GraphEDU-RAG: A Hierarchical Knowledge-Graph Retrieval–Augmented Generation Workflow for Education Question-Answering

Khang Nguyen[1], Long Nguyen[1], Quynh Vo[1], Dung Le[1], Tho Quan[1*]

[1]URA Research Group, Faculty of Computer Science and Engineering, Ho Chi Minh City University of Technology (HCMUT), Ho Chi Minh City, Vietnam.

*Corresponding author(s). E-mail(s): qttho@hcmut.edu.vn;
Contributing authors: lnkhang.sdh232@hcmut.edu.vn;
long.nguyencse2023@hcmut.edu.vn; quynh.vo01@hcmut.edu.vn;
dung.lengochung@hcmut.edu.vn;

Abstract

Question Answering (QA) in the educational domain presents unique challenges, where questions often span multiple documents, require prerequisite reasoning, and involve terminology misalignment between learners and formal instructional content. While Retrieval-Augmented Generation (RAG) has proven effective in grounding large language models with external evidence, traditional approaches remain limited in their ability to capture structural dependencies and handle redundancy across heterogeneous learning materials. We introduce GraphEDU-RAG, a novel graph-based RAG framework tailored for educational QA. GraphEDU-RAG constructs a curriculum-scale knowledge graph, applies community detection to segment it into semantically coherent clusters, and generates cluster-level summaries to guide efficient retrieval. Our system further integrates a hybrid query strategy that combines graph traversal with embedding-based search and incorporates a duplicate-aware subgraph refinement mechanism to reduce redundancy and improve generation quality. Experimental results on a benchmark of complex educational queries demonstrate that GraphEDU-RAG outperforms standard RAG and existing graph-based baselines in terms of retrieval precision, answer completeness, and factual consistency. Our findings highlight the promise of graph-structured reasoning and hybrid retrieval in building pedagogically grounded QA systems.

Keywords: Retrieval-Augmented Generation, Knowledge Graph, Educational Question Answering, Graph-Based Retrieval

1 Introduction

Question Answering (QA) is a core task in *Natural Language Processing* (NLP), lying at the intersection of *Information Retrieval* (IR), knowledge representation, logical reasoning, and machine learning []. In recent years, QA has evolved into more interactive and context-sensitive forms such as *Conversational Question Answering Systems* (CQAS), which require models to handle multi-turn dialogues and maintain coherence across interrelated questions [].

Large Language Models (LLMs) have significantly advanced QA capabilities. However, they remain prone to *hallucination*, often generating fluent yet unsupported or incorrect claims []. The *Retrieval-Augmented Generation* (RAG) paradigm addresses this issue by retrieving external evidence and using it to guide the generation process [1]. While effective in many domains, RAG shows limitations when applied to education. Educational queries often combine concepts spread across documents and rely on prerequisite relationships that surface-level matching cannot capture. Embedding-based retrieval may miss vocabulary gaps between learners' phrasing and formal textbook language, while retrieved passages frequently contain redundant or overlapping content—leading to increased computational cost and lower synthesis quality.

Graph-based RAG offers a promising direction by structuring knowledge as a graph, where nodes represent concepts or passages and edges encode semantic or dependency relationships []. This structure enables richer retrieval via graph traversal and community detection. Yet two key challenges remain: first, mitigating information overload from densely connected subgraphs; and second, integrating graph reasoning with traditional text-based search to ensure robustness across diverse question types.

In this paper, we present GraphEDU-RAG, an end-to-end framework for educational QA that combines hierarchical graph modeling with hybrid retrieval and duplicate-aware refinement. The framework constructs a curriculum-scale knowledge graph, segments it into semantically coherent communities, and summarizes each cluster to support focused, contextually relevant retrieval. Our contributions are as follows.

- We propose a hybrid query strategy that combines graph traversal with embedding-based text retrieval, enabling flexible and robust information access across educational materials.
- We introduce a duplicate-aware subgraph refinement mechanism that removes redundant or overlapping nodes using content fingerprinting and graph centrality, improving precision and synthesis clarity.
- We apply community detection via the Leiden algorithm [2] to partition the graph into conceptually cohesive clusters, each accompanied by a summary that acts as a semantic retrieval unit.

- We empirically demonstrate that GraphEDU-RAG outperforms standard RAG and existing graph-based baselines in answer completeness, factual consistency, and retrieval efficiency on a benchmark of educational QA queries.

2 Preliminaries

2.1 Language Models (LMs)

Language Models (LMs) [3] are neural sequence-to-sequence models containing billions of parameters, which grant them the ability to capture complex linguistic patterns and generate highly coherent text. Modern LMs are predominantly built upon the Transformer architecture, whose core functionality relies on a series of specialized components. The process begins with embedding layers, which map discrete input tokens to continuous vectors, allowing the model to learn rich semantic representations during training. These embeddings are then processed by attention layers, which are crucial for capturing long-range dependencies by computing pairwise token weights without the need for recurrence. Finally, feed-forward sublayers apply per-token linear transformations and non-linear activations, serving to further extract and refine features from the token representations. These architectures are optimized for different types of tasks:

- **Encoder-Only Models:** Designed primarily for understanding and encoding input text. They excel at tasks that require deep comprehension of a given sequence, such as classification, sentiment analysis, and named entity recognition. Popular examples include **BERT** and its variants like **RoBERTa**, and Electra.
- **Decoder-Only Models:** Built for generative tasks, these models produce text autoregressively (predicting the next token based on previous ones). They are widely used for text generation, creative writing, chatbots, and more. **GPT-3**, GPT-4, LLaMA, and PaLM are prominent examples of this architecture.
- **Encoder-Decoder Models (Sequence-to-Sequence Models):** These combine both an encoder and a decoder, making them suitable for tasks where an input sequence needs to be understood and transformed into a different output sequence. Common applications include machine translation, summarization, and question answering. **T5** and **BART** are well-known models in this category.

Regardless of their specific architecture, LLMs are typically pretrained on massive unlabeled corpora (e.g., web text) to learn language structure and then fine-tuned or prompted for downstream tasks.

2.2 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) [4] has emerged as a popular and cost-effective paradigm for building question-answering (QA) systems with language models (LMs), especially in knowledge-intensive tasks. RAG augments a pretrained language model with an external retrieval step, grounding generation in up-to-date and task-relevant documents. A typical RAG pipeline consists of two main components: a Retriever, which identifies and ranks relevant documents from a large corpus

based on the input query using either sparse lexical methods such as TF–IDF [5] and BM25 [6], or dense embedding-based approaches [7]; and a Generator, typically a Transformer-based architecture such as BART [8], T5 [9], or GPT, which generates a fluent, grounded response conditioned on the query and the retrieved documents. One of the earliest and most widely used RAG variants, often referred to as Naive RAG, gained traction with the emergence of ChatGPT. This approach operates by enriching the user's original query with top-ranked retrieved documents and passing the augmented prompt to the generator. Despite its simplicity, Naive RAG has proven to be a strong baseline, offering enhanced factual accuracy, broader information coverage, and adaptability across diverse tasks such as QA and summarization by leveraging both retrieval-based precision and generative flexibility.

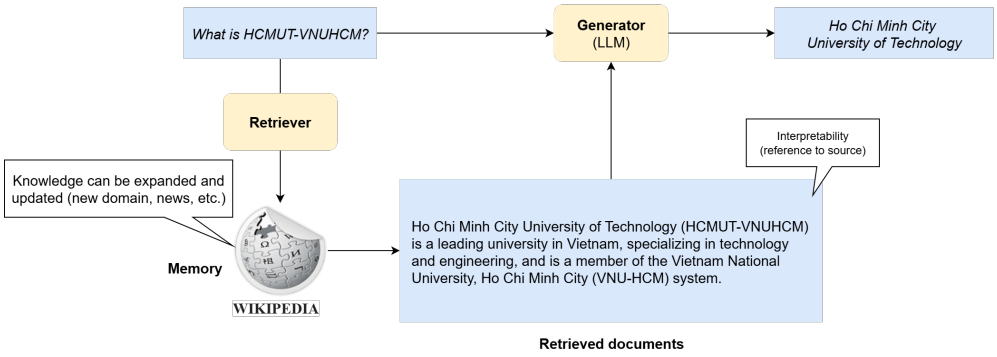

Fig. 1 An example of a typical RAG pipeline.

2.3 Leiden Algorithm

To partition our knowledge graph into meaningful, self-contained modules, we utilize the Leiden algorithm [2], a sophisticated community detection technique that offers significant advantages over its predecessor, the Louvain method [10]. The algorithm's primary objective is to find a partition of the graph that maximizes modularity, a quality score that measures the density of intra-community connections against a random baseline. This is initially achieved through a local optimization phase, where each node is considered for a move to a neighboring community. A move is executed only if it yields an increase in the global modularity score, Q, calculated as follows:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j). \tag{1}$$

Here, A_{ij} is the edge weight between nodes i and j, k_i and k_j are the summed edge weights (degree) of the respective nodes, m is the total sum of edge weights in the graph, and $\delta(c_i, c_j)$ is an indicator function that equals 1 if nodes i and j share the same community assignment.

A key innovation of the Leiden algorithm is its multi-stage workflow designed to overcome the shortcomings of simpler greedy approaches. After the initial local optimization, a refinement phase is performed. During this phase, the algorithm inspects each resulting community and, if any are not fully connected, splits them into their constituent connected components. The local optimization is then reapplied to this refined partition, ensuring that all final communities are internally cohesive. Following refinement, the algorithm enters a coarsening phase where each well-defined community is collapsed into a single node. The graph is rebuilt at this aggregate level, with new edge weights corresponding to the total weight of connections between the original communities. This entire procedure—local moving, refinement, and coarsening—is iterated until the partition stabilizes and no further improvements in modularity can be achieved.

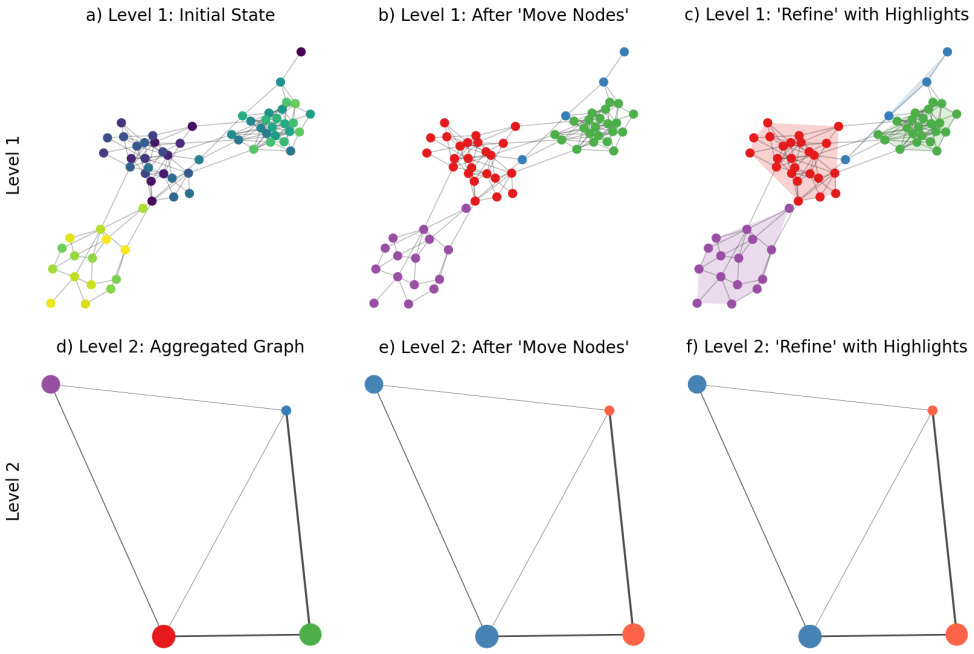

Fig. 2 Illustration of the local optimization process in the Leiden algorithm

This robust framework makes the Leiden algorithm particularly well-suited for the GraphEDU-RAG pipeline. Its guarantees of producing well-connected communities, combined with faster convergence properties, ensure that our knowledge graph is partitioned into structurally sound and semantically relevant subgraphs. This efficient and principled partitioning is a crucial prerequisite for the subsequent stages of our retrieval and generation model.

3 Related Works

3.1 Techniques and Frameworks for RAG Systems

Retrieval-Augmented Generation (RAG) enhances Large Language Models (LLMs) by first retrieving relevant information from external data and then using that data to inform the model's final response. This allows LLMs to overcome knowledge limitations and handle information corpora far larger than their internal memory [11]

- **Vector RAG**: This traditional approach involves retrieving a fixed number of documents whose vector embeddings are mathematically closest to the user's query. The LLM then generates its answer based solely on these retrieved snippets of text. While effective for specific queries, this method provides a fragmented view of the data.
- **GraphRAG**: This technique builds builds an explicit graph index over the entire dataset and applies community detection to segment the graph into thematic clusters. It then uses an LLM to summarize each cluster and iteratively merge those summaries into a comprehensive global overview. This graph-based, hierarchical method supports truly corpus-wide reasoning, rather than the local, fragmentary view of vector RAG.Building on advanced RAG techniques, GraphRAG also leverages large-scale "self-memory" summaries generated in parallel over graph segments—then aggregates them [1] and other hierarchical indexing approaches [12, 13].

Unlike flat or purely vector methods, GraphEDU-RAG's community-aware indexing enables both global and local sense-making across the full corpus related to education.

3.2 Leveraging Knowledge Graphs in LLMs and RAG Systems

Integrating Knowledge Graphs (KGs) provides LLMs and RAG systems with structured, interconnected data, which significantly enhances their capabilities. Methods for extracting knowledge graphs from natural-language text have traditionally relied on rule matching, statistical pattern recognition, clustering, and embedding techniques [14, 15]. More recent work employs LLMs to construct KGs directly from text [16].

- **KGs as an Index for Enhanced Retrieval:** Some RAG systems use a knowledge graph as a sophisticated index to improve information retrieval. For example, the StructuGraphRAG framework uses a document's inherent structure to inform the creation of a knowledge graph, which then enhances the RAG process [17].
- **Embedding Graph Features in Prompts:** Certain methods directly embed subgraphs or graph features into the LLM's prompt. The G-Retriever framework, for instance, constructs a relevant subgraph and encodes it to provide the LLM with rich, structured context for generating an answer [18].
- **Factual Grounding with KGs:** Knowledge graphs can be used to ground the LLM's output in facts, which helps to reduce the generation of inaccurate information [19].

- **Dynamic Graph Traversal:** More advanced RAG systems can employ an LLM-based agent that dynamically navigates the knowledge graph. This agent treats nodes as document elements and edges as their relationships, allowing it to actively explore the graph to collect the most relevant information for a given query [20].
- **Community Detection in GraphRAG:** A key feature of GraphRAG is its use of community detection algorithms, such as the Louvain [10] and Leiden [2] methods, to partition the graph index into thematically coherent clusters. The Leiden algorithm is a refinement of Louvain that guarantees communities are well-connected and is often faster. An LLM then summarizes these communities and aggregates the summaries hierarchically, enabling a comprehensive understanding of the entire dataset.

Our GraphEDU-RAG framework further incorporates a local querying step to perform deeper inference over entity relationships specifically within the context of educational question answering.

4 GraphEDU-RAG framework

4.1 Overview of the framework

In this work, we propose GraphEDU-RAG, a hierarchical graph-based RAG framework tailored for educational applications. GraphEDU-RAG enhances both retrieval precision and synthesis quality by integrating graph-based knowledge representation and summarization. The pipeline consists of four main steps:

- Step 1: Build a knowledge representation from text documents (based on entities and relationships between them).
- Step 2: Use community detection algorithms to divide this representation into smaller groups.
- Step 3: Each group will be summarized into concise answers.
- Step 4: These concise summaries will be combined into a comprehensive overall answer for the complete question.

4.2 GraphEDU-RAG Approach & Pipeline

4.2.1 Document Chunking

The first important step in the process of building a knowledge graph and answering questions in the Graph RAG method is to split source documents into smaller text chunks for easier processing and analysis in subsequent steps.

The method used to divide documents into short segments is based on the topic change detection described in section 3.1.

After being divided, these text chunks are stored according to separate topics and almost have no topic overlap to improve processing efficiency in subsequent steps.

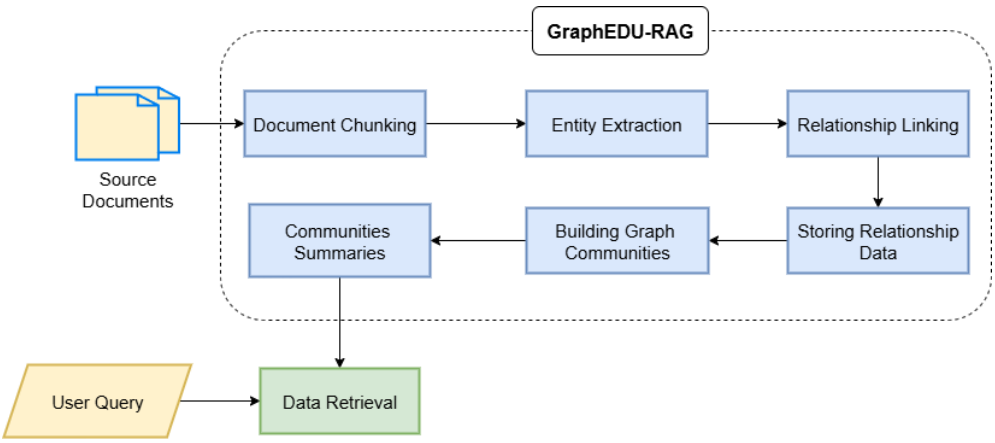

Fig. 3 Illustration of the GraphEDU-RAG pipeline.

4.2.2 Entity Extraction

After completing the chunking process, we will proceed to identify and extract entities and relationships from each text chunk.

Implementation process:

1. Entity Extraction:
   - **Entity**: Important objects mentioned in the text (e.g., people, places, organizations, or other concepts).
   - Entity information includes:
     - **Entity name**: E.g., "Albert Einstein".
     - **Entity type**: E.g., "scientist" or "location".
     - **Entity description**: E.g., "physicist known for the theory of relativity".
2. Relationship Extraction:
   - After identifying entities, we search for relationships between them, for example:
     - **Source**: The entity that initiates the relationship.
     - **Target**: The entity that receives the relationship.
     - **Relationship description**: E.g., "is a teacher of", "was developed by".
3. Additional Information Extraction (covariates):
   - Beyond entities and relationships, additional properties can be extracted, including:
     - **Subject** and **object**.
     - **Information type** and **description**.
     - **Source text span**.
     - **Start dates** and **end dates**.
     - **Embeddings**.

4.2.3 Relationship Linking

Relationship linking is the process of extracting and building relationship networks (relations) from entities, where relationships and related information are combined to create triplets.

For example, given a sentence like: "The doctoral training program only accepts applicants with undergraduate degrees. (The program lasts 3 years from the date of enrollment)."

From this sentence, following the process described above, we obtain:

- **Entities**: Doctoral training program, Undergraduate degree.
- **Relations**: Accepts/Admits.
- **Additional information**: The program lasts 3 years from enrollment date.

4.2.4 Storing Relationship Data

After obtaining triplets representing relationship data in the form of JSON chains, we will use these triplets as the foundation to build a knowledge graph.

To import these hyper-relational knowledge graphs into a Neo4j database standard, with each node being an entity, and each edge representing a relationship between entities. However, to manage and store data, careful consideration of the data above is needed.

De-duplication: Ensuring that each entity is represented uniquely and accurately, avoiding language overlap or duplicate representations leading to a single entity in the real world. Maintaining data consistency and quality in representation. Without De-duplication, knowledge representation will be fragmented with non-uniform data, leading to errors and missing crucial information.

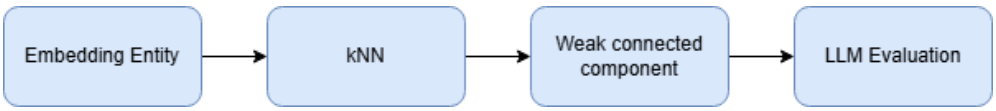

Fig. 4 De-duplication.

De-duplication process:

1. **Embedding Entity** — Begin with all entities in the representation, then add embedding properties containing vector information of the entities.
2. **kNN** — Build a kNN graph, connecting similar entities based on embeddings.
3. **Weak Connected Component** — Identify weakly connected components in the kNN graph, grouping entities that are potentially similar to each other. Further filtering by distance after these components have been determined.
4. **LLM Evaluation** — Using LLM to evaluate these components to decide whether the entities in each component should be merged or not, thereby making the final decision on entity resolution.

After building the storage system, retrieval and representation of knowledge graphs will become more efficient. Cypher query language is used to extract information from these graphs.

4.2.5 Building Graph Communities

Community detection is the process of separating or segmenting vertices in a network or graph into groups or communities based on network connection relationships between them. The goal of community detection is to find structures or organizations with strong relationships within the network, looking for similar groups or organizations with similar functions. Communities in a network can be defined based on different criteria, including relationships between vertices, proximity, or frequency of interactions within the community.

We use community detection algorithms to group related entities and relationships into communities, based on strong connections between them.

The research employs the Leiden algorithm, along with Graphdatascience as the tool used to implement this algorithm.

The Leiden algorithm allows dividing the graph into multiple levels of communities, from low-level (detailed) to high-level (overview).

Leaf-level communities: These are initial communities, typically containing groups of nodes strongly connected to each other.

Higher-level communities: After the low-level communities are merged, the algorithm creates larger communities, with corresponding nodes from the low-level communities being merged.

4.2.6 Communities Summaries

The purpose of this step is to create detailed summaries for each community, providing a comprehensive view of each part of the data and ensuring that these summaries can be used to answer questions or serve as a basis for further processing.

Implementation process:

- Leaf-level communities: For each community at the lowest level (most detailed), all entities, relationships, and related properties are shared with an LLM, which then creates a community summary based on this information.
- Higher-level communities: For larger communities, summaries of child communities are used instead of detailed information about each entity, to ensure the information doesn't exceed token limits.

When aggregating information, entities and relationships with high importance scores (for example, based on connection frequency) are prioritized. Information is added to the summary in order of priority until the model's token limit is reached.

4.2.7 Data Retrieval

Data retrieval in Graph RAG includes two parts: global retrieval and local retrieval.

Global Retrieval:

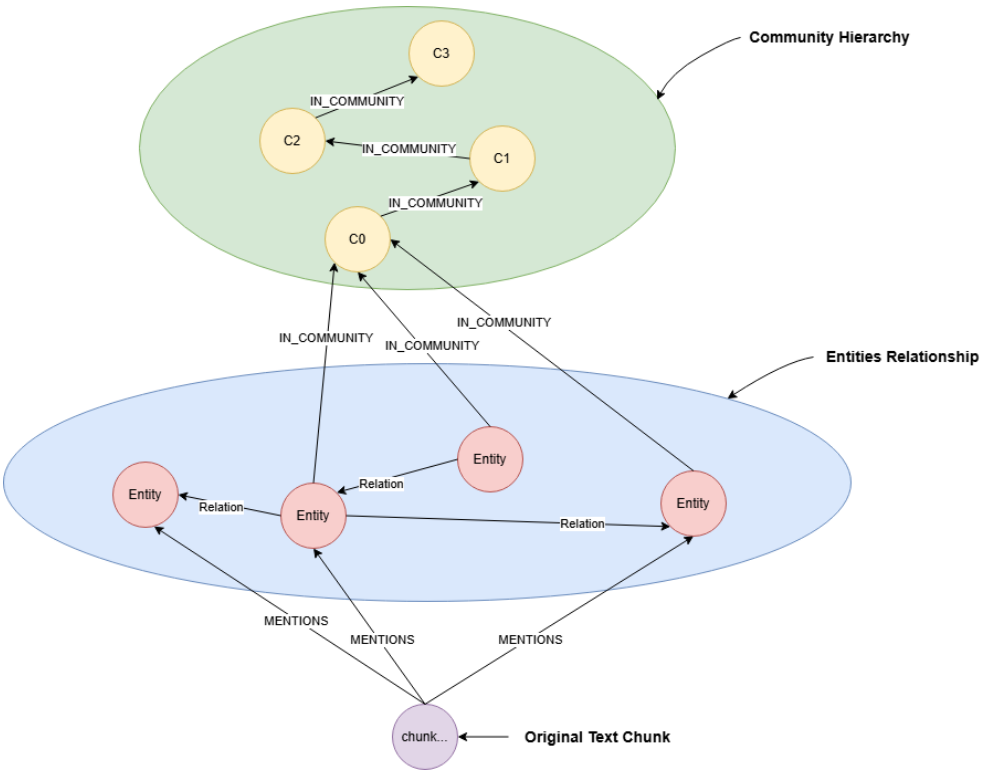

Fig. 5 Illustration of a community built using the Leiden algorithm.

- Used with questions that have a comprehensive nature, requiring an overview of the data, questions that traditional RAG would typically struggle to handle effectively.
- The retrieval work is based on the Communities Summaries previously created:
  - Retrieves the Community Summaries most relevant to the initial question.
  - These Community Summaries are divided into larger chunks with sizes appropriately matched to the context window of the LLM.
  - The chunks are randomly shuffled to distribute information evenly, avoiding situations where all related information is concentrated in one chunk that might be omitted during processing.
  - Using an LLM to synthesize each chunk and proceed with evaluation.
  - The assessments are rated and will be fed into the LLM to generate answers to parts of the question until the token limit is reached.

Local Retrieval

- Suitable for in-depth questions about a specific topic, more appropriate for deeper inquiries about entities and relationships.

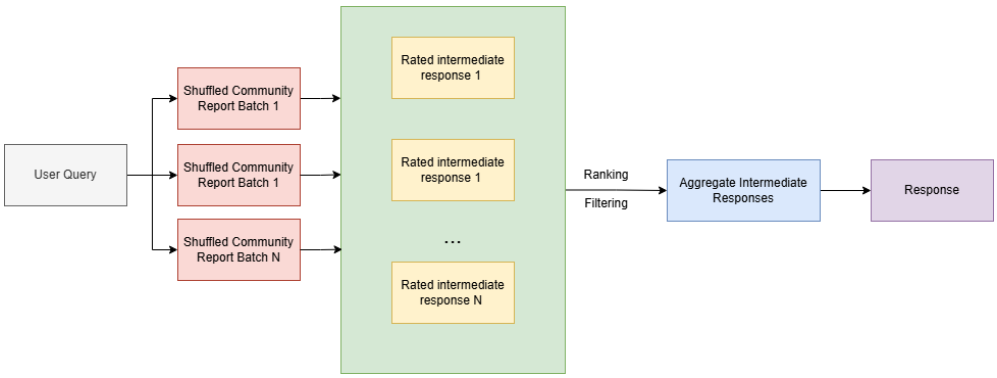

Fig. 6 Illustration of the global retrieval flow.

– Identifies a set of entities from the knowledge graph relevant to the initial query. These entities are nodes in the graph, allowing extraction of additional relevant information to form a complete answer, such as relationships, connected entities, and supplementary information.
– Additionally, it can extract text passages linked to entities (if any) to enrich the answer.
– The results from these retrievals will be used in combination with an LLM to generate the final comprehensive answer.

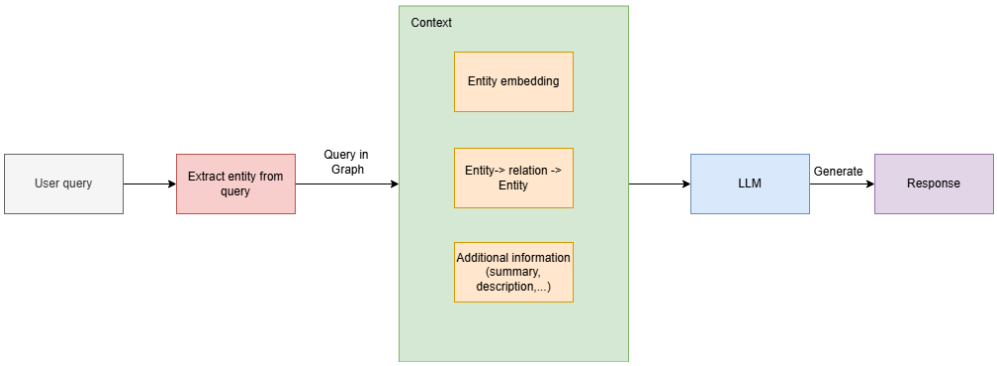

Fig. 7 Illustration of the local retrieval flow.

5 Experimentation

5.1 Experimental Data

The input data source for the Graph RAG pipeline consists of educational documents collected from the official website of Bach Khoa University in Ho Chi Minh City. In

this collection, 25 documents with main content related to regulations and rules of the university were used.

A dataset containing approximately 3000 questions and corresponding expected answers related to the documents above was also constructed to facilitate result evaluation.

5.2 Evaluation Methods

BERTScore [**?**] is an evaluation method that measures semantic similarity between two text segments, based on vector embeddings generated by language models such as BERT, RoBERTa, etc.

Unlike traditional metrics like BLEU, ROUGE (which compare by word or n-gram), BERTScore calculates semantic similarity using contextual embeddings.

The process for calculating BERTScore:

Assuming:

- **Candidate**: The model's predicted response.
- **Reference**: The standard answer (Ground Truth).

Calculation steps:

1. **Tokenize**: Both Candidate and Reference are tokenized into smaller tokens. For example:
   - Candidate tokens: [sinh, viên, cần, 135, tín, chỉ]
   - Reference tokens: [sinh, viên, phải, hoàn, thành, 135, tín, chỉ]
2. **Embedding**: Each token is transformed into a vector embedding using language models like BERT. The vector represents the semantic meaning of the token.
3. **Calculate similarity**: For each token in the Candidate, find the token in the Reference with the highest cosine similarity. Similarly, match each Reference token to a Candidate token.
4. **Calculate Precision, Recall and F1**:
   - **Precision**: The average highest similarity of each token in the Candidate compared to the Reference, with value in the range [0:1].
   - **Recall**: The average highest similarity of each token in the Reference compared to the Candidate, with value in the range [0:1].
   - **F1**: The harmonic mean between Precision and Recall, with value in the range [0:1].

Mathematical formulas Let:

- C be the set of tokens in Candidate,
- R be the set of tokens in Reference,

then:

$$\text{Precision} = \frac{1}{|C|} \sum_{c \in C} \max_{r \in R} \text{cosine\_similarity}(c, r)$$

$$\text{Recall} = \frac{1}{|R|} \sum_{r \in R} \max_{c \in C} \text{cosine\_similarity}(r, c)$$

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Advantages of BERTScore:

- Semantic understanding: No need for exact word matching.
- Language independence: Works well for many different languages.
- Good content detection: Detects added or missing information.

5.2.1 Results

When running experiments on the related dataset, an F1 score of 0.776 was obtained. If a model has a high F1 Score, it means it is producing many correct predictions with good coverage.

Method	Precision	Recall	F1
Graph RAG	0.819	0.739	0.776
RAG	0.813	0.702	0.753
GPT-4o	0.698	0.570	0.628

Table 1 BERTScore Evaluation Results

- With the GPT-4o model, which was not trained on the experimental dataset, the answers were mostly "hallucinated," resulting in a low F1 score of 0.628, which is considered relatively low.
- When using RAG and Graph RAG on the original dataset, the results obtained had much higher accuracy, with Graph RAG achieving the highest results.

6 Discussion

6.1 Summary

This Master's thesis has presented about the model, related research works to propose a method that allows the implementation of the Graph RAG - GraphEDU-RAG pipeline in education to solve challenges when connecting information points, summarizing tasks, extracting knowledge, and providing a comprehensive overview of data, while also implementing and evaluating results according to the proposed method.

The initial results obtained show positive trends, with the proposed pipeline demonstrating the ability to improve answer quality based on knowledge graphs, bringing superior performance compared to RAG in many scenarios.

However, during the research process, there are still many challenges that need to be addressed in the future:

- Requires much more storage space compared to RAG.

- Complex deployment. While RAG only needs to store text chunks and their embeddings, GraphEDU-RAG needs to implement many steps, store knowledge graphs and embeddings. Additionally, entity linking is an important step in the Graph RAG pipeline. However, the process of extracting data to obtain the triplets (triples) still faces many challenges. Models like Transformer or phoBERT are used to support the process, but the results obtained so far are not yet really good. Therefore, tasks in this stage often have to be processed multiple times.
- Answer quality depends on the information extracted from the [data retrieval process]. Therefore, in some scenarios, the database containing knowledge graphs does not carry enough necessary information, it is necessary to supplement data into the database to enrich the context of the answers.
- The method has addressed issues related to information summarization and data extraction compared to baseline RAG, however, retrieval speed for these cases is also an issue that needs attention.

6.2 Future Development Directions

Although it has achieved considerable results, the Graph RAG pipeline can still be improved and further developed to increase efficiency. Below are some potential development directions:

- Improve the capabilities of the entity and relationship extraction module.
- Build structured data linked to standard datasets for specialized terminology, technical terms...
- Create a Graph RAG pipeline for structured data.

By focusing on these development directions, the research can be expanded and further improved. This study has confirmed the great potential of Graph RAG in retrieving information from external sources to improve the efficiency and accuracy of QA systems, promising to open new directions for the field of natural language processing.

The thesis demonstrates that graph-based approaches to retrieval-augmented generation can significantly enhance educational question-answering systems by better capturing the relationships between concepts and providing more contextually relevant answers than traditional methods. While challenges remain in implementation complexity and resource requirements, the improved answer quality and semantic understanding make Graph RAG a promising approach for educational applications.

References

[1] Mao, Y., He, P., Liu, X., Shen, Y., Gao, J., Han, J., Chen, W.: Generation-augmented retrieval for open-domain question answering. In: Zong, C., Xia, F., Li, W., Navigli, R. (eds.) Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pp. 4089–4100. Association for Computational Linguistics, Online (2021). https://doi.org/10.18653/v1/2021.acl-long.316

[2] Traag, V.A., Waltman, L., Eck, N.J.: From Louvain to Leiden: guaranteeing well-connected communities. Scientific Reports **9**(1), 5233 (2019) https://doi.org/10.1038/s41598-019-41695-z

[3] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., *et al.*: Language models are few-shot learners. Advances in Neural Information Processing Systems **33**, 1877–1901 (2020)

[4] Lewis, M., Liu, Y., Goyal, N., Ruder, S., Gromann, D., Moosavi, S.K., Bassil, N., Jernite, Y., *et al.*: Retrieval-augmented generation for knowledge-intensive nlp tasks. In: Proceedings of NeurIPS 2020 (2020)

[5] Salton, G., Buckley, C.: Term-weighting approaches in automatic text retrieval. Information Processing & Management **24**(5), 513–523 (1988) https://doi.org/10.1016/0306-4573(88)90021-0

[6] Robertson, S., Zaragoza, H.: The probabilistic relevance framework: BM25 and beyond. Foundations and Trends in Information Retrieval **3**(4), 333–389 (2009) https://doi.org/10.1561/1500000019

[7] Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., Yih, W.-t.: Dense passage retrieval for open-domain question answering. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 6769–6781 (2020)

[8] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., Zettlemoyer, L.: BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In: Jurafsky, D., Chai, J., Schluter, N., Tetreault, J. (eds.) Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 7871–7880. Association for Computational Linguistics, Online (2020). https://doi.org/10.18653/v1/2020.acl-main.703

[9] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J.: Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of Machine Learning Research **21**(140), 1–67 (2020)

[10] Blondel, V.D., Guillaume, J.-L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of statistical mechanics: theory and experiment **2008**(10), 10008 (2008)

[11] Ram, O., Levine, Y., Dalmedigos, I., Muhlgay, D., Shashua, A., Leyton-Brown, K., Shoham, Y.: In-context retrieval-augmented language models. Transactions of the Association for Computational Linguistics **11**, 1316–1331 (2023) https://doi.org/10.1162/tacl_a_00605

[12] Feng, Z., Feng, X., Zhao, D., Yang, M.: Retrieval-generation synergy augmented large language models, pp. 11661–11665 (2024). https://doi.org/10.1109/ICASSP48485.2024.10448015

[13] Wang, S., Khramtsova, E., Zhuang, S., Zuccon, G.: Feb4rag: Evaluating federated search in the context of retrieval augmented generation, pp. 763–773 (2024). https://doi.org/10.1145/3626772.3657853

[14] Etzioni, O., Cafarella, M., Downey, D., Kok, S., Popescu, A.-M., Shaked, T., Soderland, S., Weld, D.S., Yates, A.: Web-scale information extraction in knowitall: (preliminary results). In: Proceedings of the 13th International Conference on World Wide Web. WWW '04, pp. 100–110. Association for Computing Machinery, New York, NY, USA (2004). https://doi.org/10.1145/988672.988687

[15] Kim, D., Xie, L., Ong, C.S.: Probabilistic knowledge graph construction: Compositional and incremental approaches. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. CIKM '16, pp. 2257–2262. Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2983323.2983677

[16] Melnyk, I., Dognin, P., Das, P.: Knowledge graph generation from text. In: Goldberg, Y., Kozareva, Z., Zhang, Y. (eds.) Findings of the Association for Computational Linguistics: EMNLP 2022, pp. 1610–1622. Association for Computational Linguistics, Abu Dhabi, United Arab Emirates (2022). https://doi.org/10.18653/v1/2022.findings-emnlp.116

[17] Zhu, X., Guo, X., Cao, S., Li, S., Gong, J.: StructuGraphRAG: Structured Document-Informed Knowledge Graphs for Retrieval-Augmented Generation. In: Proceedings of the AAAI Symposium Series, vol. 4, pp. 242–251 (2024). https://doi.org/10.1609/aaaiss.v4i1.31798

[18] He, X., Tian, Y., Sun, Y., Chawla, N.V., Laurent, T., LeCun, Y., Bresson, X., Hooi, B.: G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. In: The Thirty-eighth Annual Conference on Neural Information Processing Systems (2024)

[19] Dehal, R.S., Sharma, M., Rajabi, E.: Knowledge Graphs and Their Reciprocal Relationship with Large Language Models. Machine Learning and Knowledge Extraction **7**(2), 38 (2025) https://doi.org/10.3390/make7020038

[20] Wang, Y., Lipka, N., Rossi, R., Siu, A., Zhang, R., Derr, T.: Knowledge graph prompting for multi-document question answering. Proceedings of the AAAI Conference on Artificial Intelligence **38**, 19206–19214 (2024) https://doi.org/10.1609/aaai.v38i17.29889