Advanced RAG and Modular RAG were aimed at addressing specific deficiencies in the Naive RAG.

### 3.1 Naive RAG

The Naive RAG research paradigm represents the earliest methodology gained prominence shortly after the widespread adoption of ChatGPT. The naive RAG involves traditional process: indexing, retrieval, and generation. Naive RAG is also summarized as a "Retrieve"-"Read" framework [Ma *et al.*, 2023a].

**Indexing**

The pipeline for obtaining data from the source and building an index for it generally occurs in an offline state. Specifically, the construction of a data index involves the following steps:

**1.Data Indexing:**This involves cleaning and extracting the original data, converting different file formats such as PDF, HTML, Word, Markdown, etc., into plain text.

**2.Chunking:** This involves dividing the loaded text into smaller chunks. This is necessary because language models typically have a limit on the amount of context they can handle, so it is necessary to create as small text chunks as possible.

**3. Embedding and Creating Index:** This is the process of encoding text into vectors through a language model. The resulting vectors will be used in the subsequent retrieval process to calculate the similarity between the vector and the problem vector.The embedding models require a high inference speed. Since it is necessary to encode a large amount of corpus and encode the problem in real time when the user asks a question,

the parameter size of the model should not be too large.After generating the embedding, the next step is to create an index, storing the original corpus chunks and embedding in the form of key-value pairs for quick and frequent searches in the future.

**Retrieve**

Given a user's input, the same encoding model as in the first stage is used to convert the query into a vector. The similarity between the question embedding and the embedding of the document blocks in the corpus is calculated. The top K document blocks are chosen as the augmented context information for the current question based on the level of similarity.

**Generation**

The given question and related documents are combined into a new prompt. The large language model is then tasked with answering the question based on the provided information. It may be decided whether to allow the large model to use its knowledge or only to answer based on the given information, depending on the needs of different tasks. If there is historical dialogue information, it can also be merged into the prompt for multi-round dialogues.

**Drawbacks in Naive RAG**

The Naive RAG confronts principal challenges in three areas: retrieval quality, response generation quality, and the augmentation process.

Regarding retrieval quality, the issues are multifaceted. The primary concern is low precision, where not all blocks within the retrieval set correlate with the query, leading to potential hallucination and mid-air drop issues. A secondary issue is low recall, which arises when not all relevant blocks are retrieved, thereby preventing the LLM from obtaining sufficient context to synthesize an answer. Additionally, outdated information presents another challenge, where data redundancy or out-of-date data can result in inaccurate retrieval outcomes.

In terms of response generation quality, the issues are equally diverse. Hallucination is a prominent issue where the model fabricates an answer that doesn't exist in the context. Irrelevance is another concern where the model generates an answer that fails to address the query. Further, toxicity or bias, where the model generates a harmful or offensive response, is another problem.

Finally, the augmentation process also faces several challenges. Crucially, the effective integration of the context from retrieved passages with the current generation task is of utmost importance. If mishandled, the output might appear incoherent or disjointed. Redundancy and repetition are another issue, particularly when multiple retrieved passages contain similar information, leading to content repetition in the generation step. Moreover, determining the importance or relevance of multiple retrieved passages to the generation task is challenging, and the augmentation process needs to balance the value of each passage appropriately. The retrieved content may also come from different writing styles or tones, and the augmentation process needs to reconcile these differences to ensure output consistency. Lastly, generation models may overly rely on augmented information, resulting in output that merely repeats the retrieved content, without providing new value or synthesized information.

## 3.2 Advanced RAG

Advanced RAG has made targeted improvements to overcome the deficiencies of Naive RAG. In terms of the quality of retrieval generation, Advanced RAG has incorporated pre-retrieval and post-retrieval methods. To address the indexing issues encountered by Naive RAG, Advanced RAG has optimized indexing through methods such as sliding window, fine-grained segmentation, and metadata. Concurrently, it has put forward various methods to optimize the retrieval process. In terms of specific implementation, Advanced RAG can be adjusted either through a pipeline or in an end-to-end manner.

**Pre-Retrieval Process**

- **Optimizing Data Indexing**

  The purpose of optimizing data indexing is to enhance the quality of indexed content. Currently, there are five main strategies employed for this purpose: increasing the granularity of indexed data, optimizing index structures, adding metadata, alignment optimization, and mixed retrieval.

  1. **Enhancing Data Granularity:** The objective of pre-index optimization is to improve text standardization, consistency, and ensure factual accuracy and contextual richness to guarantee the performance of the RAG system. Text standardization involves removing irrelevant information and special characters to enhance the efficiency of the retriever. In terms of consistency, the primary task is to eliminate ambiguity in entities and terms, along with eliminating duplicate or redundant information to simplify the retriever's focus. Ensuring factual accuracy is crucial, and whenever possible, the accuracy of each piece of data should be verified. Context retention, to adapt to the system's interaction context in the real world, can be achieved by adding another layer of context with domain-specific annotations, coupled with continuous updates through user feedback loops. Time sensitivity is essential contextual information, and mechanisms should be designed to refresh outdated documents. In summary, the focus of optimizing indexed data should be on clarity, context, and correctness to make the system efficient and reliable. The following introduces best practices.
  2. **Optimizing Index Structures:** This can be achieved by adjusting the size of the chunks, altering the index paths, and incorporating graph structure information. The method of adjusting chunks (Small to Big) involves collecting as much relevant context as possible and minimizing noise. When constructing a RAG system, the chunk size is a key parameter. There are different evaluation frameworks comparing the size of individual chunks. LlamaIndex[2] uses GPT4 to assess fidelity and rele-

[2]https://www.llamaindex.ai

vance, and the LLaMA[Touvron *et al.*, 2023] index has an automatic evaluation feature for different chunking methods. The method of querying across multiple index paths is closely related to previous metadata filtering and chunking methods, and may involve querying across different indexes simultaneously. A standard index can be used to query specific queries, or a standalone index can be used to search or filter based on metadata keywords, such as a specific "date" index.

Introducing a graph structure involves transforming entities into nodes and their relationships into relations. This can improve accuracy by leveraging the relationships between nodes, especially for multi-hop questions. Using a graph data index can increase the relevance of the retrieval.

3. **Adding Metadata Information:** The focus here is to embed referenced metadata into chunks, such as dates and purposes used for filtering. Adding metadata like chapters and subsections of references could also be beneficial for improving retrieval. When we divide the index into numerous chunks, retrieval efficiency becomes a concern. Filtering through metadata first can enhance efficiency and relevance.
4. **Alignment Optimization:** This strategy primarily addresses alignment issues and differences between documents. The alignment concept involves introducing *hypothetical questions*, creating questions which are suitable to answer with each document, and embedding (or replacing) these questions with the documents. This helps address alignment problems and discrepancies between documents.
5. **Mixed Retrieval:** The advantage of this strategy lies in leveraging the strengths of different retrieval technologies. Intelligently combining various techniques, including keyword-based search, semantic search, and vector search, adapts to different query types and information needs, ensuring consistent retrieval of the most relevant and context-rich information. Mixed retrieval can serve as a robust complement to retrieval strategies, enhancing the overall performance of the RAG pipeline.

**Embedding**

- **Fine-turning Embedding:** Fine-tuning embedding models directly impacts the effectiveness of RAG. The purpose of fine-tuning is to enhance the relevance between retrieved content and query. The role of fine-tuning embedding is akin to adjusting ears before generating speech, optimizing the influence of retrieval content on the generated output. Generally, methods for fine-tuning embedding fall into the categories of adjusting embedding in domain-specific contexts and optimizing retrieval steps. Especially in professional domains dealing with evolving or rare terms, these customized embedding methods can improve retrieval relevance. The BGE[BAAI, 2023]embedding model is a fine-tunning and high-performance embedding model, such as BGE-large-EN developed by the BAAI [3]. To create training data for fine-tuning the BGE model, start by using LLMs like gpt-3.5-turbo to formulate questions based on document chunks, where questions and answers (document chunks) form fine-tuning pairs for the fine-tuning process.
- **Dynamic Embedding:** Dynamic embedding adjust based on the context in which words appear, differing from static embedding that use a single vector for each word. For instance, in transformer models like BERT, the same word can have varied embeddings depending on surrounding words. Evidence indicates unexpected high cosine similarity results, especially with text lengths less than 5 tokens, in OpenAI's text-embedding-ada-002 model[4]. Ideally, embedding should encompass as much context as possible to ensure "healthy" outcomes.Built upon the principles of large language models like GPT, OpenAI's embeddings-ada-02 is more sophisticated than static embedding models, capturing a certain level of context. While it excels in contextual understanding, it may not exhibit the same sensitivity to context as the latest full-size language models like GPT-4.

**Post-Retrieval Process**

After retrieving valuable context from the database, merging it with the query for input into LLM poses challenges. Presenting all relevant documents to the LLM at once may exceed the context window limit. Concatenating numerous documents to form a lengthy retrieval prompt is ineffective, introducing noise and hindering the LLM's focus on crucial information. Additional processing of the retrieved content is necessary to address these issues.

- **ReRank:** Re-ranking to relocate the most relevant information to the edges of the prompt is a straightforward idea. This concept has been implemented in frameworks such as LlamaIndex, LangChain, and HayStack [Blagojevi, 2023]. For instance, Diversity Ranker prioritizes reordering based on document diversity, while LostInTheMiddleRanker alternates placing the best document at the beginning and end of the context window. Simultaneously, addressing the challenge of interpreting vector-based simulated searches for semantic similarity, approaches like cohereAI rerank [Cohere, 2023], bge-rerank[5], or LongLLMLingua [Jiang *et al.*, 2023a] recalculate the semantic similarity between relevant text and query.
- **Prompt Compression** Research indicates that noise in retrieved documents adversely affects RAG performance. In post-processing, the emphasis lies in compressing irrelevant context, highlighting pivotal paragraphs, and reducing the overall context length. Approaches such as Selective Context[Litman *et al.*, 2020] and LLMLingua [Anderson *et al.*, 2022]utilize small

[3]https://huggingface.co/BAAI/bge-large-en

[4]https://platform.openai.com/docs/guides/embeddings

[5]https://huggingface.co/BAAI/bge-reranker-large

language models to calculate prompt mutual information or perplexity, estimating element importance. However, these methods may lose key information in RAG or long-context scenarios. Recomp [Xu *et al.*, 2023a]addresses this by training compressors at different granularities. Long Context [Xu *et al.*, 2023b], in dealing with extensive contexts, decomposes and compresses, while "Walking in the Memory Maze" [Chen *et al.*, 2023a]designs a hierarchical summarization tree to enhance LLM's key information perception.

**RAG Pipeline Optimization**

The optimization of the retrieval process aims to enhance the efficiency and information quality of RAG systems, Current research primarily focuses on intelligently combining various search technologies, optimizing retrieval steps, introducing the concept of cognitive backtracking, flexibly applying diverse query strategies, and leveraging embedding similarity. These efforts collectively strive to achieve a balance between efficiency and the richness of contextual information in RAG retrieval.

- **Exploring Hybrid Search:** By intelligently blending various techniques such as keyword-based search, semantic search, and vector search, the RAG system can leverage the strengths of each method. This approach enables the RAG system to adapt to different query types and information needs, ensuring consistent retrieval of the most relevant and context-rich information. Hybrid search serves as a robust complement to retrieval strategies, enhancing the overall performance of the RAG pipeline.
- **Recursive Retrieval and Query Engine:**Another powerful method to optimize retrieval in the RAG system involves implementing recursive retrieval and a sophisticated query engine. Recursive retrieval entails acquiring smaller document blocks during the initial retrieval phase to capture key semantic meanings. In the later stages of this process, larger blocks with more contextual information are provided to the language model (LM). This two-step retrieval method helps strike a balance between efficiency and contextually rich responses.
- **StepBack-prompt:** Integrated into the RAG process, the StepBack-prompt approach[Zheng *et al.*, 2023] encourages LLM to step back from specific instances and engage in reasoning about the underlying general concepts or principles. Experimental findings indicate a significant performance improvement in various challenging, inference-intensive tasks with the incorporation of backward prompts, showcasing its natural adaptability to RAG. The retrieval-enhancing steps can be applied in both the generation of answers to backward prompts and the final question-answering process.
- **Subqueries:**Various query strategies can be employed in different scenarios, including using query engines provided by frameworks like LlamaIndex, employing tree queries, utilizing vector queries, or employing the most basic sequential querying of chunks.
- **HyDE:** This approach is grounded on the assumption that the generated answers may be closer in the embedding space than a direct query. Utilizing LLM, HyDE generates a hypothetical document (answer) in response to a query, embeds the document, and employs this embedding to retrieve real documents similar to the hypothetical one. In contrast to seeking embedding similarity based on the query, this method emphasizes the embedding similarity from answer to answer. However, it may not consistently yield favorable results, particularly in instances where the language model is unfamiliar with the discussed topic, potentially leading to an increased generation of error-prone instances.

**Modular RAG**

The modular RAG structure breaks away from the traditional Naive RAG framework of indexing, retrieval, and generation, offering greater diversity and flexibility in the overall process. On one hand, it integrates various methods to expand functional modules, such as incorporating a search module in similarity retrieval and applying a fine-tuning approach in the retriever[Lin *et al.*, 2023]. Additionally, specific problems have led to the emergence of restructured RAG modules [Yu *et al.*, 2022] and iterative approaches like [Shao *et al.*, 2023]. The modular RAG paradigm is becoming the mainstream in the RAG domain, allowing for either a serialized pipeline or an end-to-end training approach across multiple modules.The comparison between three RAG paradigms is illustrated in Fig 3.

**New Modules**

- **Search Module:** Diverging from the similarity retrieval between queries and corpora in Naive/Advanced RAG, the search module, tailored to specific scenarios, incorporates direct searches on (additional) corpora in the process using LLM-generated code, query languages (e.g., SQL, Cypher), or other custom tools. The data sources for searching can include search engines, text data, tabular data, or knowledge graphs[Wang *et al.*, 2023c].
- **Memory Module:** Leveraging the memory capabilities of LLM itself to guide retrieval, the principle involves finding memories most similar to the current input. Self-mem [Cheng *et al.*, 2023b]iteratively employs a retrieval-enhanced generator to create an unbounded memory pool, combining the "original question" and "dual question." A retrieval-enhanced generative model can use its own outputs to enhance itself, making the text closer to the data distribution in the reasoning process, with the model's own outputs rather than training data[Wang *et al.*, 2022a].
- **Extra Generation Module:** In retrieved content, redundancy and noise are common issues. Instead of directly retrieving from a data source, the Extra Generation Module leverages LLM to generate the required context[Yu *et al.*, 2022]. Content generated by LLM is more likely to contain relevant information compared to direct retrieval.

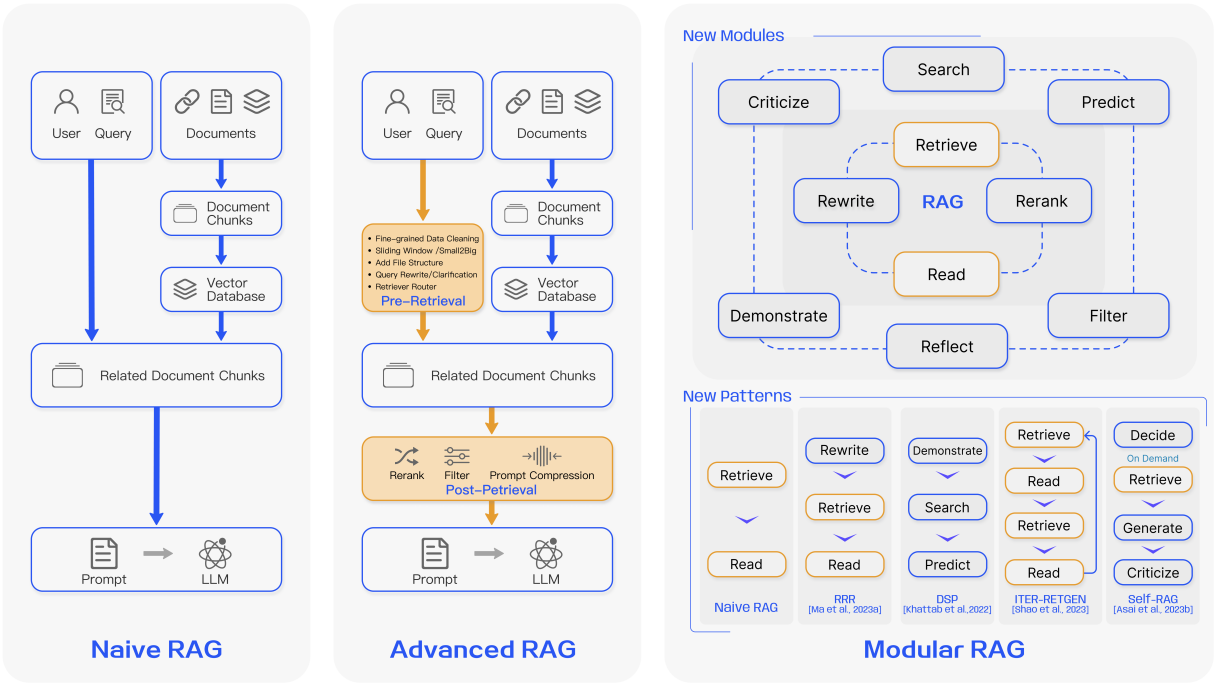


Figure 3: Comparison between the three paradigms of RAG

- **Task Adaptable Module:** Focused on transforming RAG to adapt to various downstream tasks, UPRISE[Cheng *et al.*, 2023a] automatically retrieves prompts for given zero-shot task inputs from a pre-constructed data pool, enhancing universality across tasks and models. PROMPTAGATOR[Dai *et al.*, 2022]utilizes LLM as a few-shot query generator and, based on the generated data, creates task-specific retrievers. Leveraging the generalization capability of LLM, PROMPTAGATOR enables the creation of task-specific end-to-end retrievers with just a few examples.
- **Alignment Module:** The alignment between queries and texts has consistently been a critical issue influencing the effectiveness of RAG. In the era of Modular RAG, researchers have discovered that adding a trainable Adapter module to the retriever can effectively mitigate alignment issues. PRCA[Yang *et al.*, 2023b] leveraged reinforcement learning to train a context adapter driven by LLM rewards, positioned between the retriever and generator. It optimizes the retrieved information by maximizing rewards in the reinforcement learning phase within the labeled autoregressive policy. AAR[Yu *et al.*, 2023b]proposed a universal plugin that learns LM preferences from known-source LLMs to assist unknown or non-co-finetuned LLMs. RRR[Ma *et al.*, 2023a]designed a module for rewriting queries based on reinforcement learning to align queries with documents in the corpus.
- **Validation Module:** In real-world scenarios, it is not always guaranteed that the retrieved information is reliable. Retrieving irrelevant data may lead to the occurrence of illusions in LLM. Therefore, an additional validation module can be introduced after retrieving documents to assess the relevance between the retrieved documents and the query. This enhances the robustness of RAG[Yu *et al.*, 2023a].

**New Pattern**

The organizational approach of Modular RAG is flexible, allowing for the substitution or reconfiguration of modules within the RAG process based on specific problem contexts. For Naive RAG, which consists of the two modules of retrieval and generation ( referred as read or sythesis in some literature), this framework offers adaptability and abundance. Present research primarily explores two organizational paradigms, involving the addition or replacement of modules, as well as the adjustment of the organizational flow between modules.

- **Adding or Replacing Modules**

  The strategy of adding or replacing modules entails maintaining the structure of Retrieval-Read while introducing additional modules to enhance specific functionalities. RRR[Ma *et al.*, 2023a] proposes the Rewrite-Retrieve-Read process, utilizing LLM performance as a reward in reinforcement learning for a rewritter module. This allows the rewritter to adjust retrieval queries, improving the downstream task performance of the reader. Similarly, modules can be selectively replaced in approaches like Generate-Read[Yu *et al.*, 2022], where the LLM generation module replaces the retrieval module.

Recite-Read [Sun *et al.*, 2022] transforms external retrieval into retrieval from model weights, initially having LLM memorize task-relevant information and generate output for handling knowledge-intensive natural language processing tasks.

- **Adjusting the Flow between Modules** In the realm of adjusting the flow between modules, there is an emphasis on enhancing interaction between language models and retrieval models. DSP[Khattab *et al.*, 2022] introduces the Demonstrate-Search-predict framework, treating the context learning system as an explicit program rather than a terminal task prompt to address knowledge-intensive tasks. ITER-RETGEN [Shao *et al.*, 2023] utilizes generated content to guide retrieval, iteratively performing "retrieval-enhanced generation" and "generation-enhanced retrieval" in a Retrieve-Read-Retrieve-Read flow. Self-RAG[Asai *et al.*, 2023b] follows the decide-retrieve-reflect-read process, introducing a module for active judgment. This adaptive and diverse approach allows for the dynamic organization of modules within the Modular RAG framework.

# 4 Retriever

In the context of RAG, the "R" stands for retrieval, serving the role in the RAG pipeline of retrieving the top-k relevant documents from a vast knowledge base. However, crafting a high-quality retriever is a non-trivial task. In this chapter, we organize our discussions around three key questions: 1) How to acquire accurate semantic representations? 2) How to match the semantic spaces of queries and documents? 3) How to align the output of the retriever with the preferences of the Large Language Model ?

## 4.1 How to acquire accurate semantic representations?

In RAG, semantic space is the multidimensional space where query and Document are mapped. When we perform retrieval, it is measured within the semantic space. If the semantic expression is not accurate, then its effect on RAG is fatal, this section will introduce two methods to help us build a accurate semantic space.

**Chunk optimization**

When processing external documents, the first step is chunking to obtain fine-grained features. Then the chunks are Embedded. However, Embedding too large or too small text chunks may not achieve good results. Therefore, finding the optimal chunk size for the documents in the corpus is crucial to ensure the accuracy and relevance of the search results.

When choosing a chunking strategy, important considerations include: the characteristics of the content being indexed, the embedding model used and its optimal block size, the expected length and complexity of user queries, and how the retrieval results are used in a specific application. For example, different chunking models should be selected for longer or shorter content. Additionally, different embedding models perform differently at different block sizes; for example, sentence-transformer is more suitable for single sentences, while text-embedding-ada-002 is better for blocks containing 256 or 512 tokens. Furthermore, the length and complexity of the user's input question text, as well as the specific needs of your application such as semantic search or Q&A, will all affect the choice of chunking strategy. This might directly correlate with the token limits of your chosen LLM, and may require you to adjust the block size. In fact, accurate query results are achieved by adaptively applying several chunking strategies; there is no best, only most suitable.

Current research in RAG employs diverse block optimization methods to improve retrieval efficiency and accuracy. Techniques such as sliding window technology implement layered retrieval by aggregating globally related information through multiple retrievals. The Small2big technique utilizes small text blocks during the search process and provides larger affiliated text blocks to the language model for processing. The Abstract embedding technique performs Top K retrieval on document abstracts, offering full document context. The Metadata Filtering technique leverages document metadata for filtering. The Graph Indexing technique converts entities and relationships into nodes and connections, significantly enhancing relevance in the context of multi-hop issues. The amalgamation of these methods has resulted in improved retrieval outcomes and enhanced performance for RAG.

**Fine-tuning Embedding Models**

After getting the proper size of Chunks, we need to Embedding the chunks and query in the semantic space by an Embedding model, so it is crucial whether Embedding can represent the corpus effectively. Nowadays, excellent Embedding models have appeared, such as [UAE[AngIE, 2023], Voyage[VoyageAI, 2023], BGE[BAAI, 2023], etc.], they have been pre-trained on large-scale corpus, but they may not accurately represent domain-specific corpus information when applied to specific domains. Furthermore, task-specific fine-tuning of Embedding models is critical to ensure that the model understands the user query in relation to the content relevance, whereas an un-fine-tuned model may not be able to fulfill the needs of a specific task. Thus, fine-tuning an Embedding model is essential for downstream applications. There are two basic paradigms in Embedding fine-tuning methods

**Domain Knowledge Fine-tuning** In order for an Embedding model to correctly understand domain-specific information, we need to construct domain-specific datasets to fine-tune the Embedding model. However fine-tuning an Embedding model is different from an ordinary language model, mainly in that the datasets used are different. In the current main method of fine-tuning Embedding models, the dataset used consists of three parts, including Queries, Corpus and Relevant Docs. The Embedding model looks up relevant documents in Corpus based on the Query, and then whether the Relevant Docs of the query hit or not is used as a metric for the model.

In the construction of datasets, fine-tuning models, and evaluation, numerous challenges may arise in each of these three components. In the LlamaIndex [Liu, 2023], a series of key classes and functions have been introduced specifi-

cally for the fine-tuning process of embedding models, significantly streamlining this procedure. By preparing a corpus of domain knowledge and utilizing the methods it provides, we can easily obtain the specialized embedding model tailored to our desired domain.

**Fine-tuning of downstream tasks** It is equally important to adapt Embedding models to downstream tasks. When using RAG in downstream tasks, some works have fine-tuned Embedding models by using the capabilities of LLMs.PROMPTAGATOR[Dai *et al.*, 2022] utilizes the Large Language Model (LLM) as a few-shot query generator and creates task-specific retrievers based on the generated data, and alleviates the problem of supervised fine-tuning, which is difficult in some domains due to data scarcity.LLM-Embedder[Zhang *et al.*, 2023a]uses the Large Language Model to output reward values for data from multiple downstream tasks, fine-tuning the retriever with two different supervised signals via hard labeling of the dataset and the soft reward derived from LLM.

This somewhat improves the semantic representation through both domain knowledge injection and downstream task fine-tuning. However, the retrievers trained by this approach are not intuitively helpful for large language models, so some work has been done to supervise the fine-tuning of Embedding models directly through feedback signals from the LLM. (This section will be presented in 4.4)

## 4.2 How to Match the Semantic Space of Queries and Documents

In the RAG application, some retrievers use the same embedding model to encode the query and doc, while others use two models to separately encode the query and doc. Moreover, the original query of the user may have problems of poor expression and lack of semantic information. Therefore, aligning the semantic space of the user's query and documents is very necessary. This section introduces two key technologies to achieve this goal.

### Query Rewrite

The most intuitive way to align the semantics of query and document is to rewrite the query. As mentioned in Query2Doc[Wang *et al.*, 2023b] and ITER-RETGEN[Shao *et al.*, 2023], the inherent capabilities of large language models are utilized to generate a pseudo-document by guiding it, and then the original query is merged with this pseudo-document to form a new query. In HyDE[Gao *et al.*, 2022], query vectors are established through the use of text indicators, using these indicators to generate a 'hypothetical' document that is relevant, yet may not truly exist, it only needs to capture the relevant pattern. RRR[Ma *et al.*, 2023a]introduced a new framework that inverts the order of retrieval and reading, focusing on query rewriting. This method generates a query using a large language model, then uses a web search engine to retrieve context, and finally uses a small language model as a training rewriter to serve the frozen large language model. The STEP-BACKPROMPTING[Zheng *et al.*, 2023] method can make large language models carry out abstract reasoning, extract high-level concepts and principles, and conduct retrieval based on this. Lastly, the method in Multi Query Retrieval involves using large language models to generate multiple search queries, these queries can be executed in parallel, and the retrieval results are input together, which is very useful for single problems that rely on multiple sub-problems

### Embedding Transformation

If there is a coarse-grained method like rewriting queries, there should also be a finer-grained implementation specific for embedding operations. In LlamaIndex[Liu, 2023], it is possible to connect an adapter after the query encoder, and fine-tune the adapter to optimize the representation of query embeddings, mapping it to a latent space that is better suited for specific tasks.When the data structure of a query and an external document are different, such as an unstructured query and a structured external document, it is very important to enable the query to align with the document.SANTA[Li *et al.*, 2023d] proposes two pre-training methods to make the retriever aware of structured information 1) Using the natural alignment relationship between structured data and unstructured data for contrastive learning for structured-aware pre-training. 2) Masked Entity Prediction, which designs an entity-oriented mask strategy and asks language models to fill in the masked entities.

## 4.3 How to Aligning Retriever's Output and LLM's Preference

In the RAG pipeline, even if we employ the above techniques to enhance the retrieval hit rate, it may still not improve the final effect of RAG, because the retrieved documents may not be what LLM needs. Thus, this section introduces two methods to align the outputs of the retriever and the preferences of the LLM.

**LLM supervised training** Many works leverage various feedback signals from large language models to fine-tune embedding models. AAR[Yu *et al.*, 2023b] provides supervisory signals for a pre-trained retriever through an encoder-decoder architecture LM. By determining the LM's preferred documents through FiD cross-attention scores, the retriever is then fine-tuned with hard negative sampling and standard cross-entropy loss. Ultimately, the fine-tuned retriever can directly be used to enhance unseen target LMs, thereby performing better in the target task. The training loss of retriever as:

$$\zeta = \sum_{q} \sum_{d^+ \in D^{a^+}} \sum_{d^- \in D^-} l\left(f\left(q, d^+\right), f\left(q, d^-\right)\right) \quad (1)$$

where $D^{a^+}$ is the documents preferred by the LLM in the retrieved set and $D^{a^-}$ is not preferred. $l$ is the standard cross entropy loss. In the end,it is suggested that LLMs may have a preference for focusing on readable rather than information-rich documents

REPLUG[Shi *et al.*, 2023] uses a retriever and an LLM to calculate the probability distributions of the retrieved documents, and then performs supervised training by calculating the KL divergence. This simple and effective training method enhances the performance of the retrieval model by using an LM as the supervisory signal, eliminating the need for any

specific cross-attention mechanisms. The training loss of the retriever is as follows:

$$\zeta = \frac{1}{|D|} \sum_{x \in D} KL\left(P_R\left(d|x\right) || Q_{LM}\left(d|x, y\right)\right) \tag{2}$$

where $D$ is a set of input contexts, $P_R$is the retrieval likelihood, $Q_{LM}$ is the LM likelihood of each document.

UPRISE[Cheng *et al.*, 2023a] also employs frozen large language models to fine-tune the Prompt Retriever. But both the language model and the retriever take Prompt-Input Pairs as inputs, then uses the scores given by the large language model to supervise the training of the retriever, equivalent to using the large language model to label the dataset. Atlas[Izacard *et al.*, 2022] proposes four methods of fine-tuning supervised embedding models, among them, Attention Distillation distills using the cross-attention scores that the language model generates during the output. EMDR2 employs the Expectation-Maximization algorithm to train with the retrieved documents as latent variables. Perplexity Distillation directly trains using the perplexity of the model-generated tokens as an indicator.LOOP introduces a new loss function based on the effect of document deletion on LM prediction, providing an effective training strategy for better adapting the model to specific tasks.

**Plug in an adapter** However, fine-tuning an embedding model can be challenging due to factors such as utilizing an API to implement embedding functionality or insufficient local computational resources. Therefore, some works choose to externally attach an adapter for alignment.PRCA[Yang *et al.*, 2023b] trains the Adapter through the Contextual Extraction Stage and the Reward-Driven Stage, and optimizes the output of the retriever based on a token-based autoregressive strategy. TokenFiltering[Berchansky *et al.*, 2023] method calculates cross-attention scores, selecting the highest scoring input tokens to effectively filter tokens. RECOMP[Xu *et al.*, 2023a] proposes extractive and generative compressors, which generate summaries by selecting relevant sentences or synthesizing document information to achieve multi-document query focus summaries.In addition to that, a novel approach, PKG[Luo *et al.*, 2023], infuses knowledge into a white-box model through directive fine-tuning, and directly replaces the retriever module, used to directly output relevant documents based on the query.

# 5 Generator

Another core component in RAG is the generator, responsible for transforming retrieved information into natural and fluent text. Its design is inspired by traditional language models, but in comparison to conventional generative models, RAG's generator enhances accuracy and relevance by leveraging the retrieved information. In RAG, the generator's input includes not only traditional contextual information but also relevant text segments obtained through the retriever. This allows the generator to better comprehend the context behind the question and produce responses that are more information-rich. Furthermore, the generator is guided by the retrieved text to ensure consistency between the generated content and the retrieved information. It is the diversity of input data that has led to a series of targeted efforts during the generation phase, all aimed at better adapting the large model to the input data from queries and documents. We will delve into the introduction of the generator through aspects of post-retrieval processing and fine-tuning.

## 5.1 How Can Retrieval Results be Enhanced via Post-retrieval Processing?

In terms of untuned large language models, most studies rely on well-recognized large language models like GPT-4[OpenAI, 2023] to leverage their robust internal knowledge for the comprehensive retrieval of document knowledge. However, inherent issues of these large models, such as context length restrictions and vulnerability to redundant information, persist. To mitigate these issues, some research has made efforts in post-retrieval processing. Post-retrieval processing refers to the process of further treating, filtering, or optimizing the relevant information retrieved by the retriever from a large document database. Its primary purpose is to enhance the quality of retrieval results to better meet user needs or for subsequent tasks. It can be understood as a process of reprocessing the documents obtained in the retrieval phase. The operations of post-retrieval processing usually involve information compression and result rerank.

**Information Compression**

Even though the retriever can fetch relevant information from a vast knowledge base, we are still confronted with the challenge of dealing with a substantial amount of information in retrieval documents. Some existing research attempts to solve this problem by increasing the context length of large language models, but current large models still confront context limitations. Thus, in certain situations, information condensation is necessary. In short, the importance of information condensation mainly embodies in the following aspects: reduction of noise, coping with context length restrictions, and enhancing generation effects.

PRCA [Yang *et al.*, 2023b] addressed this issue by training an information extractor. In the context extraction stage, given an input text $S_{input}$, it can generate an output sequence $C_{extracted}$, which represents the condensed context from the input document. The objective of the training process is to minimize the discrepancy between $C_{extracted}$ and the actual context $C_{truth}$ as much as possible. The loss function they adopted is as follows:

$$minL(\theta) = -\frac{1}{N} \sum_{i=1}^{N} C_{truth}^{(i)} log(f_{\cdot}(S_{input}^{(i)}; \theta)) \tag{3}$$

where $f_{\cdot}$ is the information extractor and $\theta$ is the parameter of the extractor. RECOMP[Xu *et al.*, 2023a] similarly trains an information condenser by leveraging contrastive learning. For each training data point, there exists one positive sample and five negative samples. The encoder is trained using contrastive loss [Karpukhin *et al.*, 2020] during this process.The specific optimization goals are as follows:

$$-log \frac{e^{sim(x_i, p_i)}}{sim(x_i, p_i) + \sum_{n_j \in N_i} e^{sim(x_i, p_i)}} \tag{4}$$

where $x_i$ is the training data, $p_i$ is the positive sample, and $n_j$ is the negative sample,sim(x,y) is to calculate the similarity between x and y. Another study has chosen to further streamline the quantity of documents, aiming to enhance the model's answer accuracy by reducing the number of retrieved documents. [Ma *et al.*, 2023b] proposed the "Filter-Ranker" paradigm, which integrates the strengths of Large Language Models (LLMs) and Small Language Models (SLMs). In this paradigm, SLMs serve as filters, while LLMs function as reordering agents. By prompting LLMs to rearrange portions of difficult samples identified by SLMs, the research results indicate significant improvements across various Information Extraction (IE) tasks.

**Rerank**

The pivotal role of the reordering model lies in optimizing the set of documents retrieved from retriever. LLMs experience performance degradation with retrospective performance when additional context is added, and reordering provides an effective solution to address this issue. The core idea involves rearranging document records to place the most relevant items at the top, thereby reducing the total number of documents to a fixed quantity. This not only resolves the issue of context window expansion that may be encountered during retrieval but also contributes to improving retrieval efficiency and responsiveness[Zhuang *et al.*, 2023].

Introducing context compression as part of the reordering aims to return relevant information solely based on the given query context. The dual significance of this approach lies in concentrating the display of the most relevant information in the retrieval results by reducing the content of individual documents and filtering entire documents. Thus, the reordering model plays an optimizing and refining role throughout the information retrieval process, providing more effective and accurate inputs for subsequent LLM processing.

## 5.2 How to Optimize a Generator to Adapt Input Data?

In the RAG model, the optimization of the generator is a crucial component of the architecture. The generator's task is to take the retrieved information and generate relevant text, thereby providing the final output of the model. The goal of optimizing the generator is to ensure that the generated text is both natural and effectively utilizes the retrieved documents, in order to better satisfy the user's query needs.

In typical Large Language Model (LLM) generation tasks, the input is usually a query. In RAG, the main difference lies in the fact that the input includes not only a query but also various documents retrieved by the retriever (structured/unstructured). The introduction of additional information may have a significant impact on the model's understanding, especially for smaller models. In such scenarios, fine-tuning the model to adapt to the input of query + retrieved documents becomes particularly important. Specifically, before providing the input to the fine-tuned model, there is usually post-retrieval processing of the documents retrieved by the retriever. It is essential to note that the method of fine-tuning the generator in RAG is essentially similar to the general fine-tuning approach for LLMs. Here, we will briefly introduce some representative works, including data (formatted/unformatted) and optimization functions.

**General Optimization Process**

Refers to the training data containing pairs of (input, output), aiming to train the model's ability to generate output y given input x. In the work of Self-mem[Cheng *et al.*, 2023b], a relatively classical training process is employed. Given input x, relevant documents z are retrieved (selecting Top-1 in the paper), and after integrating (x, z), the model generates output y. The paper utilizes two common paradigms for fine-tuning, namely Joint-Encoder [Arora *et al.*, 2023, Wang *et al.*, 2022b, Lewis *et al.*, 2020] and Dual-Encoder [Xia *et al.*, 2019, Cai *et al.*, 2021, Cheng *et al.*, 2022]. For Joint-Encoder, a standard model based on encoder-decoder is used, where the encoder initially encodes the input, and the decoder, through attention mechanisms, combines the encoded results to generate tokens in an autoregressive manner:

$$H = Encoder(x[SEP]m) \tag{5}$$

$$h^i = Decoder(CrossAttn(H), y < i) \tag{6}$$

$$P_{G_\xi}(.|x, y < i) = Softmax(h^i) \tag{7}$$

For the Dual-Encoder, the system establishes two independent encoders, each responsible for encoding the input (query, context) and the document, respectively. The output is then subject to bidirectional cross-attention processing by the decoder in sequence. The authors choose to use the Transformer [Vaswani *et al.*, 2017] as the building block for both architectures and optimize $G_\xi$ Negative Log-Likelihood (NLL) loss.

$$H_x = SourceEncoder(x) H_m = MemoryEncoder(x) \tag{8}$$

$$h^i = Decoder(CrossAttn(H_x, H_m), y < i) \tag{9}$$

$$\mathfrak{L}_{nll} = -\sum_{t=1}^{|y|} logP_{G_\xi}(y_t|x, m, y < t) \tag{10}$$

**Utilizing Contrastive Learning**

In the phase of preparing training data, usually generated are pairs of interactions between inputs and outputs. Under this circumstance, the model can only access a unique real output which might induce the "exposure bias" problem [Ranzato *et al.*, 2015]: during the training phase, the model only exposes to a single true feedback without accessing any other generated tokens. This can impair the model's performance in application as it might excessively fit to specific feedback in the training data without effectively generalizing to other scenarios. Therefore, a graph-text contrastive learning method has been proposed by SURGE [Kang *et al.*, 2023]. For any given pair of interactions between inputs and outputs, the objective of this contrastive learning approach can be defined as follows:

$$\mathfrak{L}_{cont} = \frac{1}{2} log \frac{e^{sim(\zeta(z),\xi(h))/\iota}}{\sum_{h'} e^{sim(\zeta(z),\xi(h'))/\iota}} + \frac{1}{2} log \frac{e^{sim(\zeta(z),\xi(h))/\iota}}{\sum_{z'} e^{sim(\zeta(z'),\xi(h))/\iota}} \tag{11}$$

Where $\zeta$,$\xi$ are learnable linear projection layers.z is the average representations of the graph from Encoder,h is the mean of decoder representations.$z'$,$h'$ represent the corresponding negative samples respectively. In the given text, 'h" and 'z" represent negative samples. By introducing a contrastive learning objective, the model can learn to generate diverse and reasonable replies better, rather than just the one seen in the training data. This helps to mitigate the risk of overfitting and improves the model's generalization ability in real-world scenarios.

When dealing with retrieval tasks that involve structured data, the work of SANTA[Li *et al.*, 2023d] utilized a three-stage training process to fully understand the structural and semantic information. Specifically, in the training phase of the retriever, contrastive learning was adopted, with the main goal of optimizing the embedding representations of the queries and documents. The specific optimization objectives are as follows:

$$\mathfrak{L}_{DR} = -log\frac{e^{sim(q,d^+)}}{e^{f(q,d^+)} + \sum_{d^- \in D^-} e^{sim(q,d^-)}} \tag{12}$$

where q and d are the query and document encoded by the encoder.$d^-$,$d^+$ represent negative samples and positive samples respectively. In the initial training stage of the generator, we utilize contrastive learning to align structured data and the corresponding document description of unstructured data. The optimization objective is as above.

Moreover, in the later training stage of the generator, inspired by references [Sciavolino *et al.*, 2021, Zhang *et al.*, 2019], we recognized the remarkable effectiveness of entity semantics in learning textual data representations in retrieval. Thus, we first perform entity identification in the structured data, subsequently applying a mask to the entities in the input section of the generator's training data, enabling the generator to predict these masks. The optimization objective hereafter is:

$$\mathfrak{L}_{MEP} = \sum_{j=1}^{k} -logP(Y_d(t_j)|X_d^{mask}, Y_d(t_1, ..., j-1)) \tag{13}$$

where $Y_d(y_j$ denotes the j-th token in the sequence $Y_d$. And $Y_d$ = $< mask >_1$, $ent_1$, ..., $< mask >_n$, $ent_n$ denotes the ground truth sequence that contains masked entities. Throughout the training process, we recover the masked entities by acquiring necessary information from the context, understand the structural semantics of textual data, and align the relevant entities in the structured data. We optimize the language model to fill the concealed spans and to better comprehend the entity semantics[Ye *et al.*, 2020].

# 6 Augmentation in RAG

This chapter is primarily organized into three dimensions: the stage of augmentation, augmentation data sources, and the process of augmentation, to elaborate on the key technologies in the development of RAG.Taxonomy of RAG's Core Components is illustrated in Fig 4.

## 6.1 RAG in Augmentation Stages

As a knowledge-intensive task, RAG employs different technical approaches during the language model training's pre-training, fine-tuning, and inference stages.

**Pre-training Stage**

Since the emergence of pre-trained models, researchers have delved into enhancing the performance of Pre-trained Language Models (PTMs) in open-domain Question Answering (QA) through retrieval methods at the pre-training stage. Recognizing and expanding implicit knowledge in pre-trained models can be challenging. REALM[Arora *et al.*, 2023] introduces a more modular and interpretable knowledge embedding approach. Following the Masked Language Model (MLM) paradigm, REALM models both pre-training and fine-tuning as a retrieve-then-predict process, where the language model pre-trains by predicting masked tokens $y$ based on masked sentences $x$, modeling $P(x|y)$.

RETRO[Borgeaud *et al.*, 2022]leverages retrieval augmentation for pre-training a self-regressive language model, enabling large-scale pre-training from scratch by retrieving from a massive set of labeled data and significantly reducing model parameters. RETRO shares the backbone structure with GPT models and introduces an additional RETRO encoder to encode features of neighboring entities retrieved from an external knowledge base. Additionally, RETRO incorporates block-wise cross-attention layers in its decoder transformer structure to effectively integrate retrieval information from the RETRO encoder. RETRO achieves lower perplexity than standard GPT models. Moreover, it provides flexibility in updating knowledge stored in the language models by updating the retrieval database without the need for retraining the language models [Petroni *et al.*, 2019].

Atla[Izacard *et al.*, 2022]employs a similar approach, incorporating a retrieval mechanism using the T5 architecture [Raffel *et al.*, 2020]in both the pre-training and fine-tuning stages. Prior to pre-training, it initializes the encoder-decoder LM backbone with a pre-trained T5, and initializes the dense retriever with a pre-trained Contriever. During the pre-training process, it refreshes the asynchronous index every 1000 steps.

COG [Vaze *et al.*, 2021]is a text generation model that formalizes its generation process by gradually copying text fragments (such as words or phrases) from an existing collection of text. Unlike traditional text generation models that select words sequentially, COG utilizes efficient vector search tools to calculate meaningful context representations of text fragments and index them. Consequently, the text generation task is decomposed into a series of copy and paste operations, where at each time step, relevant text fragments are sought from the text collection instead of selecting from an independent vocabulary. COG demonstrates superior performance to RETRO in various aspects, including question-answering, domain adaptation, and expanded phrase indexing.

On the other hand, following the discovery of the scaling law, there has been a rapid increase in model parameters, making autoregressive models the mainstream. Researchers are also exploring whether larger models can be pretrained using the RAG approach. RETRO++[Wang *et al.*, 2023a], an

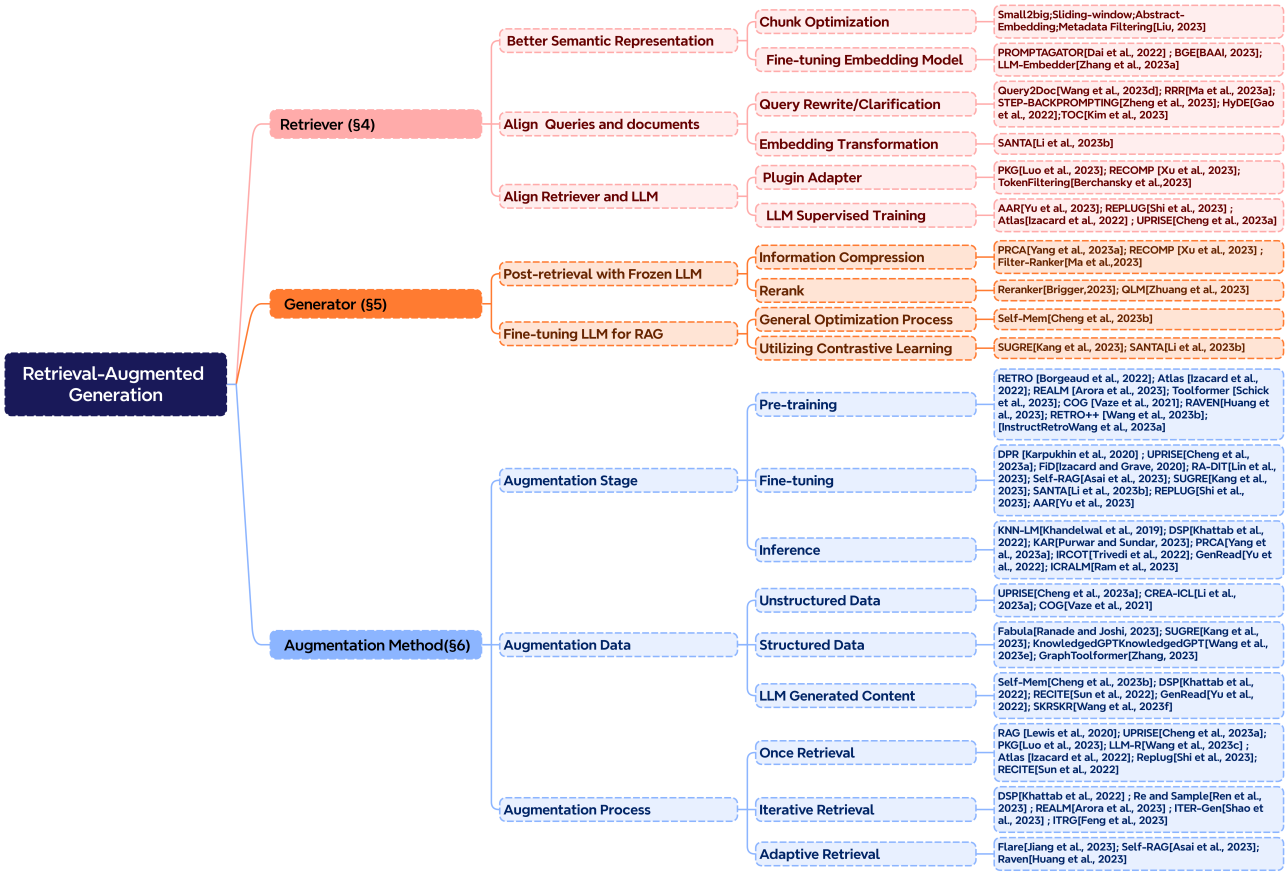


Figure 4: Taxonomy of RAG's Core Components

extension of RETRO, increased the model's parameter scale. Studies have found consistent improvements in text generation quality, factual accuracy, low toxicity, and downstream task accuracy, particularly in knowledge-intensive tasks such as open-domain question answering. These research findings highlight the promising direction of pretraining autoregressive language models in conjunction with retrieval for future foundational models.

In summary, the advantages and limitations of augmented pre-training are evident. On the positive side, this approach offers a more powerful foundational model, outperforming standard GPT models in perplexity, text generation quality, and downstream task performance. Moreover, it achieves higher efficiency by utilizing fewer parameters compared to purely pre-trained models. It particularly excels in handling knowledge-intensive tasks, allowing the creation of domain-specific models through training on domain-specific corpora. However, there are drawbacks, including the requirement for a substantial amount of pre-training data and larger training resources, as well as the issue of slower update speeds. Especially as model size increases, the cost of retrieval-enhanced training becomes relatively higher. Despite these limitations, this method demonstrates notable characteristics in terms of model robustness. Once trained, retrieval-enhanced models based on pure pre-training eliminate the need for external library dependencies, enhancing both generation speed and operational efficiency.

### Fine-tuning Stage

During the downstream fine-tuning phase, researchers have employed various methods to fine-tune retrievers and generators for improved information retrieval, primarily in open-domain question-answering tasks. Concerning retriever fine-tuning, REPlUG[Shi *et al.*, 2023] treats the language model (LM) as a black box and enhances it through an adjustable retrieval model. By obtaining feedback from the black-box language model through supervised signals, REPLUG improves the initial retrieval model. UPRISE[Cheng *et al.*, 2023a], on the other hand, fine-tunes retrievers by creating a lightweight and versatile retriever through fine-tuning on diverse task sets. This retriever can automatically provide retrieval prompts for zero-shot tasks, showcasing its universality and improved performance across tasks and models.

Simultaneously, methods for fine-tuning generators include Self-Mem[Cheng *et al.*, 2023b], which fine-tunes the generator through a memory pool of examples, and Self-RAG[Asai *et al.*, 2023b], which satisfies active retrieval needs by generating reflection tokens. The RA-DIT[Lin *et al.*, 2023] method fine-tunes both the generator and retriever by maximizing the probability of correct an-

swers given a retrieval-enhanced directive. It updates the generator and retriever to minimize the semantic similarity between documents and queries, effectively leveraging relevant background knowledge.

Additionally, SUGRE[Kang *et al.*, 2023] introduces the concept of contrastive learning. It conducts end-to-end fine-tuning of both retriever and generator, ensuring highly detailed text generation and retrieved subgraphs. Using a context-aware subgraph retriever based on Graph Neural Networks (GNN), SURGE extracts relevant knowledge from a knowledge graph corresponding to an ongoing conversation. This ensures the generated responses faithfully reflect the retrieved knowledge. SURGE employs an invariant yet efficient graph encoder and a graph-text contrastive learning objective for this purpose.

In summary, the enhancement methods during the fine-tuning phase exhibit several characteristics. Firstly, fine-tuning both LLM and retriever allows better adaptation to specific tasks, offering the flexibility to fine-tune either one or both simultaneously, as seen in methods like RePlug[Shi *et al.*, 2023] and RA-DIT[Lin *et al.*, 2023]. Secondly, the benefits of this fine-tuning extend to adapting to diverse downstream tasks, as demonstrated by UPRISE[Cheng *et al.*, 2023a], making the model more versatile. Additionally, fine-tuning enables models to better accommodate different data structures in various corpora, particularly advantageous for graph-structured corpora, as highlighted by the SUGRE method.

However, fine-tuning during this phase comes with limitations, such as the need for datasets specifically prepared for RAG fine-tuning and the requirement for substantial computational resources compared to the RAG during the inference phase. Overall, during fine-tuning, researchers have the flexibility to tailor models according to specific requirements and data formats, reducing the resource consumption compared to the pre-training phase while retaining the ability to adjust the model's output style.

**Inference Stage**

The integration of RAG methods with LLM has become a prevalent research direction in the inference phase. Notably, the research paradigm of Naive RAG relies on incorporating retrieval content during the inference stage.

To overcome the limitations of Naive RAG, researchers have introduced richer context in the RAG during the inference phase. The DSP[Khattab *et al.*, 2022] framework relies on a complex pipeline that involves passing natural language text between a frozen Language Model (LM) and a Retrieval Model (RM), providing the model with more informative context to enhance generation quality. PKG equips LLMs with a knowledge-guided module that allows access to relevant knowledge without altering the parameters of LLMs, enabling the model to perform more sophisticated tasks. Additionally, CREA-ICL[Li *et al.*, 2023b] leverages synchronous retrieval of cross-lingual knowledge to assist in acquiring additional information, while RECITE forms context by sampling one or more paragraphs from LLMs.

During the inference phase, optimizing the process of RAG can benefit adaptation to more challenging tasks. For example, ITRG[Feng *et al.*, 2023a] enhances adaptability for tasks requiring multiple-step reasoning by iteratively retrieving and searching for the correct reasoning path. ITER-RETGEN[Shao *et al.*, 2023] employs an iterative approach to coalesce retrieval and generation, achieving an alternating process of "retrieval-enhanced generation" and "generation-enhanced retrieval."

On the other hand, IRCOT[Trivedi *et al.*, 2022] merges the concepts of RAG and CoT[Wei *et al.*, 2022], employing alternate CoT-guided retrievals and using retrieval results to improve CoT. This method significantly improves the performance of GPT-3 across various QA tasks, highlighting the potential advantages of integrating retrieval and generation.

In summary, inference-stage enhancement methods offer the advantages of being lightweight, cost-effective, requiring no additional training, and utilizing powerful pre-trained models. The main strength lies in freezing the parameters of the LLMs during fine-tuning, focusing on providing context that better suits the requirements, with the characteristics of being fast and low-cost. However, this approach also has some limitations, including the need for additional data processing and process optimization, while being constrained by the foundation model's capabilities. Typically, this method is often combined with process optimization techniques such as step-wise reasoning , iterative reasoning, and adaptive retrieval to better meet the requirements of different tasks.

## 6.2 Augmentation Data Source

Data source is crucial factors for RAG effectiveness. Various data sources offer distinct granularities and dimensions of knowledge, requiring different processing methods. They primarily fall into three categories: unstructured data, structured data, and content generated by LLMs.

**Augmented with Unstructured Data**

Unstructured data mainly encompasses textual data , typically derived from pure text corpora. Additionally, other text data can serve as retrieval sources, such as Prompt data used for large model fine-tuning[Cheng *et al.*, 2023a] and cross-language data[Li *et al.*, 2023b].

In terms of text granularity, beyond the common chunks (including sentences), the retrieval unit can be tokens (e.g., kNN-LM[Khandelwal *et al.*, 2019]), phrases (e.g., NPM[Lee *et al.*, 2020], COG[Vaze *et al.*, 2021]), and document paragraphs. Finer-grained retrieval units can often better handle rare patterns and out-of-domain scenarios but come with an increase in retrieval costs.

At the word level, FLARE employs an active retrieval strategy, conducting retrieval only when the LM generates low-probability words. The method involves generating a temporary next sentence for retrieval of relevant documents, then re-generating the next sentence under the condition of the retrieved documents to predict subsequent sentences.

At the chunk level, RETRO uses the previous chunk to retrieve the nearest neighboring chunk and integrates this information with the contextual information of the previous chunk to guide the generation of the next chunk. RETRO achieves this by retrieving the nearest neighboring block $N(C_{i-1})$ from the retrieval database, then fusing the contextual in-

formation of the preceding blocks $(C_1, \ldots, C_{i-1})$ and the retrieval information of $N(C_{i-1})$ through cross-attention to guide the generation of the next block $C_i$. To maintain causality, the autoregressive generation of the $i$-th block $C_i$ can only use the nearest neighbor of the previous block $N(C_{i-1})$ and not $N(C_i)$.

**Augmented with Structured Data**

Structured data sources like Knowledge Graphs (KG) are gradually integrated into the paradigm of RAG. Verified KGs can offer higher-quality context, reducing the likelihood of model hallucinations.

RET-LLM [Modarressi *et al.*, 2023] constructs a personalized knowledge graph memory by extracting relation triples from past dialogues for future use. SUGRE[Kang *et al.*, 2023] embeds relevant subgraphs retrieved from the knowledge graph using Graph Neural Networks (GNN) to prevent the model from generating contextually irrelevant replies. SUGRE[Kang *et al.*, 2023] employs a graph encoding method that reflects the graph structure into PTMs' representation space and utilizes a multi-modal contrastive learning objective between graph-text modes to ensure consistency between retrieved facts and generated text. KnowledgeGPT[Wang *et al.*, 2023c] generates search queries for Knowledge Bases (KB) in code format and includes predefined KB operation functions. Apart from retrieval, KnowledgeGPT also offers the capability to store knowledge in a personalized knowledge base to meet individual user needs. These structured data sources provide RAG with richer knowledge and context, contributing to improved model performance.

**LLM Generated Content RAG**

Observing that the auxiliary information recalled by RAG is not always effective and may even have negative effects, some studies have expanded the paradigm of RAG by delving deeper into the internal knowledge of LLM. This approach utilizes the content generated by LLM itself for retrieval, aiming to enhance performance in downstream tasks. The following outlines notable studies within this category:

SKR[Wang *et al.*, 2023d] employs a labeled training set, categorizing questions that the model can directly answer as known and those requiring retrieval enhancement as unknown. The model is trained to discern whether a question is known, applying retrieval enhancement only to inputs identified as unknown, while directly answering the rest.

GenRead[Yu *et al.*, 2022] substitutes the LLM generator for the retriever. Experimental results indicate that situations where the generated context document contains correct answers are more prevalent than those retrieved by Naive RAG. The generated answers also demonstrate superior quality. The authors attribute this to the alignment between the task of generating document-level context and the pre-training objective of causal language modeling, allowing for better utilization of world knowledge stored in the model parameters.

Selfmem[Cheng *et al.*, 2023b] iteratively uses a retrieval-enhanced generator to create an unbounded memory pool. A memory selector is employed to choose an output as the memory for subsequent generations. This output serves as the dual problem to the original question. By combining the original and dual problems, a retrieval-enhanced generative model can leverage its own output to enhance itself.

These diverse approaches showcase innovative strategies in RAG retrieval enhancement, aiming to elevate model performance and effectiveness.

## 6.3 Augmentation Process

Most RAG research typically only performs a single retrieval and generation process. However, single retrievals may contain redundant information, leading to a "lost in the middle" phenomenon[Liu *et al.*, 2023]. This redundant information can obscure key information or contain information contrary to the real answer, negatively impacting the generation effect[Yoran *et al.*, 2023]. Additionally, the information obtained from a single retrieval is limited in problems requiring multi-step reasoning.

Current methods to optimize the retrieval process mainly include iterative retrieval and adaptive retrieval. These allow the model to iterate multiple times during the retrieval process or adaptively adjust the retrieval process to better accommodate different tasks and scenarios.

**Iterative Retrieval**

Regularly collecting documents based on the original query and generated text can provide additional materials for LLMs[Borgeaud *et al.*, 2022, Arora *et al.*, 2023]. Providing additional references in multiple iterative retrievals has improved the robustness of subsequent answer generation. However, this method may be semantically discontinuous and potentially lead to the collection of noisy and useless information, as it primarily relies on a sequence of n tokens to separate the generated and retrieved documents.

Recursive retrieval and multi-hop retrieval are used for specific data scenarios. Recursive retrieval can first process data through a structured index, then retrieve it level by level. When retrieving hierarchically rich documents, a summary can be made for each section in an entire document or long PDF. A retrieval is then performed based on the summary. After determining the document, a second retrieval is conducted for the internal chunks, thus realizing recursive retrieval. Multi-hop retrieval is often used to further mine information in graph-structured data sources[Li *et al.*, 2023c].

Some methods iterate the steps of retrieval and generation. ITER-RETGEN [Shao *et al.*, 2023] collaboratively utilizes "retrieval-enhanced generation" and "generation-enhanced retrieval" for tasks requiring reproduction of information. That is, the model uses the content needed to complete the task to respond to the input task, and these target contents serve as the information context for retrieving more relevant knowledge. This helps to generate better responses in another iteration.

IRCoT[Trivedi *et al.*, 2022] also explores retrieving documents for each generated sentence, introducing retrieval at every step of the thought chain. It uses CoT to guide the retrieval and uses the retrieval results to improve CoT, ensuring semantic completeness.

**Adaptive Retrieval**

Indeed, the RAG methods described in the previous two sections follow a passive approach where retrieval is prior-

itized. This method, which involves querying related documents and inputting into a LLM based on context, may lead to efficiency issues. Adaptive retrieval methods such as those introduced by Flare[Jiang *et al.*, 2023b] and Self-RAG[Asai *et al.*, 2023b], optimize the RAG retrieval process, enabling the LLM to actively judge the timing and content of retrieval. This helps to improve the efficiency and relevance of the information retrieved.

In fact, the way in which LLM actively uses tools and makes judgments is not originated from RAG but has been widely used in the agents of large models[Yang *et al.*, 2023c, Schick *et al.*, 2023, Zhang, 2023]. The retrieval steps of Graph-Toolformer[Zhang, 2023] are roughly divided into: LLMs actively use the retriever, Self-Ask and DSP[Khattab *et al.*, 2022] try to use few-shot prompts to trigger LLM search queries. When LLMs think it is necessary, they can decide to search for a relevant query to collect the necessary materials, similar to the tool call of the agent.

WebGPT[Nakano *et al.*, 2021] employs a reinforcement learning framework to automatically train the GPT-3 model to use a search engine for text generation. It uses special tokens to perform actions, including querying on a search engine, scrolling rankings, and citing references. This allows GPT-3 to leverage a search engine for text generation.

Flare[Jiang *et al.*, 2023b], on the other hand, automates the timing of retrieval and addresses the cost of periodic document retrieval based on the probability of the generated text. It uses probability as an indicator of LLMs' confidence during the generation process. When the probability of a term falls below a predefined threshold, the information retrieval system would retrieve references and removes terms with lower probabilities. This approach is designed to handle situations where LLMs might need additional knowledge.

Self-RAG[Asai *et al.*, 2023b] introduces an important innovation called Reflection tokens. These special tokens are generated to review the output and come in two types: Retrieve and Critic. The model can autonomously decide when to retrieve paragraphs or use a set threshold to trigger retrieval. When retrieval is needed, the generator processes multiple paragraphs simultaneously, performing fragment-level beam search to obtain the best sequence. The scores for each subdivision are updated using Critic scores, and these weights can be adjusted during the inference process to customize the model's behavior. The Self-RAG framework also allows the LLM to autonomously determine whether recall is necessary, avoiding training additional classifiers or relying on NLI models. This enhances the model's ability to autonomously judge inputs and generate accurate answers.

# 7 RAG Evaluation

In exploring the development and optimization of RAG, effectively evaluating its performance has emerged as a central issue. This chapter primarily discusses the methods of evaluation, key metrics for RAG, the abilities it should possess, and some mainstream evaluation frameworks.

## 7.1 Evaluation Methods

There are primarily two approaches to evaluating the effectiveness of RAG: independent evaluation and end-to-end evaluation[Liu, 2023].

**Independent Evaluation**

Independent evaluation includes assessing the retrieval module and the generation (read/synthesis) module.

1. **Retrieval Module**

   A suite of metrics that measure the effectiveness of systems (like search engines, recommendation systems, or information retrieval systems) in ranking items according to queries or tasks are commonly used to evaluate the performance of the RAG retrieval module. Examples include Hit Rate, MRR, NDCG, Precision, etc.

2. **Generation Module**

   The generation module here refers to the enhanced or synthesized input formed by supplementing the retrieved documents into the query, distinct from the final answer/response generation, which is typically evaluated end-to-end. The evaluation metrics for the generation module mainly focus on context relevance, measuring the relatedness of retrieved documents to the query question.

**End-to-End Evaluation**

End-to-end evaluation assesses the final response generated by the RAG model for a given input, involving the relevance and alignment of the model-generated answers with the input query. From the perspective of content generation goals, evaluation can be divided into unlabeled and labeled content. Unlabeled content evaluation metrics include answer fidelity, answer relevance, harmlessness, etc., while labeled content evaluation metrics include Accuracy and EM. Additionally, from the perspective of evaluation methods, end-to-end evaluation can be divided into manual evaluation and automated evaluation using LLMs. The above summarizes the general case of end-to-end evaluation for RAG. Furthermore, specific evaluation metrics are adopted based on the application of RAG in particular domains, such as EM for question-answering tasks[Borgeaud *et al.*, 2022, Izacard *et al.*, 2022], UniEval and E-F1 for summarization tasks[Jiang *et al.*, 2023b], and BLEU for machine translation[Zhong *et al.*, 2022]. These metrics help in understanding the performance of RAG in various specific application scenarios.

## 7.2 Key Metrics and Abilities

Existing research often lacks rigorous evaluation of the impact of retrieval-augmented generation on different LLMs. In most cases, the evaluaion of RAG's application in various downstream tasks and with different retrievers may yield divergent results. However, some academic and engineering practices have focused on general evaluation metrics for RAG and the abilities required for its effective use. This section primarily introduces key metrics for evaluating RAG's effectiveness and essential abilities for assessing its performance.

**Key Metrics**

Recent OpenAI report[Jarvis and Allard, 2023] have mentioned various techniques for optimizing large language models (LLMs), including RAG and its

evaluation metrics. Additionally, the latest evaluation frameworks like RAGAS[Es *et al.*, 2023] and ARES[Saad-Falcon *et al.*, 2023] also involve RAG evaluation metrics. Summarizing these works, three core metrics are primarily focused on: Faithfulness of the answer, Answer Relevance, and Context Relevance.

1. **Faithfulness**

   This metric emphasizes that the answers generated by the model must remain true to the given context, ensuring that the answers are consistent with the context information and do not deviate or contradict it. This aspect of evaluation is vital for addressing illusions in large models.

2. **Answer Relevance**

   This metric stresses that the generated answers need to be directly related to the posed question.

3. **Context Relevance**

   This metric demands that the retrieved contextual information be as accurate and targeted as possible, avoiding irrelevant content. After all, processing long texts is costly for LLMs, and too much irrelevant information can reduce the efficiency of LLMs in utilizing context.

   The OpenAI report also mentioned "Context Recall" as a supplementary metric, measuring the model's ability to retrieve all relevant information needed to answer a question. This metric reflects the search optimization level of the RAG retrieval module. A low recall rate indicates a potential need for optimization of the search functionality, such as introducing re-ranking mechanisms or fine-tuning embeddings, to ensure more relevant content retrieval.

**Key abilities**

The work of RGB[Chen *et al.*, 2023b] analyzed the performance of different large language models in terms of four basic abilities required for RAG, including Noise Robustness, Negative Rejection, Information Integration, and Counterfactual Robustness, establishing a benchmark for retrieval-augmented generation.RGB focuses on the following four abilities:

1. **Noise Robustness**

   This capability measures the model's efficiency in handling noisy documents, which are those related to the question but do not contain useful information.

2. **Negative Rejection**

   When documents retrieved by the model lack the knowledge required to answer a question, the model should correctly refuse to respond. In the test setting for negative rejection, external documents contain only noise. Ideally, the LLM should issue a "lack of information" or similar refusal signal.

3. **Information Integration**

   This ability assesses whether the model can integrate information from multiple documents to answer more complex questions.

4. **Counterfactual Robustness**

   This test aims to evaluate whether the model can identify and deal with known erroneous information in documents when receiving instructions about potential risks in retrieved information. Counterfactual robustness tests include questions that the LLM can answer directly, but the related external documents contain factual errors.

## 7.3 Evaluation Frameworks

Recently, the LLM community has been exploring the use of "LLMs as judge" for automatic assessment, with many utilizing powerful LLMs (such as GPT-4) to evaluate their own LLM applications outputs. Practices by Databricks using GPT-3.5 and GPT-4 as LLM judges to assess their chatbot applications suggest that using LLMs as automatic evaluation tools is effective[Leng *et al.*, 2023]. They believe this method can also efficiently and cost-effectively evaluate RAG-based applications.

In the field of RAG evaluation frameworks, RAGAS and ARES are relatively new. The core focus of these evaluations is on three main metrics: Faithfulness of the answer, answer relevance, and context relevance. Additionally, TruLens, an open-source library proposed by the industry, also offers a similar evaluation mode. These frameworks all use LLMs as judges for evaluation. As TruLens is similar to RAGAS, this chapter will specifically introduce RAGAS and ARES.

**RAGAS**

This framework considers the retrieval system's ability to identify relevant and key context paragraphs, the LLM's ability to use these paragraphs faithfully, and the quality of the generation itself. RAGAS is an evaluation framework based on simple handwritten prompts, using these prompts to measure the three aspects of quality - answer faithfulness, answer relevance, and context relevance - in a fully automated manner. In the implementation and experimentation of this framework, all prompts are evaluated using the gpt-3.5-turbo-16k model, which is available through the OpenAI API[Es *et al.*, 2023].

**Algorithm Principles**

1. Assessing Answer Faithfulness: Decompose the answer into individual statements using an LLM and verify whether each statement is consistent with the context. Ultimately, a "Faithfulness Score" is calculated by comparing the number of supported statements to the total number of statements.

2. Assessing Answer Relevance: Generate potential questions using an LLM and calculate the similarity between these questions and the original question. The Answer Relevance Score is derived by calculating the average similarity of all generated questions to the original question.

3. Assessing Context Relevance: Extract sentences directly relevant to the question using an LLM, and use the ratio of these sentences to the total number of sentences in the context as the Context Relevance Score.

**ARES**

ARES aims to automatically evaluate the performance of RAG systems in three aspects: Context Relevance, Answer Faithfulness, and Answer Relevance. These evaluation metrics are similar to those in RAGAS. However, RAGAS, being a newer evaluation framework based on simple handwritten prompts, has limited adaptability to new RAG evaluation settings, which is one of the significances of the ARES work. Furthermore, as demonstrated in its assessments, ARES performs significantly lower than RAGAS.

ARES reduces the cost of evaluation by using a small amount of manually annotated data and synthetic data, and utilizes Predictive-Driven Reasoning (PDR) to provide statistical confidence intervals, enhancing the accuracy of evaluation[Saad-Falcon *et al.*, 2023].

**Algorithm Principles**

1. Generating Synthetic Dataset: ARES initially generates synthetic questions and answers from documents in the target corpus using a language model to create positive and negative samples.
2. Preparing LLM Judges: Next, ARES fine-tunes lightweight language models using the synthetic dataset to train them to evaluate Context Relevance, Answer Faithfulness, and Answer Relevance.
3. Ranking RAG Systems Using Confidence Intervals: Finally, ARES applies these judge models to score RAG systems and combines them with a manually annotated validation set using the PPI method to generate confidence intervals, reliably estimating the performance of RAG systems.

# 8 Future Prospects

In this chapter, we delve into three future prospects for RAG, namely vertical optimization, horizontal expansion and ecosystem of RAG.

## 8.1 Vertical Optimization of RAG

Despite the rapid advancements in RAG technology over the past year, there are still several areas in its vertical domain that require further investigation.

Firstly, the issue of long context in RAG is a significant challenge. As mentioned in the literature [Xu *et al.*, 2023c], RAG's generation phase is constrained by the context window of LLMs. If the window is too short, it may not contain enough relevant information; if it's too long, it might lead to information loss. Currently, expanding the context window of LLMs, even to the extent of limitless context, is a critical direction in LLM development. However, once the context window constraint is removed, how RAG should adapt remains a noteworthy question.

Secondly, the robustness of RAG is another important research focus. If irrelevant noise appears during retrieval, or if the retrieved content contradicts facts, it can significantly impact RAG's effectiveness. This situation is figuratively referred to as "opening a book to a poisonous mushroom". Therefore, enhancing the robustness of RAG has increasingly gained researchers' attention, as represented in studies such as [Yu *et al.*, 2023a, Glass *et al.*, 2021, Baek *et al.*, 2023].

Thirdly, the issue of RAG and Fine-tuning's synergy is also a primary research point. Hybrid has gradually become one of the mainstream methods in RAG, exemplified by RA-DIT [Lin *et al.*, 2023]. How to coordinate the relationship between the two to simultaneously obtain the advantages of parameterization and non-parameterization is a problem that needs addressing.

Lastly, the engineering practice of RAG is a significant area of interest. The ease of implementation and alignment with corporate engineering needs have contributed to RAG's rise. However, in engineering practice, questions like how to improve retrieval efficiency and document recall rate in large-scale knowledge base scenarios, and how to ensure enterprise data security, such as preventing LLMs from being induced to disclose the source, metadata, or other information of documents, are crucial issues that need resolution[Alon *et al.*, 2022].

**Horizontal expansion of RAG**

Research on RAG has rapidly expanded in the horizontal field. Starting from the initial text question answering domain, RAG's ideas have gradually been applied to more modal data, such as images, code, structured knowledge, audio and video, and so on. There are already many works in this regard.

In the image field, the propozhiyosal of BLIP-2[Li *et al.*, 2023a], which uses frozen image encoders and large-scale language models for visual language pre-training, has lowered the cost of model training. Additionally, the model can generate image-to-text conversions from zero samples. In the field of text generation, the VBR[Zhu *et al.*, 2022] method is used to generate images to guide the text generation of the language model, which has significant effects in open text generation tasks.

In the code field, RBPS[Nashid *et al.*, 2023] is used for small-scale learning related to code. By encoding or frequency analysis, similar code examples to the developers' tasks are automatically retrieved. This technique has proven its effectiveness in test assertion generation and program repair tasks. In the field of structured knowledge, methods like CoK[Li *et al.*, 2023c] hints first retrieve facts related to the input question from the knowledge graph and then add these facts to the input in the form of hints. This method has performed well in knowledge graph question answering tasks.

For the field of audio and video, the GSS[Zhao *et al.*, 2022] method retrieves and concatenates audio clips from the spoken vocabulary bank, immediately transforming MT data into ST data. UEOP[Chan *et al.*, 2023] introduces a new breakthrough in end-to-end automatic speech recognition by introducing external offline strategies for voice-to-text mapping. Audio embeddings and semantic text embeddings generated by text-to-speech methods can bias ASR through KNN-based attention fusion, effectively shortening domain adaptation time. The Vid2Seq[Yang *et al.*, 2023a] architecture enhances the language model by introducing special time markings, enabling it to seamlessly predict event boundaries and text descriptions

within the same output sequence.

## 8.2 Ecosystem of RAG

**Downstream Tasks and Evaluation**

By integrating relevant information from a broad knowledge base, RAG has demonstrated significant potential in enhancing language models' ability to process complex queries and generate information-rich responses. Numerous studies have shown that RAG performs well in various downstream tasks, such as open-ended question answering and fact verification. RAG models not only improve the accuracy and relevance of information in downstream applications but also increase the diversity and depth of responses.

Given the success of RAG, exploring the model's adaptability and universality in multi-domain applications will be part of future work. This includes its use in professional domain knowledge question-answering, such as in medicine, law, and education. In the application of downstream tasks such as professional domain knowledge question-answering, RAG might offer lower training costs and better performance benefits than fine-tuning.

Simultaneously, improving the evaluation system of RAG for assessing and optimizing its application in different downstream tasks is crucial for the model's efficiency and benefits in specific tasks. This includes developing more accurate evaluation metrics and frameworks for different downstream tasks, such as context relevance, content creativity, and harmlessness, among others.

Furthermore, enhancing the interpretability of models through RAG, allowing users to better understand how and why the model makes specific responses, is also a meaningful task.

**Technical Stack**

In the ecosystem of RAG, the development of the related technical stack has played a driving role. For instance, LangChain and LLamaIndex have become widely known quickly with the popularity of ChatGPT. They both offer a rich set of RAG-related APIs, gradually becoming one of the indispensable technologies in the era of large models. Meanwhile, new types of technical stacks are constantly being developed. Although they do not offer as many features as LangChain and LLamaIndex, they focus more on their unique characteristics. For example, Flowise AI[6] emphasizes low-code, allowing users to implement various AI applications represented by RAG without writing code, simply by dragging and dropping. Other emerging technologies include HayStack, Meltno, and Cohere Coral.

In addition to AI-native frameworks, traditional software or cloud service providers have also expanded their service range. For instance, Verba[7], provided by the vector database company Weaviate, focuses on personal assistants. Amazon offers its users the intelligent enterprise search service tool Kendra, based on RAG thinking. Users can search in different content repositories through built-in connectors.

The development of the technical stack and RAG are mutually reinforcing. New technologies pose higher demands on the existing technical stack, while the optimization of the technical stack's functions further promotes the development of RAG technology. Overall, the technical stack of RAG's toolchain has initially formed, and many enterprise-level applications have gradually emerged, but an all-in-one platform still needs to be refined.

[6]https://flowiseai.com

[7]https://github.com/weaviate/Verba

# 9 Conclusion

This paper thoroughly explores Retrieval-Augmented Generation (RAG), a technique that uses an external knowledge base to supplement the context of Large Language Models (LLMs) and generate responses. Notably, RAG combines parameterized knowledge from LLMs and non-parameterized external knowledge, alleviates hallucination issues, identifies timely information via retrieval technology, and enhances response accuracy. Additionally, by citing sources, RAG increases transparency and user trust in model outputs. RAG can also be customized based on specific domains by indexing relevant text corpora. RAG's development and characteristics are summarized into three paradigms: Naive RAG, Advanced RAG, and Modular RAG, each with its models, methods, and shortcomings. Naive RAG primarily involves the 'retrieval-reading' process. Advanced RAG uses more refined data processing, optimizes the knowledge base indexing, and introduces multiple or iterative retrievals. As exploration deepens, RAG integrates other techniques like fine-tuning, leading to the emergence of the Modular RAG paradigm, which enriches the RAG process with new modules and offers more flexibility.

In the subsequent chapters, we further analyze three key parts of RAG in detail. Chapter 4 introduces the retriever of RAG, how to process corpora to obtain better semantic representations, how to mitigate the semantic gap between Query and documents, and how to adjust the retriever to fit the generator. Chapter 5 explains how the generator obtains better generation results by post-processing retrieved documents, avoiding the "Lost in the middle" issue, as well as methods to adjust the generator to fit the retriever. Subsequently, in Chapter 6, we review the current retrieval enhancement methods from the aspects of the retrieval stage, retrieval data sources, and retrieval process.

Chapter 7 explains how to evaluate current RAG methods, including evaluation, key indicators, and current evaluation frameworks Finally, we provided an outlook on the potential future research directions for RAG. As a method that combines retrieval and generation, RAG has numerous potential development directions in future research. By continuously improving the technology and expanding its applications, the performance and practicality of RAG can be further enhanced.

# References

[Alon *et al.*, 2022] Uri Alon, Frank Xu, Junxian He, Sudipta Sengupta, Dan Roth, and Graham Neubig. Neuro-symbolic language modeling with automaton-augmented retrieval. In *International Conference on Machine Learning*, pages 468–485. PMLR, 2022.