A Study on Improving Mathematical Reasoning using Retrieval Augmented Generation

1st Adithya Gaurav H C 2nd Praveen Kumar R 3rd Kiran Kumar 4th Sanish Samuel 5th Ganesh 6th Naresh

***Abstract*—Interactive question-answering with human tutors is a proven method for teaching maths. There is a potential for large language models (LLMs) to automate parts of this process, including interactive discussions. Though LLMs have demonstrated significant potential, they are limited by problems such as hallucination, accessing outdated knowledge, knowledge gaps, and insufficient data for training. LLMs can produce incorrect or irrelevant answers. To address this, retrieval-augmented generation (RAG) can enhance LLM responses by incorporating verified external knowledge into the model's prompts. RAG systems integrate information retrieval mechanisms with LLMs, augmenting prompts with relevant passages or documents retrieved from external sources, resulting in more relevant generations from LLMs as a response. In this paper, we designed experiments on mathematical reasoning using GPT 3.5 on the GSM8K dataset. We determine the base performance of GPT 3.5 and evaluate the efficacy of naive and advanced RAG systems implemented using the llama index in solving the GSM8K dataset. We demonstrate that naive RAG and advanced RAG using Hyde can improve response accuracy and reduce hallucinations.**

Keywords: RAG, RAGAS, LLM, GSM8K, OpenAI, Chroma DB, Vector Database, llama Index

I. Introduction and Literature Survey

While Large Language Models (LLMs) have shown fantastic success, their limitations in domain-specific tasks and knowledge-intensive queries persist. Using LLMs has become especially evident in how they can hallucinate (i.e., generate wrong outputs on cases outside their training data. Retrieval-augmented generation (RAG) is a framework to address this shortcoming in LLM by allowing them to retrieve relevant document snippets from external knowledge bases using semantic similarity matching. RAG prevents generation inconsistencies (in LLMs), making RAG a key enabler for the progress of chatbots and scaling up LLMs to serve real-world use cases.

RAG currently can be classified into 3 stage paradigms such as Naive RAG - Advanced RAG- Modular RAG [6]

Naive RAG is a base step towards accurate and informative LLMs, where the RAG system extracts information from an external source and provides additional context that improves generation. [12]

It mainly involves three steps:

- **Indexing:** Indexing involves processing data inputs from various sources like text documents or Knowledge Graphs and organizing them to be easily converted using Tokenization and Vectorization.
- **Retrieval:** When the user provides the query or prompt, the system looks up the Indexed Data to search for similar relevant chunks of information through ranking algorithms or similarity-based to identify such chunks.
- **Generation:** Finally, the retrieved information and the user's query are given to the LLM to process as context.

This context helps the LLM to generate a more informative and accurate answer.

Advanced RAG tries to solve the problems on naive rag using the following techniques: [7]

Addressing Retrieval Issues:

- **Precision and Recall:** Techniques like query expansion (searching for topically relevant terms) or iterative retrieval (applying successive searches based on the initial set of results) accuracy (precision) and recall of retrieved information
- **Contextual Understanding:** Advanced methods like sentence-window retrieval (i.e., relevant sentences from documents rather than whole articles) ensure high precision for the information returned in response to a certain user query.

Improving Generation:

- **Attention Mechanisms:** Advanced RAG also incorporates attention mechanisms inside the LLM used to build restrictions into the LLM to extract data better and reduce generation to a few factors that are likely most important in recovery, which should yield more low-level understanding output.
- **Data Filtering and Ranking:** Advanced RAG includes post-retrieval filtration that prunes out all irrelevant/redundant information before feeding it to the LLM. Furthermore, the importance of ranking a retrieved passage can assist the LLM in determining which context might or might not be so relevant.

Modular RAG is the most advanced and flexible iteration of the Retrieval-Augmented Generation (RAG) approach. It builds upon the strengths of Naive and Advanced RAG while addressing their limitations by introducing a modular architecture. [19]

Modular RAG breaks down the RAG process into independent modules, Which Are interchangeable and customizable (depending on the nature of the task or data source). These modules encompass, for instance, search and memory interfaces—search over different types of data (text searches, code completion, or even tabular-style information), memories to store past outputs, and fusion of incoming retrieval results. The modular design is more scalable and adaptable.

A. RETRIEVAL

Retrieval refers to the process of retrieving the correct information. The model must also decide the right-size "chunks" of information it should retrieve (retrieval granularity) and ensure they are neither too large nor too small. Further, retrieval preprocessing is necessary for the model to start using it. Last but not least, the embedding model used needs to be as close to perfect as it can.

Retrieval Source: Retrieval-augmented generation (i.e., using a question to retrieve external knowledge followed by answering it) uses large language models and additional background both during training and while generating outputs. The Retrieval source, type, and granularity of each specific evaluation influence the quality of the generated text. Initially, text was the primary source for RAG. However, semi-structured data like PDF and structured data like knowledge graphs can also be used with RAG for improved performance. Interestingly, recent research explores using text generated by LLMs as an information source within RAG.

Retrieval Granularity: Retrieval Granularity refers to selecting the right level of detail during retrieval, which can significantly improve the retrieval performance and any tasks that rely on the retrieved information. While larger chunks of text (coarse-grained retrieval) might provide more relevant context, they can also include unnecessary details that confuse the retrieval system and subsequent language models. The information's size, level of detail, and format determine the retrieval granularity.

Chunking Strategy: Text Chunking is a common approach to capture context around the words. The document is chunked into pieces of a fixed number of tokens in RAG systems. But there's a trade-off: bigger chunks capture more context but also include irrelevant information, and smaller chunks are less noisy but might miss important context. [15] To address this, more advanced chunking methods can split sentences recursively or use a sliding window approach. These methods allow for a more nuanced approach to chunking, where you can capture local and global information within the document.

By attaching metadata to document chunks, we can enrich them with valuable information like page number, file name, author, category, and even timestamps. This metadata allows us to filter searches and narrow down the retrieved results. Additionally, assigning different weights to timestamps during retrieval makes it possible to prioritize fresh information. This "time-aware RAG" (RAG stands for "Red, Amber, Green" - potentially a ranking system) ensures that searches primarily surface the most up-to-date knowledge and avoid outdated content.

Indexing Optimization: The first step was to process the documents, splinter them into embeddings during Indexing, and store them in vector-specific databases. The way the CIT process works, in turn, determines how well we can retrieve information by using that system later. Once the Indexing is adequate during the evaluation phase, and if it is in the right context, on finding,

- **Structural Index:** By structuring documents, The writing is difficult to search, but if the document can be organized in a hierarchy as structural index and indexes are positive. So, we have a layered system, similar to an outline, in how most documents are organized based on their content. Structural Index is what it means to have a structural index, and it can assist the RAG (Retrieval-Augmented Generation) system in rapidly accessing and processing information during retrieval.
- **Hierarchical index structure:** Hierarchical Indexing: It organizes the file like a tree family and maintains the relationship among one or more index levels (Index block). In other words, the files look like nodes; from there, parent nodes summarize child nodes (files). They are fast access points in the Retrieval-Augmented Generation (RAG) data structure. Hierarchical Indexing not only helps to get the required subset faster but also helps to determine what an optimal data chunk would be in that extract. Summaries can also help prevent the problems of doing a partial extract of information out of context in individual chunks by providing this extracted info with some grounding through deeper understanding elsewhere via hierarchical keys.
- **Knowledge Graph index** A Knowledge Graph is a map that shows relationships between various concepts and entities in the documents. With this map, the system can compare chunks of data and greatly reduce the chances that illusions could come from isolated reads. Even better, the knowledge graph enables it to translate questions into instructions for a Large Language Model (LLM), but more on that later. Knowledge Graph makes the LLM better equipped to gather knowledge more precisely and provide topical responses. Ultimately, a knowledge graph index allows RAG to work much more efficiently.[18]

Query Optimization: A significant limitation of Naive RAG is its dependence on the user's query for retrieval. Crafting a perfect question can be tricky, and poorly phrased queries can lead to inaccurate results. Complex questions and the inherent ambiguity in languages compound this challenge.

Query Expansion: Query expansion tackles a fundamental weakness of Naive RAG: its reliance on a single, potentially imprecise user query. This method enriches the original query with additional queries, effectively providing more context. By addressing potential ambiguities and capturing missing nuances, query expansion helps ensure the retrieved information is highly relevant to the user's intent.

- **Multi Query:** leverages Large Language Models (LLMs) and prompt engineering to generate multiple queries [16] used to retrieve a wider range of relevant information.
- **Sub Query:** Sub-query planning involves breaking down a complex question into a series of simpler, more focused sub-questions. The system can understand the user's intent by progressively adding context through these sub-questions (using a "least-to-most prompting" method) [20]. Ultimately, sub-query planning helps ensure the retrieved

information is highly relevant to the user's true information need.[21]

- **Chain-of-Verification (CoVe):** A technique called Chain-of-Verification (CoVe) can further improve the reliability of information retrieved by RAG systems. CoVe employs Large Language Models (LLMs) to validate the expanded queries generated during query expansion. This validation step helps to reduce the likelihood of hallucinations, where the system outputs seemingly correct but factually incorrect information. By incorporating CoVe, RAG systems can ensure that the expanded queries used for information retrieval are more trustworthy, ultimately leading to more reliable responses.[3]

Query Transformation: Query Transformation involves prompting LLMs to rewrite the queries into a more suitable format for information retrieval in query transformation. Instead of relying solely on the user's original query, query transformation retrieves document chunks based on a reformulated version of the query, as the original queries might not always be ideal for retrieval by large language models (LLMs) in real-world situations. e.g., HyDE system.[5] involves using prompts carefully crafted to guide the LLM into generating an effective query based on the user's original question.

Query Routing: This approach directs user queries to specialized RAG pipelines designed for specific scenarios. There are two main methods for query routing:

- **Metadata Router/Filter:** extracts keywords (entities) from the user's query and then filters document chunks based on these keywords and metadata. Uses this metadata for routing the query.
- **Semantic Router:** This method leverages the semantic meaning of the query for routing.

Embedding: RAG systems rely on embeddings to retrieve relevant information. Embeddings are essentially compressed text representations, capturing the semantic meaning of words and phrases. During retrieval, the system calculates the similarity (often using cosine similarity) between the embedding of the user's question and the embeddings of document chunks. This process heavily relies on the quality of the embedding model.

There are two common types of embedding models used in RAG:

- **Sparse encoders:** models like BM25, work well for identifying keywords and achieving efficient retrieval.
- **Dense retrievers:** models based on pre-trained language models like BERT provide a more nuanced understanding of semantics, leading to more accurate retrieval of relevant information.

B. AUGMENTATION

Traditionally, RAG systems perform retrieval only once before the generation step. This single retrieval can be inefficient for complex issues requiring multi-step reasoning, as it limits the information available to the system. Many studies have focused on optimizing the retrieval process within RAG to address this. These optimizations often involve multiple retrieval steps, allowing the system to gather a broader range of relevant information and potentially leading to more explained and accurate responses.

Iterative retrieval: Iterative retrieval involves repeatedly querying the knowledge base based on the initial user query and the text generated [14]. By incorporating the generated text into the retrieval process, iterative retrieval allows the system to build a more comprehensive understanding of the user's intent. Iterative retrieval can lead to more robust answer generation, as the Large Language Model (LLM) can access a broader range of contextual references.

However, iterative retrieval can be susceptible to two issues:

- **Semantic discontinuity:** If the generated text deviates significantly from the original query, the retrieved information might not be relevant.
- **Accumulation of irrelevant information:** Each retrieval iteration adds new information to the pool. Irrelevant information retrieved in earlier iterations can be carried forward, negatively impacting the quality of subsequent retrieval steps.

Recursive retrieval: Recursive retrieval is a powerful technique used in information retrieval (IR) and Natural Language Processing (NLP) to improve the relevance of search results. Unlike iterative retrieval, which repeats the search based on the original query and generated text, recursive retrieval refines the search queries based on the information retrieved in previous rounds. This iterative process creates a feedback loop where the system continually hones in on the most relevant information.

One example system, IRCoT, leverages a "chain-of-thought" concept to guide the retrieval process [17]. This chain of thought essentially tracks the system's reasoning steps. IRCoT then refines this chain of thought based on the information retrieved in each round, ensuring the retrieval process stays focused and relevant to the user's intent.

Adaptive retrieval: Adaptive retrieval takes the concept of retrieval in RAG a step further. This approach allows Large Language Models (LLMs) to participate actively in the retrieval process. Pioneered by systems like Flare and Self-RAG, adaptive retrieval empowers LLMs to decide when and what information to retrieve. Adaptive retrieval not only improves the efficiency of the retrieval process but also ensures the retrieved data is highly relevant to the user's needs. Adaptive retrieval aligns with a growing trend in NLP, where LLMs take on a more active role. These models can process information and judge how to utilize it best.

C. GENERATION

Simply feeding all retrieved information to a Large Language Model (LLM) after the retrieval stage isn't ideal for question answering. To improve the quality of responses, we can make adjustments from two critical angles: tailoring the retrieved content and adapting the LLM itself. The following sections will explore these approaches in more detail.

Context Curation: Context Curation involves filtering out redundant information and summarizing the remaining passages, providing the LLM with more focused and digestible information for answer generation. During Retrieval-Augmented Generation (RAG), directly feeding all retrieved information to the Large Language Model (LLM) can be problematic. Redundant passages can hinder the LLM's ability to generate a clear answer, and excessively long contexts can lead to the "Lost in the Middle" problem [10], thus causing it to lose accuracy and effectiveness. Like humans struggling to retain information in lengthy passages, LLMs focus on the beginning and end, forgetting the crucial middle section. To address these issues, RAG systems typically employ context curation techniques to process retrieved content.

Re-ranking: Re-ranking is crucial in optimizing the information fed to the LLM within an RAG system. This process reorders the retrieved document chunks, placing the most relevant results at the forefront. Re-ranking acts as a dual-purpose filter and enhancer by reducing the overall document pool. It delivers a refined set of inputs more suitable for precise language model processing. There are two main approaches to re-ranking:

- **Rule-based methods:** These rely on pre-defined metrics like relevance, diversity, and Mean Reciprocal Rank (MRR) to reorder the document chunks.
- **Model-based methods:** This approach utilizes various machine learning models for re-ranking. Examples include encoder-decoder models like SpanBERT from the BERT family, specialized re-ranking models like Cohere Re-rank, and general large language models like GPT.

Context selection and Compression techniques: A common misconception in RAG is that retrieving every relevant document and feeding it all to the LLM is ideal. However, this approach can backfire. An overwhelming amount of context can introduce irrelevant information, making it difficult for the LLM to identify the key points. Context selection and compression techniques are employed to address this. One approach, exemplified by (Long) LLMLingua, utilizes small language models (SLMs) like GPT-2 Small or LLaMA-7B [9]. These SLMs act as information filters, identifying and removing unimportant details from the retrieved content. While the resulting compressed context might seem nonsensical to humans, it's optimized for efficient processing by the LLM.

LLM Fine-tuning: One way to improve the performance of LLMs in RAG systems is through targeted fine-tuning. This technique involves tailoring the LLM to a specific domain or task using relevant data. LLM Fine-tuning is a significant advantage, especially for on-premise LLMs where domain-specific data is readily available. By incorporating additional information, fine-tuning can address situations where an LLM lacks sufficient knowledge in a particular domain. Resources like fine-tuning data from Hugging Face can be a starting point for this process. An additional benefit of fine-tuning is that it allows you to customize the LLM's input and output to suit your specific needs better.

Evaluation of RAG: Practical RAG evaluation requires assessing both retrieval quality and generation quality.

- **Retrieval Quality:** We focus on how well the retriever component finds relevant information. Essentially, we want to ensure the Retriever provides the Generator with high-quality source material.
- **Generation Quality:** This evaluation looks at the Generator's ability to create coherent and relevant answers from the retrieved context. The approach differs depending on whether we deal with unlabeled or labeled content. For unlabeled content, we care about faithfulness (adheres to retrieved info), relevance (addresses the question), and non-harmfulness of the generated answers. On the other hand, labeled content evaluation focuses on the factual accuracy of the information the model produces [8].

Modern RAG evaluation hinges on three critical quality scores and four crucial capabilities. These metrics work together to assess the effectiveness of retrieval (finding relevant information) and generation (creating coherent answers) within the RAG model.

Three critical quality scores are fundamental to evaluating RAG models: context relevance, answer faithfulness, and answer relevance. [4] These metrics assess the RAG system's effectiveness in information retrieval and generation from different angles.

- **Context Relevance:** This score measures how well the retrieved information aligns with the user's query. It essentially evaluates the precision and focus of the retrieved context, ensuring it's relevant to the question and avoids unnecessary processing of extraneous content. [8]
- **Answer Faithfulness:** This score assesses how closely the generated answers adhere to the retrieved information. It ensures consistency and prevents the model from introducing contradictions or hallucinations not supported by the retrieved context. [8]
- **Answer Relevance:** This score focuses on whether the generated answers directly address the user's question. It evaluates if the response truly answers the core inquiry presented. [8]

RAG evaluation also considers four critical abilities that reflect the model's adaptability and efficiency: noise robustness, negative rejection, information integration, and counterfactual robustness [11] [1]. These abilities are crucial for a model's performance in complex scenarios and ultimately impact the quality scores mentioned earlier.

- **Noise Robustness:** This ability assesses how well the model handles "noisy" documents - documents related to the user's query but lacking substantial information. A robust model can avoid being misled by such irrelevant content.
- **Negative Rejection:** This ability evaluates the model's ability not to provide an answer when the retrieved documents don't have the data to answer the question. A

good model should recognize when it cannot provide a helpful response.

- **Information Integration:** This ability assesses the model's proficiency in synthesizing information from multiple retrieved documents to create a comprehensive answer to complex questions. An effective model can draw connections and combine knowledge from various sources.
- **Counterfactual Robustness:** This ability tests the model's capacity to disregard known inaccuracies within documents, even when prompted about potential misinformation. A robust model should avoid generating responses based on factual errors.

The interplay between these abilities and the quality scores (context relevance, answer faithfulness, and answer relevance) ultimately determines an RAG system's effectiveness. For instance, high context relevance and noise robustness are essential for accurate retrieval, while answering faithfulness, answer relevance, negative rejection, information integration, and counterfactual robustness all contribute to high-quality generation.

Evaluation Benchmarks and Tools: Researchers have developed several vital benchmarks and tools to assess an RAG model's strengths and weaknesses. Benchmarks like RGB, RECALL, and CRUD [11] [1] specifically evaluate the four essential abilities of RAG models: noise robustness, negative rejection, information integration, and counterfactual robustness. These benchmarks provide standardized testing grounds to compare the performance of different RAG systems.

In addition to benchmarks, state-of-the-art automated tools like RAGAS [4], ARES [13], and TruLens8 leverage Large Language Models (LLMs) to judge the quality scores of context relevance, answer faithfulness, and answer relevance. Essentially, these LLM-powered tools act as automated assessors, providing valuable insights into the effectiveness of an RAG system's retrieval and generation capabilities.

II. Materials and Methods

A. DATASET

The GSM8K dataset comprises 8,792 human-written elementary school math word problems spanning various languages. It's divided into training (7,473 problems) and testing (1,319 problems) sets. Each problem requires 2 to 8 steps to solve using basic arithmetic operations. [2]

Unlike many datasets that use mathematical expressions, GSM8K provides solutions in plain language. This makes the data easily understandable for humans and allows for broader application of our research methods. https://github.com/openai/grade-school-math

B. MODEL

This research leverages the capabilities of the GPT-3.5-Turbo language model, accessed through OpenAI's API and Llama Index.

C. EXPERIMENTS

The section below details the experiments performed using the GSM8K dataset.

Experiment 1: Baseline Performance of GPT-3.5 on GSM8K train set

In this experiment, we assess GPT 3.5's overall performance in answering the questions in the GSM8K train set by prompting each question exactly once and accepting a single answer. We found that GPT 3.5 answered 69% of the questions correctly.

Experiment 2: Multi Attempt Answering

In this experiment, we assess GPT 3.5's overall performance in answering the questions in the GSM8K train set by prompting each question exactly once and accepting multiple answers to the same question. The answer was considered correct even if a single attempt contained the correct answer. We found that GPT 3.5 answered 92% of the questions correctly.

Experiment 3: Effect of Prompting

In this experiment, we assessed GPT 3.5's overall performance by prompting each question with five examples from the training set and accepting a single answer. We found that GPT 3.5 answered 84% of the questions correctly. We noticed a significant improvement in the accuracy from 69 percent of Experiment 1 to 84%, indicating that the performance of GPT 3.5 increased with the presence of hints in the prompt.

Experiment 4: Naive RAG

In this experiment, we selected the questions and answers from the GSM8K train dataset and created an index using Chroma, a vector database. When querying GPT 3.5 with questions from the train and test dataset set, the vector database was also queried for the top three matching documents and used to construct the prompt for GPT 3.5. The questions were prompted only once, and a single answer was recorded. The accuracy of the answers on the training dataset increased to 96% compared to 69% in Experiment 1. The accuracy of answers on the test dataset was found to be 76%. It's important to note these differences in the accuracy of the test and train datasets, as the vector database contained only data from the training dataset.

Experiment 5: Naive RAG without the top matching document in the context

In this experiment, we used the previously created index using Chroma on the training dataset. When querying GPT 3.5 with questions from the train set, the vector database was also queried for the top three matching documents; the document with the highest match was removed, and only two documents were used to construct the prompt. The questions were prompted only once, and a single answer was recorded. The accuracy of the answers on the training dataset was 83%, a significant drop in accuracy compared to 96% in Experiment 5. The superior performance in experiment 5 can be mainly attributed to the presence of direct hints on how to solve the problem.

Experiment 6: Naive RAG evaluation using RAGAS

In this experiment, we used the vector database generated using the train set. When querying GPT 3.5 with 2,500

questions from the train set, the vector database was also queried for the top three matching documents and used to construct the prompt for GPT 3.5. The questions were prompted only once, and a single answer was recorded. The performance was evaluated using RAGAS. The accuracy of answers on the dataset was found to be 76%.

Metric	Values
Faithfulness	31.7%
Answer Relevancy	79.2%
Context Recall	84.16%
Context Precision	91.88%
Harmfulness	0.5%
Answer Similarity	89.10%

Experiment 7: Advanced RAG using Hyde

Hyde is a query transformation technique that prompts LLMs to rewrite the user's queries into a more suitable format for information retrieval. [5] Instead of relying solely on the user's original query, query transformation retrieves document chunks based on a reformulated version of the query, as the original queries might not always be ideal for retrieval by large language models (LLMs) in real-world situations.

In this experiment, we used the previously created index using Chroma on the training dataset. When querying GPT 3.5 with questions from the train set, the vector database was also queried for the top three matching documents using the query transformation generated using Hyde. The questions were prompted only once, and a single answer was recorded. The accuracy of the answers on the training dataset was 99%, a significant increase in accuracy compared to 96% in Experiment 4. The superior performance can be mainly attributed to direct hints in the prompt and a transformed query to find the matching documents from the vector database instead of using the question.

D. ANALYSIS

Experiment	Accuracy
Baseline Performance of GPT-3.5	69%
Multi Attempt Answering	91%
Effect of Prompting	84%
Naive RAG	96%
Naive RAG without the top matching document	84%
Advanced RAG using Hyde	99%

GPT 3.5's performance improves significantly with RAG. When advanced RAG techniques like Hyde are used, accuracy increases pronouncedly.

III. Conclusion

The main idea of the research was to study the effect of RAG on mathematical reasoning. We can see that RAG significantly improves the accuracy of solving the GSM8K dataset compared to using only GPT 3.5. The performance of the RAG systems can be attributed to direct hints from the retrieved context. A possible direction of future research would be to explore knowledge graphs to retrieve the context for generating prompts, as this would enable us to abstract and group or classify similar problems.

References

[1] Jiawei Chen et al. "Benchmarking large language models in retrieval-augmented generation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 16. 2024, pp. 17754–17762.

[2] Karl Cobbe et al. "Training verifiers to solve math word problems". In: *arXiv preprint arXiv:2110.14168* (2021).

[3] Shehzaad Dhuliawala et al. "Chain-of-verification reduces hallucination in large language models". In: *arXiv preprint arXiv:2309.11495* (2023).

[4] Shahul Es et al. "Ragas: Automated evaluation of retrieval augmented generation". In: *arXiv preprint arXiv:2309.15217* (2023).

[5] Luyu Gao et al. "Precise zero-shot dense retrieval without relevance labels". In: *arXiv preprint arXiv:2212.10496* (2022).

[6] Yunfan Gao et al. "Retrieval-augmented generation for large language models: A survey". In: *arXiv preprint arXiv:2312.10997* (2023).

[7] IVAN ILIN. *Advanced RAG Techniques: an Illustrated Overview*. 2023. URL: https://pub.towardsai.net/advanced-rag-techniques-an-illustrated-overview-04d193d8fec6.

[8] C. Jarvis and J. Allard. *A survey of techniques for maximizing llm performance*. 2023. URL: https://www.youtube.com/watch?v=ahnGLM-RC1Y&t=1s.

[9] Huiqiang Jiang et al. "Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression". In: *arXiv preprint arXiv:2310.06839* (2023).

[10] Nelson F Liu et al. "Lost in the middle: How language models use long contexts". In: *Transactions of the Association for Computational Linguistics* 12 (2024), pp. 157–173.

[11] Yi Liu et al. "Recall: A benchmark for llms robustness against external counterfactual knowledge". In: *arXiv preprint arXiv:2311.08147* (2023).

[12] Xinbei Ma et al. "Query rewriting for retrieval-augmented large language models". In: *arXiv preprint arXiv:2305.14283* (2023).

[13] Jon Saad-Falcon et al. "Ares: An automated evaluation framework for retrieval-augmented generation systems". In: *arXiv preprint arXiv:2311.09476* (2023).

[14] Zhihong Shao et al. "Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy". In: *arXiv preprint arXiv:2305.15294* (2023).

[15] Freda Shi et al. "Large language models can be easily distracted by irrelevant context". In: *International Conference on Machine Learning*. PMLR. 2023, pp. 31210–31227.

[16] Yixuan Tang and Yi Yang. *MultiHop-RAG: Benchmarking Retrieval-Augmented Generation for Multi-Hop Queries*. 2024. arXiv: 2401.15391 [cs.CL]. URL: https://arxiv.org/abs/2401.15391.

[17] Harsh Trivedi et al. "Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions". In: *arXiv preprint arXiv:2212.10509* (2022).

[18] Yu Wang et al. "Knowledge graph prompting for multi-document question answering". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 17. 2024, pp. 19206–19214.

[19] Wenhao Yu et al. "Generate rather than retrieve: Large language models are strong context generators". In: *arXiv preprint arXiv:2209.10063* (2022).

[20] Denny Zhou et al. *Least-to-Most Prompting Enables Complex Reasoning in Large Language Models*. 2023. arXiv: 2205.10625 [cs.AI]. URL: https://arxiv.org/abs/2205.10625.

[21] Denny Zhou et al. "Least-to-most prompting enables complex reasoning in large language models". In: *arXiv preprint arXiv:2205.10625* (2022).