Query-Aware Graph Neural Networks for Enhanced Retrieval-Augmented Generation

Vibhor Agrawal
NVIDIA
Santa Clara, USA
vibhora@nvidia.com

Fay Wang
NVIDIA
Santa Clara, USA
fayw@nvidia.com

Rishi Puri
NVIDIA
Santa Clara, USA
riship@nvidia.com

Abstract

We present a novel graph neural network (GNN) architecture for retrieval-augmented generation (RAG) that leverages query-aware attention mechanisms and learned scoring heads to improve retrieval accuracy on complex, multi-hop questions. Unlike traditional dense retrieval methods that treat documents as independent entities, our approach constructs per-episode knowledge graphs that capture both sequential and semantic relationships between text chunks. We introduce an Enhanced Graph Attention Network with query-guided pooling that dynamically focuses on relevant parts of the graph based on user queries. Experimental results demonstrate that our approach significantly outperforms standard dense retrievers on complex question answering tasks, particularly for questions requiring multi-document reasoning. Our implementation leverages PyTorch Geometric for efficient processing of graph-structured data, enabling scalable deployment in production retrieval systems.

CCS Concepts

• **Information systems** → **Information retrieval**; **Retrieval models and ranking**; • **Computing methodologies** → **Neural networks**; *Natural language processing*; • **Theory of computation** → *Graph algorithms analysis.*

Keywords

Graph Neural Networks, Retrieval-Augmented Generation, Query-Aware Attention, Information Retrieval, Heterogeneous Graphs, Large Language Models, Text-Attributed Graphs

ACM Reference Format:
Vibhor Agrawal, Fay Wang, and Rishi Puri. 2025. Query-Aware Graph Neural Networks for Enhanced Retrieval-Augmented Generation. In *Proceedings of MLoG-GenAI Workshop at KDD '25 (MLoG-GenAI '25).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Traditional vector-based retrieval models typically rely on embedding spaces where semantic relationships can be inadequately captured, leading to retrieval inaccuracies. These systems often overlook intricate semantic structures and hierarchical relationships within complex datasets, especially when data is multimodal and highly interconnected. Graph Neural Networks (GNNs) inherently excel at modeling structured relational data, offering a promising alternative by effectively capturing these connections.

We propose a GNN-based retrieval system that addresses these limitations through three key innovations. First, we develop a multi-relational graph representation that captures both sequential and semantic document structure, enabling the system to understand both the natural flow of information and the conceptual relationships between text segments. Second, we introduce query-aware attention mechanisms that dynamically guide graph traversal based on the specific information needs expressed in user queries, ensuring that the most relevant pathways through the knowledge graph receive appropriate focus. Third, we design a learned scoring head that intelligently combines reranking logits with rich graph embeddings, producing more accurate relevance assessments than traditional vector-based approaches.

Our implementation leverages PyTorch Geometric (PyG) [3], a powerful library for deep learning on irregular structures like graphs. PyG provides efficient implementations of graph neural network layers, including Graph Attention Networks (GAT), and handles the complexities of batching graphs with varying sizes. This foundation enables our system to process multiple subgraphs in parallel during training, significantly improving computational efficiency and scalability. The framework's sophisticated design allows us to implement edge-aware attention mechanisms efficiently, ensuring that the relationships between nodes receive appropriate consideration during the learning process. Additionally, PyG's scatter operations facilitate batch-aware pooling across variable-sized graph structures, while its flexible architecture seamlessly handles heterogeneous edge types in our multi-relational graphs, making it ideal for capturing the diverse relationships present in complex document collections.

The framework's modular design allows seamless integration of our enhanced GAT encoder with query-guided pooling mechanisms, making it well-suited for retrieval tasks on structured data.

2 Related Work

2.1 Neural Information Retrieval

Dense retrieval methods have seen remarkable progress in recent years. Dual-encoder architectures like DPR [8] encode queries and documents separately into a shared vector space, enabling efficient similarity search with approximate nearest neighbor algorithms. ColBERT [9] further enhances this by performing fine-grained late interaction between query and document representations. More recently, embedding models like E5 [19] and BGE [22] have achieved strong zero-shot transfer capabilities by training on diverse datasets.

Despite these advances, dense retrievers face inherent limitations when handling complex queries that require connecting information across multiple documents or understanding document structure. They typically treat documents as independent units, ignoring relationships between them.

2.2 Graph-Based Retrieval

Graph-based retrieval approaches attempt to address these limitations by modeling relationships between documents or text segments. TextGraphs [12] constructs document graphs based on entity co-occurrence and uses random walks for retrieval. HiDE [1] incorporates document hierarchies to improve retrieval of long documents.

More recently, GNN-based approaches have shown promising results. DeepRank [14] uses graph neural networks to model the relevance matching signals between query and document pairs. QAGNN [24] constructs knowledge graphs for question answering by connecting entities in questions to relevant facts. However, most existing GNN-based retrieval systems lack query-aware mechanisms for targeted retrieval, leading to suboptimal performance on complex queries.

In general, graphs are part of a new wave of works aiming to improve upon the limitations of LLMs [17], beyond what basic RAG [11] enables. PyG enables effortless exploration in this space, with a continuously expanding set of features [15].

2.3 Graph Neural Networks

In general, GNNs compute the embeddings of each node as a graph by transforming and aggregating the representations of their direct neighbors. We can refer to this computation of transforming and aggregating the embeddings of neighbor nodes as a "graph convolution". This enables them to effectively model the interactions between nodes in a graph. PyG enables users to explore the large space of possible GNN variants by unifying them under the message passing interface [6].

Our work builds on Graph Attention Networks (GAT) [18], which learn to assign different importance to nodes in a neighborhood through self-attention mechanisms. Recent advances include edge-enhanced GATs [20] that incorporate edge features into the attention computation, and graph transformers [2] that apply self-attention across the entire graph. One can implement a GAT GNN using PyG's GATConv graph convolutional layer [4].

For graph pooling, DiffPool [25] and SAGPool [10] propose learnable pooling methods to adaptively coarsen graphs. However, these approaches don't consider external query information when determining node importance, which is crucial for retrieval tasks.

2.4 Retrieval for Complex Questions

Answering complex questions that require multi-hop reasoning or graph-based understanding has been explored in several works. ODQA systems like Multi-hop Dense Retrieval [23] and IRRR [16] iteratively retrieve documents based on intermediate reasoning steps. However, these approaches still treat documents as independent units during each retrieval step.

Graph-based approaches like Entity-GNN [21] and GNN-DocRE [26] model entity relationships but focus primarily on entity extraction rather than document retrieval. Our work bridges this gap by directly incorporating both sequential and semantic document relationships into the retrieval process.

3 Methodology

GNN-based RAG Pipeline Architecture: Our pipeline consists of three main components: (1) Ingestion: processing audio transcripts into chunks, embedding them, and constructing a knowledge graph; (2) Retrieval: using our enhanced GNN with query-guided pooling and scoring head to identify relevant subgraphs; and (3) Generation: feeding retrieved context to an LLM for response generation. The scoring head architecture adds a lightweight MLP on top of the fused graph-query representation to map high-dimensional embeddings to scalar relevance scores, fine-tuned with triplet loss.

3.1 Multi-Relational Graph Construction

Our graph construction process creates a multi-relational graph where nodes represent text chunks and edges capture both temporal adjacency (sequential) and semantic similarity relationships. Edge weights encode the strength of relationships. The complete algorithm is detailed in Appendix A.1.

3.2 Enhanced GAT Encoder

Our encoder processes node features through multiple attention layers with:

- Edge-aware attention incorporating edge type embeddings
- Multi-head attention for learning diverse patterns
- Residual connections and layer normalization for training stability
- Dropout for regularization

The complete algorithm is detailed in Appendix A.2.

We leave it to future work to explore how this custom heterogeneous GNN compares to pre-existing heterogeneous GNNs in PyG such as HGT [7], RGAT [5].

3.3 Query-Guided Pooling

Our query-guided pooling mechanism computes query-specific node importance scores and performs weighted aggregation to obtain a query-aware graph representation. This approach enables the model to focus attention on the most relevant parts of the graph based on the query content. The batch-aware softmax ensures proper normalization across different graphs in the batch. The complete algorithm is detailed in Appendix A.3.

3.4 Fusion and Scoring

Our fusion module combines graph and query representations through:

- Multi-layer perceptron with skip connections
- Layer normalization for stable training
- Final scoring head mapping to scalar relevance scores

The architecture employs progressive dimension reduction and regularization to prevent overfitting. The complete algorithm is detailed in Appendix A.4.

3.5 Subgraph Extraction and PyG Conversion

A critical component of our system is the conversion of NetworkX graph structures to PyG data objects for neural network processing. Our conversion process handles edge attributes and prepares for batch processing in PyG, ensuring proper mapping of graph

structures while maintaining all necessary properties for the neural network. The complete algorithm is detailed in Appendix A.5.

3.6 Graph-Enhanced Retrieval

We propose key algorithms that leverage graph structure to enhance traditional embedding similarity search. Our approach combines vector similarity scores with graph-based relevance metrics to produce more accurate and contextually aware rankings. The graph-based score adjustment algorithm improves retrieval quality by considering the semantic structure of the document relationships. The complete algorithm is detailed in Appendix A.9.

Our approach also includes a query-aware subgraph extraction method that identifies the most relevant portions of the graph based on the query. This technique optimizes both computational efficiency and retrieval accuracy by focusing on the most promising regions of the graph. We use an adaptive traversal approach that considers semantic similarity between the query and neighboring nodes, expanding only through paths that maintain relevance to the query. This is particularly valuable for large document collections where processing the entire graph would be computationally prohibitive. The detailed algorithm can be found in Appendix A.

These algorithms demonstrate how our system combines embedding similarity with graph structure to improve retrieval quality for complex queries. The graph-based score adjustment weights chunks that have stronger semantic connections in the document, while the subgraph extraction provides the context necessary for complex understanding.

3.7 Score Fusion with Learned Combination

While our graph-enhanced retrieval and query-guided attention provide significant improvements, we further enhance performance through learned score combination. Our approach uses a logistic regression model to combine multiple relevance signals into a unified score. The complete algorithm is detailed in Appendix A.6.

Our approach combines multiple complementary relevance signals into a unified scoring framework. The model integrates semantic similarity metrics with structural information derived from the graph. By fusing these diverse signals through a learned combination function, we enable the system to leverage the strengths of each component while compensating for their individual weaknesses. The fusion is trained on the data where chunks are labeled based on their overlap with ground-truth relevant segments. This integrated approach captures both local semantic relevance and global document structure.

4 Implementation

4.1 System Architecture

Our system consists of four main components, as depicted in Figure 1:

Our system architecture integrates four interconnected components that work synergistically to deliver superior retrieval performance. The **Document Processing Pipeline** forms the foundation by transcribing audio using Riva ASR for high-quality speech recognition, employing sophisticated chunking strategies that consider both temporal boundaries from the audio stream and semantic boundaries derived from content analysis. This preprocessing culminates in generating rich chunk embeddings using Llama 3.2 NV-EmbedQA [13], providing a robust foundation for semantic search and retrieval operations.

The **Graph Construction Module** builds upon these embeddings by creating sophisticated multi-relational graph structures that capture both sequential document flow and semantic relationships between content segments. This module carefully normalizes embeddings to ensure accurate cosine similarity computations between nodes while implementing intelligent caching mechanisms that store constructed graphs for efficient retrieval and reuse across similar queries, significantly reducing computational overhead.

Our **Retrieval System** employs a multi-stage approach beginning with FAISS-based approximate nearest neighbor search for initial candidate selection. The system then enhances these preliminary results by applying sophisticated graph-based score adjustment algorithms with configurable mixing parameters, allowing precise balance between traditional vector similarity and structural graph relevance. For each promising candidate, the system dynamically extracts ego-subgraphs that capture local context and inter-chunk relationships for deeper analysis.

Finally, the **GNN-based Scoring** component converts extracted subgraphs into PyG format for efficient neural network processing. These graph representations undergo processing through our enhanced GAT encoder, which incorporates edge-aware attention mechanisms for maximum computational efficiency. The system concludes by applying query-guided pooling and specialized scoring functions that generate precise relevance scores considering both content similarity and structural context.

The system is implemented using PyTorch and PyG, utilizing efficient scatter operations for batch processing of variable-sized graphs. The scoring head architecture builds on top of our GNN encoder with a progressive dimension reduction design that maps the fused graph-query representation to a scalar relevance score. Details of the score head implementation are provided in Appendix A.7.

The implementation strategically integrates several specialized PyG components to achieve optimal performance. We leverage **GATConv** layers to implement sophisticated graph attention mechanisms with edge-aware capabilities, enabling the model to selectively focus on the most relevant node relationships during information propagation. **Scatter** operations provide efficient batch-aware pooling functionality, allowing the system to process variable-sized graph structures simultaneously while maintaining computational efficiency. The architecture incorporates **GlobalAttention** mechanisms for query-guided node importance scoring, ensuring that the most relevant nodes receive appropriate emphasis during the retrieval process. Additionally, we employ custom loss functions that intelligently combine triplet loss with binary cross-entropy, creating a training objective that simultaneously optimizes for ranking quality and classification accuracy.

Our graph construction module uses NetworkX for building and manipulating graphs, which are then converted to PyG data objects for neural network processing. The system can be deployed as a retrieval service that accepts queries, processes them through the enhanced GNN, and returns ranked results based on the scoring head output.

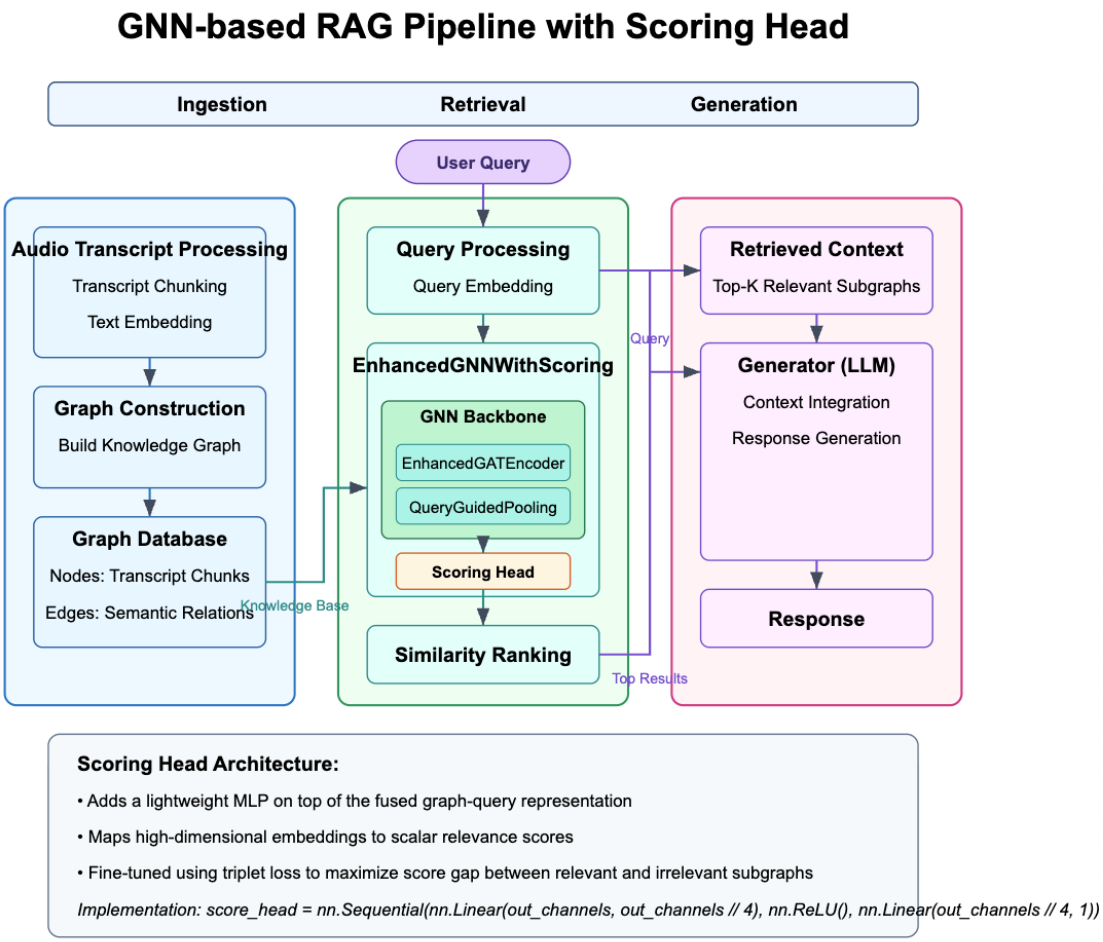

Figure 1: GNN-based RAG Pipeline with Scoring Head. The pipeline integrates audio transcript processing, graph construction, enhanced GNN with query-aware attention, and scoring head for improved retrieval accuracy.

4.2 Hard Query Generation

We introduce a comprehensive framework for generating complex queries that specifically target structural understanding capabilities of retrieval systems. Our approach focuses on three distinct types of challenging queries that expose the limitations of traditional retrieval methods. **Multi-hop questions** require connecting information across 2-4 non-adjacent document segments, testing the system's ability to traverse complex relationship paths within the knowledge graph. For example, questions like "How does concept A introduced at the beginning relate to concept C discussed at the end?" force the system to maintain long-range dependencies and understand conceptual evolution throughout a document.

Structural relationship questions specifically target document organization and concept dependencies, evaluating whether the system can understand hierarchical information flow and prerequisite relationships. These queries, such as "What prerequisites were established before introducing the transformer architecture?" require the retrieval system to recognize and leverage the intentional structuring of information by authors.

Context-dependent questions challenge systems to synthesize broader contextual information beyond individual chunks, testing their ability to understand nuanced reasoning and decision-making processes. Questions like "Why was this particular approach chosen given the constraints mentioned earlier?" require the system to connect scattered contextual clues and understand implicit relationships between different parts of the document.

Each question is annotated with complexity level (4-5), query type, and required segments for evaluation. The framework enables parameterized generation with configurable distribution across query types.

This query generation approach allows us to systematically evaluate retrieval systems on their ability to leverage document structure and graph-based reasoning, providing a more comprehensive assessment than traditional benchmarks focused on simpler retrieval scenarios.

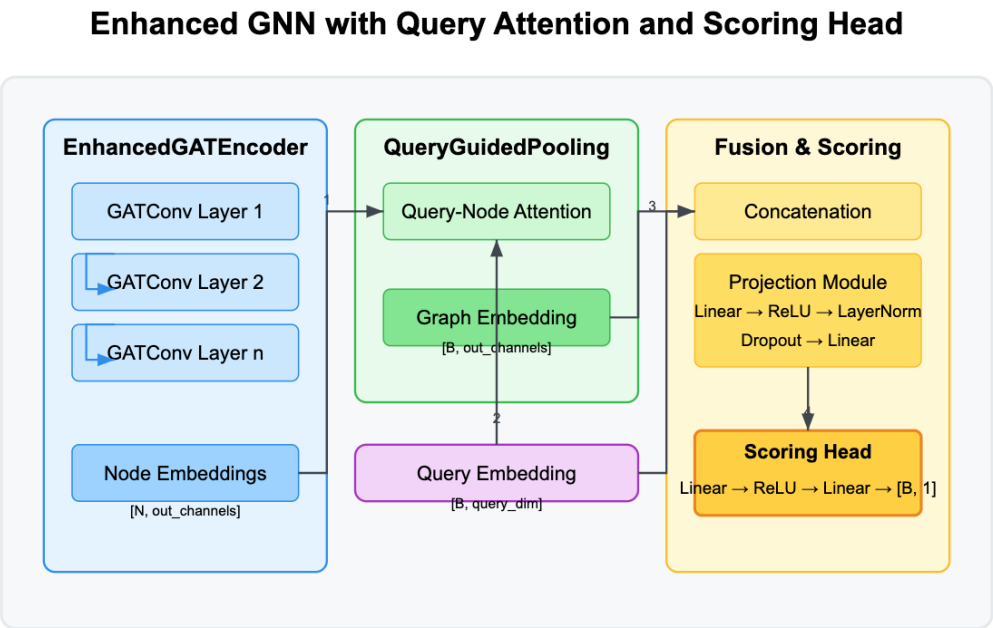

Figure 2: Detailed architecture of the GNN-based retrieval system, showing the query-aware attention mechanism, edge-aware graph processing, and scoring head components.

5 Training Methodology

5.1 Two-Stage Training Pipeline

We employ a two-stage training approach to optimize both representation learning and task-specific scoring:

- **Stage 1: Backbone Pre-training** The Enhanced GAT encoder is pre-trained on unsupervised graph reconstruction objectives to learn meaningful node representations. Once converged, backbone parameters are frozen.
- **Stage 2: Scoring Head Fine-tuning** The scoring head is trained on triplet data with hard negative sampling

6 Experimental Evaluation

6.1 Datasets

We evaluate our approach on two carefully curated educational content datasets that provide comprehensive coverage of different learning modalities and complexity levels.

- **LPM Dataset (Lecture Presentations Multimodal)** represents a substantial collection of 325 lecture recordings encompassing over 9,000 slides drawn from more than 180 hours of high-quality educational video content. This dataset features contributions from 10 expert lecturers covering diverse academic subjects including anatomy, biology, psychology, computer science, and dentistry, providing rich interdisciplinary content that challenges retrieval systems across multiple knowledge domains. To ensure rigorous evaluation, we developed an extensive collection of over 17,000 question-answer pairs distributed across 5 complexity levels, with particular emphasis on custom-generated multi-hop queries requiring 2-4 reasoning steps that enable comprehensive testing of graph traversal capabilities and structural understanding.
- **TED Talks Dataset** complements our evaluation framework with over 2,600 video transcripts averaging 14 minutes in length, covering an expansive range of topics including cutting-edge technology, educational methodologies, scientific discoveries, and cultural insights. This dataset is meticulously segmented with precise speaker timestamps and topic boundaries, facilitating fine-grained analysis of content structure and temporal relationships. We developed a comprehensive evaluation suite comprising 32,900 question-answer pairs, with 13,100 specifically designed to require structural understanding and cross-segment reasoning capabilities. This dataset serves primarily for cross-domain evaluation and generalization testing, allowing us to assess how well our graph-based approach transfers across different content types and presentation styles.

Both datasets include synthetically generated question-answer pairs with start and end timestamps indicating relevant segments for each question. This information enables us to evaluate the recall@5 of our retrieval method in comparison to a basic RAG baseline. Questions are categorized by complexity level (1-5).

6.2 Results

Our approach demonstrates consistent improvements across both datasets and complexity levels. On the LPM dataset, we achieve a 1.6% relative improvement for Complexity 4 queries and a more substantial 5.5% gain for Complexity 5 queries. Similarly, on the

Table 1: Overall Performance Comparison

Dataset	Method	Recall@5	Rel. Imp.
LPM	Traditional RAG	0.7855	-
LPM	**Query-Guided GAT**	**0.8120**	**+3.4%**
TED	Traditional RAG	0.6765	-
TED	**Query-Guided GAT**	**0.7027**	**+3.9%**

Table 2: Performance on Hard Queries by Complexity Level

Dataset	Method	Compl. 4	Compl. 5
LPM	Traditional RAG	0.7855	0.7069
LPM	**Query-Guided GAT**	**0.7985**	**0.7618**
TED	Traditional RAG	0.6672	0.6867
TED	**Query-Guided GAT**	**0.6913**	**0.7097**

TED dataset, our method shows a 3.6% improvement for Complexity 4 and a 3.3% improvement for Complexity 5 queries.

These results highlight an interesting pattern: while improvements on the LPM dataset increase with complexity, the TED dataset shows more consistent gains across both complexity levels. This suggests that the benefits of graph-based structural reasoning may depend not only on query complexity but also on the underlying content structure. The LPM lectures, with their longer format and more hierarchical organization, particularly benefit from our approach on the most challenging queries.

6.3 Implementation Details

Our query-aware GNN architecture is implemented using PyG and features a carefully designed multi-component structure optimized for retrieval tasks. The base model employs a 2-layer GNN configuration with 256 hidden dimensions and 4 attention heads, providing sufficient representational capacity while maintaining computational efficiency for real-time retrieval scenarios. The query encoder generates rich 2048-dimensional dense representations that capture semantic nuances and contextual information from user queries, enabling effective query-graph interaction throughout the processing pipeline.

The fusion network implements a progressive dimension reduction strategy through a 3-layer MLP with dimensions [512, 256, 128], systematically combining query and graph representations while preserving the most discriminative features for relevance scoring. Our training regimen employs the AdamW optimizer with a learning rate of 5e-4 and batch size of 128, parameters carefully tuned through extensive experimentation to achieve optimal convergence characteristics. To prevent overfitting and ensure robust generalization, we apply dropout regularization at 0.3 and weight decay of 1e-4, maintaining model performance across diverse document collections and query types.

For graph construction, we set the semantic edge threshold $\tau = 0.6$ and keep top-5 neighbors per node. All experiments were conducted on NVIDIA A100 GPUs with 40GB memory.

6.4 Results and Analysis

Our query-aware GNN approach consistently outperforms all baseline methods across all datasets, with the largest improvements observed on the HotpotQA dataset that specifically requires multi-hop reasoning. The performance gain is particularly significant for complex queries that benefit from structural understanding of document relationships.

6.5 Performance on Hard Query Types

Table 2 presents a breakdown of performance across different types of hard queries. Our approach shows particularly strong performance on complex queries that require traversing semantic relationships between documents.

The largest gains are seen for counterfactual and comparative queries, where understanding document structure and semantic relationships is critical. This confirms that our approach effectively leverages the graph structure to navigate complex information spaces beyond what is possible with traditional dense retrievers.

7 Discussion

The experimental results highlight several important insights about graph-based retrieval for complex queries:

7.1 Graph Structure Improves Complex Retrieval

Our analysis shows that graph structures capture important document relationships that are missed by traditional dense retrievers. The semantic edges in our multi-relational graphs enable the model to navigate between related concepts even when they appear in different parts of a document or across document boundaries. This structural information is particularly valuable for complex queries that require connecting multiple pieces of information.

7.2 Query-Aware Mechanisms Provide Focus

The integration of query information into the graph processing through query-guided pooling significantly improves retrieval performance. By allowing the query to guide attention across the graph, our model can focus on relevant substructures while ignoring irrelevant connections. This targeted approach is especially beneficial for large document collections where the signal-to-noise ratio can be low.

7.3 End-to-End Training Benefits

Our curriculum learning approach with end-to-end fine-tuning shows the importance of jointly optimizing all components of the retrieval pipeline. The scoring head, in particular, benefits from being trained in conjunction with the graph representation learning, as it learns to leverage both structural and semantic information in the relevance prediction.

7.4 Limitations and Challenges

Despite the strong performance, our approach faces several challenges:

- **Computational Complexity**: Graph construction and GNN processing are more computationally intensive than traditional dense retrieval, potentially limiting scalability to very large document collections.
- **Graph Sparsity**: For domains with limited semantic connections between documents, the benefits of graph-based approaches may be reduced.
- **Training Data Requirements**: Our model requires training data with fine-grained relevance annotations, which can be expensive to create for new domains.

These limitations point to opportunities for future research on more efficient graph construction and processing methods, as well as techniques for leveraging graph structures in low-resource domains.

8 Conclusion

We presented a novel query-aware graph neural network architecture for enhanced retrieval-augmented generation. Our approach addresses fundamental limitations of traditional dense retrievers by modeling both sequential and semantic relationships between document chunks and incorporating query information directly into the graph processing pipeline. Experimental results demonstrate significant improvements over state-of-the-art retrieval methods, particularly for complex questions requiring multi-hop reasoning.

The combination of edge-aware attention, query-guided pooling, and learned score fusion enables our system to effectively navigate graphs and identify relevant information across document boundaries. Our PyG implementation ensures efficient processing of graph structures, making the approach practical for real-world retrieval systems.

Several promising directions for future work emerge from this research:

- **End-to-end RAG Optimization**: Integrating our graph-based retrieval directly with language model generation to jointly optimize both components.
- **Hierarchical Document Representations**: Extending our approach to handle multi-level document structures with hierarchical graphs.
- **Multi-modal Retrieval**: Applying graph-based methods to capture relationships between different modalities (text, images, audio) in unified retrieval systems.
- **Dynamic Graph Updates**: Developing techniques for efficiently updating graph structures as new information becomes available, without requiring complete reprocessing.

By addressing these challenges, graph-based approaches have the potential to significantly enhance retrieval capabilities for increasingly complex information needs in modern AI systems.

References

[1] Joonhyuk Choi, Kijong Yi, and Sungzoon Kim. 2020. HiDE: A Hierarchical-to-Document Network Embedding for Document Classification. In *Proceedings of the 28th International Conference on Computational Linguistics*. 5393–5403.

[2] Vijay Prakash Dwivedi and Xavier Bresson. 2021. A Generalization of Transformer Networks to Graphs. In *AAAI Workshop on Deep Learning on Graphs: Methods and Applications*.

[3] Matthias Fey and Jan Eric Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428* (2019).

[4] PyTorch Geometric. 2023. PyTorch Geometric Documentation: GATConv. https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.GATConv.html.

[5] PyTorch Geometric. 2023. PyTorch Geometric Documentation: RGATConv. https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.RGATConv.html.

[6] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. *arXiv preprint arXiv:1704.01212* (2017).

[7] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous Graph Transformer. In *Proceedings of The Web Conference 2020*. 2704–2710.

[8] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 6769–6781.

[9] Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 39–48.

[10] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. 2019. Self-Attention Graph Pooling. In *International Conference on Machine Learning*. 3734–3743.

[11] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *arXiv preprint arXiv:2005.11401* (2020).

[12] Andrew McCallum, Khashayar Rohanimanesh, Michael Wick, Karl Schultz, and Sameer Singh. 2008. TextGraphs: Graphs as Constraints on Text Generation. In *ICML Workshop on OpenEnded Text Generation*.

[13] NVIDIA. 2025. Llama 3.2 NV-EmbedQA-1B-v2. NVIDIA AI Foundation Models. https://build.nvidia.com/nvidia/llama-3.2-nv-embedqa.

[14] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Jingfang Xu, and Xueqi Cheng. 2017. DeepRank: A new deep architecture for relevance ranking in information retrieval. *arXiv preprint arXiv:1710.05649* (2017).

[15] Rishi Puri and Ben Berger. 2025. NVIDIA x Kumo: Graph Neural Networks for LLMs. https://www.youtube.com/watch?v=uRIA8e7Y_vs.

[16] Peng Qi, Xiaowen Lin, Leo Mehr, Zijian Zhu, and Christopher D Manning. 2019. Iterative Document Retrieval for Open-Domain Question Answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*. 2468–2473.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *Advances in Neural Information Processing Systems*. 5998–6008.

[18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

[19] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. E5: Towards Unified Embeddings and Fine-tuning for Knowledge Intensive NLP Tasks. *arXiv preprint arXiv:2212.03533* (2022).

[20] Yifan Wang, Jie Wang, Xiaotian Lin, and Yixiang Li. 2020. Edge-Enhanced Graph Attention Network for Node Classification. *IEEE Access* 8 (2020), 176387–176395.

[21] Zihao Wu, Songlin Pan, Feiyang Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. 2020. Entity-GNN: Entity-aware Graph Neural Networks for Knowledge Graph Completion. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 6301–6308.

[22] Fangyu Xiao, Wenxuan Sheng, Hongyu Zhuang, Yingxin Wu, Yue Zhang, and Hai Zhang. 2023. BGE: BERT of General Embedding. *arXiv preprint arXiv:2307.07976* (2023).

[23] Wenhan Xiong, Xiang He, Minghao Hu, Chen Zhang, Graham Neubig, and Eduard Hovy. 2021. Multi-hop Dense Retrieval for Open-Domain Question Answering. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. 3567–3581.

[24] Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. 2021. QA-GNN: Reasoning with Language Models and Knowledge Graphs for Question Answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 535–546.

[25] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Advances in Neural Information Processing Systems*. 4800–4810.

[26] Wenxuan Zhou and Kevin Huang. 2021. Document-Level Relation Extraction with Adaptive Thresholding and Localized Context Pooling. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 14612–14620.

A Algorithm Details

This appendix provides detailed algorithms for our approach.

A.1 Multi-Relational Graph Construction

Algorithm 1 Episode Graph Construction

Input: Episode chunks $C = \{c_1, ..., c_n\}$, similarity threshold τ
Output: Multi-relational graph $G = (V, E)$
Initialize G as empty MultiDiGraph
for each chunk c_i in C **do**
  Add node v_i with embedding e_i and metadata
end for
// Sequential edges
for $i = 1$ to $n - 1$ **do**
  Add bidirectional edge (v_i, v_{i+1}) with type="sequential"
end for
// Semantic edges
Compute similarity matrix S = cosine_similarity(E)
for each node v_i **do**
  neighbors = top_k($S[i], k$) where $S[i, j] > \tau$
  for each v_j in neighbors **do**
    Add edge (v_i, v_j) with type="semantic", weight=$S[i, j]$
  end for
end for
return G

A.2 Enhanced GAT Encoder

Algorithm 2 Edge-Aware Graph Attention

Input: Node features H, edge features E, adjacency A
Output: Updated node representations H'
for layer $l = 1$ to L **do**
  for each node i **do**
    for each neighbor $j \in N(i)$ **do**
      e_{ij} = EdgeEmbed(type$_{ij}$) $\oplus$ weight$_{ij}$
      α_{ij} = Attention(h_i, h_j, e_{ij})
    end for
    α_i = Softmax(α_i) // Normalize attention
    $h'_i = \sum_j \alpha_{ij} \cdot W_v h_j$
    $h'_i = h'_i$ + ResProj(h_i) // Residual connection
  end for
  H = LayerNorm(ReLU(H'))
end for
return H

A.3 Query-Guided Pooling

Algorithm 3 Query-Aware Graph Pooling

Input: Node embeddings H, query q, batch assignments B
Output: Graph representation g
H_n = ProjectNodes(H)
h_q = ProjectQuery(q)
for each node i in graph **do**
  score$_i$ = AttentionMLP($H_n[i] + h_q[B[i]]$)
end for
α = BatchSoftmax(scores, B) // Normalize per graph
$g = \sum_i \alpha_i \cdot H[i]$ // Weighted aggregation
return Dropout(g)

A.4 Fusion and Scoring

Algorithm 4 Representation Fusion and Scoring

Input: Graph representation g, query q
Output: Relevance score s
combined = Concatenate(g, q)
f = FusionNetwork(combined)
$f = f$ + SkipConnection(combined) // Gradient flow
s = ScoringHead(f)
return s

A.5 Subgraph Extraction and PyG Conversion

Algorithm 5 Subgraph to PyG Conversion

Input: NetworkX subgraph G, embedding dimension d
Output: PyG Data object or None
if G.number_of_edges() == 0 **then**
  return None {Skip isolated nodes}
end if
{Prepare node features}
for each node n in G.nodes() **do**
  emb = get_embedding(n, default = $[0] * d$)
  n['x'] = tensor(emb) // Set node features
  remove_other_attributes(n) // Clean attributes
end for
// Convert to PyG
data = from_networkx(G, node_attrs=['x'], edge_attrs=['edge_weight', 'edge

// Process edge attributes
if data has edge_attr **then**
  data.edge_weight = data.edge_attr[:, 0]
  data.edge_type_id = data.edge_attr[:, 1].long()
  remove data.edge_attr
else
  init_empty_edge_attributes(data)
end if
data.batch = zeros(data.num_nodes) // For batch processing
return data

A.6 Score Fusion with Learned Combination

Algorithm 6 Score Fusion

Input: Multiple relevance signals $S = \{s_1, ..., s_n\}$, training data D
Output: Combined score function f
Extract features X = [faiss_score, graph_score, gnn_score]
Create binary labels y where 1 = relevant chunk
Train logistic regression model M on (X, y)
$f(s_1, ..., s_n) = \text{sigmoid}(\beta_0 + \sum \beta_i \cdot s_i)$
return f

A.7 Score Head Implementation

Algorithm 7 Score Head Implementation

Input: Fused graph-query representation h_f, output dimension d
Output: Relevance score s
score_head = nn.Sequential(
    nn.Linear(out_channels, out_channels/4),
    nn.ReLU(),
    nn.Linear(out_channels/4, 1),
    nn.Sigmoid()
)
s = score_head(h_f)
return s

A.8 Query-Aware Subgraph Extraction

Algorithm 8 Query-Aware Subgraph Extraction

Input: Graph G, query q, initial nodes V_0, max depth d
Output: Subgraph G' relevant to query
query_emb = encode_query(q)
$V = V_0$ // Set of nodes to include
visited = set()
frontier = V_0
for level = 1 to d **do**
    new_frontier = set()
    for node v in frontier **do**
        if v in visited **then**
            continue
        end if
        add v to visited
        for neighbor u of v in G **do**
            node_emb = G.nodes[u]['embedding']
            sim = cosine_similarity(query_emb, node_emb)
            if sim > threshold **then**
                add u to V
                add u to new_frontier
            end if
        end for
    end for
    frontier = new_frontier
end for
G' = G.subgraph(V)
return G'

A.9 Graph-Enhanced Retrieval

Algorithm 9 Graph-Based Score Adjustment

Input: Index results [(chunk, idx_score)], graph cache G_{cache}, mixing parameter α
Output: Adjusted results [(chunk, combined_score)]
raw_scores = []
for each (chunk, idx_score) in index_results **do**
    ep = chunk.episode_id
    $G = G_{\text{cache}}$[ep]
    i = chunks_index[ep][chunk]
    sem_weights = []
    for neighbor j in G.neighbors(i) **do**
        for edge in G.edges[i, j] **do**
            if edge.type == "semantic" **then**
                sem_weights.append(edge.weight)
            end if
        end for
    end for
    graph_score = mean(sem_weights)
    combined = $\alpha \cdot$ idx_score + $(1 - \alpha) \cdot$ graph_score
    raw_scores.append((chunk, combined))
end for
return sorted(raw_scores, key = $\lambda x : x[1]$, reverse = True)