A Study of Knowledge Graph Integration in Retrieval-Augmented Generation

Nathaniel Christian - 13524122
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: nathaniel.christian@gmail.com, 13524122@std.stei.itb.ac.id

***Abstract*—Most of Large Language Models (LLMs) struggle with hallucinations due to their limited and static internal knowledge. Retrieval-Augmented Generation (RAG) improves this by providing models with relevant external memory. However, standard RAG typically uses flat vector search without preserving semantic relationships. GraphRAG addresses this limitation by integrating Knowledge Graphs (KGs) into the retrieval process. In this paper, we examine how knowledge graphs can be constructed from textual sources (e.g., support logs or FAQs), how relevant subgraphs can be retrieved using algorithms such as breadth-first search, and how this structured context can improve question-answering performance.**

***Keywords*—GraphRAG, Retrieval-Augmented Generation, Question Answering, Knowledge Graphs**

I. Introduction

In recent years, Large Language Models (LLMs) have improved exponentially. These modern generative models, trained on a massive scale of data, are capable of performing complex tasks such as summarization, translation, question answering, and more. However, they still have some downsides—most notably, hallucination, which in the context of LLMs refers to the generation of content that is inconsistent with reality or factual information [1]. To address this issue, researchers have developed methods to augment generative models with external knowledge, enabling them to retrieve supporting information from a database or document collection during inference. This approached is known as Retrieval-Augmented Generation (RAG) [2]. By referencing external sources, RAG reduces the problem of generating false or incorrect text or content.

RAG enables LLMs to improve factuality and maintain up-to-date knowledge by incorporating a large document corpus—as context for generating answers. While this is an effective improvement over standalone generation, traditional RAG still treats retrieved documents as unstructured text [3]. As a result, they often overlook the underlying relationships and structures present within or across documents, making it difficult for the model to perform multi-step reasoning or link related concepts.

In response to this limitation, more structured retrieval methods have been proposed, one of which is GraphRAG. This method integrates knowledge graphs—graph-based representations of entities and their relationships—into the RAG pipeline. By using these graphs, GraphRAG improves the quality and clarity of the information retrieved to support LLM-generated responses.

II. RAG Overview

Naïve RAG combines two major components: a retriever and a generator. The retriever selects relevant documents or passages from an external knowledge base based on a given input query, while the generator uses the retrieved texts to produce an answer.

A naïve RAG pipeline operates as follows: Let a user provide a query q. The retriever searches a document collection D, which is processed into smaller chunks $\{c_1, c_2, \dots, c_k\}$. These chunks are embedded into vectors using an embedding model, which is stored in a vector database, such as FAISS. The query q is similarly embedded, and the retriever then compares these embeddings with the stored vectors using similarity metrics, such as cosine similarity, to identify the most relevant passages [5]. This part returns top-k most relevant document chunks $\{d_1, d_2, \dots, d_k\} \subseteq D$, that most likely contain the answer to the user's query. A visual overview of the entire pipeline is illustrated in Fig. 1.

The generator, typically a large language model, then takes the original query q and the retrieved passages as input context to generate an output a. This can be represented as:

$$a = \text{Generator}\ (q \mid \{d_1, d_2, \dots, d_k\}) \qquad (1)$$

This particular framework often lacks an understanding of deeper semantic structures or relationships across documents. Each passage is treated independently, and any interconnections—such as entity relationships or causal links—are ignored.

III. Knowledge Graphs

A knowledge graph (KG) is a structured representation of information in the form of a directed graph, where nodes represent entities and edges represent relationships between them [6]. Each edge in a knowledge graph typically captures a fact in the form of a triplet: *subject*, *predicate*, and *object*.

As early as 2012, Google introduced the knowledge graph as a semantic enhancement of Google's search function. Instead of just matching strings of text, it allowed the system to understand and search for "things"—real-world entities like people, places, or objects [7].

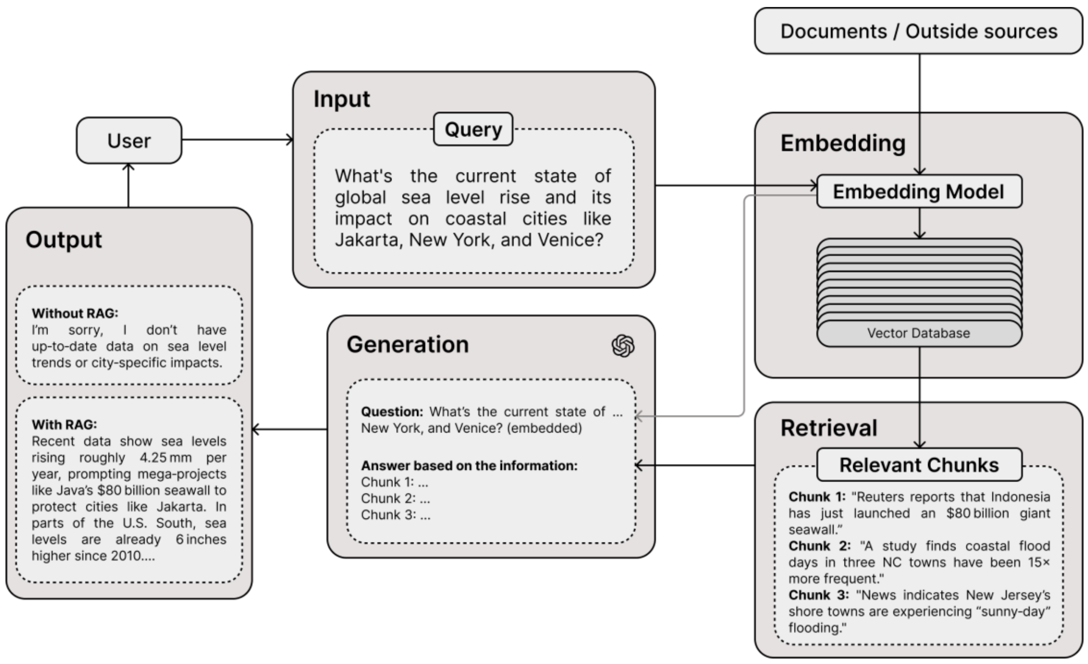

Documents / Outside sources
User
Input
Query
What's the current state of global sea level rise and its impact on coastal cities like Jakarta, New York, and Venice?
Embedding
Embedding Model
Vector Database
Output
Without RAG:
I'm sorry, I don't have up-to-date data on sea level trends or city-specific impacts.
With RAG:
Recent data show sea levels rising roughly 4.25 mm per year, prompting mega-projects like Java's $80 billion seawall to protect cities like Jakarta. In parts of the U.S. South, sea levels are already 6 inches higher since 2010....
Generation
Question: What's the current state of ... New York, and Venice? (embedded)
Answer based on the information:
Chunk 1: ...
Chunk 2: ...
Chunk 3: ...
Retrieval
Relevant Chunks
Chunk 1: "Reuters reports that Indonesia has just launched an $80 billion giant seawall."
Chunk 2: "A study finds coastal flood days in three NC towns have been 15× more frequent."
Chunk 3: "News indicates New Jersey's shore towns are experiencing "sunny-day" flooding."

Fig. 1. Overview of a naïve Retrieval-Augmented Generation (RAG) pipeline example. The system consists of a retriever and a generator. A user query is first embedded and used to retrieve relevant document chunks from a vector database. 1) Retrieval: retrieve the top-k chunks based on the user's query. 2) Generator: the retrieved chunks are combined with the query and passed to a LLM for answer generation. *Note: Figure layout design inspired by prior RAG architecture visualizations in [5], adapted and modified.*

This graph-based structure allows for efficient organization, querying, and inference over complex semantic relationships that are difficult to capture using plain text.

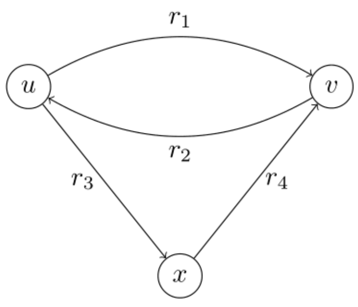


Fig. 2. An example of a knowledge graph represented as a labeled directed graph. Each node (e.g., u, v, x) represents an entity, and each labeled edge (e.g., r1, r2, r3, r4) encodes a semantic relation between entities.

Formally, a knowledge graph can be defined as a labeled directed graph:

$$G = (V, E) \quad (2)$$

where:

- V is the set of nodes (entities),
- R is the set of relation labels,
- $E \subseteq V \times R \times V$ is the set of directed, labeled edges (triplets), where each edge $(u, r, v) \in E$ represents a fact: entity u is related to entity v through relation r.

This triplet format is commonly used in Resource Description Framework (RDF) and is the basis for many graph-based knowledge systems such as DBpedia, Wikidata, YAGO, Google's Knowledge Graph, Facebook entity graphs, etc.

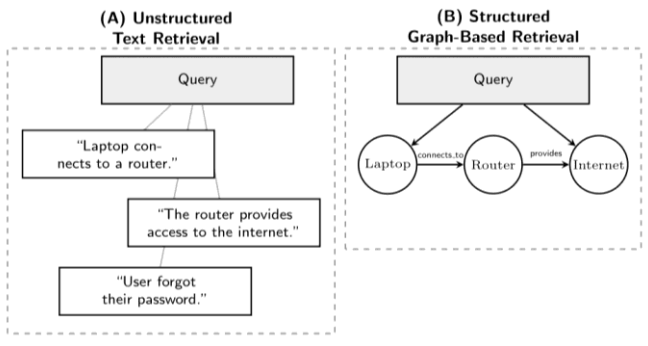
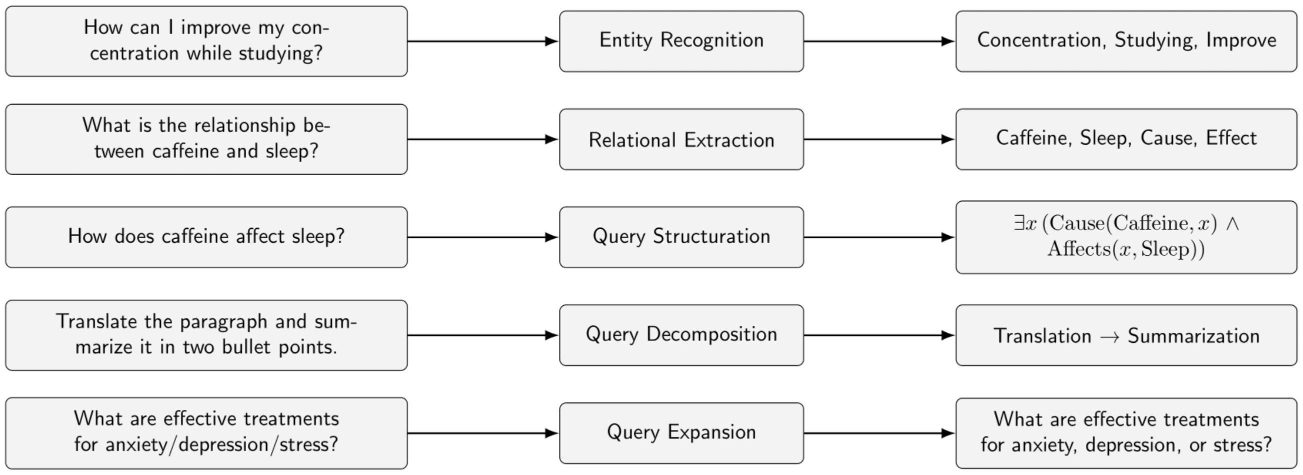

(A) Unstructured Text Retrieval
Query
"Laptop connects to a router."
"The router provides access to the internet."
"User forgot their password."
(B) Structured Graph-Based Retrieval
Query
Laptop
connects to
Router
provides
Internet

Fig. 3. Comparison between unstructured text retrieval (left), which selects passages independently, and structured graph-based retrieval (right), which captures semantic relationships through edges.

IV. GRAPHRAG

GraphRAG is a refinement of the RAG framework that introduces structure into the retrieval process by replacing flat document vectors with a knowledge graph. Instead of retrieving top-k passages based on vector similarity, GraphRAG retrieves subgraphs of semantically connected entities and relations that are relevant to a given query, thus naming it as a graph search problem.



How can I improve my concentration while studying?
Entity Recognition
Concentration, Studying, Improve
What is the relationship between caffeine and sleep?
Relational Extraction
Caffeine, Sleep, Cause, Effect
How does caffeine affect sleep?
Query Structuration
∃x (Cause(Caffeine, x) ∧ Affects(x, Sleep))
Translate the paragraph and summarize it in two bullet points.
Query Decomposition
Translation → Summarization
What are effective treatments for anxiety/depression/stress?
Query Expansion
What are effective treatments for anxiety, depression, or stress?

Fig. 4. Examples of query processor in GraphRAG. These include entity recognition, relation extraction, query structuration, decomposition, and expansion. Each step helps convert a raw user query into a more formal or structured representation suitable for graph-based reasoning and retrieval. *Note: Figure layout design inspired by query processor visualizations in [8], adapted and modified.*

With the rise of LLMs, the way knowledge graphs are used has started to change. Instead of serving only as static data sources, they now play a more active role in RAG pipelines.

Knowledge graphs help establish links between concepts, which can reduce hallucinations, add context, and support memory or personalization features for LLMs. This significantly improves RAG systems to be more robust and easier to scale—especially as the graph construction process becomes increasingly automated. Knowledge graphs not only serve as data stores for information retrieval but also as semantic structures that organize vector-based chunks.

Following the general structure of RAG, a GraphRAG system can be decomposed into few key components operating over a graph-structured data source G. Given a user-defined query q, the pipeline proceeds through the following components [8]:

- Query processor: transforms the input into a form compatible with the knowledge graph G.

$$\hat{q} = \text{Process}(q) \tag{3}$$

- Retriever: retrieve subgraphs relevant to the processed query $\hat{q}$ from the knowledge graph G.

$$C = \text{Retrieve}(\hat{q}, G) \tag{4}$$

- Organizer: refines and filters the retrieved graph C.

$$\hat{C} = \text{Organize}(\hat{q}, C) \tag{5}$$

- Generator: produces the final output a using both processed query $\hat{q}$ and refined subgraph $\hat{C}$.

$$a = \text{Generate}(\hat{q}, \hat{C}) \tag{6}$$

In a full implementation, this graph is typically constructed from unstructured documents through processes such as entity and relation extraction. However, in this example, we assume that the data is already organized in graph format.

Among these components, the query processor plays a crucial role in transforming text documents into a structured form that can interact meaningfully with a knowledge graph.

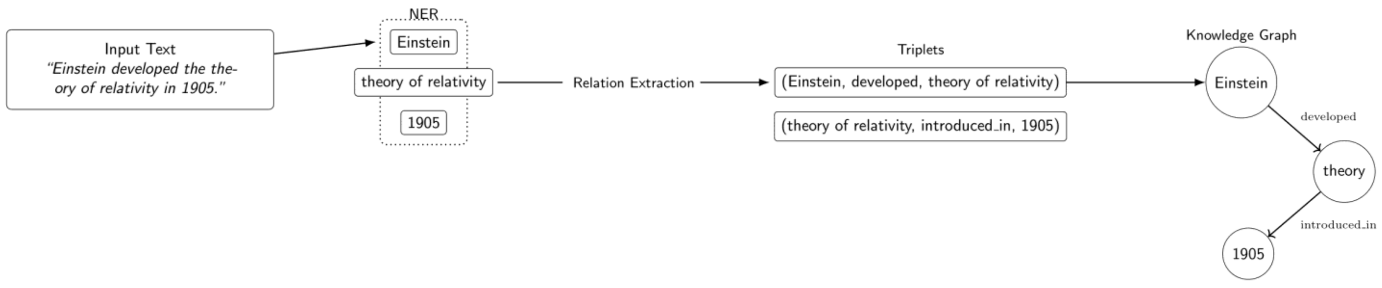
This transformation often involves a pipeline of sub-tasks such as entity recognition, relation extraction, query structuration, and query expansion. Each task helps reduce ambiguity and makes the meaning of the query clearer. These sub-tasks are shown in Fig. 4.

While GraphRAG primarily uses knowledge graphs built from entities and relations, the same framework can also be applied to other types of structured data, depending on the implementation domain. These include:

- Document graphs, where nodes represent sections and edges represent semantic links.
- Molecular graphs, which model atoms and chemical bonds for tasks like drug discovery.
- Social graphs, where nodes represent users and edges capture interactions such as friend status, shared interests, or message replies.

Each graph type comes with its own structure, semantics, and retrieval challenges, to make domain-specific implementation of the GraphRAG architecture [8, 10].

This structured retrieval process allows GraphRAG to preserve the semantic relationships between concepts, enabling more accurate and interpretable responses. By using graph searching rather than the usual similarity search, GraphRAG supports multi-hop reasoning, captures latent relationships, and organizes context in a way that improves robustness and scalability. These capabilities—multi-hop reasoning and latent relation capture—are explained in more detail in later section.



Input Text
"Einstein developed the theory of relativity in 1905."
NER
Einstein
theory of relativity
1905
Relation Extraction
Triplets
(Einstein, developed, theory of relativity)
(theory of relativity, introduced_in, 1905)
Knowledge Graph
Einstein
developed
theory
introduced_in
1905

Fig. 5. End-to-end pipeline for constructing a knowledge graph from text. The process starts with raw unstructured text, followed by named entity recognition and relation extraction, which are used to form subject–predicate–object triplets. These triplets are then assembled into a graph structure representing semantic relationships.

Overall, GraphRAG helps connect unstructured search with structured reasoning. It gives clear advantages in accuracy, clarity, and results for tasks that need a lot of knowledge in natural language processing.

V. Methodology

A. Graph Construction

The first step is to transform unstructured documents—such as FAQs, support logs, or research summaries—into a structured knowledge graph. This process usually involves the following stages:

1. *Named Entity Recognition (NER)*

   NER is the task of identifying real-world entities in the text, such as people, organizations, locations, or technical concepts. This process is usually done using natural language processing tools such as spaCy, BERT-based models, Stanford NER, domain-specific tagger, and many more.

2. *Relation Extraction*

   This stage extracts or identifies semantic relations between pairs of entities, typically using dependency parsing, pattern-based matching, or trained relation classifiers.

   For example, a sentence like "The router connects to the internet" would produce a semantic relation, as shown on the right side of Fig. 3.

3. *Triplet Formation*

   Present the extracted relation into the format of triplets (subject, predicate, object). Each extracted triplet becomes a labeled edge in the graph, as shown in (2), where $E \subseteq V \times R \times V$, as mentioned in Section III.

4. *Building the Graph*

   Map the triplets into a graph data structure (2), where each node in the graph represents an entity and each edge represents a semantic relation between two entities.

   This graph may be stored in a structured data format using graph libraries or graph databases. The resulting knowledge graph is used as the retrieval backbone in GraphRAG.

B. Query Preprocessing

When a user inputs a query q, the system applies a query processing pipeline to align it with the already-built knowledge graph. This includes:

1. *Entity Linking*

   Identify which entities in the KG are mentioned or implied in q.

2. *Relation Mapping*

   Match keywords or verbs in q to relation types in the knowledge graph.

3. *Query Structuring & Expansion*

   In some cases, we need to reduce the complexity of the questions into sub-questions or expand the query with synonyms, aliases, or inferred terms, as seen in Fig. 4.

This results in a structured query $\hat{q}$, which serves as input to the graph retriever, as mentioned in (3) and (4).

C. Subgraph Retrieval

The system retrieves a relevant subgraph $C \subseteq G$, as in (4), using graph traversal methods based on the structured query $\hat{q}$. Retrieval strategies may include [11, 12, 13]:

1. *k-hop Neighborhood Search*

   Collect all nodes within k steps from a source node (typically $k = 1,2,3$). This process generally employs the breadth-first search (BFS) algorithm to find the shortest paths and identify the k-hop neighbors efficiently. The equation is as follows:

$$N^k(v) = \{u \in V \mid \text{distance}(v,u) \leq k\} \quad (7)$$

   There are several important characteristics of k-hop neighborhood retrieval [14]:

- The value of k is determined by the shortest distance and it is unique. For example, in Fig. 6, suppose there are multiple paths between nodes A and C (such as A–C, A–D–C, and A–D–E–C). Since the shortest distance is 1 (A–C), node C will appear only in the 1-hop neighborhood of A and will not be included in the results of 2-hop or 3-hop queries.
- The k-hop results are deduplicated. For instance, if there are two distinct shortest paths between nodes A and E (e.g., A–C–E and A–D–E), node E will still appear only once in the 2-hop neighborhood of A.

This strategy ensures that subgraph retrieval remains both accurate and efficient, avoiding redundant traversal and maintaining consistent structural interpretation of node distances.

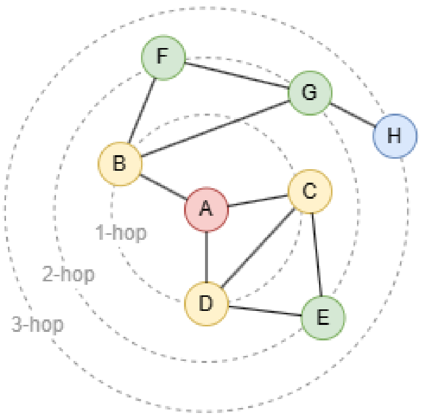


Fig. 6. An illustration of k-hop neighborhood around node A. Nodes $\{B, C, D\}$ are the 1-hop neighbors, $\{E, F, G\}$ are 2-hop neighbors, and node $\{H\}$ is the 3-hop neighbor. *Image taken from [14].*

2. *Path-based Retrieval*

Path-based retrieval aims to find connections between multiple entities mentioned in a user query. Rather than retrieving unrelated facts about each entity individually, the system identifies simple paths—non-repeating sequences of nodes—that link these entities together within the graph.

By chaining knowledge across multiple entities, we would get better answer. For example, consider the question: "What protein is associated with the disease treated by Drug X?". Here, the answer involves connecting Drug X → Disease Y → Protein Z.

Path-based retrieval captures this connection by identifying:

$$\mathrm{Path}(X, Z) = \{X \to Y \to Z\} \tag{8}$$

Given a graph G as in (2), and a source node s and target node t, a simple path is a sequence:

$$s = v_0, v_1, \ldots, v_k = t \tag{9}$$

such that:

- $(v_i, v_{i+1}) \in E$ for all $i \in [0, k-1]$,
- All v_i are distinct.

This approach enables multi-hop reasoning, where the answer is obtained by following meaningful chains of entities and relations, improving accuracy and semantic relevance.

3. *Subgraph Filtering*

Subgraph filtering refers to the post-processing step after retrieval, where the goal is to remove irrelevant nodes—or so-called *noise*—and edges from the subgraph. This step is usually done based on relation types, semantic types, or relevance score.

For example, even after subgraph retrieval, some nodes may remain off-topic, contain duplicate or redundant information, or offer minimal value to the query context. These nodes would overwhelm the overall structure, thus bring back the hallucination problem—similar to what occurs in naïve RAG system.

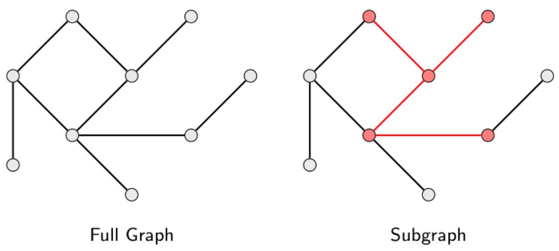


Fig. 7. Visualization of a subgraph within a larger knowledge graph. The full graph (left) contains various entities and relations. The subgraph highlighted in red (right) represents a reasoning path relevant to a specific query [15].

The final filtered subgraph $\hat{C} \subseteq G$, as in (5), is smaller, more coherent, and optimized for interpretability and answer generation.

D. Graph-based Contextualization

After retrieving a subgraph, the information is converted into natural language prompts. This contextual information is then combined with the original query and passed to a language model. There are two primary strategies:

1. *Text-based Serialization*

In this approach, the retrieved subgraph is verbalized—that is, each triplet is transformed into a natural language sentence. For instance, each triplet shown in Fig. 3, can be serialized into: "Laptop connects to Router. Router provides Internet."

This process is repeated for all triplets in the filtered subgraph $\hat{C}$. The resulting statements are then combined into a single text as a context.

2. *Logical Representation*

In settings where symbolic reasoning or exact logic is desired, the subgraph may instead be expressed in a structured knowledge representation language, such as Resource Description Framework (RDF) triples [17].

For example:

```
:Router :connects_to :Internet .
:Modem :provides :Internet .
```

After converting the subgraph into either natural language or a logical format, the next step is to combine the contextual information with user query. This combined input serves as the complete prompt for the generator module. Usually, this integration is done by prepending the contextual knowledge to the query, ensuring that the model processes the relevant information first and incorporates it into the generation process.

E. Answer Generation

The final stage of the GraphRAG pipeline is to generate an output a, conditioned on the structured query $\hat{q}$ and the context $\hat{C}$, as shown in (6). The generator usually takes the form of a LLM, such as GPT, BART, LLaMA, etc. These models receive the combined input—including both the retrieved contextual knowledge and user query—and generate a response based on this prompt.

VI. Conclusion

In this paper, we explored the use of knowledge graphs to improve Retrieval-Augmented Generation (RAG) systems, focusing on the GraphRAG framework. Unlike traditional RAG, which uses vector similarity to find unstructured document chunks, GraphRAG uses structured graph-based retrieval to capture more semantic relationships between entities. This allows it to perform multi-hop reasoning and deliver more accurate and connected answers.

We explained how a GraphRAG system works—from building the graph using entity and relation extraction, to retrieving subgraphs based on user queries, and finally generating answers with structured context.

Although building knowledge graphs can be challenging, especially in domain-specific cases, GraphRAG offers a strong foundation for making large language models more reliable, especially as graph construction becomes easier with new tools.

In conclusion, GraphRAG combines unstructured text search with structured reasoning, making use of the best of both worlds. Its ability to retrieve information in a more meaningful way makes it a strong foundation for future systems that rely on external knowledge.

Video Link at YouTube

https://youtu.be/VqhosB7FxQs

Acknowledgment

The author would like to thank Dr. Ir. Rinaldi Munir and Arrival Dwi Sentosa, for their guidance and support throughout the IF1220 Discrete Mathematics course. Appreciation is also extended to fellow students for their discussions and feedback during the preparation of this work.

References

[1] P. Wang, Y. Liu, Y. Lu, J. Hong, and Y. Wu, “What are Models Thinking about? Understanding Large Language Model Hallucinations ‘Psychology’ through Model Inner State Analysis,” *arXiv preprint arXiv:2502.13490*, 2025.

[2] P. Lewis et al., “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” *arXiv preprint arXiv:2005.11401*, 2021.

[3] Z. Xu, M. J. Cruz, M. Guevara, T. Wang, M. Deshpande, X. Wang, and Z. Li, “Retrieval-Augmented Generation with Knowledge Graphs for Customer Service Question Answering,” *arXiv preprint arXiv:2404.17723*, 2024.

[4] D. Edge et al., “From local to global: A Graph RAG approach to query-focused summarization,” *arXiv preprint arXiv:2404.16130*, 2024.

[5] S. Wu et al., “Retrieval-Augmented Generation for Natural Language Processing: A Survey,” *arXiv preprint arXiv:2407.13193*, 2024.

[6] L. Ehrlinger and W. Wöß, "Towards a definition of knowledge graphs," in Joint Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems (SEMANTiCS 2016) and the 1st International Workshop on Semantic Change & Evolving Semantics (SuCCESS16), Leipzig, Germany, Sept. 2016, vol. 1695.

[7] A. Singhal, “Introducing the Knowledge Graph: things, not strings,” Google Blog, 2012. [Online]. Available: https://blog.google/products/search/introducing-knowledge-graph-things-not/

[8] H. Han et al., "Retrieval-Augmented Generation with Graphs (GraphRAG)," *arXiv preprint arXiv:2501.00309*, 2025.

[9] H. Han et al., "RAG vs. GraphRAG: A Systematic Evaluation and Key Insights," *arXiv preprint arXiv:2502.11371,* 2025.

[10] https://github.com/Graph-RAG/GraphRAG

[11] Y. Tian, “Graph Neural Prompting with Large Language Models”, AAAI, vol. 38, no. 17, pp. 19080-19088, Mar. 2024.

[12] M. Yasunaga, H. Ren, A. Bosselut, P. Liang, and J. Leskovec, “QA-GNN: Reasoning with language models and knowledge graphs for question answering,” *arXiv preprint arXiv:2104.06378*, 2022.

[13] M. Yasunaga, A. Bosselut, H. Ren, X. Zhang, C. D. Manning, P. Liang, and J. Leskovec, “Deep bidirectional language-knowledge graph pretraining,” *arXiv preprint arXiv:2210.09338*, 2022.

[14] Ultipa, “K-Hop Traversal - Ultipa Documentation.” [Online]. Available: https://www.ultipa.com/docs/uql/k-hop. Accessed: Jun. 18, 2025.

[15] K. S. Yow, N. Liao, S. Luo, and R. Cheng, “Machine Learning for Subgraph Extraction: Methods, Applications and Challenges,” Proc. VLDB Endow., vol. 16, no. 12, pp. 3864–3867, Aug. 2023.

[16] IBM Technology, “GraphRAG vs. Traditional RAG: Higher Accuracy & Insight with LLM,” YouTube, Feb 17, 2025. [Online video]. Available: https://www.youtube.com/watch?v=Aw7iQjKAX2k

[17] RDF/JS, “Data Model Specification,” RDF/JS: JavaScript RDF Library Interfaces, 2022. [Online]. Available: https://rdf.js.org/data-model-spec/. Accessed: Jun. 18, 2025.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 20 Juni 2025

[illegible]

Nathaniel Christian – 13524122