Retreival Augmented Generation: A Comprehensive Survey on Bridging Language Models and External Knowledge

Shrey Ganatra and Pushpak Bhattacharyya
Indian Institute of Technology, Bombay

Abstract

Large Language Models (LLMs) have demonstrated remarkable capabilities in natural language understanding and generation. However, their effectiveness is constrained by the static nature of their training data, leading to issues such as knowledge cutoff, factual inaccuracies (hallucinations), and a lack of domain-specific expertise. Retrieval-Augmented Generation (RAG) has emerged as a powerful and prevalent paradigm to mitigate these limitations by dynamically integrating external knowledge into the LLM's generative process. This survey provides a systematic and comprehensive overview of the RAG landscape. We begin by deconstructing the foundational RAG pipeline into its core components: the indexer, the retriever, and the generator. We then propose a taxonomy that charts the evolution of RAG systems, from Naïve and Advanced RAG to the more sophisticated Modular, Agentic, and Graph-based paradigms. For each paradigm, we analyze its architecture, key innovations, and inherent trade-offs. Furthermore, we delve into the critical challenges facing the field, including retrieval quality, generation faithfulness, and the complexity of evaluation. Finally, we outline promising future research directions, such as end-to-end optimization, adaptive retrieval strategies, and the development of robust, automated evaluation frameworks. This survey serves as a foundational resource for both newcomers and experienced researchers, providing a structured understanding of the current state and future potential of Retrieval-Augmented Generation.

1 Introduction

The advent of Large Language Models (LLMs) such as OpenAI's GPT-4 (Achiam et al., 2023), Meta's Llama (Touvron et al., 2023) series, and Google's Gemini (Team et al., 2023) has catalyzed a paradigm shift in artificial intelligence. These models, built on the Transformer (Vaswani et al., 2017) architecture and trained on web-scale datasets, exhibit an unprecedented ability to comprehend, reason, and generate human-like text. Their impact spans a vast array of applications, from sophisticated conversational AI to automated content creation and scientific research. However, the power of these models is intrinsically tied to their training methodology. LLMs encode knowledge parametrically within their network weights during a static, computationally intensive pre-training phase. This design choice gives rise to several fundamental limitations that hinder their deployment in real-world, knowledge-intensive applications:

- **Knowledge Cutoff:** An LLM's knowledge is static and becomes outdated the moment its training is complete. It has no awareness of events, discoveries, or data that emerged post-training.
- **Factual Hallucination:** When prompted on topics outside their internalized knowledge or when forced to make high-precision claims, LLMs are prone to generating "hallucinations"—plausible-sounding but factually incorrect or nonsensical statements. This severely erodes user trust and limits their utility in high-stakes domains.
- **Lack of Domain-Specificity and Personalization:** LLMs are trained on public web data and thus lack the specialized, proprietary knowledge of a specific enterprise (e.g., internal documentation, product specs) or the personal context of an individual user.
- **Opacity and Lack of Attribution:** The knowledge generation process is a black box, making it nearly impossible to trace a generated fact back to its source. This lack of attribution is a major barrier to verification and accountability.

To overcome these inherent weaknesses, Retrieval-Augmented Generation (RAG) was proposed (Lewis et al., 2020). RAG enhances LLMs by providing them with dynamic, non-parametric knowledge. Instead of relying solely on its static internal knowledge, a RAG system first retrieves relevant information from an external knowledge base (e.g., a collection of documents, a database, or the web) and then uses this retrieved information as grounding context to generate an informed, accurate, and attributable response. This survey provides a comprehensive and structured exploration of the RAG landscape, which has evolved far beyond its initial conception. Our primary contributions are: A detailed deconstruction of the foundational RAG pipeline, explaining the technical nuances of its core components: Indexing, Retrieval, and Generation. A structured taxonomy of RAG paradigms that captures the field's evolution from simple pipelines to complex, agentic systems. A thorough analysis of the key challenges and open research questions in retrieval optimization, generation faithfulness, and, critically, system evaluation. A forward-looking perspective on the future directions that are poised to define the next generation of RAG, including end-to-end learning and adaptive control.

2 RAG Pipeline

The canonical RAG workflow, often referred to as "Naïve RAG," consists of three primary stages.

2.1 Indexing

The indexing stage is the offline process of preparing the knowledge corpus for real-time querying.

1. **Data Loading:** Ingesting documents from various sources (e.g., text files, PDFs, websites, databases).

2. **Chunking (or Segmentation):** This is a critical step where large documents are divided into smaller, manageable pieces. The chunking strategy directly impacts the quality of retrieval. Key strategies include:

    - **Fixed-Size Chunking:** Simply splitting text by a fixed number of characters or tokens, often with some overlap to preserve context across chunks.
    - **Content-Aware Chunking:** Splitting based on the semantic structure of the document, such as by paragraphs, sections, or specific markers (e.g., HTML tags).
    - **Recursive Character Splitting:** A common and robust method that recursively splits text by a series of user-defined separators (e.g.,
    n, ,) until chunks are of a manageable size.

3. **Embedding and Storing:** Each chunk is passed through an embedding model (e.g., text-embedding-ada-002, Sentence-BERT) to create a high-dimensional vector representation. These vector embeddings, along with the source text and metadata, are loaded into a vector database (e.g., Pinecone, Weaviate, Milvus, Chroma). These databases are optimized for efficient vector similarity search, typically using algorithms like HNSW (Hierarchical Navigable Small World).

2.2 Retrieval

This is the online process that finds relevant context for a given user query.
Query Embedding: The input query from the user is embedded into a vector using the same embedding model used during indexing.
Vector Similarity Search: The system executes a Maximum Inner Product Search (MIPS) to find the vectors in the database that are most similar to the query vector. Common similarity metrics include Cosine Similarity, Dot Product, and Euclidean Distance.
Top-k Context Selection: The k chunks corresponding to the most similar vectors are retrieved. This set of k documents forms the context that will be provided to the LLM.

2.3 Generation

This final stage leverages the LLM to produce an answer based on the retrieved context.
Prompt Augmentation: The retrieved chunks are formatted and inserted into a prompt template. A typical prompt might look like:

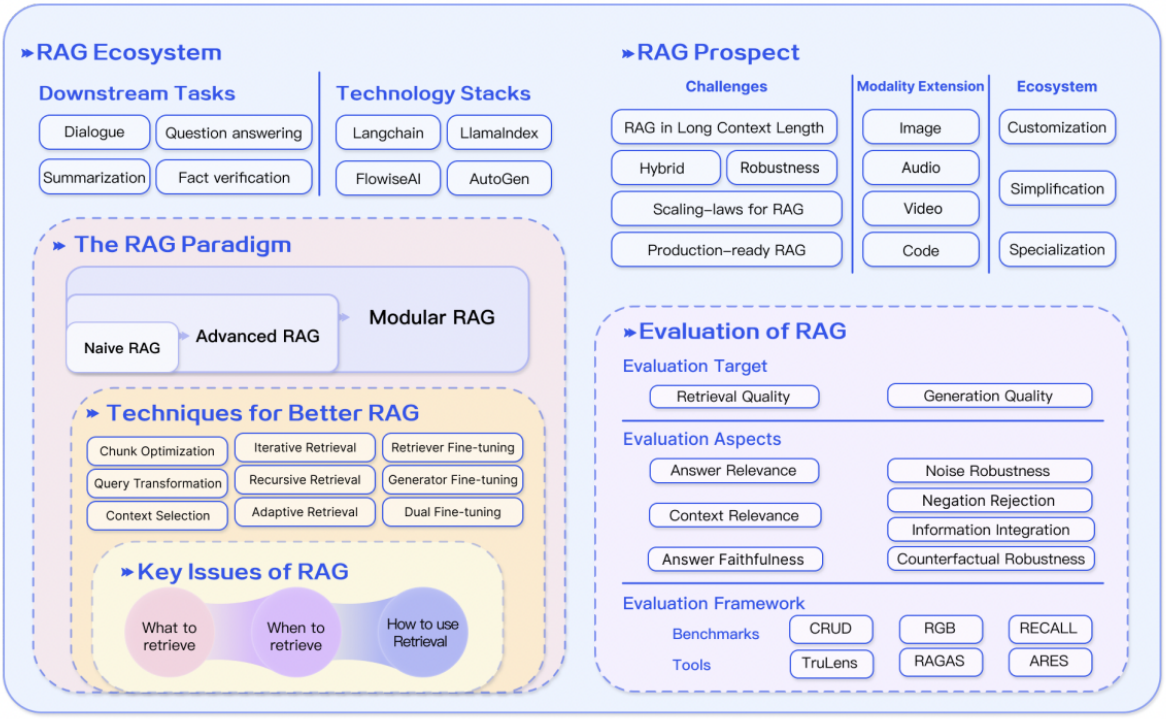

Figure 1: Summary of RAG ecosytem by (Gao et al., 2023b)

> Context:
>
> Retrieved Chunk 1
>
> Retrieved Chunk 2
>
> Based on the context provided above, please answer the following question.
> Question: [User Query]
> Answer:

LLM Generation: The augmented prompt is sent to an LLM. The model is instructed to synthesize the information from the provided context to formulate a comprehensive, factually grounded answer, ideally citing its sources from the context.

3 Taxanomy

The RAG field has evolved significantly beyond the naïve pipeline. We can categorize this evolution into several distinct paradigms.

3.1 Naïve RAG

This is the simple, foundational retrieve-then-read pipeline described above. It is a powerful baseline but often fails in complex scenarios due to retrieval imprecision (the "needle in a haystack" problem) or the generator's inability to synthesize information effectively, especially when the context is noisy or lengthy (the "lost in the middle" (Liu et al., 2023) problem).

3.2 Advanced RAG

This paradigm introduces enhancements to the retrieval stage to improve the quality of the context provided to the LLM.

3.2.1 Pre-Retrieval Enhancements

Query Transformation: Using an LLM to refine the input query. This can involve breaking a complex question into several sub-queries (Zheng et al., 2023), rewriting a vague query for clarity (Peng et al., 2024b), or generating a hypothetical document that would answer the query (HyDE (Gao et al., 2023a)) and using its embedding for the search.

3.2.2 Post-Retrieval Enhancements

Re-Ranking: (Glass et al., 2022) After retrieving an initial set of k documents (e.g., k=20), a more powerful but slower model, like a cross-encoder, is used to re-rank these candidates and select a

smaller, more relevant subset (e.g., top 3) to pass to the LLM.

Context Filtering & Compression: Using an LLM to identify and remove redundant or irrelevant information from the retrieved chunks, or to summarize them, thereby reducing noise and focusing the generator on the most critical facts.

3.3 Modular RAG

This paradigm views RAG not as a fixed pipeline but as a flexible framework of interchangeable modules. This allows for greater adaptability and the integration of diverse functionalities. Key modules can be dynamically orchestrated:

Search Module: Can choose between different retrieval methods (e.g., vector search, sparse retrieval with BM25, or keyword search) or use them in parallel.

Memory Module: Integrates conversational history to enable multi-turn, context-aware RAG.

Routing Module: Acts as a decision-making layer, directing a query to the most appropriate downstream task. For example, it might route a query to a RAG pipeline, a direct LLM call (for conversational queries), a web search, or a structured data query engine (e.g., text-to-SQL). Frameworks like LlamaIndex and LangChain are instrumental in building such modular systems.

3.4 Agentic RAG

Agentic RAG (Singh et al., 2024) represents the current state-of-the-art, where the LLM transcends its role as a mere generator and becomes a reasoning agent that can plan, use tools, and iteratively self-correct (Gou et al., 2023). Reasoning and Planning (Chain-of-Thought): The agent decomposes a complex problem into a sequence of logical steps. It plans its actions before executing them.

Tool Use: Retrieval is framed as just one of many possible "tools." An agent can decide which tool is best for a given task, such as a VectorStoreRetriever, a WebSearchAPI, a Calculator, or a CodeInterpreter.

Iterative Self-Correction: This is the defining feature. The agent can execute a tool (e.g., retrieve a document), observe the output, and then reflect on whether the information is sufficient. If not, it can generate a new plan—such as rewriting the query, trying a different tool, or breaking the problem down further—and execute it. This introduces loops and sophisticated error-correction capabilities, as seen in frameworks like ReAct (Reason + Act) and Self-RAG.

4 Graph-RAG

Distinct from the above, Graph RAG leverages knowledge graphs (KGs) for retrieval (Liang et al., 2025; Procko and Ochoa, 2024; Peng et al., 2024a). KGs store information as nodes (entities) and edges (relationships), providing an explicit, structured representation of knowledge.

Knowledge Representation: Instead of unstructured text, the knowledge base is a graph. This excels at capturing complex, multi-hop relationships. Retrieval: The retrieval process involves graph traversal algorithms to find relevant entities, paths, or entire subgraphs. This is fundamentally different from vector similarity search and allows for more precise, explainable reasoning by following explicit connections.

Generation: The retrieved subgraph is typically "linearized"—converted into a descriptive text format—before being passed to the LLM. This provides the generator with a rich, relational context that is difficult to extract from flat documents. Graph RAG is particularly powerful for domains with highly interconnected data, such as biomedical research or financial analysis.

5 Reasoning with RAG

While the foundational RAG pipeline excels at answering factoid questions where the answer is contained within a single retrieved chunk, its true power is unlocked when applied to complex queries that require reasoning. Reasoning, in this context, refers to the ability to synthesize information from multiple sources, perform multi-step logical inferences, compare and contrast concepts, and decompose a complex problem into a sequence of solvable steps. Naïve RAG, with its linear "retrieve-then-read" architecture, fundamentally lacks an explicit mechanism for this kind of iterative thought. The burden of reasoning falls entirely on the final generation call to the LLM, which must perform this complex synthesis "in its head" using a potentially noisy and unstructured context.

This section explores the evolution of reasoning capabilities within RAG, tracing the shift from implicit synthesis to explicit, planned, and agentic reasoning frameworks.

5.1 The Need for Multi-Hop Reasoning

The primary driver for advanced reasoning in RAG was the challenge of Multi-Hop Question Answering (QA). These are queries where the answer cannot be found in a single document but requires connecting information across multiple sources.

Consider the query: "What is the hometown of the director of the movie that won the Oscar for Best Picture in the year Leonardo DiCaprio won his first Oscar?"

A Naïve RAG system would likely fail because no single document contains the full answer. Answering this requires a logical chain:

Hop 1: When did Leonardo DiCaprio win his first Oscar? (Answer: For the movie "The Revenant" at the 2016 Oscars).

Hop 2: Which movie won Best Picture at the 2016 Oscars? (Answer: "Spotlight").

Hop 3: Who was the director of "Spotlight"? (Answer: Tom McCarthy).

Hop 4: What is the hometown of Tom McCarthy? (Answer: New Providence, New Jersey).

This demonstrates that the retrieval process itself must be iterative and guided by the intermediate results of the reasoning chain.

5.2 Key Paradigms for Reasoning in RAG

To address this challenge, several sophisticated reasoning paradigms have been developed, each building upon the last in complexity and capability.

5.2.1 Iterative Loop

The simplest extension to Naïve RAG is iterative retrieval. Instead of a single retrieval step, the system performs a loop.

Retrieve an initial set of documents based on the original query. The LLM analyzes the retrieved context and generates an intermediate thought or a refined query.

This new query is used to perform another retrieval step. The process repeats for a fixed number of steps or until the LLM determines it has enough information to answer the final question.

While an improvement, simple iterative retrieval is often inefficient. It lacks a clear plan and can easily get sidetracked by irrelevant information retrieved in an early step, leading to a "wild goose chase."

5.2.2 Decomposition

A more structured approach is to use an LLM to explicitly decompose a complex query into a series of simpler, interdependent sub-questions.

An LLM planner is first prompted to break the main query into a sequence of sub-questions. For the example above, it might generate:

Sub-question 1: In which year did Leonardo DiCaprio win his first Oscar?

Sub-question 2: Which movie won Best Picture in that year?

Sub-question 3: Who was the director of that movie?

Sub-question 4: What is the hometown of that director?

Sequential Execution: The system then executes this plan step-by-step. It answers each sub-question in order, using the answer from the previous step to inform the retrieval for the next. Each step involves a targeted RAG call (retrieve and synthesize an answer for the sub-question).

Final Synthesis: Once all sub-questions are answered, a final LLM call synthesizes the intermediate answers into a single, coherent response. This method is more robust than simple iteration because it follows a structured plan. Frameworks like LlamaIndex provide tools to easily implement such multi-step query engines.

5.2.3 Agentic Reasoning

The most advanced and powerful form of reasoning is found in Agentic RAG. Here, the LLM is not just a planner (Huang et al., 2024) or a generator but an autonomous agent that can reason, create complex plans, use tools, and dynamically adapt its strategy based on observations.

The canonical agentic framework is ReAct (Reason + Act), which structures the LLM's process into a Thought -> Action -> Observation loop.

Thought: The agent reasons about the overall goal and decides on the immediate next step. It verbalizes its reasoning, creating a clear trace of its strategy.

Action: The agent decides to execute a specific "tool." This could be Search[query] to retrieve from a vector database, Lookup[entity] for a specific lookup, or even Finish[answer] when it believes it has a final answer. Observation: The agent receives the output from the tool it executed. This could be a set of retrieved documents or a specific value.

The agent then repeats this loop, using the new observation to inform its next thought and action, until it decides to use the Finish action. This allows for dynamic, real-time planning and error correc-

tion. If a search action yields no results, the agent can observe this failure, think about why it failed (e.g., "my query was too specific"), and formulate a new action (e.g., Search[rewritten, broader query]).

Self-RAG (Asai et al., 2023) extends this by adding more granular, self-reflective capabilities. It trains an LLM to generate special "reflection tokens" that control the retrieval and generation process. When generating text, the model can decide whether retrieval is needed. If it retrieves, it can then generate critique tokens to assess the relevance of each document ([Relevant], [Irrelevant]) and whether the retrieved documents support the claim being made ([Supported], [Contradictory], [Partial]). This allows the agent to actively filter noise and reason about the quality of its own retrieved context.

5.2.4 Reasoning with Knowledge Graphs

Reasoning with knowledge graphs represents a fundamentally different approach. Instead of finding statistical correlations in text, it leverages explicit, structured relationships. Structured Inference: Reasoning on a KG is a process of graph traversal. To answer the query "Which drug that treats diabetes is manufactured by a company based in Germany?", the system can perform a structured query that traverses the graph:

Find(Drug) WHERE (Drug) -> [treats_with]-> (Diabetes)
AND (Drug) -> [manufactured_by] -> (Company)
AND (Company) -> [based_in]-> (Germany).

Advantages:
Precision: Reasoning follows explicit, human-curated relationships, leading to highly precise and reliable answers.
Explainability: The reasoning path through the graph serves as a fully transparent and auditable explanation of how the answer was derived. Reduced Ambiguity: By linking entities to unique identifiers in the graph, it resolves the ambiguity inherent in natural language. The primary challenge in Graph RAG is converting a natural language query into a formal graph query (e.g., SPARQL or Cypher), a task known as text-to-graph. However, when successful, it provides a level of logical rigor that is difficult to achieve with text-based RAG.

5.3 Challenges and Future Directions in RAG Reasoning

Despite these advancements, significant challenges remain:

Error Propagation: In multi-hop chains, an error in an early step (e.g., a failed retrieval or an incorrect intermediate answer) will inevitably corrupt the entire subsequent reasoning process.
Latency and Cost: Agentic reasoning, with its multiple LLM calls for thoughts, plans, and syntheses, can be significantly slower and more expensive than Naïve RAG, posing a major barrier to production deployment.
Planning Rigidity vs. Flexibility: While decomposition provides structure, it can be too rigid. Agentic frameworks offer flexibility but risk inefficient planning, getting stuck in loops, or hallucinating invalid tool calls. A key research area is finding the right balance between structured planning and dynamic adaptation
Optimal Tool Selection: As the number of available retrieval tools and data sources grows, the agent faces a new retrieval problem: which tool is the best one for the current task? This requires sophisticated routing and tool-use capabilities.

The future of reasoning in RAG lies in creating more efficient and robust agents. This includes developing techniques for parallelizing actions, using smaller, specialized LLMs for planning and tool selection to reduce latency, and training agents with self-correction mechanisms that are more resilient to early-stage errors. Ultimately, the goal is to create systems that can seamlessly transition between simple retrieval and complex, agentic reasoning based on the demands of the query, all while remaining fast, cost-effective, and trustworthy.

6 Key Challenges and Open Research Areas

6.1 Retrieval Challenges

6.1.1 Optimal Chunking

Finding the right chunking strategy remains a "dark art." The optimal size and method depend heavily on the data and task, and there is no one-size-fits-all solution.

6.1.2 Embedding Model Limitations

General-purpose embedding models may not capture the nuances of specialized domains. Fine-

tuning these models is often necessary but can be data-intensive.

6.1.3 Precision-Recall Trade-off

Retrieving more documents (increasing k) improves the chance of finding the correct information (higher recall) but also introduces more noise, making it harder for the LLM to find the "needle in the haystack" (lower precision).

6.2 Generation Challenges

6.2.1 Faithfulness vs. Fluency

Enforcing strict faithfulness to the provided context can lead to stilted, extractive answers. Allowing for more abstraction and synthesis increases fluency but risks introducing hallucinations.

6.2.2 Handling Noise and Contradiction

When retrieved documents contain conflicting information, it is an open question how the LLM should reconcile these contradictions.

6.2.3 Attribution

Reliably citing sources is difficult, especially when the final answer is a synthesis of information from multiple different chunks.

6.3 Evaluation

Evaluating RAG systems is the most significant bottleneck in the field.

6.3.1 Inadequacy of Traditional Metrics

Simple metrics for retrieval (Hit Rate, MRR) do not always correlate with the quality of the final answer. Metrics for generation (BLEU, ROUGE) fail to capture factual correctness.

6.3.2 Emergence of RAG-Specific Frameworks

A new generation of evaluation frameworks like RAGAS (Es et al., 2024), ARES (Saad-Falcon et al., 2023), and TruLens has emerged. They focus on a more holistic set of metrics, often using an LLM as the evaluator:

Faithfulness: Is the generated answer factually consistent with the retrieved context?
Answer Relevance: Does the answer directly address the user's question?
Context Precision: Are the retrieved chunks relevant to the question? (Signal-to-noise ratio).
Context Recall: Were all necessary documents to answer the question retrieved?

Need for Better Benchmarks: The field needs robust, standardized benchmarks that test RAG systems on diverse, realistic, and challenging queries, especially those requiring multi-hop reasoning and the rejection of "unanswerable" questions.

7 Future Directions

7.1 End-to-End Optimization

The holy grail is to move beyond separately trained components and jointly optimize the entire RAG pipeline. This would allow the retriever to be fine-tuned to retrieve documents that are specifically most useful for the generator it is paired with.

7.2 Adaptive and Agentic Control

Systems will become more intelligent about resource allocation. They will learn if retrieval is needed at all, what to retrieve (e.g., which data source or tool to use), and how to retrieve (e.g., single-step vs. multi-step).

7.3 Multimodality

RAG will expand beyond text to retrieve and synthesize information from images, tables, audio, and video, enabling a richer and more comprehensive understanding of the world.

7.4 Scalability and Productionization

A major focus will be on making complex agentic workflows efficient and low-latency enough for real-world production systems. This involves techniques like prompt caching, parallelizing tool calls, and using smaller, specialized LLMs for intermediate reasoning steps.

8 Conslusion

Retrieval-Augmented Generation has evolved from a simple technique to a cornerstone of modern AI, providing a practical and powerful solution to the inherent limitations of static LLMs. By grounding generation in external, verifiable knowledge, RAG is the key to building factual, trustworthy, and context-aware AI systems. This survey has charted its evolution through a structured taxonomy, from the Naïve pipeline to sophisticated Agentic and Graph-based systems. While significant challenges in retrieval, generation, and especially evaluation persist, the field is a hotbed of innovation. The future of RAG lies in the development of fully

adaptive, end-to-end optimized, and multimodal systems that can reason and act as true collaborators in our quest for knowledge.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*.

Shahul Es, Jithin James, Luis Espinosa Anke, and Steven Schockaert. 2024. Ragas: Automated evaluation of retrieval augmented generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 150–158.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2023a. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1762–1777.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023b. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1).

Michael Glass, Gaetano Rossiello, Md Faisal Mahbub Chowdhury, Ankita Naik, Pengshan Cai, and Alfio Gliozzo. 2022. Re2G: Retrieve, rerank, generate. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2701–2715, Seattle, United States. Association for Computational Linguistics.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2023. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. Understanding the planning of llm agents: A survey. *arXiv preprint arXiv:2402.02716*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Lei Liang, Zhongpu Bo, Zhengke Gui, Zhongshu Zhu, Ling Zhong, Peilong Zhao, Mengshu Sun, Zhiqiang Zhang, Jun Zhou, Wenguang Chen, et al. 2025. Kag: Boosting llms in professional domains via knowledge augmented generation. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 334–343.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*.

Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. 2024a. Graph retrieval-augmented generation: A survey. *arXiv preprint arXiv:2408.08921*.

Wenjun Peng, Guiyang Li, Yue Jiang, Zilong Wang, Dan Ou, Xiaoyi Zeng, Derong Xu, Tong Xu, and Enhong Chen. 2024b. Large language model based long-tail query rewriting in taobao search. In *Companion Proceedings of the ACM Web Conference 2024*, pages 20–28.

Tyler Thomas Procko and Omar Ochoa. 2024. Graph retrieval-augmented generation for large language models: A survey. In *2024 Conference on AI, Science, Engineering, and Technology (AIxSET)*, pages 166–169. IEEE.

Jon Saad-Falcon, Omar Khattab, Christopher Potts, and Matei Zaharia. 2023. Ares: An automated evaluation framework for retrieval-augmented generation systems. *arXiv preprint arXiv:2311.09476*.

Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. 2024. Enhancing ai systems with agentic workflows patterns in large language model. In *2024 IEEE World AI IoT Congress (AIIoT)*, pages 527–532. IEEE.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H Chi, Quoc V Le, and Denny Zhou. 2023. Take a step back: Evoking reasoning via abstraction in large language models. *arXiv preprint arXiv:2310.06117*.