GNN-based Retriever for Knowledge-Grounded LLMs

Joseph Yoo

Abstract

Retrieval-Augmented Generation (RAG) over Knowledge Graphs (KGs) enhances Large Language Models (LLMs) by grounding generation in structured facts. However, frameworks such as SubgraphRAG rely on simple MLP retrievers that score triples independently, ignoring the local connectivity that often characterizes valid reasoning paths. We extend the SubgraphRAG retriever with a Graph Neural Network (GNN) encoder that applies GraphSAGE message passing over batch-induced subgraphs to refine entity embeddings before scoring candidate triples. On the WebQSP benchmark, a single GNN layer ($L = 1$) consistently improves retrieval quality, increasing Answer Recall by 2.0 points at $K = 50$ and 1.2 points at $K = 100$ compared to the MLP baseline, with minimal computational overhead. These gains in structural recall, however, do not yield clear improvements in downstream question answering metrics (e.g., Hit@1, Exact Match), suggesting a reasoning bottleneck where the LLM fails to fully exploit the richer context. Increasing the GNN depth to $L = 2$ provides no additional benefit and introduces substantial variability in training convergence and inference latency, consistent with the sparsity of batch-induced graphs. Overall, our results identify shallow message passing as a cost-effective way to improve retrieval coverage, while indicating that further end-to-end gains will require supervision that better aligns retriever behavior with LLM reasoning preferences. The codebase is available here.

1 Introduction

Retrieval-Augmented Generation (RAG) has emerged as a standard solution to the limitations of Large Language Models (LLMs), enhancing generation by conditioning on relevant external evidence. While most RAG research focuses on unstructured text, many real-world applications rely on structured Knowledge Graphs (KGs), where facts are represented as explicit entities and relationships. This has motivated the development of *GraphRAG* frameworks that retrieve subgraph structures directly from a KG to support multi-hop reasoning and factual grounding (Zhang et al., 2025).

In knowledge-based GraphRAG, the retrieval task is distinct from document search: the system must identify a compact subgraph connecting the query concepts to the correct answer. A prominent recent framework, SubgraphRAG (Li et al., 2025), addresses this by training a lightweight Multi-Layer Perceptron (MLP) retriever to score individual KG triples based on their relevance to the question. The top-ranked triples are assembled into a question-specific subgraph, which is then linearized and fed to an LLM reasoner. SubgraphRAG established that a simple MLP augmented with distance-based features could achieve strong performance on benchmarks like WebQSP, proving that effective graph retrieval does not necessarily require heavy graph encoders.

However, a key limitation of the SubgraphRAG retriever is its structure-agnostic design: it scores each candidate triple independently, conditioned only on local features. This approach ignores the rich local connectivity of the knowledge graph during the retrieval stage itself. Intuitively, valid reasoning chains correspond to connected paths; a triple that bridges two other relevant triples is likely more valuable than an isolated one. While Graph Neural Networks (GNNs) are widely proposed to capture such structural dependencies (Zhang et al., 2025), there is limited empirical evidence systematically isolating their benefit within a controlled KGQA pipeline.

This work investigates the following question: *Does adding GNN-based message passing to the SubgraphRAG retriever actually improve retrieval and end-to-end QA compared to the simple MLP baseline?* We replicate the SubgraphRAG pipeline on WebQSP (Yih et al., 2016) and extend the re-

triever with a GraphSAGE-based encoder that operates on batch-induced subgraphs of candidate entities. By varying the GNN depth ($L \in \{0, 1, 2\}$) while keeping the rest of the pipeline fixed, we perform a controlled ablation to measure the impact of local structural awareness.

Our contributions are as follows:

- We design a GNN-enhanced retriever that integrates GraphSAGE message passing over batch-induced subgraphs into the SubgraphRAG framework, maintaining efficiency while enabling local structural refinement.
- We perform a systematic evaluation on WebQSP, comparing the baseline MLP ($L = 0$) against 1-layer and 2-layer GNN variants across multiple retrieval budgets ($K \in \{50, 100, 200, 400\}$).
- We show that a single GNN layer ($L = 1$) consistently improves retrieval recall at tight budgets (e.g., $+2.0$ points Answer Recall at $K = 50$) with modest cost, but these gains do not yield clear improvements in downstream QA metrics, indicating a retrieval–reasoning mismatch. Furthermore, increasing depth to $L = 2$ offers no further gains and leads to unstable training and inference, identifying shallow message passing as the optimal configuration in this setting.

2 Related Work

2.1 Knowledge-Base Question Answering (KBQA)

Question answering over Knowledge Graphs (KGs) has evolved from semantic parsing approaches, which map natural language to logical forms, to embedding-based and generation-based methods. Standard benchmarks such as WebQSP (Yih et al., 2016) and ComplexWebQuestions (CWQ) (Talmor and Berant, 2018) were introduced to test multi-hop reasoning capabilities over large-scale KGs like Freebase. In this work, we focus on WebQSP as a representative testbed for reasoning over structured data. While earlier systems relied on complex engineered parsers, recent approaches increasingly leverage Large Language Models (LLMs) to reason directly over retrieved graph context, framing KBQA as a specialized instance of Retrieval-Augmented Generation.

2.2 Graph Retrieval-Augmented Generation (GraphRAG)

As categorized in recent surveys (Zhang et al., 2025), GraphRAG frameworks generally fall into two paradigms: *index-based*, where graphs are constructed from unstructured text to support document retrieval, and *knowledge-based*, where the system retrieves facts from an existing curated KG. Our work operates in the latter domain. Within knowledge-based GraphRAG, the retrieval mechanism is critical. State-of-the-art frameworks like SubgraphRAG (Li et al., 2025) employ lightweight retrievers—typically a text encoder plus a small MLP with distance-encoding features—to score triples or paths. Li et al. (Li et al., 2025) demonstrated that this simple architecture could achieve competitive performance without heavy graph encoders, leaving the potential benefits of more complex GNN-based retrievers as an open question.

2.3 GNNs for KG Reasoning

In the context of KGQA, models like QA-GNN use message passing to reason jointly over the question and the graph. However, these models typically function as *reasoners* (predicting the answer entity directly) rather than as *retrievers* in a RAG pipeline (selecting context for an LLM). While GNNs are theoretically appealing for capturing structural dependencies during retrieval, they introduce computational overhead and optimization challenges. In contrast to prior work that often proposes entirely new architectures, we embed GNN-based message passing into the SubgraphRAG *retriever* while keeping the dataset, candidate generation, and LLM reasoner fixed. This allows us to systematically isolate the contribution of local message passing and provide a controlled comparison against the strong MLP baseline established by previous work.

3 Methodology

3.1 Problem Setup and Notation

We consider question answering over a knowledge graph (KG). Let the KG be $G = (V, E)$, where V is the set of entities and E is the set of directed edges. Each edge corresponds to a triple $t_i = (h_i, r_i, a_i)$ with head entity $h_i \in V$, relation r_i from a relation set $\mathcal{R}$, and tail entity $a_i \in V$. The input to the system is a natural-language question q together with the KG G.

A pre-trained text encoder maps surface forms to dense embeddings: $e_q \in \mathbb{R}^d$ for the question, $e_v \in \mathbb{R}^d$ for entities ($v \in V$), and $e_r \in \mathbb{R}^d$ for relations ($r \in \mathcal{R}$). Additionally, for each triple t_i, we compute a structural feature vector $f_i^{\text{DDE}} \in \mathbb{R}^m$ (e.g., Distance-to-Description-Entity).

The retriever learns a scoring function $s_\theta(q, t_i) \in \mathbb{R}$, parameterized by θ. We introduce a hyperparameter $L \in \{0, 1, 2\}$ denoting the depth of GNN message passing; $L = 0$ corresponds to a baseline MLP, while $L \geq 1$ incorporates graph-based refinement.

3.2 SubgraphRAG Pipeline Overview

Figure 1 illustrates our SubgraphRAG-style KGQA pipeline, which consists of three stages. First, given a question q and a pool of candidate triples $\{t_i\}$, the retriever computes relevance scores $s_\theta(q, t_i)$ and ranks all candidates accordingly. Second, we induce a question-specific subgraph $G_K(q)$ by selecting the top-ranked triples and including their constituent entities as nodes and the selected triples as directed edges. Third, this subgraph is linearized into text and provided, together with the question, to a large language model that acts as a reasoner and produces an answer, which is then evaluated at the entity level. Across all experiments, the subgraph construction and reasoning stages are kept fixed; our contribution is entirely in the design of the retriever in Stage 1, where we vary the amount of graph message passing via the GNN depth parameter $L \in \{0, 1, 2\}$.

3.3 Baseline Retriever: MLP over Text and DDE (L = 0)

In the $L = 0$ configuration, we use a purely feed-forward retriever without any graph message passing. For each candidate triple $t_i = (h_i, r_i, a_i)$, we form a feature vector by concatenating the question, entity, relation, and DDE features defined in Section 3.1:

$$x_i = [\, e_q \,;\, e_{h_i} \,;\, e_{r_i} \,;\, e_{a_i} \,;\, f_i^{\text{DDE}} \,].$$

We project x_i to a scalar relevance score $s_\theta(q, t_i)$ using a multi-layer perceptron (MLP) with ReLU activations and dropout. This baseline corresponds to the case where no GNN layers are applied, i.e., $h_{\text{final}} = h_{\text{initial}}$, and thus serves as the $L = 0$ reference in our layer ablations.

3.4 GNN-Based Retriever Extension

3.4.1 Motivation

The baseline retriever scores triples independently, ignoring the local connectivity structure among candidate triples. We inject a GNN over the candidate-induced entity graph so that triples sharing entities can exchange information, refining entity embeddings based on the topology observed in the current batch.

3.4.2 Batch-Induced Graph Construction

Let $\mathcal{B}$ denote the set of candidate triples in the current minibatch, where each triple is $t_i = (h_i, r_i, a_i)$. We define the node set of the batch-induced graph as

$$V_{\mathcal{B}} = \{h_i : t_i \in \mathcal{B}\} \cup \{a_i : t_i \in \mathcal{B}\}.$$

Each node $v \in V_{\mathcal{B}}$ is initialized with an entity embedding $h_{\text{initial}}(v) \in \mathbb{R}^d$, given by the text encoder embedding e_v (optionally concatenated with e_q). The edge set consists of directed edges induced by the candidate triples: for each $t_i = (h_i, r_i, a_i) \in \mathcal{B}$, we add a directed edge from h_i to a_i. Message passing is performed only on this batch-induced graph.

3.4.3 GraphSAGE Message Passing

Let $h_v^0 = h_{\text{initial}}(v)$ be the input representation of node v. For each GNN layer ℓ and node v, we compute a mean-pooled neighbor representation

$$\text{AGG}_\ell(v) = \text{MEAN}\{h_u^\ell : u \in \mathcal{N}(v)\},$$

and apply an additive GraphSAGE update

$$h_v^{\ell+1} = W_1^{(\ell)} h_v^\ell + W_2^{(\ell)} \text{AGG}_\ell(v) + b^{(\ell)},$$

where $W_1^{(\ell)}$ and $W_2^{(\ell)}$ are learnable weight matrices and $b^{(\ell)}$ is a bias. Between layers, we apply a nonlinearity and dropout, which we write compactly as $h_v^{\ell+1} \leftarrow \phi(h_v^{\ell+1})$ with $\phi(\cdot)$ denoting ReLU followed by dropout, except that the final GNN layer omits the ReLU. Let $x_{\text{out}}(v)$ denote the output of the last GNN layer for node v. We apply a residual connection and L2 normalization once at the end of the stack:

$$h_{\text{res}}(v) = h_{\text{initial}}(v) + x_{\text{out}}(v) \tag{1}$$

$$h_{\text{final}}(v) = \frac{h_{\text{res}}(v)}{\|h_{\text{res}}(v)\|_2} \tag{2}$$

We experiment with $L \in \{0, 1, 2\}$ GNN layers. For $L = 0$, we apply L2 normalization directly to $h_{\text{initial}}(v)$. For $L \geq 1$, we apply L layers of the update above to obtain $h_{\text{final}}(v)$.

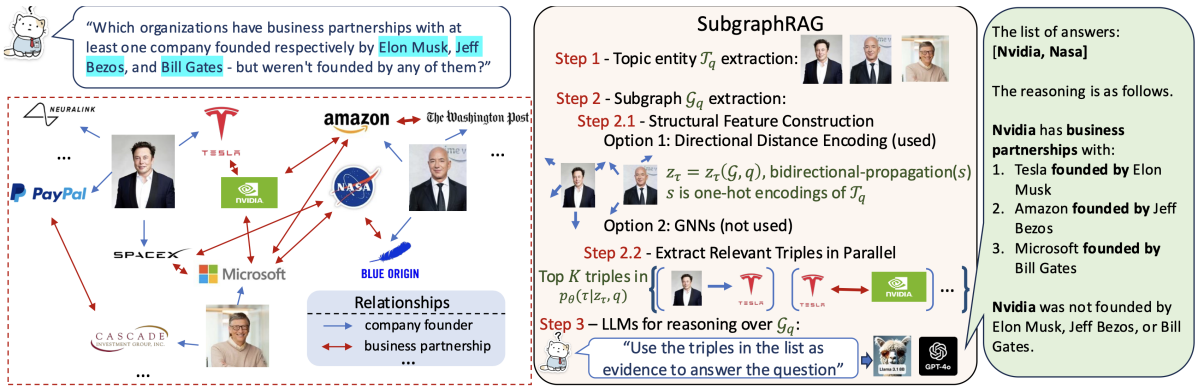

Figure 1: SubgraphRAG Pipeline

3.4.4 Triple Scoring and DDE Integration

Given final node embeddings $h_{\text{final}}(v)$, we construct a semantic representation for each triple $t_i = (h_i, r_i, a_i)$ as

$$g_i = \left[h_{\text{final}}(h_i) \,;\, h_{\text{final}}(a_i) \,;\, e_{r_i} \,;\, e_q \right].$$

We then concatenate the DDE features f_i^{DDE} to obtain the final feature vector

$$z_i = \left[g_i \,;\, f_i^{\text{DDE}} \right],$$

and compute the retrieval score using the MLP head $s_\theta(q, t_i) = \text{MLP}(z_i)$ shared with the $L = 0$ baseline.

3.5 Training Objective and Optimization

All retriever configurations share the same supervision and optimization setup. We derive proxy labels from KG shortest paths: for each question, triples that lie on at least one shortest path between a topic entity and an answer entity are marked as positives ($y = 1$), while other candidate triples for that question are treated as negatives ($y = 0$). For each minibatch, we include all available positives and sample a fixed number of negatives per positive from the candidate pool, and we minimize the binary cross-entropy loss over the resulting batch.

A key consequence of our design is that the minibatch also defines the graph on which message passing occurs when $L > 0$. Crucially, the batch size controls the density and connectivity of this induced graph: larger batches yield richer neighborhoods and more opportunities for message passing, but are limited by GPU memory; smaller batches produce sparser graphs, which can attenuate the benefits of deeper GNNs. In our experiments, we fix the batch size across all L to avoid confounding this effect.

We optimize all models using AdamW with early stopping based on validation answer recall at $K = 100$, keeping all other training hyperparameters fixed across $L \in \{0, 1, 2\}$.

3.6 Inference Configuration and RAG Integration

For inference, we rank candidate triples for each question by $s_\theta(q, t_i)$ and select the top-K triples with $K \in \{50, 100, 200, 400\}$. These triples are converted into the question-specific subgraph $G_K(q)$ using the construction described in Section 3.2 and then passed, together with the question, to a fixed prompt template for the LLM reasoner. We use Llama 3.1 8B Instruct served via vLLM for all experiments, with a shared decoding configuration (e.g., temperature $= 0$, fixed max tokens and top-p) across all values of L and all K, ensuring that differences in performance can be attributed solely to changes in the retriever.

3.7 Layer-Ablation Configurations

To isolate the effect of message passing, we evaluate three retriever configurations: the baseline MLP ($L = 0$), a single-layer GNN ($L = 1$), and a two-layer GNN ($L = 2$). We compare these variants under identical training conditions and evaluate them across retrieval budgets $K \in \{50, 100, 200, 400\}$, ensuring that any performance differences are attributable solely to the GNN depth.

4 Experiments

4.1 Experimental Setup

We evaluate on the WebQSP KGQA benchmark using the standard train/validation/test split and candidate-generation pipeline released with SubgraphRAG (Li et al., 2025). For each question, this

pipeline produces a pool of KG triples that our retriever is allowed to rank. We consider the three retriever configurations defined in Section 3: the baseline MLP ($L = 0$) and the two GNN-based extensions ($L = 1, 2$). For each L, we run training with multiple random seeds and select the checkpoint with the best validation answer recall at $K = 100$. All experiments are conducted on NVIDIA H200 GPUs in a shared-cluster environment.

At test time, we report retrieval quality in terms of answer recall, shortest-path triple recall, and GPT-based triple recall at $K \in \{50, 100, 200, 400\}$. To assess downstream QA, we plug each retriever into the fixed SubgraphRAG reasoning pipeline with Llama 3.1 8B Instruct served via vLLM. The language model, prompt template, and decoding hyperparameters are held fixed across all retriever variants. We measure QA performance using entity-level Hit@1, Macro F1, Micro F1, Exact Match, Hal Score, and the proportion of "no answer" outputs.

4.2 Retriever Performance

Table 1 summarizes training loss and validation recall for the three retriever variants. Training confirms that adding message passing ($L \geq 1$) reduces loss relative to the MLP baseline, but validation metrics peak at $L = 1$. In particular, $L = 1$ achieves the best validation answer recall@100, while $L = 2$ yields diminishing returns: its validation scores are close to or slightly below $L = 1$, and in some cases slightly regress toward the baseline. This pattern suggests that a single GraphSAGE layer is sufficient to capture most of the useful local structure under our batching regime.

On the test set (Table 2), $L = 1$ consistently outperforms the baseline across retrieval budgets. The gains are most pronounced at tight budgets: at $K = 50$, $L = 1$ improves answer recall by roughly 2.0 points and shortest-path triple recall by about 0.5 points over $L = 0$, with effect sizes well above one standard deviation. As K increases, performance saturates and all models behave similarly by $K = 400$. Interestingly, while the GNN retrievers improve structural coverage, they slightly degrade GPT-based triple recall at high K: at $K = 400$, $L = 1$ lags the baseline by about 0.8 points and $L = 2$ by about 1.0 point. This suggests a trade-off between prioritizing triples that complete shortest paths in the KG and triples that align with an LLM's semantic notion of relevance.

L	Train loss	Val ans@100	Val triple@100
0	0.0059 ± 0.0034	93.4 ± 1.1	87.9 ± 1.4
1	0.0049 ± 0.0004	94.5 ± 0.5	88.4 ± 0.3
2	0.0050 ± 0.0006	94.3 ± 0.8	87.5 ± 0.5

Table 1: Training and validation metrics for the three retriever variants. Val ans@100 and Val triple@100 are answer recall and shortest-path triple recall@100 on the validation set (percent, mean $\pm$ std over runs).

4.3 Downstream Reasoning Performance

We next evaluate whether the improvements in retrieval recall translate to better end-to-end question answering. Table 3 reports QA metrics for Llama-3.1-8B-Instruct using the subgraphs retrieved at a fixed budget of $K = 100$.

Despite the 1.2 percentage point improvement in retrieval recall provided by the $L = 1$ model at this budget (95.2% vs. 94.0%), downstream QA metrics remain largely static. The $L = 1$ model achieves an Exact Match (EM) score of 50.0%, nominally higher than the baseline's 49.3%, but this difference is small relative to the variance across seeds ($\sigma \approx 1.4$ for the baseline). Similarly, Hit@1 scores are effectively tied ($\approx 81\%$), and we observed similar stagnant trends for Micro F1 and Hallucination Scores (omitted for brevity).

This result highlights a "reasoning bottleneck" in the pipeline: while the GNN retriever places the correct answer entity in the context window more frequently, the LLM reasoner is not always able to leverage this additional evidence to produce the correct string. This may be due to the trade-off observed in Section 4.2, where GNNs slightly degrade the retrieval of triples preferred by GPT-based scorers, potentially omitting semantic cues the LLM relies on. Notably, however, the $L = 1$ model maintains this parity with the baseline while being more efficient than $L = 2$, which introduces significant instability.

4.4 Efficiency and Stability

Finally, we analyze the computational cost of introducing GNN layers. Table 4 compares the wall-clock time for retriever training and end-to-end reasoning inference.

Training cost. The single-layer GNN ($L = 1$) introduces a modest training overhead of approximately 12% compared to the baseline (1031s vs. 922s), which is negligible given the total pipeline duration. Surprisingly, the $L = 2$ model recorded

L	$K = 50$			$K = 100$			$K = 200$			$K = 400$		
	Ans	S-Path	GPT	Ans	S-Path	GPT	Ans	S-Path	GPT	Ans	S-Path	GPT
0	89.6 ± 1.1	82.9 ± 1.0	80.4 ± 1.2	94.0 ± 0.7	88.6 ± 0.9	86.3 ± 1.1	96.8 ± 0.5	93.4 ± 0.8	91.5 ± 1.3	98.6 ± 0.2	96.7 ± 0.6	95.9 ± 0.9
1	91.6 ± 0.3	83.4 ± 0.3	80.9 ± 0.5	95.2 ± 0.3	89.1 ± 0.4	86.2 ± 0.1	97.4 ± 0.2	93.9 ± 0.2	91.2 ± 0.3	98.7 ± 0.1	97.0 ± 0.1	95.1 ± 0.2
2	90.6 ± 0.7	82.4 ± 0.5	79.8 ± 0.8	94.6 ± 0.6	88.4 ± 0.3	85.7 ± 0.4	97.1 ± 0.3	93.3 ± 0.3	90.7 ± 0.4	98.6 ± 0.1	96.8 ± 0.3	94.9 ± 0.3

Table 2: Test retrieval performance as a function of K for each retriever depth L (percent, mean $\pm$ std over runs). "Ans" is answer recall@K, "S-Path" is shortest-path triple recall@K, and "GPT" is GPT-based triple recall@K.

Model	Hit@1	Exact Match	Macro F1
$L = 0$	81.3 ± 0.8	49.3 ± 1.4	69.8 ± 0.7
$L = 1$	81.0 ± 0.5	$\mathbf{50.0} \pm 0.5$	70.0 ± 0.5
$L = 2$	81.3 ± 0.3	49.2 ± 0.7	$\mathbf{70.0} \pm 0.6$

Table 3: End-to-End QA Performance with $K = 100$ (Mean $\pm$ Std). Improved retrieval recall does not lead to significant gains in reasoning accuracy, suggesting an LLM reasoning bottleneck.

a lower mean training time (957s) than $L = 1$. Inspection of the training logs suggests this is due to early stopping: the $L = 2$ model often failed to improve validation recall after the initial epochs, triggering the patience counter earlier than the more robust $L = 1$ configuration.

Inference stability. Inference latency is dominated by the LLM reasoner, which takes $\approx$ 42 minutes per run regardless of the retriever. Notably, the $L = 1$ configuration exhibits the highest stability (lowest standard deviation) and a mean runtime (2471s) comparable to the baseline, indicating reliable performance. In stark contrast, the $L = 2$ configuration shows extreme instability, with a standard deviation of nearly 1200 seconds. This variance points to significant system instability or computational bottlenecks introduced by the deeper message passing on the batch-induced graphs, making $L = 2$ less suitable for production deployment.

Model	**Retriever Training (s)**	**Reasoner Inference (s)**
$L = 0$	922 ± 126	2599 ± 94
$L = 1$	1031 ± 155	$\mathbf{2471 \pm 38}$
$L = 2$	957 ± 170	2582 ± 1198

Table 4: Efficiency benchmarks (Mean $\pm$ Std). $L = 1$ adds minimal training cost and maintains highly stable inference, whereas $L = 2$ exhibits severe instability during inference.

5 Discussion

5.1 The Reasoning Bottleneck

Our experiments reveal a stark divergence between upstream retrieval metrics and downstream question answering performance. While the introduction of a single GNN layer ($L = 1$) significantly improves structural retrieval, boosting answer recall by $+2.0$ points at $K = 50$ and $+1.2$ points at $K = 100$, these gains do not translate into statistically significant improvements in end-to-end QA metrics. Hit@1, Exact Match, and F1 scores remain within one standard deviation of the baseline, indicating that the pipeline is currently limited by the reasoning capabilities of the LLM or the nature of the supervision, rather than by the raw recall of the retriever.

A primary factor driving this plateau is the high baseline performance at the operating point of $K = 100$. The MLP baseline already achieves an answer recall of 94.0%; increasing this to 95.2% with $L = 1$ converts only a small fraction of "answer missing" cases into "answer present" cases. For the vast majority of questions, the gold answer entity is already present in the baseline's context window. Consequently, the remaining errors stem largely from the LLM failing to extract or reason over the correct entity despite its presence, suggesting a ceiling effect where additional structural recall yields diminishing returns.

Furthermore, our analysis of GPT-based triple recall suggests a semantic–structural mismatch. While GNN message passing successfully prioritizes triples that complete topological shortest paths (improving shortest-path recall), it slightly degrades the retrieval of triples judged relevant by GPT-4o (decreasing GPT recall by ≈ 0.8 points at high K). This implies that the GNN optimizes for structural connectivity, favoring "bridge" edges that connect topic and answer entities, whereas the LLM reasoner may rely on semantically rich but

structurally peripheral triples (e.g., aliases, descriptive properties) to generate the correct answer. By optimizing strictly for shortest-path inclusion, the GNN may inadvertently displace semantic cues that the LLM prefers, neutralizing the benefits of improved answer coverage.

5.2 The Limits of Local Message Passing

Our experiments identify a clear "sweet spot" at $L = 1$: a single layer of message passing provides robust retrieval gains, whereas adding a second layer ($L = 2$) yields diminishing returns and introduces significant instability. This behavior can be attributed to the topological properties of the batch-induced graphs on which our GNN operates.

As defined in Section 3.4, the GNN does not query the full knowledge graph but instead operates on a temporary graph induced by the candidate triples in the current minibatch. Given the sampling strategy, where a few dozen positive triples are mixed with hundreds of negatives, the resulting graph is often sparse and fragmented. A single hop ($L = 1$) effectively aggregates information from immediate neighbors, allowing triples that share entities (e.g., alternative paths between the same topic–answer pair) to reinforce each other's representations. However, a second hop ($L = 2$) expands the receptive field into a broader neighborhood dominated by randomly sampled negative triples. On such a noisy graph, deeper propagation likely dilutes the relevant signal with noise rather than capturing useful multi-hop structural dependencies. This hypothesis is consistent with our training observations, where $L = 2$ models often triggered early stopping sooner than $L = 1$, suggesting they struggled to generalize from the noisy higher-order features.

Furthermore, the deeper architecture incurs a disproportionate systems cost. While the mean inference time for $L = 2$ is similar to the baseline, it exhibits extreme variance ($\sigma \approx 1200$s), with some runs taking nearly four times longer than the median. This instability likely stems from the irregular computational workload of message passing on variable-sized connected components within different batches, which can cause GPU memory fragmentation or pipeline stalls. Consequently, for GraphRAG-style retrievers operating on candidate-induced subgraphs, we conclude that shallow, well-tuned message passing ($L = 1$) offers the optimal balance of performance and reliability.

6 Conclusion and Future Work

We investigated the efficacy of adding local graph message passing to the retrieval stage of a SubgraphRAG-style KGQA pipeline on WebQSP. Our experiments show that a single GraphSAGE layer ($L = 1$) consistently improves retrieval quality compared to the MLP baseline, achieving gains of $+2.0$ points in Answer Recall at $K = 50$ and $+1.2$ points at $K = 100$. However, these structural improvements do not translate into higher end-to-end QA accuracy; downstream metrics such as Hit@1 and Exact Match remain within one standard deviation of the baseline. This plateau suggests that once retrieval coverage exceeds $\sim 94\%$, the remaining errors are dominated by the LLM's semantic interpretation rather than missing evidence. Furthermore, we find that deeper message passing ($L = 2$) fails to provide additional benefit and introduces severe instability in training convergence and inference latency, identifying $L = 1$ as the practical sweet spot for this architecture.

A primary limitation of our current approach is the reliance on sparse batch-induced graphs. As discussed in Section 5.2, restricting message passing to small minibatches creates fragmented neighborhoods that dilute the signal for deeper GNNs ($L = 2$). Future work should explore richer graph construction strategies to address this, such as fetching global k-hop neighborhoods for candidate entities or employing structure-aware batching that clusters topologically related triples together. Architectures robust to sparsity, such as relation-aware attention, may also help the model distinguish valuable structural context from batch noise.

Finally, our results highlight a misalignment between the retriever's topological objective and the reasoner's semantic needs. Shortest-path supervision optimizes for connectivity but correlates imperfectly with the LLM's notion of relevance, as evidenced by the regression in GPT-based triple recall. To bridge the gap between improved retrieval and stagnant QA performance, future research should move beyond topological proxy labels. Promising directions include distilling relevance scores from LLMs into the retriever's supervision signal or exploring reinforcement learning approaches where the retriever is directly rewarded for downstream QA correctness.

References

Mufei Li, Siqi Miao, and Pan Li. 2025. Simple Is Effective: The Roles of Graphs and Large Language Models in Knowledge-Graph-Based Retrieval-Augmented Generation. *arXiv preprint*. ArXiv:2410.20724 [cs] version: 4.

Alon Talmor and Jonathan Berant. 2018. The Web as a Knowledge-Base for Answering Complex Questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 641–651, New Orleans, Louisiana. Association for Computational Linguistics.

Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. 2016. The Value of Semantic Parse Labeling for Knowledge Base Question Answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 201–206, Berlin, Germany. Association for Computational Linguistics.

Qinggang Zhang, Shengyuan Chen, Yuanchen Bei, Zheng Yuan, Huachi Zhou, Zijin Hong, Hao Chen, Yilin Xiao, Chuang Zhou, Junnan Dong, Yi Chang, and Xiao Huang. 2025. A Survey of Graph Retrieval-Augmented Generation for Customized Large Language Models. *arXiv preprint*. ArXiv:2501.13958 [cs].