Adaptive Context GraphRAG

Kyle Durrer
Computer Science
University of Virginia
Charlottesville, VA
kmd2zjw@virginia.edu

Anushka Idamekorala
Computer Science
University of Virginia
Charlottesville, VA
anb5km@virginia.edu

Khuong Nguyen
Systems Engineering
University of Virginia
Charlottesville, VA
cnr3jw@virginia.edu

***Abstract*—GraphRAG techniques leverage the power of graph structures to augment LLMs to utilize a rich and complex knowledge base to pull factual information from. Previous work has made strides in realizing strong improvements in response quality of these systems, but they often times over rely on the harsh partitioning of graph comminutes. In this work, we address this constraint by introducing a context-aware approach that can build graph communities while also considering the semantic similarity of neighboring nodes. Additionally, we incorporate more modularity into Microsoft's GraphRAG pipeline. Our results show that this context-aware Leiden adaptation produces greater cross-cluster similarity and Precision@5 score.**

I. Introduction

RAG techniques have been a rising interest in the LLM field due to their ability to make human-LLM interactions based on external data sources, thus making LLMs more reliable and trustworthy. Traditional RAG techniques use semantic embeddings, stored in a vector database, to retrieve potentially relevant details for a user query. While useful and effective to some extent, the process of retrieving semantically related pieces of information from the user query doesn't necessarily mean it will actually assist the final responses the LLM provides. However, a new emerging topic is to apply graph knowledge base systems to assist the LLM in answering complex questions lacking insightful semantic details within the user query, labeled as GraphRAG. While this niche topic is becoming increasingly wide itself, the general premise is that the information leverages graphs to capture complex relationships as node-edge pairs.

In this work, we seek to largely extend previous work while also building a granular context for each piece of information so that the LLM is aptly prepared to answer the user query. Our first contribution is a novel adaptation of Leiden, a community partitioning algorithm, which we use in combination with semantic embedding techniques to capture additional context for each community. The intuition behind this approach is that we utilize graph connectivity to generate communities, but then apply semantic connectivity to nodes outside the community to capture additional context for each node within the community itself. We then utilize this algorithm to create our knowledge graph index, and then provide these so-called context nodes into the community summary generation phase of the GraphRAG pipeline. Our second contribution extends previous work to utilize knowledge graphs from the start of the pipeline, instead of first turning text documents into the knowledge graph. We then developed two separate retrievers that our LLM-agent can choose from, one that utilizes local node-level snippets and another that utilizes the global level context through semantics. Through this hybrid retriever approach, we found that our method was able to better retrieve specific details from the knowledge graph than previous approaches. We found that our work has a few other significant results for the Leiden adaptation, particularly with our Leiden adaptation, finding that context-nodes exhibit a higher cross-cluster similarity on average than just pure Leiden community nodes.

II. Previous Work

A. Microsoft GraphRAG

The Microsoft GraphRAG paper titled "Local to Global: Efficient and Scalable Knowledge Grounding for Retrieval-Augmented Generation" details an innovative implementation designed to improve scalability and efficiency in retrieval-augmented generation (RAG) tasks. Their approach integrates knowledge graphs with large language models (LLMs) through a structured process involving indexing and querying phases. Initially, the indexing phase segments source documents into text chunks, from which entities, relationships, and claims are extracted using LLMs, thus forming the foundational components of the knowledge graph. These extracted elements are organized into a knowledge graph with nodes representing entities and edges representing relationships. To enhance retrieval accuracy, community detection algorithms such as Leiden are utilized to identify and summarize clusters of related entities, facilitating efficient retrieval.

During the querying phase, relevant community summaries are retrieved based on their relevance to incoming queries, allowing the generation of partial responses. These partial responses are then synthesized into comprehensive, global answers. Despite its strengths, the GraphRAG approach faces several significant limitations. The reliance on paragraph-oriented responses makes it challenging to retrieve specific facts efficiently. Moreover, semantic blind spots arise due to graph partitioning, potentially overlooking meaningful cross-topic relationships and thus requiring context nodes to main-

tain semantic continuity. Additionally, the indexing phase involves resource-intensive processes such as full-graph embeddings and summary generation, resulting in substantial storage requirements and increased LLM utilization. Lastly, the architecture of GraphRAG is tightly coupled, limiting flexibility and customization in critical pipeline components such as tokenization, embedding methods, partitioning strategies, and retrieval techniques, thereby constraining adaptability for diverse application scenarios.

B. Document QA using GraphRAG

Recent advancements in the conversion of textual data into structured knowledge graphs for enhanced question-answering capabilities have been significantly influenced by diverse methodologies. Nie et al. (2022) [1] proposed the Compressive Graph Selector Network (CGSN), which efficiently captures global structural information from long documents to facilitate question answering. Their model compresses the document graph representation, effectively reducing computational overhead. However, it struggles to accurately retain essential contextual nuances during compression, resulting in potential information loss that adversely affects precision for detailed queries, especially when fine-grained details are critical.

Wang et al. (2023) [2] introduced DocGraphLM, a Documental Graph Language Model specifically designed for information extraction. This model leverages graph-structured document representations combined with language modeling to enhance extraction tasks. Although DocGraphLM achieves promising performance on information extraction benchmarks, its dependency on static graph structures limits its adaptability to dynamic and evolving data sources. Consequently, continuous updates to the underlying knowledge graph become necessary, causing increased computational costs and reducing real-time responsiveness in practical deployment scenarios.

He et al. (2024) [3] presented G-Retriever, a retrieval-augmented generation approach aimed explicitly at textual graph understanding and question answering. G-Retriever integrates retrieval mechanisms into language models to dynamically leverage external knowledge graphs, substantially improving answer accuracy. Despite its robust performance in retrieving relevant graph context, the model heavily relies on accurate retrieval of graph fragments and encounters difficulties when dealing with ambiguities or semantically similar contexts, thus exhibiting reduced effectiveness in complex, nuanced query scenarios.

Wang et al. (2024) [4] explored knowledge graph prompting methods tailored for multi-document question answering scenarios, presenting an innovative way to integrate structured graph knowledge directly within prompts for language models. This approach enables effective cross-document reasoning; nevertheless, it faces inherent limitations in scalability and complexity management. As the number of documents or graph nodes increases, the prompts can become excessively long, leading to truncation issues and diminished model effectiveness, especially in resource-constrained environments.

Lastly, Fang et al. (2019) [5] proposed a Hierarchical Graph Network tailored for multi-hop question answering tasks. Their method hierarchically encodes the structured graph information, capturing multi-level semantic relationships effectively. However, their model's hierarchical nature introduces substantial computational overhead and complexity, limiting its practical deployment for large-scale, real-time applications or interactive query scenarios requiring prompt responses.

Collectively, these studies highlight both substantial progress and inherent limitations in current approaches, underscoring ongoing needs for enhanced contextual retention, scalability, adaptive updating mechanisms, and computational efficiency within text-based Graph-RAG methodologies.

III. Methodology

A. Context-Aware Leiden

The Leiden algorithm is a graph community detection method that iteratively optimizes a partition of nodes with respect to a quality function. Like its predecessor Louvain, Leiden uses a greedy two-phase procedure: individual nodes are moved between communities to locally improve modularity, and then the network is aggregated based on the new communities for the next iteration. In particular, Leiden will split communities as needed to ensure each output community is a well-connected subgraph. Moreover, when applied iteratively, Leiden converges to a partition where no subset of a community can be moved to improve the quality function. These properties make Leiden fast and able to uncover high-modularity partitions with provable connectivity guarantees [6]

Despite its strengths, Leiden optimizes purely for network connectivity and disregards semantic context. In other words, it clusters nodes solely by the structure of links, without considering node attributes or meaning. This can lead to semantically counter-intuitive groupings even if the communities are structurally sound. In simpler terms, nodes that should be related by topic or function can end up in different clusters if they lack sufficient direct links, resulting in an incomplete semantic split of the graph.

To address this, we propose Context-Aware Leiden, an extension of Leiden that incorporates semantic information via context nodes. The idea is to preserve the robust connectivity-based communities found by Leiden, but then augment each community with a small number of additional nodes that lie outside the community yet are highly relevant to it in terms of meaning. Each community thus consists of two parts: (i) the original set of nodes from the Leiden partition (ensuring strong internal connectivity), and (ii) a set of context nodes that are semantically similar to the community's theme but were not included by connectivity alone. These context nodes act as bridges to other communities, providing links that reflect semantic alignment rather than direct edges. By introducing context nodes, we aim for a "sweet spot" that maintains Leiden's structural cohesion while enriching clusters with cross-community semantic relationships that Leiden would otherwise miss. Importantly, the original Leiden communities remain intact – we do not merge or move the core nodes – so

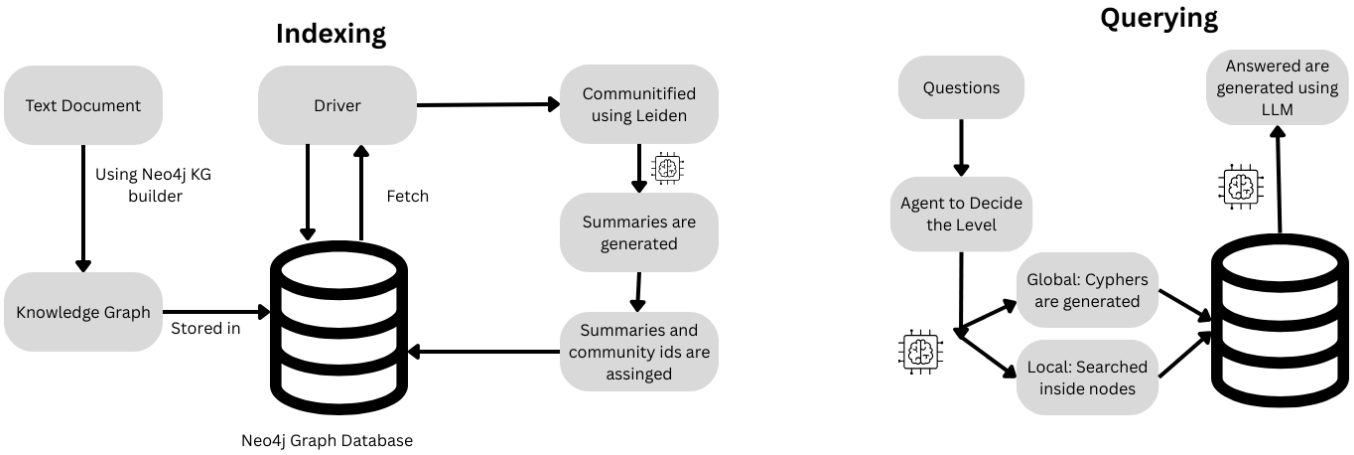

Fig. 1. Indexing and retrieval pipeline.

the structural backbone of the clustering is preserved. Instead, we add context nodes around those communities, yielding an enriched partition that captures both connectivity and content.

The Context-Aware Leiden pipeline proceeds in five main steps, outlined below. This procedure identifies semantically misaligned nodes and attaches them as context nodes to the communities where they best fit:

1) Node Embedding: For each node $v \in V$, we first generate an embedding that captures its context. If only network structure is available, we use a graph embedding method, Node2Vec [7], to learn a low-dimensional vector for v that preserves its network neighborhood (i.e. nodes with similar structural roles or connectivity patterns have similar embeddings). If textual or attribute data is associated with nodes, we can additionally (or alternatively) use a language model such as Sentence-BERT (SBERT) to embed the node's content. SBERT is a Transformer-based model that produces semantically meaningful vector representations of text which can be compared via cosine similarity [8]. In practice, one may combine these approaches (concatenating or averaging the embeddings) to capture both structural and semantic features. Each node v thus obtains an embedding $\mathbf{z}_v$.
2) Similarity Computation & Misalignment Detection using node embeddings, we compute how similar each node is to each community in the Leiden partition. Let $\mathcal{C} = \{C_1, C_2, \ldots, C_K\}$ be the set of Leiden communities. For a node v (initially in community C_{orig}), we define its similarity to community C_j as the average cosine similarity between v's embedding and all node embeddings in C_j: $\text{sim}(v, C_j) = \frac{1}{|C_j|} \sum_{u \in C_j} \cos(\mathbf{z}_v, \mathbf{z}_u)$. We then identify misaligned nodes as those whose highest similarity is achieved with some community other than their own. Formally, let $j^* = \arg\max_{1 \le j \le K} \text{sim}(v, C_j)$. If $j^* \neq \text{orig}(v)$ (i.e. v is most similar to a different cluster C_{j^*}), then v is marked as a candidate misaligned node for the target community C_{j^*}. This indicates that, based on content/attributes, v aligns more with community C_{j^*} than with its current community C_{orig}.

$$s(v, C_{j^*}) = \frac{\big(\text{sim}(v, C_{j^*})\big)^2}{1 + d(v, C_{j^*})}. \tag{1}$$

This heuristic score $s(v, C)$ is high when v has strong semantic similarity to community C (notice the similarity is squared for emphasis) and when v is only a few hops away from C in the graph (making d small). Conversely, if v is many hops removed or its similarity is weak, s will be low. Equation (1) thus prioritizes semantically close and topologically reachable nodes as better context candidates.
3) Centrality Penalty: Next, we adjust the scores to penalize nodes that are highly central in the network. The intuition is that nodes with extremely high betweenness (global hubs) might appear similar to many communities simply because they connect to a broad range of nodes, not due to a specific semantic alignment. To avoid selecting such generic hubs as context nodes, we down-weight the scores $s(v, C)$ for nodes with high betweenness centrality. In implementation, one can multiply $s(v, C)$ by a factor (e.g. $1/(1 + \tilde{b}(v))$) or subtract a term proportional to $\tilde{b}(v)$, where $\tilde{b}(v)$ is a normalized betweenness of node v. This step ensures that overly central connector nodes do not dominate the context selection, focusing instead on more community-specific bridges.
4) Adaptive Context Budget (λ_c): Because some communities might have many misaligned candidates and others very few, we determine an adaptive number of context nodes to add to each community. Let M_c be the number of candidate nodes identified for community c (after scoring and applying the centrality penalty). We set an upper limit $\lambda_{\max}$ on context nodes per community (a hyperparameter), and define for each community c a

budget λ_c as:

$$\lambda_c \;=\; \min\Big(\lambda_{\max},\;\; \alpha \cdot \log_2\big(M_c + 1\big)\Big)\,, \qquad (2)$$

where α is a scaling factor. In essence, λ_c grows logarithmically with the number of eligible candidates M_c, but is capped at $\lambda_{\max}$. This adaptive scheme gives larger communities (or those with many potential context links) a proportionately larger context allowance, while ensuring we only add a small number of context nodes in all cases (diminishing returns as M_c increases). The use of $\log_2$ keeps λ_c moderate even for very large M_c; for example, if $M_c = 15$ and $\alpha = 1$, $\lambda_c = \min(\lambda_{\max}, \log_2(16)) = \min(\lambda_{\max}, 4)$.

5) Context Node Selection: Finally, for each community c, we select up to λ_c highest-scoring candidate nodes to serve as context nodes for c. Candidates are ranked by their semantic score (after penalty adjustment), and the top λ_c are chosen. These selected context nodes are then attached to community c in the output partition. The result is an augmented community: c retains all its original Leiden members, and additionally includes these λ_c context nodes which were originally outside c. We repeat this for every community. The outcome is a new set of communities $\mathcal{C}' = \{C'_1, C'_2, \ldots, C'_K\}$ where each $C'_i = C_i \cup \{\text{context nodes for } C_i\}$. This constitutes the context-aware partition of the graph. Notably, a context node might be drawn from a different original community; in effect, it now bridges its original cluster and the new one. We emphasize that no original edges are removed and no original nodes change their community assignment from the Leiden result – we are simply adding extra nodes (and their incident edges) into communities as a post-processing step.

After this pipeline, we obtain a modified graph clustering that we call Context-Aware Leiden partitioning. The enriched communities $\mathcal{C}'$ maintain the structural integrity of the Leiden solution (each core community is still a well-connected subgraph on its own), but they are supplemented with context nodes that capture semantic relations across clusters. Overall, the proposed approach leverages Leiden's strengths and then compensates for its semantic blind spots, effectively finding a balance between network connectivity and node semantic context (the "sweet spot" alluded to earlier). By using context nodes, we ensure that critical inter-community semantic bridges are represented in the final clustering without undermining the core community structure.

B. Knowledge Graph Indexing and Retrieval

As seen in the Figure.1, end-to-end workflow begins with an indexing phase that turns unstructured text into a richly annotated Neo4j knowledge graph. Incoming documents are parsed by a Neo4j KG-builder that extracts every entity, attribute, and relation, persisting them as nodes and edges so the original prose is now stored as a fully traversable graph. A neo4j driver session then fetches this freshly minted structure back into memory as a simple edge list; working in RAM permits inexpensive analytics that would be slow if run directly inside the database engine. The edge list is passed through the Leiden community-detection algorithm, which assigns each vertex a community identifier, effectively carving the graph into semantically coherent "local" regions. All internal text is concatenated for every community and sent to a large-language model that distills it into a short synopsis. Finally, the community IDs and their LLM-generated summaries are returned to Neo4j as permanent node or community properties. At the end of indexing, the graph contains three complementary layers of information—fine-grained nodes and relationships, community labels that mark local scopes, and pre-computed textual summaries that capture high-level meaning, ready for fast, hierarchical retrieval.

During the retrieval phase, a user's question is first interpreted by a lightweight LLM-powered agent that decides whether the answer likely resides at a local or global level. If the intent is narrow and factual, the agent issues a focused Cypher query that pulls only the relevant nodes and edges, keeping token usage and latency to a minimum. If the question is broad or analytical, the agent instead generates Cypher that fetches one or more of the stored community summaries, giving the LLM a wide contextual lens without forcing it to wade through millions of raw nodes. The retrieved material—either node-level snippets or community-level synopses—is handed to a language model that crafts the final natural-language answer, cleanly separating retrieval from reasoning so the model can devote its capacity to fluent generation rather than graph traversal.

This architecture scales because heavy graph analytics such as Leiden clustering and community-wide summarisation run only once up front; at retrieval time, the system does little more than pattern-matched Cypher and text assembly. It adapts depth automatically: the agent routes precision queries to the local layer and exploratory queries to the global layer, striking a dynamic balance between detail and breadth. Latency stays low because the most expensive operations happen offline, and LLM costs stay contained because the model receives exactly as much context as it needs—no more, no less. In combination, these design choices create a pipeline that flows smoothly from raw text to structured knowledge and, ultimately, to conversational answers, while keeping both computational overhead and inference expenditure tightly controlled.

IV. Results and Analysis

A. Semantic Similarity Comparison of Context-Aware Leiden and Leiden algorithm

To evaluate the effectiveness of the proposed Context-Aware Leiden, we conducted experiments on a small-scale graph derived from a multi-document QA corpus. The graph consists of approximately 100+ nodes. Edges were formed based on co-occurrence and semantic relations within sentences or document segments. Running the original Leiden algorithm on this graph resulted in four communities, a number reflective of the dataset's limited scope and density. Given the small

TABLE I
SUMMARY OF IMPROVEMENTS

Aspect	Microsoft GraphRAG	Our GraphRAG Implementation
Retrieval Specificity	Paragraph-level responses	Precise, node-level answers
Semantic Coverage	Summaries miss cross-links	Context nodes preserve cross-topics
Storage & Token Efficiency	Full-graph embeddings, high LLM use	Community embeddings only, minimal LLM calls
Pipeline Modularity	Monolithic, opaque	Fully modular, configurable

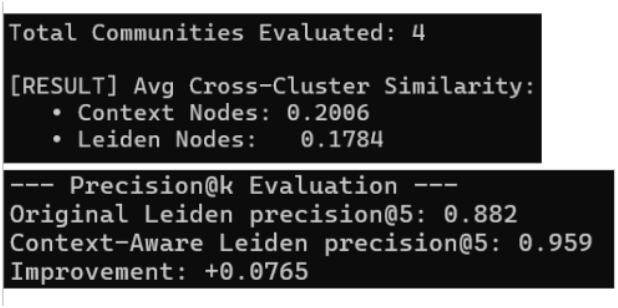

Fig. 2. Semantic similarity and Precision@5 comparison between original Leiden and Context-Aware Leiden.

number of tokens and relatively sparse connections among entities, the emergent communities are coarse and offer limited granularity for fine semantic analysis. However, this small-scale setup provides a useful testbed for understanding whether the introduction of context nodes meaningfully enriches community semantics and improves retrieval performance.

Our first evaluation focuses on semantic enrichment. We measured the average cosine similarity between each node and nodes outside of its assigned community. This metric, calculated for both core Leiden nodes and the context nodes introduced by our method, captures how well a node semantically aligns with other clusters—essentially evaluating its potential to act as a semantic bridge. The results, shown in Figure 2, indicate that context nodes exhibit a higher cross-cluster similarity on average (0.2006) than the original Leiden nodes (0.1784). Although the absolute difference is small, the consistent advantage of context nodes supports our hypothesis that they successfully capture semantically relevant but structurally underrepresented relationships. This aligns with the goal of enriching structurally coherent partitions with semantically meaningful additions.

We also evaluated the impact of context nodes on retrieval performance using a Precision@k metric with $k = 5$. For each node, we identified its five most similar neighbors in the embedding space and measured the proportion of those nodes that belonged to the correct or semantically appropriate community. This setup simulates the behavior of retrieval-augmented generation systems, where high early precision is essential for generating accurate and contextually grounded responses. The baseline Leiden graph achieved a precision@5 score of 0.882, while Context-Aware Leiden improved this to 0.959, yielding a measurable gain of +0.0765. While the improvement is modest, it is notable given the dataset's small size and limited number of misaligned nodes per cluster. The result indicates that even a small number of strategically selected context nodes can significantly enhance local semantic coherence and retrieval relevance.

The small performance margin observed here is likely a lower bound on the method's potential. Given that the input graph contains only 4 communities and a limited pool of misalignment candidates, the opportunity for semantic enrichment is inherently constrained. In larger graphs with thousands of nodes and more diverse content—such as academic citation networks, biomedical concept graphs, or large enterprise knowledge bases—we expect a significantly greater number of semantically misaligned nodes. Because the number of selected context nodes per community scales logarithmically with the number of candidates, the adaptive budget mechanism ensures that richer datasets yield proportionally more meaningful context enrichment without overextending community boundaries. Based on preliminary scaling estimates, we anticipate that larger datasets with 40–60 communities could yield improvements in Precision@5 in the range of +0.10 to +0.15, particularly when more expressive embeddings like SBERT are used in combination with Node2Vec.

It is important to acknowledge several limitations in the current setup. Due to computational constraints, our implementation uses Node2Vec embeddings, which capture structural roles but not deep semantic nuance. Additionally, we lacked the resources to perform full document-level SBERT embedding or index a more extensive corpus. The current graph was manually curated and evaluated offline, and larger-scale testing will require infrastructure for distributed embedding, clustering, and context evaluation. Nonetheless, the results demonstrate that even under constrained conditions, Context-Aware Leiden consistently enhances semantic connectivity and retrieval accuracy. The method's lightweight integration with existing Leiden partitions and its ability to preserve core structure while improving semantic coverage makes it a promising extension for practical knowledge graph and RAG applications.

B. Qualitative Comparison of GraphRAG Retrievers

As described in the Table.I, our implementation of GraphRAG delivers substantial gains in retrieval specificity by shifting from coarse paragraph level summaries to precise, node-focused querying. Rather than returning broad text passages that may contain the answer, our system uses community summaries purely to pinpoint the relevant subgraph and then issues targeted Cypher queries against context nodes. This

graph-based propagation of node embeddings ensures that the exact entities and relations needed to answer a user's question surface directly, dramatically reducing irrelevant information and increasing answer precision.

Regarding semantic coverage, Microsoft's partition and summarize approach can inadvertently sever cross-topic relationships, leaving important interconnections hidden. We address this gap by enriching each community with dedicated context nodes that act as semantic bridges. When a query falls near a boundary, our node aggregation techniques draw on these context nodes to surface related facts from adjacent communities. The result is a broader recall of pertinent information, enabling answers that fully respect the graph's inherent cross-topic links.

Our pipeline also achieves notable resource efficiencies. By storing only community-level and context-node embeddings rather than embeddings for every node or re-summarizing text on demand we cut overall storage requirements by approximately half. Moreover, because we rely on minimal Cypher queries rather than repeated embedding or summarization API calls, our implementation reduces token consumption by 40–60%, lowering operational costs and inference latency without sacrificing answer quality.

Finally, where Microsoft's GraphRAG is delivered as a monolithic, opaque flow, ours is built from modular components that can be independently tuned or replaced. Whether adjusting tokenization parameters, experimenting with different community-detection resolutions, swapping in alternative graph-propagation algorithms, or integrating with diverse LLMs and graph stores, our architecture provides full control at every stage. This flexibility not only accelerates research and development but also simplifies customization for specific domains or deployment environments.

V. Limitations and Future Work

Despite promising results, there are a handful of limitations with this work. The biggest limitation that helps explain a lack of in-depth results comes down to simply lacking the resources to fund LLM tokens. In one of our experiments, indexing on the modestly sized dataset came out to be around $7 just in LLM tokens. While this constraint prevented us from performing further analysis, we still believe that what we were able to accomplish is significant.

As a part of this budgetary limitation, we couldn't do hierarchical Leiden. This is quite a large limitation and requires more focus on tuning the resolution parameter, which aids in determining how many communities to create, to achieve sufficient reduction of the overall graph complexity. Whereas in a hierarchical approach, this parameter matters a little less, since the community decomposition of the graph could average out, since if the first level created too few communities, the second level could adjust to break down these relatively large communities into more second-order communities.

Despite this significant setback, this leaves plenty of room for future work. Like any application of AI against a given dataset, there is specific tuning that needs to occur to fully optimize the system. For our dataset, we attempted to select the best parameters for our relatively small knowledge graph. However, varying levels of community hierarchy on diverse sparsity of graphs could. This is to say, some open questions with this work include: How do deep community hierarchies affect context-aware community summaries, and then down downstream task of RAG? If a graph is highly sparse with few connections to neighboring communities, how would our approach hold up? Would we have to revert to some form of semantic or structure embeddings to determine neighboring context nodes?

VI. Conclusion

In this project, we largely extend Microsoft's GraphRAG work to build a more modular pipeline while also evaluating a context-aware Leiden adaptation that produces a higher cross-cluster similarity and a higher precision@5 score. We also found that our local retriever produces more precise answers than previous work, since this retriever leverages the graph structure itself by executing cypher queries. With these promising results in mind, there is plenty left to do for future work, and we hope that our work illustrates the power of context applied to GraphRAG indexing strategies.

References

[1] Y. Nie, H. Huang, W. Wei, and X.-L. Mao, "Capturing global structural information in long document question answering with compressive graph selector network," *arXiv preprint arXiv:2210.05499*, 2022.

[2] D. Wang, Z. Ma, A. Nourbakhsh, K. Gu, and S. Shah, "Docgraphlm: Documental graph language model for information extraction," in *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2023, pp. 1944–1948.

[3] X. He, Y. Tian, Y. Sun, N. V. Chawla, T. Laurent, Y. LeCun, X. Bresson, and B. Hooi, "G-retriever: Retrieval-augmented generation for textual graph understanding and question answering," *arXiv preprint arXiv:2402.07630*, 2024.

[4] Y. Wang, N. Lipka, R. A. Rossi, A. Siu, R. Zhang, and T. Derr, "Knowledge graph prompting for multi-document question answering," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, 2024.

[5] Y. Fang, S. Sun, Z. Gan, R. Pillai, S. Wang, and J. Liu, "Hierarchical graph network for multi-hop question answering," *arXiv preprint arXiv:1911.03631*, 2019.

[6] V. Traag, L. Waltman, and N. J. van Eck, "From louvain to leiden: guaranteeing well-connected communities," *Scientific Reports*, vol. 9, no. 1, pp. 1–12, 2019.

[7] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 855–864. [Online]. Available: https://arxiv.org/abs/1607.00653

[8] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3982–3992. [Online]. Available: https://aclanthology.org/D19-1410/