

Towards a Learnable Retriever for Textual Graph Question Answering

Kaifeng Jin¹, Yuchen Yan¹, Zhichen Zeng¹, Xiao Lin¹, Lihui Liu², Yinglong Xia³, Hanghang Tong¹

¹University of Illinois Urbana-Champaign

²Wayne State University

³Meta

Abstract

Textual graph question answering (GraphQA) and graph-based retrieval-augmented generation (GraphRAG) have gained increasing attention as a way to ground large language models (LLMs) on structured knowledge. Despite extensive efforts, existing methods still face two key gaps. The first is an *architectural gap*: existing methods typically fine-tune an LLM-based generator, but do not also make the retriever itself a learnable LLM for graph evidence selection within one end-to-end retrieve-then-generate pipeline. The second is a *retrieval-supervision gap*: existing retrievers rarely use gold answer signals to supervise evidence selection. As a result, they may fail to recall evidence nodes required for reasoning, resulting in insufficient *coverage* of the gold answer set; meanwhile, naive attempts to boost recall often introduce excessive irrelevant information that compromises *compactness* and overwhelms the LLM. To address these gaps, we propose TALENT, an LLM-based end-to-end retriever-generator framework for textual GraphQA. To bridge the architectural gap, TALENT pairs a learnable LLM-based retriever for evidence selection with a fine-tuned LLM-based generator within one unified retrieve-then-generate pipeline. To bridge the retrieval-supervision gap, we leverage gold answer signals as retriever supervision. Specifically, we stratify graph nodes into gold answers, answer-related nodes, and irrelevant noise. We then adopt a weighted pseudo-label loss to prioritize gold answers and preserve answer-related nodes for coverage, while discarding irrelevant noise for compactness. Experimental results across three GraphQA datasets demonstrate that TALENT consistently improves downstream performance.

1 Introduction

Large language models (LLMs) have achieved impressive progress in general-purpose generation and reasoning (Thopilan et al. 2022; Bang et al. 2023), but their outputs are bounded by what is encoded in model parameters. Trained on large-scale open-domain corpora collected before a fixed cutoff date, LLMs may hallucinate or omit critical evidence when a query requires precise, up-to-date, or domain-specific knowledge (Zhao et al. 2026; Wang et al. 2023b).

To mitigate these limitations, Retrieval-Augmented Generation (RAG) has emerged as a dominant paradigm that grounds LLM outputs in external evidence by dynamically retrieving relevant knowledge snippets for context-aware generation (Lewis et al. 2020; Shi et al. 2024; Gao et al.

2023). While conventional RAG methods (Lewis et al. 2020; Shi et al. 2024) mainly retrieve evidence from unstructured text corpora, recent work has shifted toward textual graphs as a more expressive and structured knowledge source. This trend has spurred the development of Graph-based RAG (GraphRAG) and Graph Question Answering (GraphQA). By exploiting the explicit semantics and relational structure of textual graphs, graph-based RAG methods support more grounded reasoning and complex information synthesis (IV et al. 2019; Luo et al. 2025; He et al. 2024).

A graph-based RAG pipeline generally consists of two stages: graph evidence retrieval and answer generation. On the generation side, existing methods often use an LLM-based generator and fine-tune it over retrieved graph contexts. This makes graph evidence retrieval a critical step, since the generator can only reason over the evidence selected for it.

Conventional retrieval methods in GraphRAG can be categorized into non-parametric, graph neural network (GNN)-based, and LM-based approaches. Non-parametric retrievers construct subgraphs using graph search algorithms or predefined rules (Taunk et al. 2023; Yasunaga et al. 2021). GNN-based methods encode graph structure with GNNs and identify task-relevant components to form the retrieved subgraph (Mavromatis and Karypis 2024). LM-based methods often use LMs to assist graph retrieval, for example by reading structured data, generating retrieval cues, or using LM reasoning to guide evidence selection (Baek, Aji, and Saffari 2023; Jiang et al. 2023).

However, with these retrieval designs, existing graph-based RAG pipelines still leave an architectural gap. While the generator is often fine-tuned as an LLM-based module, the retriever itself is usually not made a learnable LLM for graph evidence selection. Instead, evidence selection is still handled by non-parametric, GNN-based, or LM-assisted procedures. As a result, the retriever and generator are not both trained as LLM modules within one end-to-end retrieve-then-generate pipeline.

Beyond this architectural gap, these conventional retrievers also face a retrieval-supervision gap. They rely primarily on query similarity while rarely using gold answers to supervise evidence selection. Query similarity can favor nodes matching query terms over answer-critical evidence, leading to two limitations: *coverage*, where a query-based retriever may fail to cover the gold answer set in the retrieved subgraph, and

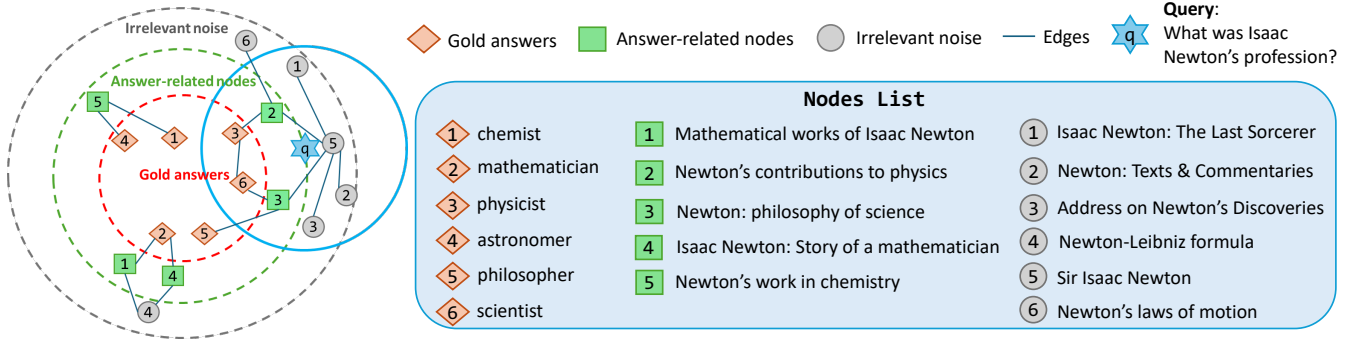


Figure 1: An illustrative example of two limitations of query-based graph retrieval. For the query “What was Isaac Newton’s profession?”, a query-similarity retriever returns a Newton-centric subgraph (blue circle). We partition the nodes into three groups: *Gold answers* (only a small subset of the gold answer set is retrieved by the subgraph in the blue circle, showing limited coverage), *Answer-related nodes* (topically relevant but not exact gold answers), and *Irrelevant noise* (nodes that match the query tokens such as “Newton” yet are not answer-related). The presence of substantial noise alongside partial gold coverage highlights the *compactness* issue: improving recall by expanding query-similar candidates can inflate the subgraph with distractors.

compactness, where it may introduce excessive irrelevant noise. An illustrative example is shown in Figure 1. A key observation is that the *gold answer set* provides precisely the missing supervision for retrieval. Gold answers specify what should be covered and also help distinguish answer-related evidence from irrelevant noise, and can therefore help alleviate these two limitations. However, such gold signals are rarely exploited to guide or train graph retrievers.

In this work, we address both gaps by proposing TALENT, an LLM-based end-to-end retriever-generator framework for textual GraphQA. To bridge the architectural gap, TALENT makes the retriever itself a learnable LLM for graph evidence selection, and pairs it with a fine-tuned LLM-based generator within one unified retrieve-then-generate pipeline. Within the full pipeline, we first apply a rule-based preprocessing step to construct a candidate subgraph, providing a high-recall pool of nodes under the LLM context budget. The learnable LLM-based retriever is then trained to select relevant evidence nodes, and the generator is fine-tuned on contexts constructed from the trained retriever’s outputs. To bridge the retrieval-supervision gap, we use gold answer signals as retriever supervision to improve both retrieval coverage and compactness. To improve retrieval coverage, we train a retriever based on a weighted pseudo-label objective, incorporating gold answer signals to guide the retrieval process. To promote answer-critical evidence, we prioritize gold answer and down-weight other rest signals to improve retrieval compactness. Intuitively, the loss assigns the highest weight to gold answers to ensure coverage, assigns a much smaller weight to answer-related nodes to retain useful context, and ignores irrelevant noise to enforce compactness. In this way, TALENT improves evidence selection before generation, leading to better downstream QA performance. Our contributions are summarized as follows:

- **Idea.** We introduce an LLM-based end-to-end retriever-generator framework for textual GraphQA, where a learnable LLM-based retriever and a fine-tuned LLM-based

generator are placed within one unified retrieve-then-generate pipeline. This design fills the architectural gap by making the retriever itself a learnable LLM for graph evidence selection, rather than only fine-tuning the downstream generator.

- **Method.** We train the learnable LLM-based retriever with a weighted pseudo-label objective that leverages gold answer signals. By prioritizing gold answer nodes and down-weighting weakly related nodes, the objective fills the retrieval-supervision gap and leads to better downstream QA performance.
- **Evaluation.** We conduct extensive experiments on three GraphQA benchmarks to validate both retrieval effectiveness and end-task QA performance. Our method outperforms the best competitor by at least 3.18%.

2 Preliminaries

Notation. We use calligraphic letters to denote sets, e.g., $\mathcal{V}$ and $\mathcal{E}$ for node and edge sets. A textual graph is represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each node $v \in \mathcal{V}$ has a textual attribute $\text{text}(v)$, e.g., an entity name or concept, and each edge $e = (u, r, v) \in \mathcal{E}$ denotes a relation r connecting the source node u to the destination node v . The textual attribute of an edge is denoted as $\text{text}(e)$. We use $\mathcal{S}$ to denote the subgraph of $\mathcal{G}$, where $\mathcal{S} \subseteq \mathcal{G}$ indicates that its node set $\mathcal{V}_{\mathcal{S}} \subseteq \mathcal{V}$ and its edge set $\mathcal{E}_{\mathcal{S}} \subseteq \mathcal{E}$. We use q for the query and a for the final answer from the generator. We use a^* to denote the token sequence of the gold answer in the dataset and use $\mathcal{V}^*$ as the set of nodes appearing in the gold answer, i.e., the gold answer set. We denote the retriever and generator by R_θ and G_ϕ , respectively.

2.1 Problem Setup

We consider a general retrieve-then-generate setup (He et al. 2024) for textual graph question answering (GraphQA). Each instance in the dataset consists of a natural-language query q and a textual graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as the knowledge base. Due to

the large scale of $\mathcal{G}$, it is impractical to directly serialize $\mathcal{G}$ into an LLM prompt, hence the system first retrieves a relevant *subgraph*, conditioned on which an answer is generated.

Formally, a retriever R_θ extracts a subgraph $\mathcal{S}$ from the textual graph $\mathcal{G}$ based on the query q :

$$\mathcal{S} = R_\theta(q, \mathcal{G}). \quad (1)$$

A generator G_ϕ then predicts the answer a based on the query q and the retrieved subgraph $\mathcal{S}$:

$$a = G_\phi(q, \mathcal{S}). \quad (2)$$

2.2 Candidate Connected Subgraph

For large textual graphs, directly serializing the full graph often exceeds the LLM context budget. As in prior graph-based QA pipelines, we include a preprocessing stage. Specifically, following G-Retriever (He et al. 2024), we first construct a *candidate connected subgraph* $\mathcal{S}$ using the Prize-Collecting Steiner Tree (PCST) algorithm (Bienstock et al. 1993), and use $\mathcal{S}$ as the input context for downstream modules.

Indexing and similarity retrieval. Given a query q and a textual graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we embed the query and textual attributes of node $\mathcal{V}$ and edges $\mathcal{E}$ using a sentence encoder such as SentenceBERT (Reimers and Gurevych 2019):

$$\mathbf{z}_q = \text{LM}(q), \quad (3)$$

$$\mathbf{z}_v = \text{LM}(\text{text}(v)), \quad v \in \mathcal{V}, \quad (4)$$

$$\mathbf{z}_e = \text{LM}(\text{text}(e)), \quad e \in \mathcal{E}. \quad (5)$$

We compute cosine similarity scores and retrieve the top- k_v nodes and top- k_e edges:

$$\mathcal{V}_{\text{sim}} = \arg \text{topk}_{v \in \mathcal{V}, k=k_v} \cos(\mathbf{z}_q, \mathbf{z}_v), \quad (6)$$

$$\mathcal{E}_{\text{sim}} = \arg \text{topk}_{e \in \mathcal{E}, k=k_e} \cos(\mathbf{z}_q, \mathbf{z}_e), \quad (7)$$

where k_v and k_e control the number of retrieved nodes and edges, respectively.

Subgraph construction. We assign *prizes* to retrieved candidates based on their ranks, so higher-ranked items receive larger prizes. Concretely, for $v \in \mathcal{V}_{\text{sim}}$ with rank r_v we set $p(v) = k_v - r_v$ to prioritize those with higher similarity, and analogously define $p(e) = k_e - r_e$ for $e \in \mathcal{E}_{\text{sim}}$; items outside these retrieved lists receive zero prize. PCST then selects a connected subgraph $\mathcal{S} = (\mathcal{V}_\mathcal{S}, \mathcal{E}_\mathcal{S})$ that trades off prize coverage and compactness:

$$\mathcal{S} = \arg \max_{\substack{\mathcal{S} \subseteq \mathcal{G} \\ \text{connected}}} \left(\sum_{v \in \mathcal{V}_\mathcal{S}} p(v) + \sum_{e \in \mathcal{E}_\mathcal{S}} p(e) - c|\mathcal{E}_\mathcal{S}| \right). \quad (8)$$

where c controls the size-coverage trade-off.

3 Method

Overview. TALENT is an LLM-based end-to-end retriever-generator framework for textual GraphQA. Its two core trainable LLM modules are: (i) a learnable nodes retriever

R_θ trained with weighted pseudo-label supervision (Section 3.1), and (ii) an answer generator G_ϕ fine-tuned with standard Supervised Fine-Tuning (SFT) (Section 3.2). The full TALENT pipeline is illustrated in Figure 2. Given a query q and a full textual graph $\mathcal{G}$, a rule-based PCST preprocessing step first constructs an initial candidate connected subgraph $\mathcal{S}$ that fits the LLM context budget. We then train a learnable retriever R_θ with weighted pseudo-label supervision to produce an answer-relevant node set $\hat{\mathcal{V}}$ (Section 3.1). Using answer-relevant nodes $\hat{\mathcal{V}}$, we construct textual contexts and fine-tune a generator G_ϕ with standard supervised learning (Section 3.2).

3.1 Retriever Training

As discussed earlier, PCST and other query-similarity based retrievers often fail to cover the answer-related nodes and include irrelevant noise, resulting in limited *coverage* and poor *compactness*. We therefore use gold answer signals to guide retriever training, encouraging prioritization of gold answers while down-weighting other signals. We first describe TALENT’s retriever prompting and then introduce the weighted pseudo-label objective.

Retriever prompting. We implement our learnable retriever R_θ as an instruction-following LLM (with LoRA) that performs node selection based on a provided candidate subgraph. Given a query q and the PCST subgraph $\mathcal{S}$, we prompt the LLM to produce the nodes that are most helpful for answering q . The LLM outputs should be exactly one line in the format of “name1|name2|...”. Each name such as name1 and name2 represents the names of the nodes R_θ finally selects. This constrained format makes the output directly interpretable as a retrieved nodes set, which we denote by $\hat{\mathcal{V}} = R_\theta(q, \mathcal{S})$. A concrete retriever prompt example is provided in Appendix E.1. It is worth noting that since our retriever R_θ is an LLM, it leverages its inherent parametric knowledge during the selection process. Consequently, the output node set $\hat{\mathcal{V}}$ may include nodes that were not present in the initial candidate subgraph $\mathcal{S}$ (i.e., $\hat{\mathcal{V}} \not\subseteq \mathcal{V}_\mathcal{S}$). In our method, $\hat{\mathcal{V}}$ encompasses the complete set of nodes generated by R_θ , reflecting both the information from $\mathcal{S}$ and the LLM’s internal knowledge.

Fine-tuning with weighted pseudo-label supervision. Recall that existing works in GraphRAG use retrievers that are largely query-similarity driven, which leads to two limitations: coverage and compactness. Our goal is to train the retriever to output the node set $\hat{\mathcal{V}}$ that achieves both high *coverage* and *compactness*: it should reliably include nodes that appear in the gold answer set $\mathcal{V}^*$ (coverage) while avoiding redundant or weakly related candidates (compactness). The gold answer provides precisely the missing supervision, i.e., the gold signal, for retrieval. To inject gold signals into retrieval, we leverage the dataset-provided gold answers and stratify candidate nodes into three levels: (i) *gold answers*, which are candidate nodes that overlap with the gold answer set $\mathcal{V}^*$, and provides evidence that the retriever should prioritize to support correct reasoning, (ii) *answer-related nodes*, which are candidate nodes within $\mathcal{S}$ but are not gold answers, and may still be helpful for providing context, though

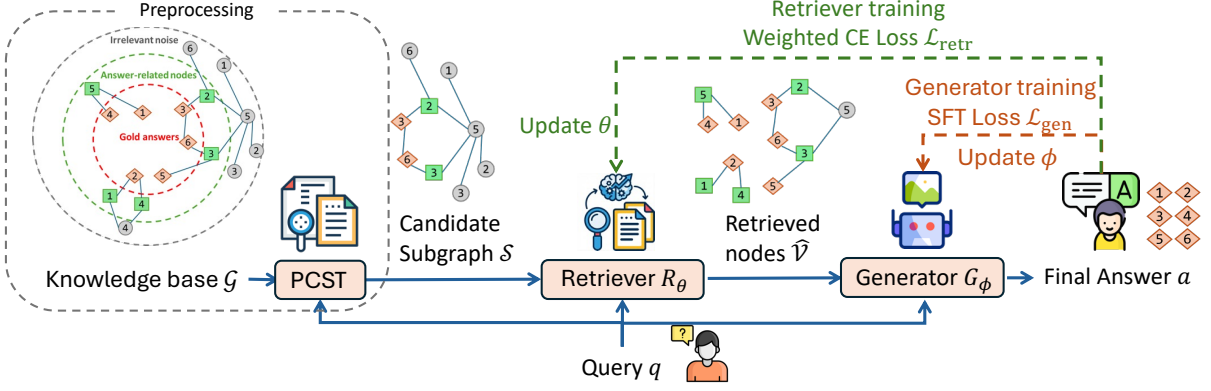


Figure 2: Overview of TALENT, an LLM-based end-to-end retriever-generator framework for textual GraphQA. The full TALENT pipeline contains three stages: (1) Preprocessing, which applies PCST to construct an initial candidate connected subgraph $\mathcal{S}$ from the knowledge base $\mathcal{G}$ based on query q . (2) Retriever training, which trains a learnable LLM-based retriever R_θ with weighted pseudo-label supervision, given the candidate subgraph $\mathcal{S}$ and query q , to produce $\hat{\mathcal{V}}$. (3) Generator training, which fine-tunes an LLM-based generator G_ϕ with standard SFT given retrieved node set $\hat{\mathcal{V}}$ and q , to produce the final answer a .

they are less directly tied to the final answer, and (iii) *irrelevant noise*, which are nodes not contained in $\mathcal{S}$, and are completely irrelevant and should be suppressed to maintain compactness. Given the candidate subgraph $\mathcal{S}$ with node set $\mathcal{V}_\mathcal{S}$, and the gold answer set $\mathcal{V}^*$, we utilize these nodes as weighted pseudo label supervision by defining

$$\mathcal{P} = \mathcal{V}^* \cap \mathcal{V}_\mathcal{S} \quad (\text{strong positives; level-1}), \quad (9)$$

$$\mathcal{W} = \mathcal{V}_\mathcal{S} \setminus \mathcal{V}^* \quad (\text{weak positives; level-2}). \quad (10)$$

All nodes outside $\mathcal{V}_\mathcal{S}$ are treated as irrelevant noise (level-3) during retriever training.

We train R_θ as conditional generation of a target string y that concatenates nodes in $\mathcal{P}$ and $\mathcal{W}$ (separated by the pipe symbol “|”; strong positives first, then weak positives)

$$y = p_1 | \dots | p_{|\mathcal{P}|} | w_1 | \dots | w_{|\mathcal{W}|}, \quad (11)$$

where $p_i, \forall i = 1, \dots, |\mathcal{P}|$ are nodes in $\mathcal{P}$ and $w_j, \forall j = 1 \dots |\mathcal{W}|$ are nodes in $\mathcal{W}$. Let x denote the retriever input prompt, which includes the query q and the textualized candidate subgraph $\mathcal{S}$, and let $y_{1:T}$ be the tokenized target sequence, where T is its token length. To prioritize level-1 (gold answer) nodes while still preserving level-2 (answer-related) candidates as a weak learning signal, we minimize the token-weighted negative log-likelihood:

$$\mathcal{L}_{\text{retr}}(\theta) = - \sum_{t=1}^T \omega_t \log p_\theta(y_t | x, y_{<t}). \quad (12)$$

Since we want to encourage R_θ to prioritize gold answer nodes while still allowing potentially useful answer-related nodes to contribute as a weak learning signal, we adopt the following token weights to achieve this

$$\omega_t = \begin{cases} 1, & \text{if } y_t \text{ belongs to a node in } \mathcal{P}, \\ \lambda, & \text{if } y_t \text{ belongs to a node in } \mathcal{W}, \\ 0, & \text{otherwise,} \end{cases} \quad (13)$$

where λ is a hyperparameter that controls the contribution of weak positives relative to strong positives.

3.2 Generator Training

Generator Prompting. Given the predicted node set $\hat{\mathcal{V}} = R_\theta(q, \mathcal{S})$ output by the trained R_θ , we form the generator input as a *prompt* that concatenates the query q with subgraphs derived from $\hat{\mathcal{V}}$. These subgraphs contain (i) the retrieved nodes in $\hat{\mathcal{V}}$ and (ii) their associated edges $\hat{\mathcal{E}}$ that appear in $\mathcal{S}$. Concretely, we include edges for which at least one endpoint is in $\hat{\mathcal{V}}$:

$$\hat{\mathcal{E}} = \{(u, r, v) \in \mathcal{E} : u \in \hat{\mathcal{V}} \text{ or } v \in \hat{\mathcal{V}}\}. \quad (14)$$

We then serialize the retrieved nodes $\hat{\mathcal{V}}$ together with the selected edges $\hat{\mathcal{E}}$ into a *serialized graph text* and prepend the query q to form the generator prompt. Importantly, this *serialized graph text* is not required to form one whole connected subgraph. This design avoids forcing unnecessary “bridge” nodes or edges solely to maintain connectivity, which often increases irrelevant context and hurts compactness. Instead, we allow the retrieved evidence to appear as multiple disconnected components, as long as it covers the nodes in $\hat{\mathcal{V}}$. Moreover, as we mentioned in Section 3.1, $\hat{\mathcal{V}}$ may include nodes that come from R_θ ’s parametric knowledge and do not appear in the initial candidate subgraph $\mathcal{S}$, we retain these nodes as additional isolated nodes in the *serialized graph text*. A concrete generator prompt example together with the graph serialization format is provided in Appendix E.2.

Fine-tuning with standard SFT. We fine-tune an LLM-based generator G_ϕ using standard supervised fine-tuning (SFT). For each training instance, we first run the trained retriever R_θ to obtain the generator prompt constructed in the previous paragraph (i.e., the *serialized graph text* formed from $\hat{\mathcal{V}}$ and the associated $\hat{\mathcal{E}}$, together with the query q). We

Dataset (Metrics)	ExplaGraphs (ACC)	WebQSP (Hit@1)	CWQ (Hit@1)
Zero-shot	56.50	41.06	18.58
Zero-CoT (Kojima et al. 2022)	57.04	51.30	25.63
CoT-BAG (Wang et al. 2023a)	57.94	39.60	18.83
KAPING (Baek, Aji, and Saffari 2023)	62.27	52.64	23.25
Graph-based Inference	33.93	47.22	18.21
Frozen LLM + Prompt Tuning (PT)	57.63	48.34	36.22
GraphToken (Perozzi et al. 2024)	85.08	57.05	33.39
G-Retriever	85.16	70.49	45.62
G-Retriever w/ LoRA	<u>87.05</u>	<u>73.79</u>	<u>49.19</u>
T _{AO} NA-A (Yan et al. 2025)	87.01	71.23	45.28
StructGPT (Jiang et al. 2023)	78.34	72.60	32.57
TALeNT	91.79	76.97	55.71

Table 1: Benchmark results (%).

then train G_ϕ to generate the gold answer a^* conditioned on this prompt:

$$\mathcal{L}_{\text{gen}}(\phi) = - \sum_{t=1}^{T_a} \log p_\phi(a_t^* | q, \hat{\mathcal{V}}, \hat{\mathcal{E}}, a_{<t}^*), \quad (15)$$

where $a^* = (a_1^*, \dots, a_{T_a}^*)$ denotes the *token* sequence of the gold answer, and T_a is its length. This SFT step transfers retriever’s improved evidence selection into downstream QA.

3.3 Inference and Evaluation

At inference time, we apply the full retrieve-then-generate pipeline using the retriever and generator fine-tuned on the training (and validation) splits. For each test instance $(q, \mathcal{G})$, we first apply the fixed PCST-based preprocessing step to construct the candidate subgraph $\mathcal{S}$, then run the learned retriever to obtain $\hat{\mathcal{V}} = R_\theta(q, \mathcal{S})$. We materialize $(\hat{\mathcal{V}}, \hat{\mathcal{E}})$ into a textual graph context and feed it to the generator to produce the final answer $a = G_\phi(q, \hat{\mathcal{V}}, \hat{\mathcal{E}})$. We report task metrics computed from the generator outputs on the test split.

4 Experiments

4.1 Experiment Setup

Datasets and Metrics. To evaluate our method on textual QA tasks, we conduct experiments on three textual graph question answering datasets: ExplaGraphs and WebQSP from the GraphQA benchmark (He et al. 2024), and CWQ (Talmon and Berant 2018). Specifically, (1) ExplaGraphs’s task is to determine whether two arguments are supportive or contradictory. (2) WebQSP is a knowledge graph question answering dataset whose primary task is to answer questions that require multi-hop reasoning. (3) CWQ is a more compositional KGQA benchmark than WebQSP, with more complex questions and longer reasoning chains. For evaluation, we adopt Accuracy for ExplaGraphs and Hit@1 for WebQSP and CWQ. Dataset statistics and additional dataset details are provided in Appendix A.

Baselines. We compare against two groups of baselines. (1) Inference-Only methods that include Zero-shot, Zero-CoT (Kojima et al. 2022), CoT-BAG (Wang et al. 2023a), KAPING (Baek, Aji, and Saffari 2023), Graph-based Inference, and StructGPT (Jiang et al. 2023). (2) Prompt-Tuning

methods that include Frozen LLM + Prompt Tuning (PT), GraphToken (Perozzi et al. 2024), G-Retriever (He et al. 2024), and T_{AO}NA-A (Yan et al. 2025). For G-Retriever, we include a variant that fine-tunes the LLM-based generator with LoRA (G-Retriever w/ LoRA). Additional discussion of our differences from other answer-aware or answer-related methods and recent graph retrieval baselines not included in Table 1 is provided in Appendix D.

Experiment pipeline. For the main experiments, we use the open-source LLaMA2-Chat-7B model (Touvron et al. 2023) as the backbone for both the retriever and the generator, and conduct all experiments on a single NVIDIA A100 GPU. We follow the same train/validation/test splits as prior work for each dataset. For WebQSP and CWQ, since the full KG exceeds the context budget, we first construct a PCST candidate subgraph with $k_v=k_e=10$ and then apply our learnable retriever. During retriever training, we use the weighted pseudo-label objective with strong-positive weight 1. The weak-positive weight is set to $\lambda = 0.001$ for WebQSP and $\lambda = 0.01$ for ExplaGraphs and CWQ. Additional implementation details, including hyperparameter configurations and computing infrastructure, are provided in Appendix B. Appendix C reports sensitivity analyses for λ and PCST size parameters k_v and k_e . A backbone ablation with Llama-3-8B (Grattafiori et al. 2024) is provided in Appendix G.

4.2 Benchmark results

Table 1 reports the benchmark results on ExplaGraphs (Accuracy), WebQSP (Hit@1), and CWQ (Hit@1). Overall, TALeNT outperforms the strongest competitors across all three datasets. We draw the following three observations.

(1) TALeNT outperforms the strongest competitors. TALeNT achieves 91.79% accuracy on ExplaGraphs, 76.97% Hit@1 on WebQSP, and 55.71% Hit@1 on CWQ, improving over the strongest baseline by 4.74% on ExplaGraphs, 3.18% on WebQSP, and 6.52% on CWQ. These results show that TALeNT generalizes well under different graph QA objectives and metrics, with the largest gain obtained on the more challenging CWQ benchmark.

(2) Comparison to inference-only methods. Inference-only baselines remain behind TALeNT on all three datasets. Even compared with the strongest inference-only baseline, StructGPT, TALeNT improves by 13.45% on ExplaGraphs, 4.37%

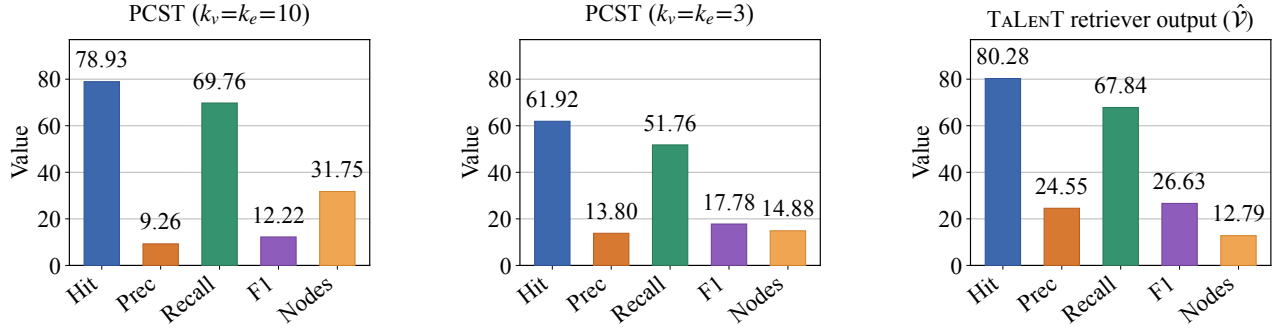


Figure 3: WebQSP test-set retrieval results (macro-averaged). We measure how well PCST subgraphs and TALENT’s learned retriever output $\hat{\mathcal{V}}$ cover the gold answer set $\mathcal{V}^*$. We compare: (1) the PCST candidate subgraph with $k_v=k_e=10$ (this is the retriever input), (2) a much smaller PCST subgraph with $k_v=k_e=3$ (a size-matched reference), and (3) the nodes set $\hat{\mathcal{V}}$ selected by TALENT’s learned retriever. Our retriever keeps coverage close to PCST($k_v=k_e=10$) while using far fewer nodes, and it also covers more gold nodes than the size-matched PCST($k_v=k_e=3$).

on WebQSP, and 23.14% on CWQ. This performance gap stems from their reliance on test-time prompting and non-trainable evidence acquisition, without task-specific fine-tuning of either the retriever or generator. In contrast, TALENT trains both components, enabling more reliable evidence selection and more accurate answers.

(3) Comparison to prompt-tuning methods. Prompt-tuning methods form substantially stronger baselines than inference-only methods overall, with G-Retriever w/ LoRA being the strongest prior prompt-tuning baseline on all three datasets. Nevertheless, TALENT consistently improves over it, from 87.05% to 91.79% on ExplaGraphs, from 73.79% to 76.97% on WebQSP, and from 49.19% to 55.71% on CWQ. These gains support the two central design choices of TALENT. First, TALENT moves beyond rule-based, query-driven evidence selection by introducing a learnable LLM-based retriever for task-adaptive graph evidence selection within an end-to-end retriever-generator framework. Second, it trains this retriever with gold-answer-guided weighted pseudo-label supervision to improve evidence coverage and compactness, thereby benefiting downstream generation.

4.3 Coverage-Compactness Tradeoff

In this section, we study how TALENT’s retriever achieves a better coverage-compactness tradeoff for its output node set $\hat{\mathcal{V}}$. We conduct experiments on WebQSP, with results shown in Figure 3, and make the following observations.

Compactness: similar coverage with substantially fewer nodes. We first compare the compactness of nodes retrieved by the initial PCST candidate subgraph ($k_v=k_e=10$) and TALENT, as shown in Figure 3. Under similar gold-node coverage, TALENT’s retriever output $\hat{\mathcal{V}}$ contains significantly fewer nodes. Both methods achieve comparable Hit@1 and Recall on WebQSP, yet $\hat{\mathcal{V}}$ contains less than half as many nodes: on average 12.79 nodes for TALENT versus 31.75 nodes for the PCST subgraph. Meanwhile, $\hat{\mathcal{V}}$ attains notably higher precision and F1, improving precision from 9.26% to 24.55% and F1 from 12.22% to 26.63%. These results sug-

Output subset	Hit@1	Recall
All retrieved nodes	80.28	67.84
Nodes within PCST	75.31	64.19
Nodes out of PCST	10.26	5.46

Table 2: Breakdown of gold answer coverage contributed by different parts of the learned retriever output on WebQSP test. All metrics are macro-averaged over examples and reported as percentages.

Retriever setting	Hit@1	Prec	Recall	F1
TALENT	80.28	24.55	67.84	26.63
Ablate retriever training	36.61	14.97	26.26	14.97

Table 3: Retriever ablation on WebQSP test set (macro-averaged over examples). “Ablate retriever training” replaces TALENT’s learnable retriever by a frozen retriever without LoRA fine-tuning.

gest that the weighted pseudo-label supervision effectively incorporates gold signals, enabling the retriever to filter redundant or weakly related nodes and produce a more compact context without sacrificing coverage.

Coverage: better gold answer coverage under comparable node number. To further compare coverage, we report results for a much smaller PCST subgraph with $k_v=k_e=3$, whose size is comparable to that of TALENT. As shown in Figure 3, under a similar node budget, TALENT’s retriever achieves substantially higher coverage. Specifically, TALENT improves Hit@1 from 61.92% to 80.28% and Recall from 51.76% to 67.84% compared to PCST. This demonstrates that, given a similar number of nodes, our gold-signal-involved weighted pseudo-label supervision enables the retriever to achieve significantly higher coverage than the rule-based, query-similarity-driven PCST algorithm.

System	Hit@1 (%)
TALeNT w/o retriever training	68.37
TALeNT w/o retriever	72.50
TALeNT	76.97

Table 4: QA ablations on WebQSP (Hit@1, %). All systems follow the same pipeline as TALeNT except for the ablated component stated in each row.

4.4 Retrieval Beyond PCST

Fine-tuning on an LLM-based retriever may leverage parametric knowledge to retrieve nodes beyond the PCST candidate subgraph. We therefore examine whether TALeNT’s retriever can retrieve such beyond-PCST nodes and how they contribute to performance. Specifically, we split the retriever output $\hat{\mathcal{V}}$ into nodes inside and outside the PCST subgraph. Shown in Table 2, nodes outside PCST alone recover a non-trivial portion of gold answer nodes, with a Hit@1 of 10.26% and a Recall of 5.46%. Moreover, incorporating beyond-PCST nodes improves Hit@1 from 75.31 to 80.28% and Recall from 64.19% to 67.84%. Results indicate that the trained LLM retriever effectively complements the PCST candidate subgraph with parametric-knowledge nodes that enhance gold-node coverage. An additional groundedness and hallucination analysis of out-of-PCST retrieval is provided in Appendix F.

5 Ablation Study

We conduct two ablation studies on WebQSP to illustrate (i) the effect of pseudo-label retriever training (ii) the effect of learnable retriever.

5.1 On the Effect of Pseudo-label Training

Retrieval quality. Table 3 compares TALeNT’s retriever against an ablated variant, which uses the same backbone LLaMA2-Chat-7B model but keeps it *frozen* (no training) and directly prompts it to select helpful nodes from the PCST candidate subgraph. Results show that the frozen retriever fails to recover gold nodes required for answering, with Hit@1, Precision, and Recall of 36.61%, 14.97%, and 26.26%, respectively. In contrast, TALeNT’s retriever explicitly prioritizes gold answer nodes during retriever fine-tuning using pseudo label training, substantially improving coverage with a Hit@1 of 80.28%, a precision of 24.55% and a recall of 67.84%, indicating that injecting gold signals into retriever training is crucial for selecting answer-relevant nodes.

Downstream QA performance. We next examine whether the retrieval improvement translates into better end-task QA. Table 4 summarizes QA Hit@1 under three systems: (i) TALeNT with a *frozen* retriever (ablate training on retriever), (ii) TALeNT *without* the retriever module (directly feeding the PCST candidate subgraph to the generator), and (iii) the full TALeNT system. Switching from a frozen retriever to our pseudo-label trained retriever improves QA Hit@1 from 68.37% to 76.97%. This gain is consistent with Table 3: pseudo-label training recovers substantially more gold nodes, providing the generator with more reliable evidence.

5.2 On the Effect of Learnable Retriever

We further isolate the contribution of the retriever by removing it from the pipeline and directly feeding the PCST candidate subgraph into the generator, i.e., without R_θ or $\hat{\mathcal{V}}$ selection. As shown in Table 4, removing the learnable retriever reduces QA performance from 76.97% to 72.50%, indicating that an explicit retriever provides clear benefits for downstream QA. Notably, using a frozen retriever performs even worse (Hit@1 of 68.37%) than removing the retriever entirely, suggesting that retrieval is not inherently beneficial when gold answer node coverage is low, as noisy context can mislead the generator. This result further underscores the importance of our weighted pseudo-label training, which substantially improves retrieval quality for end-task QA.

6 Related Work

Retrieval Augmented Generation. RAG grounds LLMs with external evidence to reduce hallucinations and improve faithfulness (Sun et al. 2024; Gao et al. 2023). Classic RAG systems follow a straightforward index-retrieve-generate workflow (Ma et al. 2023), while many recent variants strengthen retrieval with additional processing before and after retrieval. For example, pre-retrieval techniques such as query transformation, expansion, and rewriting are often adopted to better match the search space (Zheng et al. 2024; Gao et al. 2023), and post-retrieval reranking is commonly used to refine the evidence set (Blagojević 2023). More broadly, modular RAG systems use LLMs to iteratively refine retrieval queries or evidence selection (Yu et al. 2023).

RAG over Graphs. Extensive works study how to combine graph-structured information with LLMs (Li et al. 2024; Wang et al. 2023a). Prior efforts span general graph modeling (Liu et al. 2024; Lei et al. 2023), multimodal architectures that incorporate graph-like structures (Yoon et al. 2023; Li et al. 2023). Representative applications include fundamental graph reasoning (Zhao et al. 2023), node prediction (Sun et al. 2025; Chen et al. 2024), graph-level prediction (Qian et al. 2023). A related line of work discusses GraphRAG / GraphQA, where the system selects graph evidence and conditions an LLM on graph context to answer questions (IV et al. 2019; Luo et al. 2025).

7 Conclusion

We address two key gaps in existing GraphRAG systems: an *architectural gap* and a *retrieval-supervision gap*. To address both gaps, we propose TALeNT, an LLM-based end-to-end retriever-generator framework for textual GraphQA. To bridge the architectural gap, TALeNT makes the retriever itself a learnable LLM for graph evidence selection and pairs it with a fine-tuned LLM-based generator within one unified retrieve-then-generate pipeline. To bridge the retrieval-supervision gap, we train TALeNT’s retriever with a gold-answer-guided weighted pseudo-label loss that prioritizes answer-related signals to improve coverage while suppressing noise for compactness. Extensive experiments on three GraphQA benchmarks demonstrate that TALeNT consistently improves downstream QA performance.

References

- Baek, J.; Aji, A. F.; and Saffari, A. 2023. Knowledge-augmented language model prompting for zero-shot knowledge graph question answering. In *Proceedings of the 1st Workshop on Natural Language Reasoning and Structured Explanations (NLRSE)*, 78–106.
- Bang, Y.; Cahyawijaya, S.; Lee, N.; Dai, W.; Su, D.; Wilie, B.; Lovenia, H.; Ji, Z.; Yu, T.; Chung, W.; et al. 2023. A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity. In *Proceedings of the 13th international joint conference on natural language processing and the 3rd conference of the asia-pacific chapter of the association for computational linguistics (volume 1: Long papers)*, 675–718.
- Bienstock, D.; Goemans, M. X.; Simchi-Levi, D.; and Williamson, D. 1993. A note on the prize collecting traveling salesman problem. *Mathematical programming*, 59(1): 413–420.
- Blagojevi, V. 2023. Enhancing rag pipelines in haystack: Introducing diversityranker and lostinthemiddleranker.
- Bollacker, K.; Evans, C.; Paritosh, P.; Sturge, T.; and Taylor, J. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 1247–1250.
- Chai, Z.; Zhang, T.; Wu, L.; Han, K.; Hu, X.; Huang, X.; and Yang, Y. 2025. Graphllm: Boosting graph reasoning ability of large language model. *IEEE Transactions on Big Data*.
- Chen, Z.; Mao, H.; Li, H.; Jin, W.; Wen, H.; Wei, X.; Wang, S.; Yin, D.; Fan, W.; Liu, H.; et al. 2024. Exploring the potential of large language models (llms) in learning on graphs. *ACM SIGKDD Explorations Newsletter*, 25(2): 42–61.
- Gao, L.; Ma, X.; Lin, J.; and Callan, J. 2023. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1762–1777.
- Grattafiori, A.; Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Vaughan, A.; et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- He, X.; Tian, Y.; Sun, Y.; Chawla, N.; Laurent, T.; LeCun, Y.; Bresson, X.; and Hooi, B. 2024. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. *Advances in Neural Information Processing Systems*, 37: 132876–132907.
- Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; Chen, W.; et al. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2): 3.
- Hu, Y.; Lei, Z.; Dai, Z.; Zhang, A.; Angirekula, A.; Zhang, Z.; and Zhao, L. 2025. Cg-rag: Research question answering by citation graph retrieval-augmented llms. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 678–687.
- IV, R. L. L.; Liu, N. F.; Peters, M. E.; Gardner, M.; and Singh, S. 2019. Barack’s Wife Hillary: Using Knowledge-Graphs for Fact-Aware Language Modeling. *arXiv:1906.07241*.
- Jiang, J.; Zhou, K.; Dong, Z.; Ye, K.; Zhao, X.; and Wen, J.-R. 2023. Structgpt: A general framework for large language model to reason over structured data. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, 9237–9251.
- Kojima, T.; Gu, S. S.; Reid, M.; Matsuo, Y.; and Iwasawa, Y. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213.
- Lei, B.; pei Hung Lin; Liao, C.; and Ding, C. 2023. Boosting Logical Reasoning in Large Language Models through a New Framework: The Graph of Thought. *arXiv:2308.08614*.
- Lester, B.; Al-Rfou, R.; and Constant, N. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, 3045–3059.
- Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; Yih, W.-t.; Rocktäschel, T.; et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474.
- Li, X.; Lian, D.; Lu, Z.; Bai, J.; Chen, Z.; and Wang, X. 2023. Graphadapter: Tuning vision-language models with dual knowledge graph. *Advances in Neural Information Processing Systems*, 36: 13448–13466.
- Li, X. L.; and Liang, P. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *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)*, 4582–4597.
- Li, Y.; Li, Z.; Wang, P.; Li, J.; Sun, X.; Cheng, H.; and Yu, J. X. 2024. A Survey of Graph Meets Large Language Model: Progress and Future Directions. *arXiv:2311.12399*.
- Liang, J.; Zhou, P.; Zhou, W.; Qing, W.; Zhao, Q.; Wang, Z.; Song, Q.; and Li, X. 2026. SentGraph: Hierarchical Sentence Graph for Multi-hop Retrieval-Augmented Question Answering. *arXiv:2601.03014*.
- Liu, H.; Feng, J.; Kong, L.; Liang, N.; Tao, D.; Chen, Y.; and Zhang, M. 2024. One for All: Towards Training One Graph Model for All Classification Tasks. *arXiv:2310.00149*.
- Luo, L.; Ju, J.; Xiong, B.; Li, Y.-F.; Haffari, G.; and Pan, S. 2025. Chatrule: Mining logical rules with large language models for knowledge graph reasoning. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 314–325. Springer.
- Luo, L.; Li, Y.-F.; Haffari, G.; and Pan, S. 2024. Reasoning on Graphs: Faithful and Interpretable Large Language Model Reasoning. *arXiv:2310.01061*.
- Ma, X.; Gong, Y.; He, P.; Zhao, H.; and Duan, N. 2023. Query rewriting in retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 5303–5315.
- Mavromatis, C.; and Karypis, G. 2024. GNN-RAG: Graph Neural Retrieval for Large Language Model Reasoning. *arXiv:2405.20139*.

- Perozzi, B.; Fatemi, B.; Zelle, D.; Tsitsulin, A.; Kazemi, M.; Al-Rfou, R.; and Halcrow, J. 2024. Let Your Graph Do the Talking: Encoding Structured Data for LLMs. arXiv:2402.05862.
- Qian, C.; Tang, H.; Yang, Z.; Liang, H.; and Liu, Y. 2023. Can Large Language Models Empower Molecular Property Prediction? arXiv:2307.07443.
- Reimers, N.; and Gurevych, I. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, 3982–3992.
- Shi, W.; Min, S.; Yasunaga, M.; Seo, M.; James, R.; Lewis, M.; Zettlemoyer, L.; and Yih, W.-t. 2024. Replug: Retrieval-augmented black-box language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 8371–8384.
- Sun, F.; Zheng, T.; Kolekar, A.; Patki, R.; Khazaei, H.; Guo, X.; Cai, Z.; Liu, D. C.; Li, R.; Huang, Y.; Everaert, D.; Lu, H.; and Cheng, G. P. M. 2024. A Product-Aware Query Auto-Completion Framework for E-Commerce Search via Retrieval-Augmented Generation Method. In *IR-RAG@SIGIR*.
- Sun, S.; Ren, Y.; Chen, J.; and Ma, C. 2025. Large Language Models as Topological Structure Enhancers for Text-Attributed Graphs. arXiv:2311.14324.
- Talmor, A.; and Berant, J. 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)*, 641–651.
- Taunk, D.; Khanna, L.; Kandru, S. V. P. K.; Varma, V.; Sharma, C.; and Tapaswi, M. 2023. Grapeqa: Graph augmentation and pruning to enhance question-answering. In *Companion Proceedings of the ACM Web Conference 2023*, 1138–1144.
- Thoppilan, R.; Freitas, D. D.; Hall, J.; Shazeer, N.; Kulshreshtha, A.; Cheng, H.-T.; Jin, A.; Bos, T.; Baker, L.; Du, Y.; Li, Y.; Lee, H.; Zheng, H. S.; Ghafouri, A.; Menegali, M.; Huang, Y.; Krikun, M.; Lepikhin, D.; Qin, J.; Chen, D.; Xu, Y.; Chen, Z.; Roberts, A.; Bosma, M.; Zhao, V.; Zhou, Y.; Chang, C.-C.; Krivokon, I.; Rusch, W.; Pickett, M.; Srinivasan, P.; Man, L.; Meier-Hellstern, K.; Morris, M. R.; Doshi, T.; Santos, R. D.; Duke, T.; Soraker, J.; Zevenbergen, B.; Prabhakaran, V.; Diaz, M.; Hutchinson, B.; Olson, K.; Molina, A.; Hoffman-John, E.; Lee, J.; Aroyo, L.; Rajakumar, R.; Butryna, A.; Lamm, M.; Kuzmina, V.; Fenton, J.; Cohen, A.; Bernstein, R.; Kurzweil, R.; Aguerre-Arcas, B.; Cui, C.; Croak, M.; Chi, E.; and Le, Q. 2022. LaMDA: Language Models for Dialog Applications. arXiv:2201.08239.
- Tian, Y.; Song, H.; Wang, Z.; Wang, H.; Hu, Z.; Wang, F.; Chawla, N. V.; and Xu, P. 2024. Graph neural prompting with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 19080–19088.
- Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; Bikel, D.; Blecher, L.; Ferrer, C. C.; Chen, M.; Cucurull, G.; Esiobu, D.; Fernandes, J.; Fu, J.; Fu, W.; Fuller, B.; Gao, C.; Goswami, V.; Goyal, N.; Hartshorn, A.; Hosseini, S.; Hou, R.; Inan, H.; Kardas, M.; Kerkez, V.; Khabsa, M.; Kloumann, I.; Korenev, A.; Koura, P. S.; Lachaux, M.-A.; Lavril, T.; Lee, J.; Liskovich, D.; Lu, Y.; Mao, Y.; Martinet, X.; Mihaylov, T.; Mishra, P.; Molybog, I.; Nie, Y.; Poulton, A.; Reizenstein, J.; Rungta, R.; Saladi, K.; Schelten, A.; Silva, R.; Smith, E. M.; Subramanian, R.; Tan, X. E.; Tang, B.; Taylor, R.; Williams, A.; Kuan, J. X.; Xu, P.; Yan, Z.; Zarov, I.; Zhang, Y.; Fan, A.; Kambadur, M.; Narang, S.; Rodriguez, A.; Stojnic, R.; Edunov, S.; and Scialom, T. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288.
- Wang, H.; Feng, S.; He, T.; Tan, Z.; Han, X.; and Tsvetkov, Y. 2023a. Can language models solve graph problems in natural language? *Advances in Neural Information Processing Systems*, 36: 30840–30861.
- Wang, K.; Duan, F.; Wang, S.; Li, P.; Xian, Y.; Yin, C.; Rong, W.; and Xiong, Z. 2023b. Knowledge-Driven CoT: Exploring Faithful Reasoning in LLMs for Knowledge-intensive Question Answering. arXiv:2308.13259.
- Wu, S.; Fei, H.; Qu, L.; Ji, W.; and Chua, T.-S. 2024. Nextgpt: Any-to-any multimodal llm. In *Forty-first International Conference on Machine Learning*.
- Yan, Y.; Kolekar, A.; Genc, S.; Xu, W.; Huang, E. W.; Srinivasan, A.; Jain, M.; He, Q.; and Tong, H. 2025. To answer or not to answer (TAONA): A robust textual graph understanding and question answering approach. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, 6360–6376.
- Yasunaga, M.; Ren, H.; Bosselut, A.; Liang, P.; and Leskovec, J. 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.
- Yoon, M.; Koh, J. Y.; Hooi, B.; and Salakhutdinov, R. 2023. Multimodal Graph Learning for Generative Tasks. arXiv:2310.07478.
- Yu, W.; Iter, D.; Wang, S.; Xu, Y.; Ju, M.; Sanyal, S.; Zhu, C.; Zeng, M.; and Jiang, M. 2023. Generate rather than Retrieve: Large Language Models are Strong Context Generators. arXiv:2209.10063.
- Zhang, R.; Han, J.; Liu, C.; Gao, P.; Zhou, A.; Hu, X.; Yan, S.; Lu, P.; Li, H.; and Qiao, Y. 2024. LLaMA-Adapter: Efficient Fine-tuning of Language Models with Zero-init Attention. arXiv:2303.16199.
- Zhao, J.; Zhuo, L.; Shen, Y.; Qu, M.; Liu, K.; Bronstein, M.; Zhu, Z.; and Tang, J. 2023. GraphText: Graph Reasoning in Text Space. arXiv:2310.01089.
- Zhao, W. X.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; Du, Y.; Yang, C.; Chen, Y.; Chen, Z.; Jiang, J.; Ren, R.; Li, Y.; Tang, X.; Liu,

Z.; Liu, P.; Nie, J.-Y.; and Wen, J.-R. 2026. A Survey of Large Language Models. arXiv:2303.18223.

Zheng, H. S.; Mishra, S.; Chen, X.; Cheng, H.-T.; Chi, E. H.; Le, Q. V.; and Zhou, D. 2024. Take a Step Back: Evoking Reasoning via Abstraction in Large Language Models. arXiv:2310.06117.

Zhu, D.; Chen, J.; Shen, X.; Li, X.; and Elhoseiny, M. 2023. MiniGPT-4: Enhancing Vision-Language Understanding with Advanced Large Language Models. arXiv:2304.10592.

A Additional Dataset Details

A.1 Datasets

Datasets and Metrics. To evaluate the effectiveness of our method on textual QA tasks, we conduct experiments on three textual graph question answering datasets, ExplaGraphs and WebQSP from the GraphQA benchmark (He et al. 2024), and CWQ (Talmor and Berant 2018). Specifically, (1) ExplaGraphs is developed for generative commonsense reasoning, with a specific focus on the construction of explanation graphs for stance prediction in debates. By providing detailed and unambiguous commonsense-augmented graphs, it enables an evaluation of whether arguments either support or refute a specific belief. ExplaGraphs’s primary task is to determine whether the arguments are supportive or contradictory. Its node denotes commonsense concepts and the edge denotes Commonsense relations. In this dataset, the gold answer is a binary label (i.e., ‘support’ or ‘counter’), which does not provide explicit supervision for the retriever. Therefore, we leverage the LLaMA2-Chat-7B model (Touvron et al. 2023) to generate pseudo gold reasoning node sets, and treat these annotated nodes as pseudo-label supervision $\mathcal{V}^*$ for training the learnable retriever. Due to the small graph size of the graphs in ExplaGraphs, we directly use the entire graph as the candidate context for retrieval. Following GraphQA (He et al. 2024), we evaluate the end task with **Accuracy**. (2) WebQSP is a large-scale multi-hop knowledge graph QA dataset that consists of 4,737 questions. The primary task is to answer questions that necessitate multi-hop reasoning. It employs a subset of Freebase (Bollacker et al. 2008), with a specific focus on facts within 2 hops of the entities identified in the questions. The nodes denote Entities in Freebase and the edges denote Relations in Freebase. The gold answers are lists of Freebase entities, which naturally correspond to node lists and can directly serve as $\mathcal{V}^*$ without requiring additional annotation. For metrics, we report Hit@1, counting a prediction as correct if the generated answer matches any of the answers in the gold answer list. (3) CWQ (ComplexWebQuestions) is a compositional KGQA dataset constructed from WebQSP by generating more complex queries and rewriting them into natural language (Talmor and Berant 2018). It contains 34,689 questions. Compared with WebQSP, CWQ contains more complex questions and longer reasoning chains. Like WebQSP, CWQ is grounded in Freebase, where nodes denote entities and edges denote relations. The gold answers are lists of Freebase entities and can therefore directly serve as $\mathcal{V}^*$. We evaluate CWQ with **Hit@1**. Dataset statistics are summarized in Ta-

ble 5. See Appendix A.2 for Pseudo-Gold Nodes Annotation for ExplaGraphs.

A.2 Pseudo-Gold Nodes Annotation for ExplaGraphs

ExplaGraphs provides gold supervision only as a stance label (support/counter), rather than node-level evidence annotations. To make our learnable retriever pseudo label supervision applicable to this dataset, we introduce a lightweight preprocessing step on the training and validation splits: we use Llama-2-7b-chat to annotate a set of pseudo-gold reasoning nodes for each example. Concretely, we provide the model with (i) the full textual explanation graph, (ii) the two arguments, and (iii) the gold stance label, and prompt it to select nodes from the graph that form the critical reasoning path for deciding whether the arguments support or counter each other. The selected nodes are then used as $\mathcal{V}^*$ in our weighted pseudo-label objective for training the retriever. This annotation is used only to train our retriever: the generator is still fine-tuned with the original task supervision (predicting support vs. counter), and at inference time the pipeline does not require access to any node annotations.

We further conduct two sanity checks on these pseudo-gold node annotations. First, we manually inspected LLM annotations and found these LLM-selected nodes to be broadly consistent with human judgment. For example, for the argument pair “*Women should be allowed in military combat.*” and “*Females as soldiers should be acceptable.*”, whose gold label is *support*, the annotator selects critical nodes including *strength*, *military combat*, and *females* as “pseudo-gold” nodes, which we found to be reasonable and aligned with the human interpretation of key evidence needed to justify the support relation.

Second, we perform a cross-annotator consistency check using GPT-4o-mini. Specifically, we re-annotate the ExplaGraphs dataset with GPT-4o-mini using the same annotation prompt as used for Llama-2-7b-chat, and compare the resulting node sets. We report two simple agreement statistics. (1) **Any-overlap rate**: in 80.44% of graph instances, the two annotators share at least one selected node. (2) **Reference coverage**: when treating the original Llama-2-7b-chat annotations as the reference, the GPT-4o-mini annotations cover 83.75% of those nodes on average. These results suggest that the pseudo-gold node annotations are not arbitrary: a second annotator recovers most of their core evidence, while manual inspection further indicates that the selected nodes are reasonable for stance prediction. These sanity checks suggest that the Llama-2-7b-chat pseudo-gold annotations are reasonably consistent and provide useful supervision for training the ExplaGraphs retriever.

B Implementation Details

Preprocessing: Candidate-subgraph construction. For WebQSP and CWQ, we construct the PCST candidate subgraph with $k_v=k_e=10$. Because the graphs in ExplaGraphs are small, we directly use the full graph as the candidate context.

Dataset	ExplaGraphs	WebQSP	CWQ
#Graphs	2,766	4,737	34,689
Average #Nodes	5.17	1370.89	1255.71
Average #Edges	4.25	4252.37	4000.86
Node Attribute	Commonsense concepts	Entities in Freebase	Entities in Freebase
Edge Attribute	Commonsense relations	Relations in Freebase	Relations in Freebase
Task	Commonsense reasoning	KGQA	KGQA
Evaluation metrics	Accuracy	Hit@1	Hit@1

Table 5: Statistics of datasets.

Retriever fine-tuning. We fine-tune the retriever with LoRA under our token-weighted cross-entropy objective. Given a question and a candidate subgraph, we construct weighted pseudo labels over candidate nodes: nodes overlapping with gold answer nodes are treated as strong positives with weight 1.0, while the remaining candidate nodes are treated as weak positives. The weak-positive weight is set to $\lambda = 0.001$ for WebQSP and $\lambda = 0.01$ for ExplaGraphs and CWQ. The training target is a single-line, pipe-separated node list, formed by concatenating strong positives and weak positives using the delimiter “|” (strong positives first). We deduplicate node names in both retriever outputs and the contexts later used for generator training.

We use LoRA with rank $r=8$, scaling factor $\alpha=16$, and dropout 0.05. Retriever training uses 3 epochs, learning rate 5×10^{-5} , warmup ratio 0.03, per-device batch size 1 with gradient accumulation 8.

Generator fine-tuning. We fine-tune the generator with LoRA using standard supervised fine-tuning on retrieved textual graph contexts and gold targets. We use the same LoRA configuration ($r=8$, $\alpha=16$, dropout 0.05) for generator fine-tuning.

Computing infrastructure. The experiments were run on a server equipped with eight NVIDIA A100-SXM4 GPUs, each with 80 GB of memory. Each experimental run used a single GPU. The server contained two Intel Xeon Gold 6448Y processors (64 physical cores and 128 hardware threads in total) and 1 TiB of system memory, and ran Ubuntu 22.04.5 LTS. We used Python 3.10.19, NVIDIA driver 535.183.06, PyTorch 2.5.1 with CUDA runtime 12.1 and cuDNN 9.1, Transformers 4.57.1, PEFT 0.17.1, Accelerate 1.11.0, and NumPy 2.1.2.

C Sensitivity Analysis and Ablation of the Weighted Objective

We provide two complementary analyses on WebQSP. First, we study the sensitivity of our method to the weak-positive weight λ and the PCST candidate subgraph size parameters k_v and k_e . In the λ sweep, the setting $\lambda = 1$ also serves as an equal-weight ablation, since strong and weak positives receive the same training weight.

Sensitivity to the weak-positive weight λ . We vary λ while fixing the PCST input to $k_v = k_e = 10$. Table 6 reports both downstream QA performance (Hit@1) and the average number of retrieved nodes on the test set, where a smaller number indicates better compactness. Across all tested val-

λ	Hit@1 (%)	Avg. #Retrieved Nodes
0	76.23	10.19
1e-5	76.66	10.38
0.001	76.97	12.79
0.1	76.23	22.42
1.0	75.55	25.25

Table 6: Sensitivity analysis on the weak-positive weight λ on WebQSP, with PCST input fixed to $k_v = k_e = 10$. We report Hit@1 and the average number of retrieved nodes on the test set; smaller retrieved node sets indicate better compactness. Larger λ values lead to less compact retrieval.

PCST input $k_v = k_e$	Hit@1 (%)
3	74.51
5	75.43
10	76.97

Table 7: Sensitivity analysis on the PCST candidate subgraph size on WebQSP, with λ fixed to 0.001.

ues of λ , downstream Hit@1 ranges from 75.55% to 76.97%, consistently exceeding the strongest baseline reported in the main paper, G-Retriever w/ LoRA (73.79%). Our default setting, $\lambda = 0.001$, achieves the highest Hit@1 of 76.97% while retaining a compact output of 12.79 nodes on average. At the same time, the average number of retrieved nodes increases consistently as λ becomes larger, which is expected since larger weak-positive weights encourage the retriever to retain more weakly related candidate nodes. As λ increases beyond 0.001, the retrieved node sets become progressively larger without improving downstream performance. In particular, the equal-weight setting $\lambda = 1$ reduces Hit@1 to 75.55% and increases the average number of retrieved nodes from 12.79 to 25.25. These results support assigning weak positives a substantially smaller weight than strong positives, rather than weighting the two groups equally.

Sensitivity to PCST candidate subgraph size. We next vary the PCST input size $k_v = k_e$ while fixing $\lambda = 0.001$. As shown in Table 7, downstream QA Hit@1 increases from 74.51% to 76.97% as the candidate-subgraph size increases. This behavior is expected because a larger PCST subgraph provides a higher-recall candidate pool from which the learned retriever can select evidence.

D Discussion of Answer-Aware or Answer-Related Methods and Additional Recent Baselines

Table 8 summarizes several key distinctions between TAL-ENT and recent related baselines. Here, in this comparison, *LLM-based end-to-end* refers to a unified retrieve-then-generate pipeline that pairs a learnable *LLM-based retriever* for graph evidence selection with a fine-tuned *LLM-based generator* (i.e., the retriever itself is made a trainable LLM, rather than only fine-tuning the downstream generator). *Coverage-compactness* indicates whether the method explicitly aims to improve answer-relevant evidence coverage while suppressing irrelevant noise. *KGQA* indicates whether the method is directly applicable, as-is, to open-ended KGQA benchmarks such as WebQSP and CWQ. *Label-prediction GraphQA* indicates whether the method is directly applicable, as-is, to graph QA tasks whose outputs are discrete labels rather than entity answers, such as ExplaGraphs-style support/counter prediction.

Answer-aware or answer-related methods. Some related methods do exploit answer-aware or answer-related signals, but they do so in task formulations that differ substantially from ours.

QA-GNN (Yasunaga et al. 2021). QA-GNN is designed for multiple-choice QA rather than our open-ended GraphQA setting. Given a question and an answer choice $a \in C$, it forms a QA context $[q; a]$, links entities mentioned in the question and answer choice to a KG, and performs joint LM+GNN reasoning over a constructed subgraph for that question-choice pair. Therefore, QA-GNN is conditioned on *candidate answer choices*, rather than using dataset gold answers to supervise a retriever. Its main focus is answer-choice scoring in multiple-choice QA, not training a retriever for open-ended graph QA generation. Accordingly, it does not satisfy our *LLM-based end-to-end* criterion, does not explicitly optimize the *coverage-compactness* trade-off, and is not directly applicable as-is to either open-ended KGQA benchmarks such as WebQSP/CWQ or label-prediction graph QA tasks such as ExplaGraphs.

GNN-RAG (Mavromatis and Karypis 2024). GNN-RAG targets KGQA. It uses training question-answer pairs to supervise a GNN that scores answer candidates over a dense KG subgraph, and then retrieves the shortest paths connecting question entities to those candidates as context for a tuned LLM. Thus, GNN-RAG does use answer supervision during training, but in the form of answer-candidate scoring and path retrieval for KGQA, rather than weighted pseudo-label supervision for graph evidence selection. In particular, it does not contain a trainable *LLM-based retriever*, nor does it explicitly optimize the *coverage-compactness* trade-off. It is directly applicable to KGQA, but not to label-prediction graph QA tasks such as ExplaGraphs, whose outputs are discrete labels rather than KG entities.

Other recent graph retrieval baselines. We also discuss several other recent graph retrieval methods that are useful

points of comparison but differ substantially in task setting and retrieval target.

RoG (Luo et al. 2024). RoG is a planning-retrieval-reasoning framework tailored to KGQA. It first generates relation paths as plans, then retrieves valid reasoning paths from the KG and reasons over them with the same LLM. Although RoG jointly trains its planning and reasoning modules, its retrieval step itself is plan-guided KG path retrieval rather than a trainable *LLM-based retriever* under our definition above. Moreover, RoG is designed for KGQA where answers are expected to be KG entities reachable via KG paths. Therefore, it does not satisfy our *LLM-based end-to-end* criterion, does not explicitly optimize the *coverage-compactness* trade-off, and is not applicable as-is to label-prediction graph QA tasks such as ExplaGraphs.

CG-RAG (Hu et al. 2025). CG-RAG targets research question answering over citation graphs. Its retrieval module ranks relevant chunks by combining sparse and dense retrieval signals with graph encoding over contextual citation graphs, and the retrieved chunk subgraphs are then used for generation. This differs from our setting in both graph type and retrieval unit: CG-RAG performs chunk-level retrieval over citation graphs, rather than node-level evidence selection over a given QA graph. Accordingly, it neither provides an *LLM-based end-to-end* retriever-generator pipeline in our sense nor explicitly optimizes *coverage-compactness*, and it is not directly applicable as-is to either KGQA benchmarks such as WebQSP/CWQ or label-prediction graph QA tasks such as ExplaGraphs.

SentGraph (Liang et al. 2026). SentGraph is a sentence-level graph-based RAG framework for multi-hop QA over document collections. It constructs hierarchical sentence graphs with explicit logical dependencies and performs graph-guided sentence retrieval and path expansion. In contrast, our setting assumes a given QA graph whose nodes and edges already define the reasoning space, such as Freebase entities/relations in WebQSP/CWQ or commonsense concept graphs in ExplaGraphs. Thus, SentGraph differs from our setting in graph granularity, retrieval target, and benchmark formulation. It also does not provide an *LLM-based end-to-end* retriever-generator pipeline in our sense, does not explicitly optimize *coverage-compactness*, and is not directly applicable as-is to either KGQA benchmarks such as WebQSP/CWQ or label-prediction graph QA tasks such as ExplaGraphs.

Overall, the distinctive feature of TAL-ENT is not merely that it leverages answer-related signals, but that it combines a unified *LLM-based end-to-end* retriever-generator pipeline with weighted pseudo-label supervision, thereby addressing both the *architectural gap* and the *retrieval-supervision gap* while explicitly improving the *coverage-compactness* trade-off. Moreover, the framework applies to both open-ended KGQA benchmarks (e.g., WebQSP and CWQ) and label-prediction GraphQA tasks (e.g., ExplaGraphs).

Method	LLM-based end-to-end	Coverage-compactness	KGQA	Label-prediction GraphQA
TaLent	✓	✓	✓	✓
QA-GNN	✗	✗	✗	✗
GNN-RAG	✗	✗	✓	✗
RoG	✗	✗	✓	✗
CG-RAG	✗	✗	✗	✗
SentGraph	✗	✗	✗	✗

Table 8: Comparison of TaLent with answer-related and other recent graph retrieval methods discussed in Appendix D.

E Concrete Prompt Templates and Graph Serialization

In this section, we provide one concrete example of the prompts used in TaLent. We show the exact retriever prompt and the generator prompt, and clarify how retrieved nodes and associated edges are serialized for answer generation.

E.1 Retriever Prompt

The retriever is prompted with the question together with the initial PCST candidate subgraph S . Its task is to output nodes that are most useful for answering the question, and to output them as a single pipe-separated line. Below we show one concrete retriever prompt example.

Retriever Prompt

```
<s>[INST] <<SYS>>
You are an entity retriever for KGQA.
Your job is to SELECT CANDIDATE ENTITIES FROM THE
SUBGRAPH that would help an answerer solve the question.
CRITICAL CONSTRAINTS:
- Return lowercase names only.
- Select the most relevant nodes from the subgraph for
answering the question.

RETURN FORMAT (MANDATORY):
Your ENTIRE reply must be EXACTLY ONE SINGLE LINE
like:
name1|name2|name3
If you output anything other than the single pipe-separated
line, the answer is invalid.

Hard rules:
- Output EXACTLY one line; no greetings, no explanations,
no extra words.
- The line must contain ONLY lowercase entity names
separated by '|' (no labels like 'Output:').
- Keep spaces inside names; DO NOT add spaces around '|'.
- DO NOT output placeholders like A/B/C/unknown.

</SYS>>

Question:
where did kennedy's inaugural address take place

Subgraph (CSV between <DESC> tags):
<DESC>
node_id,node_attr
```

```
24,john f. kennedy
282,cold war
286,speech
287,m.05rrkys
295,anti-apartheid
352,m.04tnzwc
397,m.05r7b2w
428,john f. kennedy 1961 presidential inauguration
429,inaugural address of john f. kennedy
475,m.0b9zgjq
661,berlin wall
737,john fitzgerald kennedy memorial
771,john fitzgerald kennedy inaugural poem read by robert lee
frost
967,united states presidential inauguration
1008,john f. kennedy speech
1119,apollo program
1282,the kennedy presidential press conferences
1335,1957 national book awards
1376,m.0ypz1mv
1458,1957 pulitzer prize
1701,jfk inauguration

src,edge_attr,dst
24,event.public_speaker.speeches_or_presentations,397
352,award.award_honor.award_winner,24
24,award.award_nominee.award_nominations,475
428,common.topic.image,1701
24,book.author.works_written,1282
24,music.artist.track,1008
967,time.recurring_event.instances,428
24,base.kwebbase.kwtopic.connections_from,771
737,symbols.namesake.named_after,24
1376,event.speech_or_presentation.
type_or_format_of_presentation,286
287,event.speech_or_presentation.
type_or_format_of_presentation,286
397,event.speech_or_presentation.
type_or_format_of_presentation,286
1119,event.speech_topic.
speeches_or_presentations_on_this_topic,1376
295,event.speech_topic.
speeches_or_presentations_on_this_topic,287
661,event.speech_topic.
speeches_or_presentations_on_this_topic,397
282,event.speech_topic.
speeches_or_presentations_on_this_topic,397
428,base.inaugurations.inauguration.inaugural_address,429
397,event.speech_or_presentation.speech_topic,282
397,event.speech_or_presentation.speech_topic,661
```

```

24,base.inaugurations.inauguration_speaker.inauguration,428
475,award.award_nomination.ceremony,1335
352,award.award_honor.ceremony,1458
428,base.inaugurations.inauguration.president,24
287,event.speech_or_presentation.speech_topic,295
1376,event.speech_or_presentation.speech_topic,1119
</DESC>

```

Now output EXACTLY ONE SINGLE LOWERCASE LINE with '|' separators. Nothing else. [/INST]

E.2 Generator Prompt and Graph Serialization

For answer generation, we serialize the retrieved evidence in the format: nodes are first listed as a table with header `node_id,node_attr`, followed by edges as a table with header `src,edge_attr,dst`.

The serialized node set contains two parts. First, nodes with regular numeric identifiers are retrieved nodes that appear in the initial PCST candidate subgraph $\mathcal{S}$. Second, nodes whose identifiers start with the prefix `na` (e.g., `na1`, `na2`) are retrieved nodes that do not appear in the initial $\mathcal{S}$. These additional nodes are retained in the generator context as isolated nodes. After listing the nodes, we include the associated edges $\hat{\mathcal{E}}$ that appear in $\mathcal{S}$. Therefore, the generator context consists of the retrieved nodes $\hat{\mathcal{V}}$ together with their associated edges $\hat{\mathcal{E}}$, and it is not required to form one connected subgraph. Below we show one concrete generator prompt example.

Generator Prompt

```

<S>[INST] <<SYS>>
You are a KGQA answer generator.
PRIMARY GOAL: output the FINAL ANSWER entity name(s) to the question.
Use the provided subgraph as STRONG HINTS. If some correct answers are not in subgraph but answers the question, you may also include them.
Return all names in lowercase.

```

```

RETURN FORMAT (MANDATORY):
Your ENTIRE reply must be EXACTLY ONE SINGLE LINE like:
name1|name2|name3
If you output anything other than the single pipe-separated line, the answer is invalid.

```

Guidelines:

- Output ONLY answers; exclude distractors and the subject itself unless the subject IS the answer.
- If multiple entities correctly answer the question, include them all separated by '|'. Otherwise, output a single name.
- No greetings, no explanations, no labels (e.g., 'Answer:').
- Keep spaces inside names; DO NOT add spaces around '|'.
- DO NOT output placeholders (A/B/C/unknown).

```

<</SYS>>

```

Question:
where did kennedy's inaugural address take place

Subgraph (CSV between <DESC> tags):

```

<DESC>
node_id,node_attr
24,john f. kennedy
429,inaugural address of john f. kennedy
1008,john f. kennedy speech
1282,the kennedy presidential press conferences
1701,jfk inauguration
na1,washington d.c.
na2,capitol building
na3,united states
na4,1961 presidential inauguration
na5,united states capitol building

```

```

src,edge_attr,dst
428,common.topic.image,1701
24,book.author.works_written,1282
24,music.artist.track,1008
428,base.inaugurations.inauguration.inaugural_address,429
24,base.inaugurations.inauguration_speaker.inauguration,428
428,base.inaugurations.inauguration.president,24
</DESC>

```

Now output ONLY the final answer entity name(s) as EXACTLY ONE SINGLE LOWERCASE LINE with '|' separators. Nothing else. [/INST]

F Groundedness and Hallucination Analysis of Out-of-PCST Retrieval

Retrieving nodes beyond the initial PCST candidate subgraph can improve coverage, but it also raises a natural concern about groundedness: such nodes may introduce ungrounded evidence and potentially contribute to hallucinated answers. To assess this effect, we conduct an additional analysis on the WebQSP test set.

We use a practical *hit-aware hallucination proxy*, since groundedness is ultimately relevant insofar as it affects the final answer. For each test instance, let $\mathcal{S}$ denote the initial PCST candidate subgraph with node set $\mathcal{V}_{\mathcal{S}}$, let $\hat{\mathcal{V}}$ denote the retrieved node set output by the retriever, and let $\mathcal{V}^*$ denote the gold answer set. We define the hallucinated retrieved nodes as

$$\mathcal{H} = \begin{cases} \emptyset, & \text{if Hit@1} = 1, \\ \{v \in \hat{\mathcal{V}} \setminus \mathcal{V}_{\mathcal{S}} \mid v \notin \mathcal{V}^*\}, & \text{if Hit@1} = 0. \end{cases} \quad (16)$$

That is, if the final prediction is correct, we do not count retrieved nodes as hallucinated under this proxy. If the final prediction is incorrect, we count as hallucinated those retrieved nodes that are outside the PCST candidate subgraph and do not match any gold answer.

We then compute the overall hallucination rate over the test set as

$$\text{Hallucination Rate} = \frac{\sum |\mathcal{H}|}{\sum |\hat{\mathcal{V}}|}, \quad (17)$$

where the sums aggregate over all test instances.

On the WebQSP test set, the resulting hallucination rate is 4.97%. This suggests that although the trained retriever can retrieve nodes beyond PCST, such retrieval introduces only a

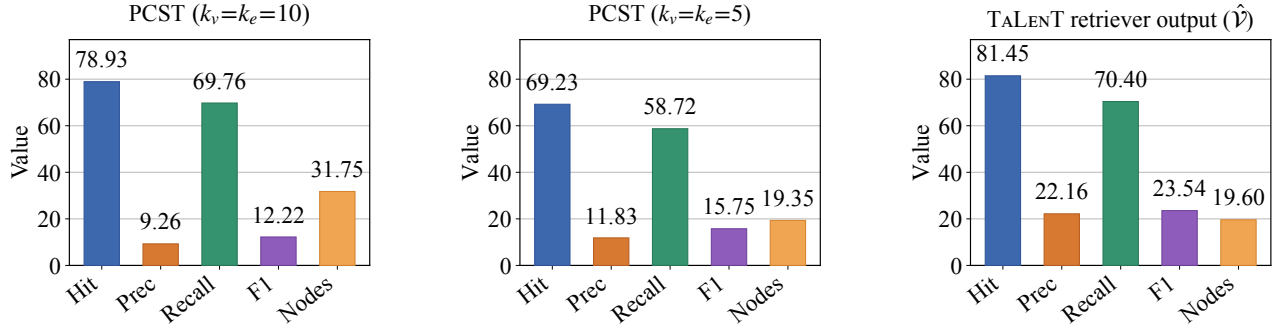


Figure 4: WebQSP test-set retrieval results with Llama-3-8B (macro-averaged). We measure how well PCST subgraphs and TALeNT’s learned retriever output $\hat{\mathcal{V}}$ cover the gold answer set $\mathcal{V}^*$. We compare: (1) the PCST candidate subgraph with $k_v=k_e=10$ (the retriever input), (2) a smaller PCST subgraph with $k_v=k_e=5$ (a size-matched reference), and (3) the node set $\hat{\mathcal{V}}$ selected by TALeNT’s learned retriever. Our retriever maintains comparable gold-answer coverage with substantially fewer nodes than the original PCST candidate subgraph ($k_v=k_e=10$), while achieving higher coverage than the size-matched PCST subgraph ($k_v=k_e=5$).

Table 9: WebQSP downstream QA results with Llama-3-8B, comparing TALeNT against the strongest baseline G-Retriever w/ LoRA in Table 1 (Hit@1, %).

Method	Hit@1
G-Retriever w/ LoRA	79.36
TALeNT	83.05

small fraction of ungrounded nodes overall. Combined with the results in Section 4.4 of the main paper, this indicates that out-of-PCST retrieval provides useful complementary coverage while maintaining a low level of hallucinated evidence.

G Backbone Ablation with Llama-3-8B

We further evaluate TALeNT with Llama-3-8B on WebQSP, comparing against G-Retriever w/ LoRA, the strongest baseline in Table 1. As shown in Table 9, TALeNT achieves 83.05% Hit@1, compared with 79.36% for G-Retriever w/ LoRA, suggesting that our framework remains effective with a stronger LLM backbone. We use $\lambda = 0.001$ for the learned retriever.

We further examine the retrieval coverage-compactness tradeoff under Llama-3-8B, as shown in Figure 4.

Compactness: comparable coverage with fewer nodes. Compared with the initial PCST candidate subgraph ($k_v=k_e=10$), TALeNT’s retriever produces a more compact node set while maintaining comparable gold-answer coverage. Specifically, the average number of nodes decreases from 31.75 to 19.60, while Hit@1 increases from 78.93% to 81.45% and Recall from 69.76% to 70.40%. Precision also improves from 9.26% to 22.16%, and F1 from 12.22% to 23.54%.

Coverage: better gold-answer coverage under a comparable node budget. We further compare against a smaller PCST subgraph with $k_v=k_e=5$, which contains 19.35 nodes on average, close to the 19.60 nodes retrieved by TALeNT. Under

Table 10: Breakdown of gold-answer coverage contributed by different parts of the learned retriever output on the WebQSP test set with Llama-3-8B ($\lambda = 0.001$). All metrics are macro-averaged over examples and reported as percentages.

Output subset	Hit@1	Recall
All retrieved nodes	81.45	70.40
Nodes within PCST	76.90	67.35
Nodes out of PCST	8.29	4.28

this comparable node budget, TALeNT improves Hit@1 from 69.23% to 81.45% and Recall from 58.72% to 70.40%. These results exhibit a similar coverage-compactness trend to the main experiments.

Retrieval beyond PCST. Table 10 further breaks down the gold-answer coverage of the learned retriever output. Nodes outside PCST alone achieve 8.29% Hit@1 and 4.28% Recall. Incorporating these nodes increases Hit@1 from 76.90% to 81.45% and Recall from 67.35% to 70.40%. This is consistent with the main-paper observations that the learned retriever can complement the PCST candidate subgraph with additional gold-answer coverage.

Overall, these results suggest that TALeNT maintains a favorable retrieval coverage-compactness tradeoff and strong downstream QA performance with Llama-3-8B.

H More Related Works

Graphs and Large Language Models A growing body of work studies how to combine graph-structured information with LLMs (Li et al. 2024; Wang et al. 2023a). Prior efforts span general graph modeling (Liu et al. 2024; Lei et al. 2023), multimodal architectures that incorporate graph-like structures (Yoon et al. 2023; Li et al. 2023). Representative applications include fundamental graph reasoning (Zhao et al. 2023), node prediction (Sun et al. 2025; Chen et al. 2024), graph-level prediction (Qian et al. 2023). A related

line of work discusses GraphRAG / GraphQA, where the system selects graph evidence and conditions an LLM on graph context to answer questions (IV et al. 2019; Luo et al. 2025).

Parameter-Efficient Fine-Tuning Parameter-efficient fine-tuning (PEFT) methods enable adapting large pretrained models with limited trainable parameters. Representative techniques include prompt tuning (Lester, Al-Rfou, and Constant 2021), prefix tuning (Li and Liang 2021), and low-rank adaptation such as LoRA (Hu et al. 2022). Related adapter-style methods (e.g., LLaMA-adapter (Zhang et al. 2024)) have also supported rapid adaptation and inspired multimodal instruction-following systems such as MiniGPT-4 (Zhu et al. 2023) and NExT-Chat (Wu et al. 2024). PEFT has recently been explored in graph+LLM settings as well, including GraphLLM (Chai et al. 2025) and GraphToken (Perozzi et al. 2024) for graph reasoning, and GNP (Tian et al. 2024) for knowledge-graph QA.