Graph-Based Retrieval-Augmented Generation (GraphRAG) for Query-Focused Summarization: A Vertical Review and Replication Study

Veronica Di Gennaro, 11090623
Daniel Yezid Guarnizo Orjuela, 10836955

Politecnico di Milano

November 20, 2025

1 Introduction

GraphRAG is an innovative approach able to integrate the structured semantics of knowledge graphs (KGs), the retrieval dynamics of RAG systems and the generative power of LLMs, enabling the latter to give more accurate and context-aware responses free of noise and hallucinations. This project has two main objectives:

1. to conduct a literature review on GraphRAG, tracing its origins, current research directions and the respective challenges;
2. to replicate and adapt Microsoft Research's paper: *From Local to Global: A GraphRAG Approach to Query-Focused Summarization* [ETC+25] on a reduced knowledge corpus for practical evaluation.

2 Literature Review

2.1 Background: RAG Motivations and Limitations

Since their advent, Large Language Models (LLMs) have demonstrated remarkable capabilities in comprehending and producing human language across a wide variety of tasks and user queries [CZ24]. This versatility makes them highly effective for general-purpose applications.

However, their utility becomes limited when addressing knowledge-intensive or domain-specific tasks due to several key shortcomings. LLMs, in fact, are prone to hallucinations [HYM+25], producing information that, while syntactically plausible, is factually incorrect. Furthermore, they often lack interpretability, providing little insight into the logic reasoning behind their predictions [HS24]. Most importantly, their knowledge is static, constrained by a fixed training cut-off, which makes them unable to incorporate up-to-date or technical information and prevents access to private or internal data, which is often essential in professional contexts. For LLMs to be practically useful in the latter, they must not only access large volumes of information but also retrieve the correct and relevant information [LPP+21]. Achieving this through continuous retraining is impractical, as it is both time-consuming and resource-intensive, since updates can be minor but still frequent.

To solve these limitations, fine-tuning on domain-specialized datasets is out of play because, apart from being costly and slow, it risks issues such as 'catastrophic forgetting' [vdVSK24], where LLM tends to overfit to the small fine-tuning dataset, resulting in loss of performance in the other more generalist tasks; or generating new hallucinations [GYA+24]. Both phenomena intensify when the new data conflict with the pre-existing knowledge.

As an alternative, prompt-based approaches have gained attention, particularly those that provide more context to the model, helping it to reason on an enriched knowledge avoiding hallucination. Techniques like Chain-of-Thought (CoT) prompting [WWS+23] guide the model through structured reasoning, while Retrieval Augmented Generation (RAG) enhances the accuracy and relevance of the responses by integrating the generative power of LLMs with data retrieval from external knowledge sources.

RAG, first introduced in [LPP+21] in the version now known as Naïve RAG features the basic pipeline visible in Figure 1, when applied to a question answering task. The process consists of several key steps. A first, offline, step is indexing during which raw data, such as documents, are converted into plain text, chunked, embedded, and stored in a vector database. Then, when a user submits a query it is embedded in the same space as the chunks, allowing for the retrieval of the top-k similar chunks based on some vector similarity or keyword matching technique. The final stage involves generating a response, where the query and retrieved chunks are formatted into prompts and submitted to the LLM to produce an enriched and coherent response.

RAG is a cost-effective solution since it allows LLMs to bypass retraining and instead draw from external, up-to-date resources. This not only enables access to technical and domain-specific information, but it also improves transparency and trust in LLMs responses, since RAG allows model to draw from specific sources, enabling users to verify any claims made.

Moreover, RAG offers a broad potential since ideally anything can become a knowledge basis for retrieval including medical indexes for healthcare professionals, technical or policy manuals for businesses, marked data for financial analysts and so on. But, despite this potential, several critical limitations have emerged for Naïve RAG:

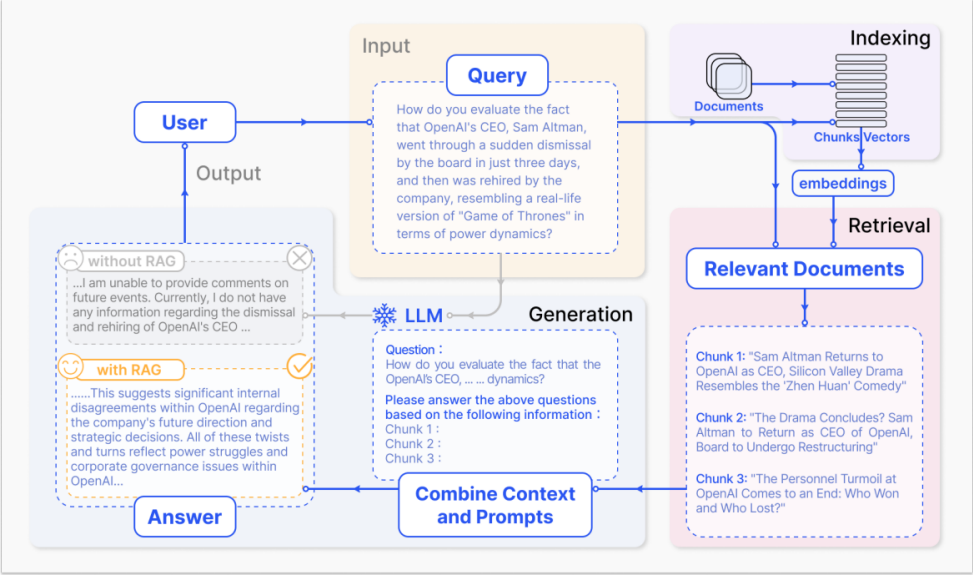

Figure 1: Naive RAG pipeline applied to a question answering task. Taken from [GXG+24].

- Complex query understanding, especially in professional contexts in which queries often involve multiple, not-explicit related concepts.
- Knowledge integration challenges especially when information is scattered across heterogeneous or distributed sources.
- System efficiency degrading as the size of the knowledge corpus grows, indicating struggles to synthesize comprehensive and meaningful information in coherent answers.

Additionally, RAG also presents some limitation inherent to LLMs, such as the inability to capture long-range dependencies in complex or large documents due to the limited context window or due to the "Lost in the Middle" problem [LLH+23].

2.2 Advancing Towards GraphRAG: Knowledge Organization

A key way to improve RAG is concentrating on the retrieval step. In fact, one weakness of Naïve RAG is their over-reliance on single-step retrieval which is insufficient for addressing complex queries that often require multi-hop reasoning and broader context integration. Some Advanced RAG methods attempt to mitigate this through both *pre-retrieval enhancements*, such as query rewriting or better indexing strategies and *post-retrieval enhancements* like chunk re-ranking and context compression. Among these, we highlight **hierarchical RAG** that aims to preserve document structures through multi-level retrieval. This approach shows a key insight: traditional RAG methods face challenges with knowledge-intensive reasoning tasks because the essential information needed for these tasks is badly distributed, making it hard to identify key information and perform global reasoning accurately [LCY+24]. As shown in prior studies, the ability to structure knowledge, through methods such as hierarchical indexing, which organizes data into parent-child relationships—significantly improves retrieval step efficiency and reasoning effectiveness [ZCB+25].

In this context, we talk about **Knowledge Organization** in RAG referring to the processes and techniques for structuring and preparing external knowledge to optimize RAG retrieval process.

2.2.1 Knowledge Graphs (KGs)

Knowledge Graphs (KGs) represent a powerful form of knowledge organization. They structure information as graphs made of nodes (entities) and edges (relationships among entities). In the context of RAG systems, they started to be used because they provide a concise but semantically rich way of

representing enormous knowledge, free of noisy and irrelevant information, allowing to enhance knowledge awareness in LLMs. Even outside the context of RAG, KGs had already been largely employed to improve the accuracy, reasoning capability, and transparency of LLMs in prior works. In the following paragraph we outline a taxonomy of KG-enhanced LLMs, drawing from prior work [PLW+24] which identifies some primary paradigms to enhance LLMs with KGs.

First, *KG-enhanced pre-training* aims to inject structured knowledge during the model training phase. This includes modifying training objectives to prioritize knowledge-rich entities by aligning textual tokens with graph entities and applying contrastive learning to reinforce factual consistency. A whole different strategy, where KGs are leveraged without altering model weights or needing retraining, involves feeding KGs (as knowledge triples made of subject entity, relationship, object entity) directly *into LLMs inputs* [SAJ23], [BAS23], enabling better access to domain-specific information. However, because LLMs are not explicitly trained to interpret graph structures, they may struggle to leverage the crucial information retained into subgraph structures or multi-hop reasoning paths embedded in KGs, underutilizing this powerful structure. To address this limitation, *KGs instruction-tuning* focuses on fine-tuning LLMs to comprehend and reason over KG structures using prompt-based learning, enabling models to generate logical queries or reasoning paths [LET+24], [LLHP24]. Finally, *KG-enhanced interpretability* utilizes KGs to improve LLM interpretability and trustworthiness. This category contains techniques to probe the factual knowledge encoded within models [LVPH23], to explore how such knowledge is represented at the neuron level [SRJ21] and to ground model reasoning in KG structures to trace and explain answer generation steps [YRB+21].

2.2.2 Structured Knowledge in RAG

Going back to RAG, *why can structured knowledge be useful?* Because external knowledge sources often consist of extensive text corpora, heterogeneous document and data collections and traditional RAG systems, which rely on flat text retrieval can be easily prone to:

- **Information overload** due to the inclusion of numerous irrelevant or redundant information;
- **Knowledge conflicts** arising from inconsistencies among *independently retrieved* text chunks;
- **Missing context** because of a limited possibility of reasoning across document.

All of these leading to a **compromised comprehension**.

In contrast, structured knowledge in the form of knowledge graphs (KGs), by explicitly modeling entity relationships and semantic hierarchies, enables context-preserving retrieval and multi-hop reasoning. This allows systems to populate the limited context window of LLMs with more relevant and interconnected information. In this set context we locate GraphRAG.

2.3 GraphRAG

GraphRAG represents a novel class of Retrieval-Augmented Generation systems that leverage knowledge graphs to improve retrieval precision and reasoning depth and to effectively overcome naive RAG's limitations by enabling broader semantic integration across documents. GraphRAG follows a workflow similar to the one of traditional RAG composed of Knowledge Organization, Knowledge Retrieval and Knowledge Integration.

In the context of **Knowledge Organization**, Graphs in GraphRAG can serve two purposes, as knowledge carriers or as retrieval indexes. We can, in fact, distinguish between two GraphRAG paradigms Index-Based GraphRAG and Knowledge-Based GraphRAG, discussed in Section 2.3.1.

Knowledge Retrieval, defined as the process of gathering relevant nodes, edges, or subgraphs in alignment with the user's query, now, should support semantic traversal, community detection, and relation-aware filtering. Retrieval can, in fact, be performed over entire graphs or focused subgraphs, which brings new challenges for preserving the structural richness of the graph while integrating results into a coherent, text-based prompt.

Knowledge Integration is about synthesizing the retrieved graph data into a textual prompt suitable for LLM input with the goal preserving the semantic richness of subgraphs (due to their relations) without introducing redundancy or misplacing emphasis on less critical aspects.

The two primary strategies used for integration are:

1. *Node-Level Knowledge Integration.* This strategy, typical in index-based GraphRAG, where each node links to a document chunk, consists in feeding node and neighboring nodes (chunks) as contextual information to the LLMs. Sometimes, embeddings are used to compress retrieved content to fit within LLM context limits since it has been demonstrated that LLMs are capable to capture information from these embeddings [ZBY+24].

2. *Subgraph-Level Knowledge Integration.* Subgraphs are composed of multiple nodes and edges, capturing rich semantic interconnections. Their integration requires methods for identifying meaningful subgraphs, summarizing their internal relationships, and constructing prompts that retain relational integrity. One example is the community grouping and summarization of [ETC+25]. In this way GraphRAG provides a more scalable and semantically grounded approach to RAG.

2.3.1 Index-Based vs. Knowledge-Based GraphRAG

Within the GraphRAG approach, two distinct paradigms have emerged according to the differing uses of knowledge graphs.

Index-Based GraphRAG

In Index-Based GraphRAG, knowledge graphs are primarly used as structural indexes for raw textual data. In this case each text chunk extracted from a documents is treated as a node within the graph. Then, relationships between chunks, such as chunks in overlap, chunks referring to same topic and chunks talking about related topics, are encoded as edges, forming a graph over the entire corpus. An important feature is that the raw text associated with each node is preserved and ultimately fed, as it is, into the language model during generation. This structure also supports knowledge hierarchy construction, enabling hierarchical indexing and facilitating both bottom-up summarization and top-down localization of relevant information to easily and rapidly localize relevant knowledge in a context-aware manner. This is particularly beneficial for large-scale corpora where semantic proximity among documents enhances retrieval efficiency.

Knowledge-Based GraphRAG

In contrast, Knowledge-Based GraphRAG treats the graph itself as a ***carrier of knowledge***. Here, we do not preserve the raw documents but instead we focus on building a semantically rich, structured representation of their content. Textual content is, in fact, pre-processed to extract and summarize factual information, typically using LLMs or traditional information extraction methods. The extracted knowledge is then condensed and fused into nodes and edges, to retain relations among entities, forming a unified, interpretable knowledge base. This condensed form enhances reasoning efficiency and interpretability by reducing noise and redundancy and allowing to reason over semantic graphs.

Within the Knowledge-Based GraphRAG paradigm, two primary lines of research can be identified:

1. **Utilizing Pre-Existing Knowledge Graphs** In general, this involves using large-scale, general-purpose knowledge graphs, such as ConceptNet or Wikidata. For this reason the main challenge here is to efficiently navigate and dynamically retrieve all the relevant information in response to user queries. A key technical and research-interesting problem is the identification of relevant subgraphs or reasoning paths within a large graph.

   Some approaches treat this as a graph planning problem and address it via techniques such as reinforcement learning [ZDC+24]. Other approaches, such as [ETC+25] that presents a Graph RAG approach to global question answering in the query-focused summarization (QFS) task, try to build systems able to perform hierarchical community detection and summarization to support long-range reasoning across multiple documents. Other systems, such as in [HTS+24], aim to combine both graph traversal and semantic retrieval . There are also some approaches that integrate Graph Neural Networks (GNNs) and LLMs, to better work with the structured knowledge retained by graphs to generate semantically informed responses [HLZ+24].

2. **Generating KGs from Document Corpora** The second line of research identified focuses on building *task-specific* knowledge graphs directly from a given corpus. This can be accomplished using both **Traditional relation extraction methods** such as rule-based, statistical,

or clustering techniques developed since the early 2000s, and **Modern LLM-based techniques**, which offer greater flexibility and adaptability across application areas. Reason why the latter is becoming increasingly prevalent, apart from the fact that it allows to construct knowledge structures from unstructured text with minimal manual work.

2.3.2 Tasks and Applications

Numerous research studies have demonstrated the versatility of GraphRAG systems across various domains, showcasing many real-world applications. As expected, these systems excel in numerous Natural Language Processing (NLP) tasks, where the integration of structured and unstructured data, advanced reasoning, and contextual understanding are fundamental aspects. The following list of tasks aims to highlight GraphRAG systems' ability to address a wide range of needs using diverse datasets while adapting to the specific requirements of various fields:

- **Question answering (QA).** This can be divided into two categories: **Simple QA** and **Large-scale complex QA**. Simple QA requires only basic evidence retrieval and does not involve complex reasoning chains. Datasets for such tasks can be useful for assessing and benchmarking retrieval accuracy. Large-scale complex QA, instead, involves broader contextual understanding. In this case, models are provided with extensive knowledge sources and asked high-level, multifaceted questions that require the synthesis of information from different domains, testing the model's ability to understand queries and to draw insights that may not be explicitly stated in any single document. An application is academic literature review systems that summarize a body of research. The system can assist in answering complex queries by linking concepts across multiple documents. In scientific research, GraphRAG systems can help create knowledge graphs that organize research papers to highlight relationships between key findings, methodologies, and datasets, offering new insights into research trends. Systems specialized for this task can be used as customer support assistants and in other real-world scenarios requiring concise but valuable (and well-supported) answers.

- **Multi-hop Reasoning.** This task requires the model to respond to questions that involve multi-step reasoning chains or comparisons across different sources. Graph-based approaches can trace evidence paths, enabling reasoning over distant yet related facts. It is incorporated by other tasks, such as the already discussed Large-scale complex QA.

- **Domain specific QA.** This includes fact-checking or knowledge validation subtasks, focusing on the model's capacity to answer questions based on available evidence. Datasets for such tasks can be used to assess query understanding and factual accuracy. An application is tutoring in the educational domain [YYF+25], or supporting decision-making in areas such as legislation and healthcare. In fact, because of the ability of knowledge graphs to integrate various knowledge sources (structured databases, semi-structured data, unstructured text, etc.), they are well suited to handle the intricate and high-stakes nature of medical and legal information.

3 Replication of the paper: *From Local to Global: A GraphRAG Approach to Query-Focused Summarization*

The paper we decided to replicate is "*From Local to Global: A GraphRAG Approach to Query-Focused Summarization*" [ETC+25] because it puts together many of the innovative techniques that we discussed in our literature review. These techniques, such as using a structured knowledge in the form of Knowledge Graphs constructed from a document corpus using an LLMs, retrieving knowledge from a structural database instead of a vector database, performing a bottom-up, coarse-to-fine summarization of a hierarchical knowledge and more, allow GraphRAG to substantially outperform traditional RAG approaches on the Query-Focused Summarization (QFS) task, targeted in the paper, and it involves synthesizing relevant information across a *broad* but *interconnected* corpus in response to a specific user query. So the QFS task requires both *sensemaking*, as the global understanding on the entire dataset, and reasoning over connections. The one proposed in the paper is a graph-based approach to question answering that *"scales with both the generality of user questions and the quantity of the source"* [ETC+25], and the basic pipeline implemented is the following:

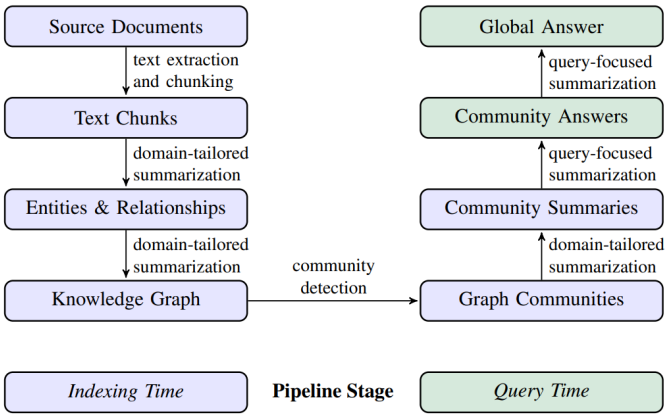

Figure 2: Schematic overview of GraphRAG pipeline implemented by [ETC+25].

- **Knowledge Graph Construction**: The input corpus is divided into text chunks, and an LLM is used to extract entities (nodes) and relationships (edges), forming a knowledge graph.
- **Community Detection**: The graph is then partitioned into a hierarchy of communities, clustering closely related entities.
- **Hierarchical Summarization**: Community-level summaries are generated using a bottom-up approach. Summaries at higher levels recursively incorporate those from lower levels, reflecting a hierarchical structure.
- **Answer Generation**: A map-reduce style process is used, where each community summary independently generates a partial response to the user query. These partial answers are then aggregated into a final, globally coherent answer.

A schematic overview of this pipeline is shown in Figure 2

3.1 Implementation Details

To replicate and adapt the GraphRAG framework proposed by Microsoft Research, we implemented a modular RAG system capable of answering user queries over a corpus of scientific papers retrieved directly from ArXiv. We implemented both vector-based and graph-based retrieval pipelines, enabling comparative analysis of their effectiveness on QFS task. The system is implemented using Python and leverages some open-source tools including LangChain, Neo4j as the graph database, Milvus as the Vector database, foor vector-based retrieval, and OpenAI's o4-mini model for various generation tasks.

The choice of the o4-mini model was based on a trade-off analysis between cost, performance, and token limits, as documented in OpenAI's model comparison page.

In the following paragraphs, we describe each step of the implemented pipeline highlighting some differences from the original implementation in [ETC+25]:

3.1.1 Documents Retrieval and Chunking

In this step we encounter the first and one of the most significant deviations from the original GraphRAG implementation: the different source used to build the knowledge base. The original paper uses two datasets, one composed of public transcripts of a podcast, and the other consisting of news article across a range of categories [TY24], both in the one million token range. Instead, we decided to use as our knowledge corpus scientific documents extracted from ArXiv.

This choice was driven by computational and more critically budgetary constraints. Constructing a large-scale knowledge graph from a broad corpus of documents each composed of a large number of tokens, would have required substantial costs, particularly during the graph construction through LLM calls. Since the high costs stem from the high number of tokens we knew we had to limit the number of tokens. But rather than drastically limiting the number of documents, which would nullify the purpose of evaluating GraphRAG's ability to generalize and summarize across a diverse corpus of

documents, we opted to work with a large enough set of documents retrieved from ArXiv, but using their summaries instead of full texts, allowing us to gather more documents, to have a graph large enough to test our pipeline, keeping the number of tokens low. This could be done through the arXiv API by issuing a keyword query. The keywords we used are "agent", "large language model" and "prompt engineering". The final dataset consists of approximately 1,000 tokens across 20 document summaries. Each summary was processed into chunks of 600 tokens, consistent with the original paper's chunking strategy designed to mitigate the "Lost in the Middle" problem. However, since the summaries were generally short, no actual chunk splitting was required. These processed chunks were then stored in a Milvus vector database, enabling efficient vector-based retrieval, as discussed later on in Section 3.1.5.

3.1.2 Entity Extraction and Knowledge Graph Construction

On the document chunks, we use OpenAI's o4-mini model via LangChain's LLMGraphTransformer to extract structured knowledge in the form of entities and relationships. Each chunk is processed independently to avoid the "lost in the middle" issue, and the resulting outputs are serialized and stored as GraphDocument objects using the Pickle format.

The storing part is crucial to avoid redundant reprocessing, since the generation of Graph Documents costs money due to the LLM calls involved.

A separate process subsequently handles the loading of these Graph Documents into a Neo4j knowledge graph, where entities become nodes, and relationships between them are encoded as directed edges. This results in a graph-based representation of the source documents.

3.1.3 Graph Communities Detection and Hierarchical Structuring

The goal of this step is to identify and organize semantically coherent regions within the knowledge graph in hierarchical communities.

Our approach uses the Louvain algorithm, provided by the Neo4j Graph Data Science library, recursively to partition the entity-level knowledge graph into nested communities (from C3 up to C0), forming a hierarchy. Each level corresponding to a progressively coarser semantic grouping of entities.

The process starts with C3-level communities that are detected directly over the *Base Graph*, which is projected over entity nodes and their related relationships.

To ensure quality and prevent the formation of trivial or noisy clusters, a pruning step is introduced that removes low-information C3 communities.

Subsequent, higher-level communities (C2 to C0) are constructed in the following way: at each level, nodes represent communities from the previous level, and edges are formed by aggregating inter-community connections weighted by the number of relationships between their constituent nodes.

An important detail is that each new layer is built only when meaningful/sufficient inter-community relationships are detected. Reason why, in our specific case, due to the smaller scale of the dataset, the process stopped at the generation of C2 level communities. However, our code is designed to adapt to deeper hierarchies when applied to larger graphs.

A view of the generated Knowledge Graph and the identified C2 and C3 level Community grahs is given in Figure 3

3.1.4 From Graph Communities to Community Summaries

The previous step resulted in a nested hierarchy of semantically cohesive communities that serves as the backbone for the multi-tiered community summarization of this step.

At leaf level (C3), each community is individually summarized by prompting an LLM with its corresponding prioritized subgraph (in decreasing order of the overall prominence of the edges given by the combined degrees of the source and the target node), consisting of entities, relationships, and claims. The prioritization aims to ensure the most influential elements are emphasized and put at the beginning of the textual summary.

The LLM generates structured outputs in JSON format, which include a descriptive community title, an executive summary, an impact severity rating (on a scale from 0 to 10), a set of detailed findings, each accompanied by grounded evidence references.

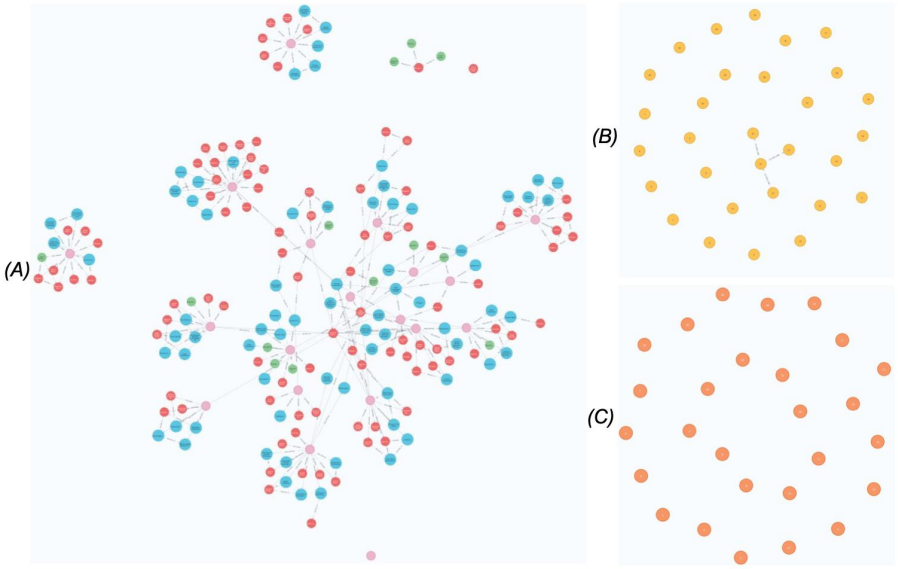

Figure 3: A) generated Knowledge Graph. B) C3 level community graph. C) C2 level community graph. In the C2 level community graph it is noticeable the absence of edges between nodes, reason why the hierarchical process of community generation stopped at this level.

Communities that fail to produce sufficiently informative summaries are filtered out at this stage to prevent noise propagation.

For higher levels, summaries are produced recursively by aggregating and synthesizing the summaries of child communities into a coherent higher-level summaries. To manage input size and relevance, sub-summaries are ranked by their token count, and the most substantial ones are prioritized for inclusion.

This bottom-up summarization approach enables the system to produce increasingly abstract yet semantically rich summaries at higher levels, supporting efficient and scalable global understanding.

3.1.5 Answer Generation: Vector RAG Vs Graph RAG Pipelines

At this stage, given a user query, we distinguish between two distinct retrieval and answer generation pipelines:

- **Vector RAG Pipeline**: The Milvus vector database, in which we stored the plain-text chunked documents enables document chunk indexing and semantic search to perform what we can call "*flat*" retrieval. Upon receiving a user question, the retriever selects the top-K most relevant chunks based on vector similarity. These chunks are then passed to an LLM (in our case, once again OpenAI's o4-mini model), to generate a concise, informative answer based on the retrieved context apart from the pre-existing knowledge.

- **GraphRAG Pipeline**: This pipeline leverages the community-based knowledge summaries constructed in previous step. The process has two main phases:

  - Map phase: For a given user query, the system processes each community independently using the LLM to generate a partial response together with a numerical helpfulness score, which quantifies how much the answer addresses the query. These scores are extracted and used to rank and filter these partial responses, retaining only the most informative ones.
  - Reduce phase: The retained partial answers are concatenated and fed into another LLM prompt that synthesizes them into a final, comprehensive response. This final output should include source references to maintain transparency and enable verification.

  This map-reduce-style summarization is what allows the GraphRAG pipeline to scale effectively with corpus size and question complexity or generality. It supports globally informed reasoning,

while preserving local evidence grounding.

3.2 Evaluation and Comparison

Disclaimer on Comparability

Because of the limitations we encountered and the consequential adaptations made to our knowledge base (as detailed in Section 3.1.1), we were unable to directly replicate the evaluation results from the original paper. Since we used a different knowledge base, in fact, the original question sets could not be reused. However, our setup still supports a comparative evaluation between the Vector RAG and Graph RAG pipelines using the same methodology (LLM-as-a-judge) and the same qualitative criteria as in the original paper.

3.2.1 Question Generation

Since we had to regenerate the questions to submit to our pipelines as queries, we employed an LLM-based persona-task-question method to generate realistic, corpus-tailored global sensemaking queries. This multi-step procedure involves the **persona generation** [SHR+24] in which an LLM generates distinct, realistic, potential user personas (e.g., `NLP PhD Student`, `Machine Learning Engineer`) based on a textual description of the corpus, the **task creation**, where for each persona, the LLM suggests concrete information-seeking tasks (e.g., `Survey recent advances in LLM interpretability and alignment`) and finally the **global question generation** in which for each task, high-level questions are generated that require synthesis of broad themes across documents, rather than fact retrieval from a single source.

Importantly, this strategy avoids generating questions directly from the corpus, reducing the risk of unfair tuning or model bias in evaluation.

3.2.2 LLM evaluator

Following the original paper's approach, we used an LLM evaluator [WLM+23] to compare responses from both the Vector RAG and Graph RAG systems to the same set of questions. Evaluations were conducted blindly, with the LLM unaware of which system generated which response.

Each answer pair was judged according to the following four evaluation criteria:

- Comprehensiveness: How fully does the answer address the question, covering all aspects and relevant details (without unnecessary redundancy)?
- Diversity: Does the answer incorporate multiple perspectives, dimensions, or evidence sources?
- Empowerment: How well does the answer support understanding and informed reasoning, while avoiding misleading or fallacious conclusions?
- Directness: How concisely and clearly does the answer address the core of the question?

The first three criteria are considered positive indicators of global *sensemaking* quality, while Directness often contrasts with Comprehensiveness, potentially favoring more focused, snippet-style responses like those from Vector RAG. We therefore expect Vector RAG to outperform Graph RAG on this last dimension in some cases.

For each criterion, the LLM returns a winner (it can also indicate a tie) and a short natural language justification explaining its decision.

3.3 Results and Discussion

We compared the performance of **Graph RAG** and **Vector RAG** across different types of queries and we could notice that each method has its own advantages depending on the structure and nature of the information required.

Graph RAG Responses varying across Community levels

One key point of the hierarchical structure of community summaries is that questions can be answered using different community levels trying to find the best balance of summary detail and scope for general sense making questions, enabling the system to adapt to diverse query types and context window limitations during global question answering. As shown in Table 3 and its description in the Appendix, the Graph RAG responses vary according to the hierarchical level of the community used to provide the partial answers to integrate in the global one.

Why could GraphRAG Outperform Vector RAG?

In general we can notice how Graph RAG performs better when a query requires reasoning over structured relationships between concepts. In Table 1, we can see that Graph RAG excels because it can follow explicit links in the knowledge graph to produce a well-organized, comparative, and evidence-based answer. In fact, the graph structure, apart from supporting the synthesis of complex information across multiple documents, allows the model to include in the answer references to data reports for each claim. These are behaviors that Vector RAG cannot replicate, as its retrieval is based only on semantic similarity and lacks awareness of how entities are related.

When could Vector RAG be more effective?

In Table 2, we can see a case in which Vector RAG wins over GraphRAG. The reason behind this is that since Vector RAG retrieves content based on semantic similarity rather than structure, it is able to gather and synthesize information from diverse sources even when the terminology differs. This leads to more complete and informative answers, while Graph RAG struggles due to the absence of relevant nodes and edges. Therefore, when information is scattered or the domain lacks a clear structure, Vector RAG is more robust and flexible.

References

[BAS23] Jinheon Baek, Alham Fikri Aji, and Amir Saffari. Knowledge-augmented language model prompting for zero-shot knowledge graph question answering, 2023.

[CZ24] Rong-Ching Chang and Jiawei Zhang. Communitykg-rag: Leveraging community structures in knowledge graphs for advanced retrieval-augmented generation in fact-checking, 2024.

[ETC+25] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization, 2025.

[GXG+24] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey, 2024.

[GYA+24] Zorik Gekhman, Gal Yona, Roee Aharoni, Matan Eyal, Amir Feder, Roi Reichart, and Jonathan Herzig. Does fine-tuning llms on new knowledge encourage hallucinations?, 2024.

[HLZ+24] Yuntong Hu, Zhihan Lei, Zheng Zhang, Bo Pan, Chen Ling, and Liang Zhao. Grag: Graph retrieval-augmented generation, 2024.

[HS24] Yue Huang and Lichao Sun. Fakegpt: Fake news generation, explanation and detection of large language models, 2024.

[HTS+24] Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V. Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering, 2024.

[HYM+25] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55, January 2025.

[LCY+24] Zhuoqun Li, Xuanang Chen, Haiyang Yu, Hongyu Lin, Yaojie Lu, Qiaoyu Tang, Fei Huang, Xianpei Han, Le Sun, and Yongbin Li. Structrag: Boosting knowledge intensive reasoning of llms via inference-time hybrid information structurization, 2024.

[LET+24] Haoran Luo, Haihong E, Zichen Tang, Shiyao Peng, Yikai Guo, Wentai Zhang, Chenghao Ma, Guanting Dong, Meina Song, Wei Lin, Yifan Zhu, and Anh Tuan Luu. Chatkbqa: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models. In *Findings of the Association for Computational Linguistics ACL 2024*, page 2039–2056. Association for Computational Linguistics, 2024.

[LLH+23] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023.

[LLHP24] Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. Reasoning on graphs: Faithful and interpretable large language model reasoning, 2024.

[LPP+21] 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. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

[LVPH23] Linhao Luo, Trang Vu, Dinh Phung, and Reza Haf. Systematic assessment of factual knowledge in large language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 13272–13286, Singapore, dec 2023. Association for Computational Linguistics.

[PLW+24] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):3580–3599, July 2024.

[SAJ23] Juan Sequeda, Dean Allemang, and Bryon Jacob. A benchmark to understand the role of knowledge graphs on large language model's accuracy for question answering on enterprise sql databases, 2023.

[SHR+24] Joongi Shin, Michael A. Hedderich, Bartlomiej Rey, Andrés Lucero, and Antti Oulasvirta. Understanding human–ai workflows for generating personas. In *Proceedings of the 2024 ACM Designing Interactive Systems Conference, DIS 2024*, Proceedings of the 2024 ACM Designing Interactive Systems Conference, DIS 2024, pages 757–781. ACM, jul 2024.

[SRJ21] Vinitra Swamy, Angelika Romanou, and Martin Jaggi. Interpreting language models through knowledge graph extraction. *arXiv (Cornell University)*, January 2021.

[TY24] Yixuan Tang and Yi Yang. Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries, 2024.

[vdVSK24] Gido M. van de Ven, Nicholas Soures, and Dhireesha Kudithipudi. Continual learning and catastrophic forgetting, 2024.

[WLM+23] Jiaan Wang, Yunlong Liang, Fandong Meng, Zengkui Sun, Haoxiang Shi, Zhixu Li, Jinan Xu, Jianfeng Qu, and Jie Zhou. Is chatgpt a good nlg evaluator? a preliminary study, 2023.

[WWS+23] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[YRB+21] M. Yasunaga, H. Ren, A. Bosselut, P. Liang, and J. Leskovec. Qagnn: Reasoning with language models and knowledge graphs for question answering. *NAACL*, pages 535–546, 2021.

[YYF+25] Rui Yang, Boming Yang, Aosong Feng, Sixun Ouyang, Moritz Blum, Tianwei She, Yuang Jiang, Freddy Lecue, Jinghui Lu, and Irene Li. Graphusion: A rag framework for knowledge graph construction with a global perspective, 2025.

[ZBY+24] Yang Zhang, Keqin Bao, Ming Yan, Wenjie Wang, Fuli Feng, and Xiangnan He. Text-like encoding of collaborative information in large language models for recommendation, 2024.

[ZCB+25] Qinggang Zhang, Shengyuan Chen, Yuanchen Bei, Zheng Yuan, Huachi Zhou, Zijin Hong, Junnan Dong, Hao Chen, Yi Chang, and Xiao Huang. A survey of graph retrieval-augmented generation for customized large language models, 2025.

[ZDC+24] Qinggang Zhang, Junnan Dong, Hao Chen, Daochen Zha, Zailiang Yu, and Xiao Huang. Knowgpt: Knowledge graph based prompting for large language models, 2024.

A Tables

Table 1: **Comparison between GraphRAG (using C2 Community Summary Level) and Vector RAG.** The question asks to compares multi-agent coordination architectures using LLMs, which involves understanding the differences between models, their usage, and evaluations. GraphRAG supports the synthesis of complex information across multiple documents, which Vector RAG cannot replicate, as its retrieval is based only on semantic similarity and lacks awareness of how entities are related.

Question	How do the papers in the corpus compare various coordination architectures for multi-agent systems built with LLMs?
Vector RAG	**Comparison of Coordination Architectures**: Discusses different architectures for coordinating multi-agent systems built with LLMs, emphasizing the interplay of *prompt engineering* and *multi-agent discussion mechanisms*. **Key Strategies**: • **Multi-Agent Discussion**: Uses symmetry of agents for complex reasoning. • **Scalable Discussion Mechanisms**: Includes conquer-and-merge strategies. • **Framework for Collaboration**: LLM agents embody expert personas for cooperative problem solving. **Applications and Effectiveness**: Applied to simulations (negotiation, games) and complex engineering projects; prompt engineering aids performance.

Graph RAG	**1. Overview**: Some papers compare coordination frameworks at an architectural level rather than empirically. [Data: Reports (188)] **2. Coordination Paradigms**: • Centralized "Hub-and-Spoke" orchestration [Data: Reports (7)] • Decentralized peer-to-peer negotiation [Data: Reports (14, 70)] • Asynchronous QA-driven assembly [Data: Reports (78)] • Symmetric role-based discussion [Data: Reports (117)] • Delegation architectures [Data: Reports (71)] **3. Case Studies & Empirical Insights**: • *Cochain framework* outperforms baselines in accuracy and collaboration. • *LLM-empowered siting*: Multi-agent setup yields more robust designs than single-agent. **4. Axes of Comparison**: • Centralization vs. Autonomy • Communication Overhead • Conflict Resolution • Adaptability vs. Coherence **5. Key Findings & Gaps**: • Most comparisons remain qualitative. • No unified benchmark. • Calls for standard metrics and datasets.
LLM Decision	**Comprehensiveness**: **Winner = Graph RAG**. Provides taxonomy, paradigms, empirical comparisons. **Diversity**: **Winner = Graph RAG**. Multiple coordination styles beyond prompt-based. **Empowerment**: **Winner = Graph RAG**. Cites data, details trade-offs, offers empirical case studies. **Directness**: **Winner = Graph RAG**. Enumerates architectures clearly and precisely.

Table 2: **Comparison between GraphRAG and Vector RAG.** In this example we see how the Vector RAG outperforms the Graph RAG using C2 Community Summary Level in almost all criteria. (Note that in the C3 this doesn't happen.) The query asks about interpretability tools, which are mentioned inconsistently across papers and not well-represented in the graph. Because Vector RAG retrieves content based on semantic similarity rather than structure, it is able to gather and synthesize information from diverse sources even when the terminology differs. This leads to a more complete answer, while Graph RAG struggles due to the absence of relevant nodes and edges.

Question	How do interpretability tools and methods differ in evaluating LLM-generated outputs across different research works?
Vector RAG	**Interpretability Across Studies**: Highlights variation in focus and efficacy of interpretability tools depending on the research objective. **Key Dimensions**: • **Prompt Engineering**: Techniques like Chain-of-Thought, Reflection, and MODP are tailored for reasoning, hallucination reduction, or secure code generation. • **Evaluation Strategies**: Benchmarks and metrics vary (e.g., hallucination rate, vulnerability detection), often with different datasets and task-specific evaluations. • **Application Specificity**: Simpler prompting strategies may outperform advanced ones depending on task (e.g., summarization vs. vulnerability analysis).
Graph RAG	**1. Overall Observation**: No mainstream interpretability tools (e.g., saliency maps, attention visualization) are reported across the corpus. [Data: Reports (4, 62)] **2. Narrow Use Cases**: • **Community 4**: Security-focused iterative prompting reveals model behavior under stress, acting as lightweight interpretability. • **Community 62**: Static vulnerability scanners and mistake taxonomies flag generated flaws as a post-hoc interpretability signal. **3. Limitations**: • Lack of probing classifiers, attention-based explanations, or internal model feature analysis. • No cross-comparison of interpretability techniques due to their absence in most reports.
LLM Decision	**Comprehensiveness**: **Winner = Vector RAG**. Covers more dimensions (prompting, metrics, context sensitivity) across studies. **Diversity**: **Winner = Vector RAG**. Encompasses multiple strategies and application contexts, not limited to security. **Empowerment**: **Winner = Graph RAG**. Provides grounded, data-backed findings from specific report clusters, avoiding overgeneralization. **Directness**: **Winner = Vector RAG**. More directly answers the core question with structured comparisons.

Table 3: **Comparison between GraphRAG responses when using `C3` or `C2` Community Summary Level.** Community level `C3` is a leaf-level subgraph composed of directly related nodes and edges. Its summaries includes detailed descriptions of prominent elements (specific nodes, edges, and claims), leading to structured, topic-specific answers grounded in fine-grained evidence. In contrast, community level `C2` represents a higher-level aggregation of multiple sub-communities. Summaries at this level are built from sub-community summaries rather than full element details, resulting in broader, more abstract content. While this makes `C2`'s response more general and inclusive of multiple perspectives, it sacrifices a bit the specificity and structural clarity seen in `C3`.

Question	Which prompt engineering techniques are shown to most effectively improve multi-step reasoning in LLMs across the corpus?
C3	**Prompt Engineering Techniques for Enhanced Multi-Step Reasoning** Based on the provided corpus, several prompt-engineering methods have been shown to yield significant improvements in multi-step (chain-of-thought) reasoning with large language models (LLMs). While direct head-to-head comparisons remain limited, the following techniques emerge as the most impactful: 1. **Formal Prompt Engineering Framework** A structured approach that decomposes complex tasks into ordered sub-steps using explicit templates. - Consistently improves accuracy on multi-hop reasoning benchmarks by guiding the model through well-defined intermediate stages. - Supported by empirical evaluations in the core framework study [Data: Reports (30)]. 2. **Multi-Objective Directional Prompting** Extends the formal framework with step-wise directional cues and multiple concurrent objectives. - Further boosts performance on complex, multi-stage reasoning tasks. - Validated against baseline prompts in LLMs [Data: Reports (30, 7)]. 3. **Dynamic Prompt Assembly** Dynamically selects, orders, and adapts prompt components at each reasoning iteration. - Enhances coherence and consistency across long reasoning chains by tailoring context in real time. - Demonstrated gains in diverse reasoning workflows [Data: Reports (30)]. 4. **Model-Agnostic Prompt Design** Crafts prompts whose structure and instructional style transfer across different LLM architectures. - Achieves robust multi-step reasoning improvements regardless of model size or family. - Confirmed through cross-model evaluations [Data: Reports (30, 8)]. 5. **Chain-of-Thought Prompting** Guides the model to articulate explicit intermediate reasoning steps in its output. - Shown to uncover deeper inference pathways and improve final answer correctness. - Documented gains on standard reasoning benchmarks [Data: Reports (70, +more)]. 6. **Reflection Prompting** Encourages the model to self-evaluate and iteratively refine its reasoning before finalizing an answer. - May reduce errors by prompting corrective feedback loops within a single interaction. - Early evidence suggests complementary benefits to chain-of-thought prompting [Data: Reports (70, +more)]. **Comparative Insights & Recommendations** - While all above techniques have demonstrated effectiveness, direct comparative metrics (e.g., % accuracy gains) across methods are not fully reported in the current corpus. - Practitioners may combine structured decomposition (Formal Framework) with iterative refinement (Reflection Prompting) to leverage both guided and self-corrective reasoning. - Further ablation studies and head-to-head benchmarks are needed to quantify which approach yields the largest absolute gains under consistent evaluation protocols. - Future work should explore hybrid designs (e.g., integrating Multi-Objective Directional cues into Dynamic Prompt Assembly) to capitalize on complementary strengths. *This synthesis consolidates the key prompt-engineering strategies supported by the available data. For detailed implementation patterns and quantitative results, consulting the full texts of the cited reports is recommended.*

C2

Key Prompt-Engineering Techniques for Multi-Step Reasoning

Below is a concise summary of the top methods shown to most effectively bolster multi-step reasoning in LLMs across the provided corpus. Each technique is backed by empirical gains on standard benchmarks.

1. **Chain-of-Thought Prompting** Asking the model to "think aloud" by generating intermediate reasoning steps will encourage more systematic problem solving and reduce error propagation. Extensively validated on math and commonsense tasks, CoT prompting has yielded consistent performance gains. [Data: Reports (14, 78, 117, 7, +more)]

2. **Few-Shot / In-Context Learning** Embedding a handful of worked examples within the prompt will help the model internalize stepwise solution patterns and generalize to novel, multi-hop tasks. [Data: Reports (14, 70, 117, 2, +more)]

3. **Least-to-Most Prompting** Decomposing a complex problem into an ordered sequence of subproblems will enable the model to tackle multi-step tasks incrementally—often yielding 7–15% gains on compositional benchmarks compared to standard CoT. [Data: Reports (16)]

4. **Self-Consistency Sampling** Sampling multiple chain-of-thought outputs and aggregating the most frequent final answer will enhance robustness and accuracy on challenging reasoning benchmarks, typically adding $\sim$7% absolute improvement. [Data: Reports (95)]

5. **Tree-of-Thoughts Search** Exploring multiple reasoning branches in a search tree—rather than following a single linear chain—will allow backtracking and parallel hypothesis testing, resulting in 15–22% performance improvements on abstract reasoning and game-playing tasks. [Data: Reports (91)]

Synergies and Best Practices – These methods often work best in combination. For example, pairing CoT prompts with self-consistency or embedding least-to-most decomposition within a tree-of-thought framework may further amplify multi-step reasoning performance. – Iterative feedback loops (e.g., critique-and-reprompt) and attention-guiding scaffolds can additionally complement the above techniques to reduce cascading errors and maintain logical coherence.