GraphRAG-LiteX: A Lightweight and Modular Framework for LLM-Based Summarization via Knowledge Graphs

1st Pathirage Puranjana Wijayarathna
Colombo, Sri Lanka

***Abstract*—This paper introduces GraphRAG-LiteX, a lightweight and modular framework for graph-based retrieval-augmented generation (RAG). The system builds on the principles of the original GraphRAG architecture by using entity and claim extraction to construct a knowledge graph, applying community detection for content structuring, and performing query-focused summarization via a map-reduce approach. Unlike the original design, GraphRAG-LiteX is optimized for local deployment and interpretability, relying on a simplified pipeline and a locally hosted open-source language model (*deepseek-r1-distill-qwen-7b*) executed via LM Studio.**

To evaluate the system, a set of global comprehension questions was posed over a recent news article corpus. Answers generated by GraphRAG-LiteX and a dense retrieval-based VectorRAG baseline were compared using an LLM-as-a-judge evaluation protocol. The proposed system outperformed the baseline in 83.3% of comparisons, particularly excelling in thematic reasoning and structural coherence.

GraphRAG-LiteX demonstrates that graph-based RAG architectures can be adapted for constrained environments without sacrificing answer quality.

***Index Terms*—GraphRAG, Knowledge Graph, Question Answering, Community Detection, InfoMap, Local Deployment, Open-Source LLMs, Summarization, Global Sensemaking**

I. Introduction

Retrieval-Augmented Generation (RAG) has gained traction as a method to enhance the factual basis and relevance of outputs generated by large language models (LLMs). Traditional RAG implementations typically rely on dense vector similarity techniques to identify relevant passages within a document corpus. While this approach has proven effective for specific factual queries, it faces limitations when confronted with broader questions that demand structural comprehension, thematic synthesis, or the integration of global context [1].

To overcome these limitations, Microsoft Research introduced GraphRAG, a graph-based framework. It processes documents to extract entities and claims, which are then used to construct a knowledge graph. Hierarchical community detection identifies meaningful substructures, and summaries are generated at various levels of abstraction to support query-focused reasoning. While promising, the original implementation of GraphRAG required substantial computational resources and complex infrastructure, including dense retrieval integration, scoring fusion mechanisms, and model-based self-reflection [2].

To address these challenges, a simplified version of the original framework, GraphRAG-LiteX, has been developed and introduced in this paper. This new approach retains the core principles of GraphRAG, such as entity extraction, claim extraction, graph construction, community detection, and summary-based query answering, while eliminating components that contribute to system complexity and computational overhead. GraphRAG-LiteX is implemented as a modular and lightweight Python framework. It doesn't rely on vector databases or distributed processing systems. Instead, all components interact through direct, prompt-based communication with a local or hosted LLM. For this study, the entire system was deployed and executed using LLM Studio, a development platform for running local LLMs efficiently. A distilled version of the DeepSeek model, deepseek-r1-distill-qwen-7b, was used as the language model across all components, including entity extraction, claim identification, community summarization, and answer generation. This model was selected for its balance between performance and resource efficiency but mainly as it was open source.

An evaluation of GraphRAG-LiteX was conducted using a limited set of global comprehension questions. The results showed that GraphRAG-LiteX can produce competitive and often superior answers for complex, synthesis-oriented queries, even though it has a simpler design.

This study describes the design and implementation of GraphRAG-LiteX, discusses its evaluation results, and highlights opportunities for future enhancements and integration with hybrid systems.

II. Background

Retrieval-Augmented Generation (RAG) has emerged as a significant advancement in the utilization of large language models (LLMs). It integrates information retrieval with generative capabilities to produce contextually relevant outputs. Traditional RAG systems retrieve relevant passages from a database based on the input query and then use generative models to synthesize and articulate responses based on the retrieved information. This dual approach offers a more robust solution compared to standalone generative models, which often rely solely on their intrinsic training to navigate queries [3]. Traditional RAG frameworks often operate on static datasets,

which limits their ability to adapt dynamically to new information or evolving retrieval needs. In real-world applications, knowledge is continuously updated, requiring systems that can seamlessly integrate emerging trends, insights, and technological advancements. A more flexible approach is needed—one that enables adaptive retrieval strategies to maintain relevance in rapidly changing domains. This work explores methods to enhance RAG systems with dynamic adaptability, ensuring robust performance even as data landscapes shift [4].

GraphRAG, a retrieval-augmented generation (RAG) methodology, addresses the limitations of conventional vector-based RAG approaches in handling global sensemaking tasks. Traditional RAG relies on retrieving semantically similar documents using vector embeddings to answer localized queries. However, it struggles with questions requiring comprehensive understanding of an entire corpus, such as identifying overarching themes or systemic patterns. GraphRAG circumvents this limitation by constructing a knowledge graph from the corpus using large language models (LLMs). Nodes in this graph represent entities, and edges capture their relationships. The graph is then hierarchically partitioned into communities of closely related entities using algorithms like Leiden [2].

Summaries of each community are generated through LLMs and organized in a hierarchical structure, enabling scalable and modular access to global content. During query time, GraphRAG utilizes a map-reduce strategy. It maps the query to each community summary to produce partial answers in parallel. These partial answers are then reduced to form a final global response. This approach supports reasoning across an entire corpus and facilitates structured summarization grounded in the community organization of the knowledge graph.GraphRAG emphasizes modularity and scalability by summarizing local communities before aggregating them. This approach improves both the comprehensiveness and diversity of the generated answers. Empirical evaluations on real-world datasets show that GraphRAG significantly outperforms vector RAG in global query tasks. Therefore, it is an effective solution for complex information synthesis over large-scale text corpora [2].

This study is upon inspriration from this particular GraphRAG study conducted by Microsoft.

III. Methodology

The system follows a structured workflow that begins with the ingestion of raw source documents. These documents are processed into manageable text chunks using a fixed token window with overlap to preserve contextual continuity. These text chunks are then passed to a lightweight entity extractor and claim extractor, both powered by local large language models, to identify key entities and their relationships. The extracted information is used to construct a knowledge graph where nodes represent entities and edges denote relational claims. Once the graph is built, a community detection algorithm is applied to group semantically related entities into distinct graph communities. For each community, an LLM is prompted to generate a tailored summary that captures the key themes and facts relevant to that cluster. When a user query is presented, these summaries are leveraged to generate partial answers from each community, which are then aggregated and synthesized into a final comprehensive response using a map-reduce-style approach. This architecture supports efficient, interpretable, and context-rich question answering over large unstructured text corpora using graph-based reasoning.

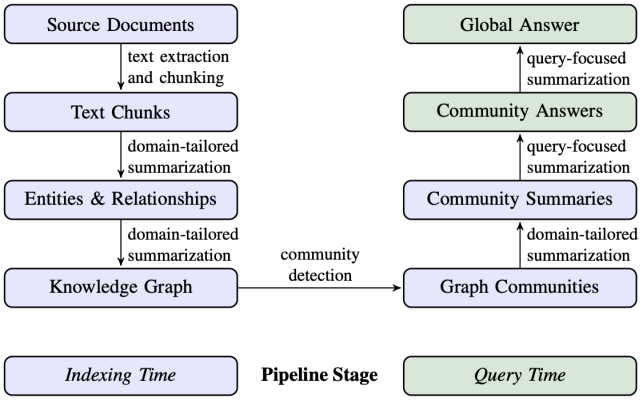

Fig. 1. Overall System Architecture [2]

A. LLM Configuration and Execution Environment

All language model operations in GraphRAG-LiteX were executed locally using **LM Studio**, a desktop interface for running open-source large language models (LLMs) without internet dependency. The model used in this study was *deepseek-r1-distill-qwen-7b*, a 7B parameter distilled variant of Qwen, optimized for lightweight deployment and instruction-following tasks. The model was served in MLX format and run on consumer-grade hardware with 18 GB RAM and GPU inference. Despite limitations in context window size and generation speed compared to commercial LLM APIs like GPT-4, this setup enabled fully offline execution, offering advantages in cost, privacy, and reproducibility. The model was used for all stages of the pipeline—including entity and claim extraction, community summarization, final answer generation and judgement, demonstrating the viability of deploying graph-based RAG systems in low-resource environments.

B. Text Chunking

To facilitate efficient processing of long documents with large language models that operate within limited context windows, we implemented a custom text chunking component called TextChunker. This module is responsible for dividing raw document content into overlapping segments based on token count, enabling manageable and semantically coherent input windows for downstream processing.

The chunker uses the cl100k_base tokenization scheme to encode input text into tokens. Each document is segmented into chunks of up to 600 tokens, with an overlap of 100 tokens between adjacent chunks. This overlap ensures that contextual continuity is preserved at the boundaries of each segment,

reducing the risk of losing important cross-sentence or cross-paragraph information.

Formally, given a tokenized document D with T tokens, the chunker produces a sequence of chunks C_i as follows:

$$C_i = \text{decode}(T_{i\cdot(S-O):i\cdot(S-O)+S})$$

where S is the fixed chunk size (600 tokens), O is the overlap size (100 tokens), and i denotes the chunk index. The process continues until the end of the document is reached. Each chunk is associated with metadata including its original source identifier and a unique chunk identifier.

For documents that contain fewer tokens than the defined chunk size, the entire document is treated as a single chunk. This approach strikes a balance between processing efficiency (by minimizing the number of LLM calls) and information fidelity (by preserving local coherence), and is well-suited for building scalable pipelines for knowledge extraction and graph-based reasoning over large text corpora.

C. Entity and Relationship Extraction

1) Entity Extraction: Following text chunking, entity and relationship extraction is applied to each chunk using two parallel components: a simplified entity extractor and a simplified claim extractor. These components utilize lightweight prompt-based interactions with a local language model to identify semantic elements from text without relying on traditional NLP pipelines.

Entities are extracted by prompting the language model to return tuples containing the entity name, type (e.g., person, organization, location), and a natural language description. Each tuple is then converted into a structured entity object. The procedure is described in Algorithm 1.

Algorithm 1 Entity Extraction Algorithm

Require: Text chunk T, source ID s
Ensure: List of extracted entities E
1: Initialize empty list E
2: Prompt LLM with T using predefined entity extraction template
3: Parse LLM response into list of $(\textit{name}, \textit{type}, \textit{description})$ tuples
4: **for all** tuple (n, t, d) in parsed output **do**
5: Generate unique ID u
6: Create entity object $e \leftarrow \texttt{Entity}(id = u, name = n, type = t, description = d, source = s)$
7: Append e to E
8: **end for**
9: **return** E

2) Relationship Extraction: In parallel, claims (representing relationships) are extracted in the form of subject-predicate-object triples. Each claim describes a directed edge between two entities with a semantic label (e.g., *"developed by"*, *"located in"*). This process is outlined in Algorithm 2.

The structured output from these components enables the creation of a knowledge graph where each node represents an entity and each edge corresponds to a claim or relationship. This method emphasizes clarity and modularity, ensuring that the outputs are easily auditable and traceable through document-level metadata and chunk associations.

Algorithm 2 Claim and Relationship Extraction Algorithm

Require: Text chunk T, source ID s
Ensure: List of extracted claims C
1: Initialize empty list C
2: Prompt LLM with T using predefined claim extraction template
3: Parse LLM response into list of $(\textit{head}, \textit{relation}, \textit{tail})$ triples
4: **for all** triple (h, r, t) in parsed output **do**
5: Generate unique ID u
6: Create claim object $c \leftarrow \texttt{Claim}(id = u, head = h, relation = r, tail = t, source = s)$
7: Append c to C
8: **end for**
9: **return** C

Both entity and claim extraction rely on deterministic prompt formats to enable easy parsing and avoid hallucination. Rather than relying on traditional named entity recognition models or syntactic parsers, this LLM-driven approach leverages in-context learning and string pattern extraction. Each extracted object is tracked with metadata such as the originating document, chunk index, and LLM response history, enabling traceability and error analysis during downstream processing.

D. Knowledge Graph Construction

To integrate extracted semantic information across multiple documents, a knowledge graph was constructed using the entities and claims derived from each text chunk. The graph builder component assembles these elements into a structured representation where nodes correspond to unique entities and edges denote relational claims.

Each entity is treated as a graph node, while each claim (represented as a subject-predicate-object triplet) is translated into a directed, labeled edge between two nodes. If an entity appears in multiple documents or chunks, it is de-duplicated using exact name matching. Each node and edge also carries associated metadata such as descriptions, source identifiers, and origin chunk references.

The construction process follows Figure 2, which operates on lists of entity and claim objects obtained from the extractor components.

This algorithm ensures that duplicate entities are merged based on their string name, while maintaining provenance through embedded metadata. Entity descriptions and claim annotations are retained as node and edge attributes, enabling interpretability and traceability of graph elements. The resulting graph serves as the core structure for subsequent operations including community detection, summarization, and query-focused reasoning.

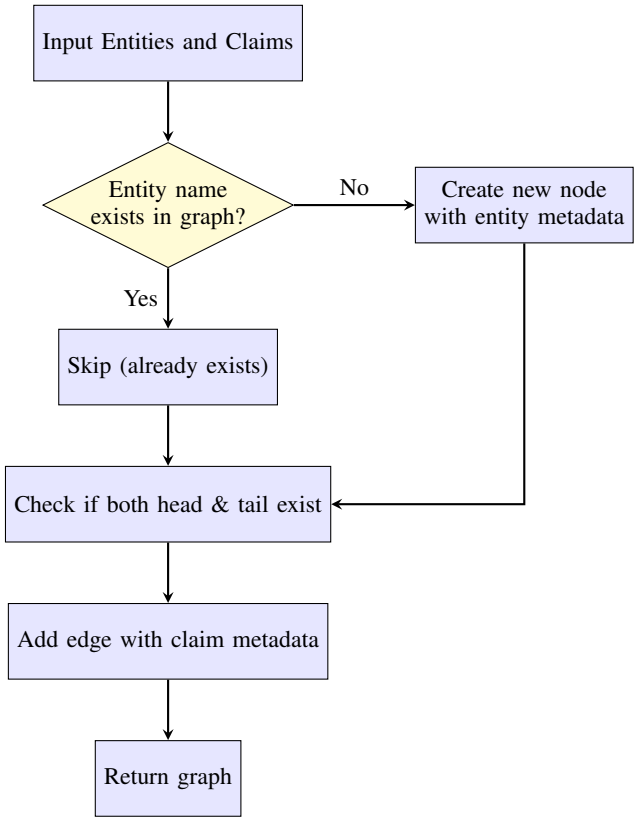

Fig. 2. Workflow for building the knowledge graph from extracted entities and claims.

E. Community Detection

To identify thematic substructures within the knowledge graph, community detection was performed using the InfoMap algorithm. InfoMap is an information-theoretic method that formulates the community detection task as a problem of compressing the description of a random walk on a graph. Communities are identified as modules that allow the most efficient encoding of the trajectory of such a walk, thus capturing groups of nodes with dense internal connectivity and sparse external links.

Given a graph $G = (V, E)$, where V represents entity nodes and E denotes relationships derived from extracted claims, the InfoMap algorithm searches for a partition $\mathcal{C}$ that minimizes the expected description length of a random walker's path. This is achieved by assigning unique codewords to modules and submodules such that the total encoding cost is minimized.

The high-level procedure is described in Algorithm 3.

This method was implemented using the InfoMap library, which provides Python bindings for efficient execution over large-scale graphs. Unlike modularity-based approaches, InfoMap does not rely on topological density alone, but leverages the flow dynamics of information within the graph. As a result, it is particularly effective in identifying communities in networks with directed or weighted edges, such as those derived from semantically rich claim-based relationships.

The resulting community assignments are used to guide localized summarization and enhance the interpretability of answer generation in the downstream stages of the system.

Algorithm 3 Graph Community Detection using InfoMap

Require: Graph $G = (V, E)$
Ensure: Partition $\mathcal{C}$ of nodes into communities
1: Represent G as a weighted undirected graph
2: Initialize each node $v \in V$ as its own community
3: **repeat**
4: Simulate random walks across G to estimate node transition probabilities
5: Compute the map equation for the current partition
6: Propose node movements and merge operations that reduce the code length
7: **if** a proposed move reduces the total description length **then**
8: Accept the move and update the community structure
9: **end if**
10: **until** no further improvement is possible
11: **return** community partition $\mathcal{C}$

F. Community Summarization

After detecting community structure in the knowledge graph, each community is summarized independently using a prompt-based large language model. The objective of this step is to transform each graph cluster—composed of semantically related entities and relationships—into a compact, human-readable summary that reflects its internal structure and content.

For each community, relevant information such as node names, node descriptions, relationship labels, and associated claims are aggregated and passed to a language model via a structured prompt. The summarizer generates an output that captures the most salient facts and relationships within that community. This process is designed to maintain interpretability while reducing the volume of data that needs to be processed during query time.

In the implementation, a rank-based sampling strategy is used to prioritize edges connecting high-degree nodes. The source and target nodes of each selected edge are collected along with their descriptions and any associated claims. These are encoded into a flat textual input, which is then submitted to the language model along with a community-level system prompt.

The resulting summaries are cached and reused during question answering. This caching mechanism ensures that summarization is computed only once per community, significantly improving the system's responsiveness at inference time. The summaries also serve as the building blocks for hierarchical reasoning and final answer synthesis, allowing the system to navigate from localized insights to broader thematic explanations efficiently.

G. Answer Generation

To support query-focused reasoning over large text corpora, answers are generated using a map-reduce strategy over precomputed community summaries. When a user submits a question, the system performs a two-stage process: (1) it generates partial answers from individual community summaries, and (2) it aggregates these partial results into a final, unified response.

The system first identifies all community summaries, then constructs a standardized prompt by combining each summary with the input question. These prompts are submitted independently to the language model, which responds with partial answers scoped to the content of each specific community. This step corresponds to the map phase and is fully parallelizable.

Each partial answer is scored according to its relevance and informativeness. In this implementation, relevance scoring is either computed explicitly by prompting the model or inferred based on heuristic ordering. Once scored, the top-ranked partial answers are selected and concatenated to form the input to the reduce phase.

In the reduce phase, the selected partial answers are jointly processed by the model to produce a single, coherent answer to the original question. This step benefits from the compressed yet diverse nature of the partial responses, allowing the model to synthesize higher-level insights that span multiple parts of the corpus.

To ensure efficiency and clarity, all prompt templates are optimized for compatibility with the `deepseek-r1-distill-qwen-7b` model running locally through LM Studio. The model's deterministic configuration, low temperature, and moderately sized token limits are specifically chosen to balance generation quality with performance in offline environments.

This map-reduce approach enables scalable and interpretable question answering over graphs derived from long, unstructured document sets. It supports both broad thematic queries and specific investigative questions without requiring document re-processing at inference time.

H. Evaluation Methodology

To evaluate the performance of the GraphRAG-LiteX system, we conducted a comparative study against a vector-based Retrieval-Augmented Generation (VectorRAG) baseline. The evaluation focuses on both structural and content-based question answering.

Dataset and Questions: The corpus consists of multiple text files stored in the directory. Three representative questions were selected to evaluate the systems:

1) What are the main themes discussed in this corpus?
2) What key entities are involved and how do they relate?
3) What insights can be derived about the topic's evolution over time?

These questions are designed to test global sensemaking, entity-relationship understanding, and temporal insight generation.

Evaluation Procedure: The evaluation process is implemented as an asynchronous Python script. It begins by reading all input documents using a helper function and then runs both retrieval-augmented systems independently.

The GraphRAG-LiteX pipeline processes documents, builds a knowledge graph, detects communities, generates summaries, and finally produces answers based on top-level summaries (community level C0). - The VectorRAG baseline retrieves semantically relevant chunks using dense embeddings and passes them to a language model to generate answers.

For each question, both systems generate an answer. These answers are compared using an automated language model judge that evaluates the quality of both responses based on predefined criteria. The judge assigns a winner for each comparison or declares a tie.

Judgment Criteria: Each pair of answers is assessed based on the following qualitative criteria:

- Relevance to the question
- Coverage and informativeness
- Faithfulness to the document contents
- Clarity and coherence

The language model selects a winner and provides a short explanation. It also generates optional scores for each system if applicable.

Scoring Formula: Let s_1 and s_2 be the scores assigned to System 1 and System 2, respectively. The winning system W is determined by:

$$W = \begin{cases} \text{System 1}, & \text{if } s_1 > s_2 \\ \text{System 2}, & \text{if } s_2 > s_1 \\ \text{Tie}, & \text{if } s_1 = s_2 \end{cases}$$

The scores s_1 and s_2 are qualitative assessments generated by the language model based on criteria such as relevance, informativeness, and coherence.

Result Logging: The results of all comparisons are collected and saved in CSV format. Each row in the output file includes the original question, both answers, the evaluation criterion, winner, and justification. This ensures traceability and allows for further quantitative or manual review if needed.

The proposed system integrates lightweight prompt-based extraction, graph construction, and community-driven summarization to enable scalable and interpretable question answering over large text corpora. Each stage—from token-based chunking to final answer synthesis—is modular and optimized for execution with a local large language model, specifically *deepseek-r1-distill-qwen-7b* via LM Studio. This architecture balances computational efficiency with semantic depth, allowing the system to perform high-quality, context-aware reasoning across diverse and unstructured datasets. The following sections present the evaluation methodology and results that demonstrate the system's performance on representative sensemaking tasks.

IV. Results

To evaluate the effectiveness of the proposed GraphRAG-LiteX framework, an experiment was conducted using a dataset of 10 news articles published 48 hours before the experiment. Three high-level analytical questions were posed to both GraphRAG-LiteX and a VectorRAG baseline. The systems' responses were evaluated using a language model-based judge across multiple qualitative criteria, including comprehensiveness, diversity, empowerment, and directness.

A total of 12 comparisons were generated. As shown in Table I, GraphRAG-LiteX was preferred in 10 of the 12 evaluations (83.3%), while VectorRAG was selected in 2 cases (16.7%). No ties were recorded in the evaluation.

TABLE I
Evaluation Summary: GraphRAG-LiteX vs VectorRAG

System	Wins	Percentage
GraphRAG-LiteX	10	83.3%
VectorRAG	2	16.7%
Total	12	100.0%

The total evaluation time was 7758.47 seconds. Of this, 7209.40 seconds (92.9%) were used for processing within GraphRAG-LiteX, 128.76 seconds (1.7%) for VectorRAG, and 420.30 seconds (5.4%) for LLM-based judgment.

TABLE II
Evaluation Results by Question and Criterion

Criterion	Winning System
What are the main themes discussed in this corpus?	
Comprehensiveness	GraphRAG-LiteX
Diversity	GraphRAG-LiteX
Empowerment	GraphRAG-LiteX
Directness	GraphRAG-LiteX
What key entities are involved and how do they relate?	
Comprehensiveness	VectorRAG
Diversity	GraphRAG-LiteX
Empowerment	VectorRAG
Directness	GraphRAG-LiteX
What insights can be derived about the topic's evolution over time?	
Comprehensiveness	GraphRAG-LiteX
Diversity	GraphRAG-LiteX
Empowerment	GraphRAG-LiteX
Directness	GraphRAG-LiteX

Table II presents the detailed evaluation results across three analytical questions and four qualitative criteria. It can be observed that GraphRAG-LiteX (labeled as GraphRAG-LiteX) consistently outperformed VectorRAG in most cases. Notably, it was the preferred system for all criteria under the first and third questions, which required high-level thematic synthesis and temporal reasoning. For the second question, which focused on entity-level relationships, performance was more balanced, with each system winning two criteria. These results suggest that GraphRAG-LiteX is particularly effective in tasks that involve summarizing complex ideas and tracking topic evolution.

The LLM-based evaluator consistently highlighted GraphRAG-LiteX's superior thematic synthesis, structural organization, and clarity when reasoning about entities and their relationships. These results demonstrate that even without the infrastructure and computational overhead of dense retrieval systems, GraphRAG-LiteX can offer effective performance on complex sensemaking tasks.

V. Discussion

The evaluation results demonstrate that GraphRAG-LiteX, despite its lightweight design, is capable of outperforming a traditional dense retrieval-based VectorRAG baseline on global comprehension tasks. In 10 out of 12 qualitative comparisons, the proposed system was judged superior, particularly in criteria associated with thematic synthesis, structural reasoning, and entity understanding. These findings validate the effectiveness of a graph-augmented pipeline in generating high-quality, globally informed answers using a locally hosted LLM.

A. Improving Answer Quality and Robustness

While GraphRAG-LiteX was the preferred system in most cases, a performance gap remains in certain criteria, such as comprehensiveness and directness. These limitations are largely attributed to deliberate simplifications made to support local deployment. Specifically, the system relies on a single-pass summarization strategy and does not maintain a multi-level hierarchical community structure.

To address these limitations, several enhancements are proposed:

- Hybrid retrieval methods that combine graph-based traversal with semantic vector search could improve the retrieval of relevant and diverse content.
- Iterative summarization or refinement could enhance abstraction quality, allowing for more comprehensive and coherent summaries.
- Scoring-based fusion could prioritize community subgraphs with higher relevance to the user query, improving answer specificity.
- Domain-adaptive prompting techniques could be used to tailor entity and claim extraction processes for specialized content domains.

These enhancements are expected to improve coverage and robustness, especially for information-dense or multi-perspective questions.

B. Comparison to the Original GraphRAG Framework

GraphRAG-LiteX follows the core methodological structure of the original GraphRAG framework [2], including prompt-driven entity and claim extraction, knowledge graph construction, community detection, and map-reduce summarization. However, significant differences exist in terms of complexity and deployment.

The Microsoft GraphRAG system integrates dense retrieval, recursive summarization across multiple community levels,

model self-reflection, and scoring fusion—all supported by proprietary APIs and GPT-4. In contrast, GraphRAG-LiteX uses a locally hosted model, *deepseek-r1-distill-qwen-7b*, executed via LM Studio. This choice prioritizes accessibility and interpretability, while sacrificing multi-stage inference and high-depth recursion due to speed constraints.

C. Design Trade-offs and the Use of InfoMap

A key divergence from the original design lies in the choice of community detection algorithm. While GraphRAG uses the Leiden algorithm to build hierarchical community graphs, GraphRAG-LiteX employs InfoMap. InfoMap is a flow-based algorithm known for its efficiency and its ability to identify semantically coherent clusters, which makes it well-suited for thematic grouping in natural language graphs. This substitution reduces computational overhead and aligns with the system's goal of being deployable in constrained environments.

D. Evaluation Strategy and Scalability

GraphRAG-LiteX adopts a scalable, low-cost evaluation method using a language model as a judge. Although this strategy lacks the rigor of human annotation, it provides consistent, reproducible insights for early-stage research. The LLM judge assesses each pair of system outputs across multiple qualitative dimensions and provides both a winner and a rationale.

Importantly, the system architecture avoids reliance on external APIs, distributed databases, or GPU clusters. This design makes GraphRAG-LiteX particularly valuable for academic, experimental, and educational contexts where resources are limited but model-based reasoning is still desired.

E. Future Directions

Future work will focus on enhancing both retrieval and reasoning capabilities. The integration of hybrid retrieval mechanisms is expected to increase semantic coverage without exceeding local model constraints. Multi-turn summarization and scoring-based fusion may allow better control over answer quality. Entity and claim extraction modules can also be refined using domain-specific schemas to improve precision in technical or specialized content. Additionally, as more efficient LLMs emerge, deeper summarization and hierarchical reasoning could be reintroduced without compromising portability. Together, these directions will bring GraphRAG-LiteX closer to the expressive power of its larger predecessor, while preserving its lightweight and interpretable design philosophy.

VI. Conclusion

This paper presented GraphRAG-LiteX, a lightweight and modular framework for graph-based retrieval-augmented generation. The system was designed to preserve the core methodological insights of the original GraphRAG architecture—namely, the use of entity-centric knowledge graphs, community detection, and map-reduce style summarization—while simplifying the implementation to support local deployment and rapid experimentation.

GraphRAG-LiteX was evaluated against a traditional vector-based RAG baseline using a corpus of recent news articles and a set of global comprehension questions. The results demonstrated that the proposed system outperformed the baseline in 83.3% of evaluations, with particular strength in areas involving thematic synthesis and structural reasoning. These findings highlight the potential of graph-based approaches to provide coherent and contextually grounded answers without relying on dense retrieval or external APIs.

By adopting a locally hosted open-source model (*deepseek-r1-distill-qwen-7b*) and avoiding the complexity of large-scale infrastructure, GraphRAG-LiteX offers a practical alternative for low-resource environments, academic research, and educational settings. While the system currently omits advanced techniques such as hierarchical summarization and scoring fusion, its modular design enables straightforward extension.

Future enhancements will explore hybrid retrieval integration, iterative summarization, domain-specific extraction, and more adaptive scoring mechanisms. These improvements aim to further close the performance gap with large-scale frameworks while maintaining transparency, portability, and ease of use.

In summary, GraphRAG-LiteX demonstrates that high-quality, structured reasoning with LLMs is achievable without industrial-scale infrastructure—making graph-augmented summarization more accessible and adaptable across diverse application domains.

References

[1] J. Chen, H. Lin, X. Han, & L. Sun, "Benchmarking large language models in retrieval-augmented generation", Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, no. 16, p. 17754-17762, 2024. https://doi.org/10.1609/aaai.v38i16.29728

[2] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, D. Metropolitansky, R. O. Ness, and J. Larson, "From Local to Global: A Graph RAG Approach to Query-Focused Summarization," arXiv preprint arXiv:2404.16130, Apr. 2024. [Online]. Available: https://arxiv.org/abs/2404.16130

[3] H. Lee and S. Kim, "Bring retrieval augmented generation to google gemini via external api: an evaluation with big-bench dataset",, 2024. https://doi.org/10.21203/rs.3.rs-4394715/v1

[4] G. Izacard and É. Grave, "Leveraging passage retrieval with generative models for open domain question answering", Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, 2021. https://doi.org/10.18653/v1/2021.eacl-main.74