

Mapping the Mind: Knowledge-Graph Augmented Retrieval

Stanford CS224N Custom Project

Nicholas Vo

Department of Computer Science
Stanford University
nickvo@stanford.edu

Abstract

Retrieval augmented generation techniques (RAG) have continued to serve as a solution to the limited context-window size and static knowledge-base of existing large language models (LLMs). However, many RAG techniques function by retrieving small document chunks based on semantic similarity to a query, failing to include context on broader themes or nuanced ideas. In this project, we investigate how to leverage knowledge graphs in producing structured information that supplements retrieval tasks. Our main contribution is a new framework for RAG that combines ideas from multi-query vector searching, knowledge graph traversal, and summarization techniques. To benchmark, we gauge accuracy over a long-context length question and answering dataset, demonstrating comparable performance to existing systems. Qualitatively, we analyze success and error modes across sample queries.

1 Key Information to include

- Mentor: Anna Goldie
- External Collaborators: None
- Sharing project: No

2 Introduction

Knowledge graphs are graph structures that represent entities and relationships where nodes represent entities and edges represent relations. For example, a knowledge graph constructed from the Harry Potter series may include a node for "Harry" with outgoing edges denoting Harry's actions, relationships, and views with respect to other entities. The information encoded within knowledge graphs is structured and offers the ability to perform efficient querying or lookups (Chen et al. (2024)).

In contrast, LLMs encode knowledge in a parametric form through their weights. As a result, while LLMs excel at encoding generalized knowledge and language processing abilities, they may fail in capturing domain-specific or up-to-date knowledge. Furthermore, they may "hallucinate" which is problem exacerbated by the fact that it is difficult to trace the source of the information. Initial approaches to provide LLMs with external knowledge involved fine-tuning on additional data. However, with larger models leading to longer fine-tuning times and ever-evolving data, this becomes unfeasible.

RAG emerged as another approach to providing external knowledge by retrieving relevant documents based on the query and providing them as additional context during inference (Lewis et al. (2021) Gao et al. (2024)). Traditional RAG techniques involve splitting a document into chunks such that they may fit within the limited context-window size of an LLM. On retrieval, documents are sourced by their semantic similarity to the query. RAG performs exceptionally well on direct, single-subject

queries such as "tell me about Harry Potter". In this case, document chunks mentioning Harry Potter would likely be retrieved, allowing the LLM to use this information as context when answering the query.

Drawbacks of this method, however, include chunking that results in a loss of semantic meaning, a lack of holistic understanding across chunks, and failure to retrieve relevant documents based on the query. For queries that call for complex reasoning across a number of concepts and ideas, these drawbacks present a challenge to using RAG for grounded, accurate results. To exemplify this challenge, we imagine the example query of "summarize the consequences of Harry Potter's actions across all books." To retrieve relevant documents, we would first need to retrieve documents about the actions of Harry Potter as well as documents that outline later effects – all across the entire corpora. And, without a holistic understanding of these documents, this would not be possible with similarity based retrieval.

Evidently, LLMs and knowledge graphs both present advantages and shortcomings that may compliment each other (Pan et al. (2024)). In this paper, we propose a novel RAG framework that involves components powered by knowledge graph databases and LLMs. By synthesizing the two, we hope to address the drawbacks of existing RAG techniques and LLMs. Specifically, our solution seeks to combine the complex, generalized reasoning abilities of LLMs with the efficient, grounded information storage of knowledge graphs.

Our framework achieves this through a process of query decomposition into sub-queries that are used to lookup relevant entities from a pre-constructed knowledge graph database. With this knowledge graph lookup, we return a structured list of entity properties and outgoing relationships. We then summarize this structured information and produce candidate neighborhood nodes to continue traversing. From candidate nodes, we continue traversal and querying for a defined depth. At each traversal step, we summarize the structured knowledge graph information and perform a vector search on relevant documents. With collected node summaries, relevant documents, and subqueries, we prompt the LLM to answer the original query. Figure 1 summarizes this approach.

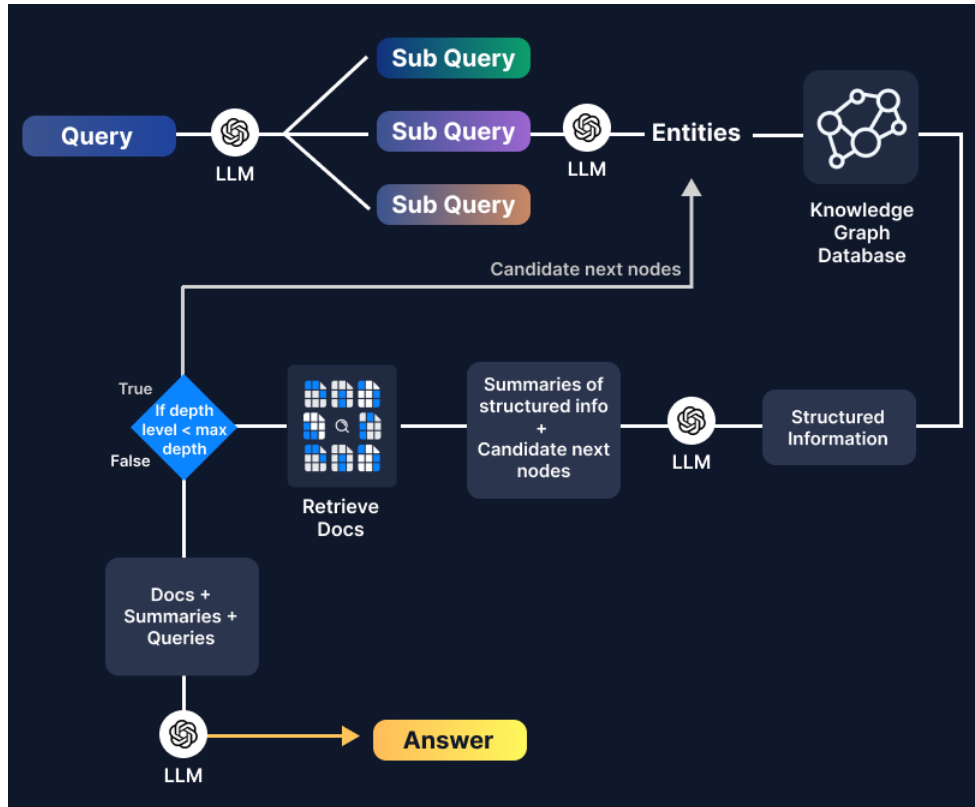


Figure 1: Complete pipeline of our approach

3 Related Work

Summarization RAG

In the realm of RAG, traditional techniques split long documents into defined chunks, resulting in a limited understanding of the entire document context. To combat this, summarization approaches have sought to provide a holistic understanding of entire documents. Through summarizing snippets or recursively summarizing chunks, these approaches allow for broader themes to be captured sequentially (Gao et al. (2023) Wu et al. (2021)). To capture broader themes or ideas that might not be adjacent to one another within the documents, RAPTOR is a recent approach that recursively generates summaries on semantically similar chunks of document text (Sarathi et al. (2024)). However, through only clustering semantically similar chunks, ideas that may interact with one another without semantic similarity are ignored. Ideally, we could summarize across the entire chunk combination space, but this would be computationally expensive.

Graph Based RAG

Knowledge graph based RAG approaches have found recent exploration in the past year. Neo4j proposes graph based RAG that simply returns queried knowledge graph information in addition to relevant documents based on query similarity. As a result, the documents returned are still based on standard RAG techniques but with grounded context from the knowledge graph. Recently, GraphRAG by Microsoft Research addresses these gaps by leveraging knowledge graphs to create summaries based on semantically similar clusters (Edge et al. (2024)). This approach allows for semantic structure to be considered and gives a higher-level view across multiple documents. Nonetheless, clustering nodes and documents to recursively create summaries presents an expensive challenge – especially as the document size scales. Our approach differs from both GraphRAG and RAPTOR by arguing that pre-summarization across clusters or chunks may not be required with proper querying at inference.

Querying

Chain of thought prompting was introduced as a way to improve the reasoning abilities of LLMs by suggesting they take a step-based approach to answering a question (Wei et al. (2023)). By combining answers to sub questions, the LLM should be guided to generate a more logical response. Similarly, multi query retrieval involves prompting an LLM to generate multiple sub queries that can be used to retrieve documents. By generating sub queries, the goal is to construct specialized queries that have may have more distinct semantic meaning to relevant documents. This idea allows for more granularity when retrieving relevant documents, and we utilize this as a part of our querying system.

4 Approach

4.1 Overview

Our approach to RAG involves two different stages. First, we must pre-process documents to construct our knowledge graph database. This step is adapted from an existing library and implementation with minor modifications to include a wider range of entities and relationships (Neo4J (2024)). After we have constructed our knowledge graph, we are then able to complete the second stage of querying. We implement this from scratch, utilizing the langchain library to construct prompt chains, structured output generation, and graph and similarity search (langchain (2024)).

4.2 Pre-processing Information

Before we are able to query, we need to construct our knowledge graph database from documents. To do this, we first split each document into smaller chunks at length of 200. Splitting is completed by recursively splitting by characters in order to preserve the semantic meaning of sentences and paragraphs (LangChain (2024)). After splitting documents, we then use GPT-4 as our LLM to parse through the chunks and extract any real-world, person, or object entities and relationships. While we use GPT-4, a smaller-model could be used to reduce costs. With the extracted nodes and edges, we then populate our knowledge graph database. It is important to note, that along with the extracted nodes and edges, we also use the knowledge graph database to store the original document chunks and respective vector embeddings of those chunks.

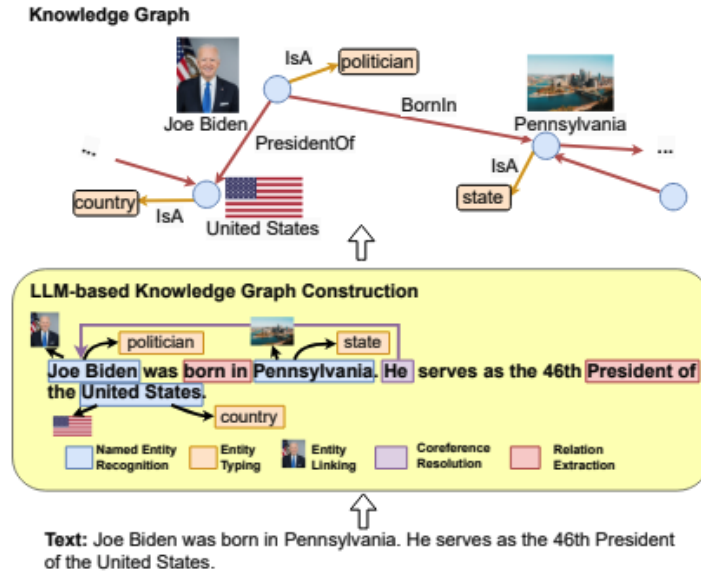


Figure 2: Constructing a knowledge graph using LLMs. Figure from Chen et al. (2024)

4.3 Querying

With a constructed knowledge graph database, we can now define a querying process for our RAG approach. The process for querying involves the following steps using GPT-4 as our LLM:

- We first query the LLM using chain-of-thought prompting to generate a series of sub queries required to answer the question, focusing on the diversity of each prompt. We set the number of *max_sub_queries* to be generated as a parameter.
- For each generated sub query, we extract relevant entity nodes and query our knowledge graph to produce structured node-relationship information. Figure 5 displays a sample of this structured output.
- Next, we (1) summarize this structured output as it pertains to each sub query and (2) ask for any neighboring nodes that could contribute towards answering the question. We then combine this summary with the original sub query into a string and convert it into a vector embedding using OpenAI Embeddings. This is then used to perform a vector similarity search using the original document chunk vector store. The specific scoring metric that is used is cosine similarity score, though this can also be modified. We return the *top_k* similar documents where *top_k* is a set parameter. Additionally, we return the next neighboring nodes.
- For efficient storage and lookup, we maintain the documents and next neighboring nodes as individual sets. With the next neighboring nodes, we conduct steps 2 and 3 for a set *max_depth* parameter where *max_depth* = 2 means we complete an additional pass.
- Lastly, we return the sub queries, node-relationship summaries, and original docs as context to the LLM to answer our original question.

5 Experiments

5.1 Data

Given we are assessing the ability of document retrieval, We use the QuALITY dataset for experimentation (Pang et al. (2022)). This dataset contains long context passages in English that range from approximately 2000-8000 tokens. Each passage contains a list of multiple-choice question and answering pairs which allows us to conduct benchmarks on accuracy as opposed to sentence

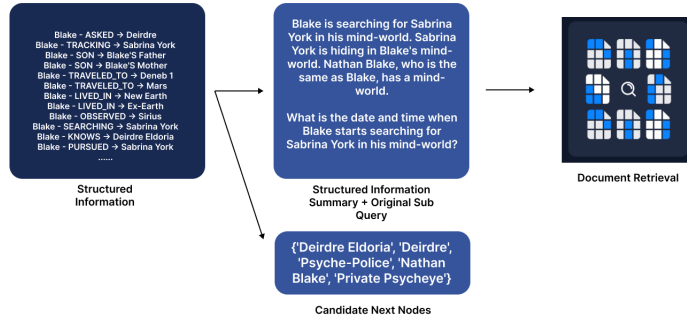


Figure 3: Sample structured output to retrieval pipeline from knowledge graph query

similarity metric scores such as BLEU or METEOR. We evaluate using the dev set as the test set does not include ground truth labels.

5.2 Evaluation method

To evaluate, we use a measure of accuracy as the answer options are multiple-choice. In order to generate answers, we utilize a structured output LLM call to produce an integer for each answer option. We evaluate using the dev set as the test set does not contain annotated ground truth labels. For qualitative evaluation, we make use of logged outputs at each step of the architecture. Evaluation occurs across different architecture choices as an ablation study.

5.3 Experimental details

For experimenting, we utilized GPT-4 across all tasks and with the following parameters across all design configurations: $max_sub_queries=3$, $top_k=3$. Our knowledge graph was first created by pre-processing the long context passages represented in the QuALITY dataset. For this split, knowledge graph creation took a considerable time as we did not parallelize requests. Using this constructed knowledge graph, we then perform testing across several architecture choices. The design paradigm that performed the best is the one proposed within this paper.



Figure 4: Knowledge graph constructed from QuALITY

5.4 Results

From our experiment, we achieved a baseline accuracy of 64.7 using GPT-4 and a standard RAG approach that uses cosine similarity to retrieve relevant documents. Next, we test using our knowledge graph retrieval system but without source documents and with a depth of 1. This means that we do not retrieve or include original source documents as context when answering the question; rather, we solely include our summaries constructed from our knowledge graph queries. With a depth of 1, this means that we do not continue to traverse along any candidate next nodes.

For this configuration, we achieve an accuracy that under performs significantly compared to our baseline. This indicates that the information returned from knowledge graph querying is not sufficient without the actual document chunks. From this test, we then continued testing the same configuration but with an increased depth. However, we only achieve a marginally improved accuracy which highlights the importance of including documents.

Our next results were retrieved by implementing the entire pipeline. We vary the max depth of traversal and retrieve accuracies of 69.3, 76.1, and 70.9 as depth increases from 1 to 3. These results are surprising and indicate that increasing the depth of traversal does not provide meaningful information that can improve accuracy. However, this could be due to the additional noise that is added to the context, or conflicting entities across all documents. Nonetheless, a notable finding is that the additional context returned from structured knowledge graph information significantly improves on our baseline accuracy. Specifically, at a depth of 2, we see a remarkable increase of more than 10 percent.

Model Architecture	Accuracy
Standard RAG	64.9
KGraph RAG (no source docs) max depth=1	55.3
KGraph RAG (no source docs) max depth=2	57.7
KGraph Augmented RAG max depth=1	69.3
KGraph Augmented RAG max depth=2	76.1
KGraph Augmented RAG max depth=3	70.9

Table 1: Model performance on dev split of QuALITY dataset using GPT-4

In comparing accuracy on the QuALITY dataset to state-of-the-art approaches, we do not have access to direct data comparisons as accuracies from RAPTOR are reported on the test dataset or on the dev dataset using GPT-3. However, as a loose comparison, RAPTOR + GPT-4 achieved an accuracy of 82.6 on the test dataset. Overall, these initial results show that we can achieve similar success on question and answering across medium-long length datasets without extensive pre-processing of the documents.

6 Analysis

For analysis, we conduct observations across several failure and success modes represented across architecture choices. Throughout the experimentation phase, we logged intermediate outputs. Thus, we are able to leverage these intermediate outputs to complete our qualitative analysis.

Failure Modes

The main failure modes we analyze are based on retrieval of structured knowledge graph information and subsequent summarizations. In understanding why our knowledge graph approach without returning source documents failed, it is evident that many sub queries produced knowledge graph entities and relationships that were unable to answer the sub query. As a result, when prompting the model with the final context, there was no concrete data to support potential claims.

In understanding the error mode of depth, we also analyze how and what candidate nodes are produced for traversal. For most queries, despite setting a max depth of 3, the model often only completes a single pass. This is because we filter for already queried entities and only choose to traverse new entities. This finding is surprising and does not allow for the knowledge and context to expand beyond

```

Why does the Tr'en's logic fail them?
Retriever Step:
Sub-Question:
What is the context or scenario involving the Tr'en's logic?
Structured data:
The context or scenario involving the Tr'en's logic is not provided in the given input.

Retriever Step:
Sub-Question:
What specific logical steps or arguments do the Tr'en use?
Structured data:
The Tr'En have several connections that might provide insight into their logical steps or arguments. They are led by a Ruler and have a government structure. They created a Lie-Detector, which could be a tool they use in their logical processes. The Tr'En have interactions with Korvin, including capturing and opposing him, which might reveal their strategic or argumentative methods.

Retriever Step:
Sub-Question:
What are the potential flaws or fallacies in the Tr'en's logic?
Structured data:
Tr'En is associated with several entities and roles: it created the Lie-Detector, is a member of Didiyak, is led by the Ruler, and has a government. Korvin has multiple interactions with Tr'En, including being captured by, opposed to, and escaping from Tr'En. The Ruler rules and leads Tr'En. Didiyak is a member of Tr'En.

{'Ruler', 'Korvin', 'Didyak', 'Lie-Detector', 'Government'}

The Tr'en's logic fails them primarily due to their interactions with Korvin. The Ruler of Tr'en, who commands and questions Korvin, relies heavily on a Lie-Detector to verify information. Korvin, who is from Earth and a member of the Comity of Planets, describes and defines concepts such as Government and Democracy to the Ruler. Despite the Ruler's use of the Lie-Detector, Korvin's responses and definitions may not align with the Tr'en's understanding or expectations, leading to a failure in their logic. Additionally, the Ruler's consultations with Experts and commands to Guards and Didiyak, who are also involved with Korvin, suggest a complex web of interactions that could contribute to the logical failure.

{'Experts'}
The Tr'en's logic fails them because the Experts, who are commanded and consulted by The Ruler, interacted with and questioned Korvin. This interaction and questioning likely led to a failure in their logic.

```

Figure 5: Sample success output of intermediate steps

the initial query. In moving forward, we may experiment with methods to retrieve a more diverse set of potential entities.

Success Modes

Success modes largely occur when the sub queries are able to be accurately answered. However, even when sub queries do not retrieve relevant information from knowledge graph querying and summarization, they are still able to answer the question through multi-query vectoring. Furthermore, in samples where subqueries are not answered, the returned documents are able to supplement as context. This indicates that there is a large advantage in returning both structured and unstructured data.

7 Conclusion

In this project, we experimented with using knowledge graphs for RAG tasks, primarily focusing on performance on multiple-choice question and answering across medium-long length documents. Our results indicate that we achieve performance similar to summarization RAG techniques without needing to pre-process our data with an expensive summarization method.

Nonetheless, the power of knowledge graphs is in their ability to connect ideas through nodes across a large number of varying documents. In testing on medium-long length documents, we may not have utilized the abilities of knowledge graphs to their full potential. Similarly, there may be more efficient ways to query a knowledge graph through LLM constructed filters or custom queries. For future-work, we hope to evaluate our approach against multi-hop datasets to further investigate how knowledge graphs can assist in augmenting LLMs with the ability to reason across an external, diverse knowledge base.

8 Ethics Statement

We first acknowledge that our architecture and exploration into knowledge graphs presents an expensive solution that is inaccessible to many. In testing for this paper, available token credits and an enterprise account allowed for testing without any rate limits. In testing other architectures such as RAPTOR, an account with increased rate limits and a nontrivial budget for tokens was also required for timely testing. To allow for more accessible testing of these solutions, providing public access to

previously constructed knowledge graph databases or recursively summarized document structures on large datasets would allow for researchers to test without the required financial investment for construction of such structures.

With large databases and knowledge stores, there is a valid potential for (1) bias to be represented within the dataset and (2) inequitable access to large datasets. With regards to bias, RAG approaches provide a method to cite retrieved documents which provides transparency on how an LLM reasons. Despite this, however, LLMs may still exhibit their own bias that is encoded in their parameters or they may hallucinate even with the added context. For users of RAG applications – especially those using RAG to make critical decisions – there should be caution taken when interpreting responses. As creators and researchers, we have a responsibility to implement safeguards and UI features such that users are aware of the risk. To address how RAG may lead to inequitable access to data, there is already an inequity in large corporations owning private datasets and documents. With the effectiveness of RAG, this will allow for these institutions to have access and the ability to draw on knowledge from this private information. Inversely, however, RAG also allows for more accessibility to utilizing external databases for analysis. Through scraping our individual data collection, a user can conduct complex analysis across large documents. Overall, to mitigate inequitable access, we suggest for institutions to make large data accessible and open-source after consulting with third-party institutions focused on AI policy. This will ensure that release of harmful data does not occur.

References

- Zhuo Chen, Yichi Zhang, Yin Fang, Yuxia Geng, Lingbing Guo, Xiang Chen, Qian Li, Wen Zhang, Jiaoyan Chen, Yushan Zhu, Jiaqi Li, Xiaoze Liu, Jeff Z. Pan, Ningyu Zhang, and Huajun Chen. 2024. Knowledge graphs meet multi-modal learning: A comprehensive survey.
- Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization.
- Tianyu Gao, Howard Yen, Jiatong Yu, and Danqi Chen. 2023. Enabling large language models to generate text with citations.
- Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-augmented generation for large language models: A survey.
- LangChain. 2024. "<https://api.python.langchain.com/en/latest/character/langchain-text-splitters.character.RecursiveCharacterTextSplitter.html>".
- langchain. 2024. Langchain documentation. <https://python.langchain.com/v0.2/docs/introduction/>.
- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-augmented generation for knowledge-intensive nlp tasks.
- Neo4J. 2024. Knowledge graphs for rag. <https://neo4j.com/developer-blog/knowledge-graph-rag-application/>.
- Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):3580–3599.
- Richard Yuanzhe Pang, Alicia Parrish, Nitish Joshi, Nikita Nangia, Jason Phang, Angelica Chen, Vishakh Padmakumar, Johnny Ma, Jana Thompson, He He, and Samuel R. Bowman. 2022. Quality: Question answering with long input texts, yes!
- Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. 2024. Raptor: Recursive abstractive processing for tree-organized retrieval.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.
- Jeff Wu, Long Ouyang, Daniel M. Ziegler, Nisan Stiennon, Ryan Lowe, Jan Leike, and Paul Christiano. 2021. Recursively summarizing books with human feedback.