RAGRAPH: A General Retrieval-Augmented Graph Learning Framework

Xinke Jiang[♠*], **Rihong Qiu**[♠*], **Yongxin Xu**[♠*], **Wentao Zhang**[♢], **Yichen Zhu**[♢], **Ruizhe Zhang**[♠]
Yuchen Fang[♡], **Xu Chu**[♠], **Junfeng Zhao**[♠♣†], **Yasha Wang**[♠★†]
{xinkejiang, rihongqiu, xuyx, ruizhezhang}@stu.pku.edu.cn
{wentaozh2001, yichenzhu2014, fyclmiss}@gmail.com
{chu_xu, zhaojf, wangyasha}@pku.edu.cn
[♠]Key Laboratory of High Confidence Software Technologies (Peking University),
School of Computer Science, Peking University, China
[♢]No Affiliation, [♡]University of Electronic Science and Technology of China
[♣]Big Data Technology Research Center, Nanhu Laboratory, Jiaxing, China
[★]Peking University Information Technology Institute, Tianjin Binhai, China

https://github.com/Artessay/RAGraph/

Abstract

Graph Neural Networks (GNNs) have become essential in interpreting relational data across various domains, yet, they often struggle to generalize to unseen graph data that differs markedly from training instances. In this paper, we introduce a novel framework called General **R**etrieval-**A**ugmented **Graph** Learning (**RAGRAPH**), which brings external graph data into the general graph foundation model to improve model generalization on unseen scenarios. On the top of our framework is a toy graph vector library that we established, which captures key attributes, such as features and task-specific label information. During inference, the **RAGRAPH** adeptly retrieves similar toy graphs based on key similarities in downstream tasks, integrating the retrieved data to enrich the learning context via the message-passing prompting mechanism. Our extensive experimental evaluations demonstrate that **RAGRAPH** significantly outperforms state-of-the-art graph learning methods in multiple tasks such as node classification, link prediction, and graph classification across both dynamic and static datasets. Furthermore, extensive testing confirms that **RAGRAPH** consistently maintains high performance without the need for task-specific fine-tuning, highlighting its adaptability, robustness, and broad applicability.

1 Introduction

Graph Neural Networks (GNNs) [5, 48, 97, 63, 126] have recently burgeoned a surge of interest in both academic and industry communities due to their robust capability to model complex, real-world data in diverse domains, including societal [72, 55, 80], biochemical [17, 111, 107], and traffic-related [54, 23, 44, 21] fields and etc [53, 37, 68, 15, 25, 24]. Utilizing a message-passing mechanism [48, 29], GNNs have transcended traditional node embedding approaches [28, 79, 95], enabling the capture of intricate relationships within data through sophisticated architectures and advanced graph representation learning techniques [48, 50, 54, 18, 97]. However, the challenge of generalizing GNNs across different modalities, domains [62, 61], and tasks remains largely unexplored [56, 113]. This is in stark contrast to the significant successes of large models such as GPTs [74, 75] in NLP and Sora [64] in CV, presenting a crucial frontier for further research and realms for graph data generalizing.

[*]Indicates equal contribution.
[†]Yasha Wang and Junfeng Zhao are corresponding authors.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

In graph learning tasks, providing the necessary context is crucial for graph generalization [129, 51, 73, 134], *i.e.,* retrieve similar shopping context as illustrated in Figure 1 (c). Therefore, our insight is to enhance the model's generalization ability and prediction accuracy by retrieving necessary contexts during graph learning through retrieval. **Retrieval-Augmented Generation (RAG)** represents a prominent methodology, significantly augmenting language model functionalities through the integration of a dynamic retrieval mechanism during the generation process [135, 77] (*e.g.*, a person asks what animal it is, and we use some visual [138] or text retrieval [2] methods to retrieve more descriptive features or even the wanted category). RAG enriches not only accurate and reliable content but also reduces factual errors, addressing challenges such as incorrect answers, hallucinations, and limited interpretability in knowledge-intensive tasks [40, 2, 1], obviating the need for updating model parameters and could be generalized even in unseen scenarios.

However, how to enable retrieval-augmented generation for graph learning, *i.e.*, retrieving the user's historical purchasing behavior to enhance recommendation ability [30, 113, 35] and identifying fraud crimes by searching for similar fraudulent relationship behaviors [85, 63], still remains unexplored and faces the following challenges ***C1& C2***.

C1. The first challenge is how to leverage the retrieved context *i.e.,* features (X) and labels (Y) into the GNNs model under dynamic changing scenarios. Previous studies, such as PRODIGY [73], have adopted the concept of in-context learning (ICL) by constructing consistent and static task graphs for each specific task or dataset. These task graphs determine labels through the calculation of similarities using hidden vectors, employing a few-shot learning approach. However, PRODIGY's reliance on a fixed set of examples as rules may not sufficiently address and generalize the variety of scenarios encountered in real-world settings, which is particularly problematic in dynamically changing environments, as the system focuses primarily on teaching the direct mapping paradigm from inputs to outputs ($X \rightarrow Y$), rather than truly integrate the input (X) and output (Y) data into the analysis. In contrast to RAG, PRODIGY struggles to incorporate external information (X and Y) related to data nodes, which is crucial for enriching the learning process in graph-based systems.

C2. Moreover, it is challenging to develop a tune-free prompt mechanism to support retrieved knowledge and be applicable to seamlessly switch unseen scenarios and multi-tasks. Numerous initiatives have been undertaken in the realm of graph pre-training [33, 116, 34, 81, 98, 36, 7, 88, 125], however, the challenge persists in designing a plug-and-play RAG module that can seamlessly interface with already pre-trained models. Insights derived from prior investigations into the graph prompt [9, 26, 90, 65, 113, 20, 94], the knowledge obtained by RAG can be facilitated and injected into prompt via a plug-and-play manner.

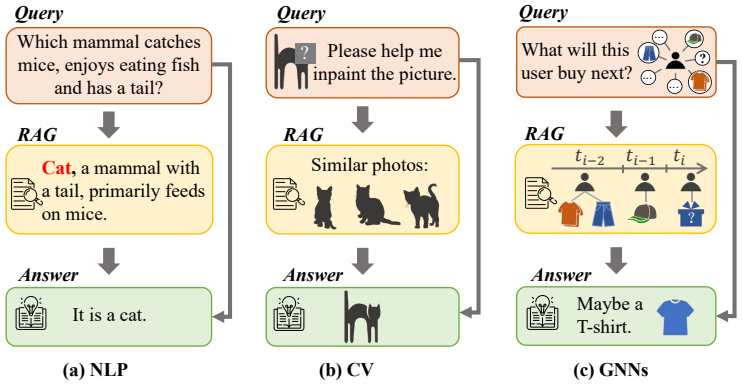


Figure 1: **(a)** RAG in NLP utilizes retrieval to enhance model responses, based on a query to retrieve related features (*e.g., a tail, primarily feeds on mice*) and answers (*e.g., Cat*). **(b)** In CV, RAG employs similar photo retrieval to enhance model comprehension, assisting in downstream tasks such as inpainting or image question answering. **(c)** For GNNs, RAG could leverage retrieval of similar historical subgraphs or scenarios to aid in graph-based tasks (*e.g.,* recommendations or fraud detection).

For endeavoring to address these two challenges previously mentioned, we put forward the General **R**etrieval- **A**ugmented **Graph** Learning Framework (**RAGRAPH**). Drawing inspiration from the success of RAG on LLMs [135] and the ICL on GNNs [73] (we detail the difference between RAG and ICL in Appendix E), we constructed a toy graphs vector library by chunking from resource graphs, where the library key stores key information, including environmental, historical, structural, and semantic details, while node features and label information (task-specific output vector) are stored as values. For downstream tasks, the key value of the query node would be leveraged to retrieve toy graphs by the key similarities,

and the stored features (X) and labels (Y) would be aggregated structurally to provide essential knowledge to the query node, instead of the mapping paradigm, to address challenge ***C1***. In prompt mechanism design, we start by transferring features and task-specific output from the toy graphs to their master nodes (the central node of the toy graph) via message-passing. Subsequently, features from the master nodes and the query node's neighbors are aggregated to the query node, along with the task-specific output from master nodes. This process could be parameter-free, indicating that our model can be applied across different tasks and datasets without the need to fine-tune for downstream tasks, effectively addressing challenge ***C2***.

In summary, our contributions are listed as follows:

- To the best of our knowledge, our proposed framework, RAGRAPH, is the first to integrate RAG with pre-trained GNNs. By constructing a key-value vector library for toy graphs, RAGRAPH facilitates explicit plug-and-play access to pre-trained GNNs, achieving commendable performance even without fine-tuning, demonstrating its superiority on cross-task and cross-dataset capabilities.
- Our RAGRAPH employs a classic message-passing mechanism and introduces a well-designed prompt mechanism to integrate knowledge. This approach effectively incorporates the retrieved knowledge X and Y from toy graphs, into the pre-trained GNNs model, enhancing the accuracy and relevance of the model's outputs.
- We have extensively tested RAGRAPH on both static and dynamic graphs across multiple levels of graph tasks (node, edge, and graph). The results validate the effectiveness of our model, showing significant improvements over state-of-the-art baselines in both fine-tuned and tuning-free scenarios, particularly in cross-dataset validations.

2 Related Work

2.1 Retrieval-Augmented Generation on Large Language Models

RAG integrates an external knowledge retrieval component and through prompt engineering into pre-trained language models to enhance factual consistency, thus improving the reliability and interpretability of LLM responses [131, 135, 49, 22, 43, 118, 57, 110, 127]. Traditional RAG approaches utilize retriever models to source relevant documents from extensive knowledge corpora [106, 82, 69, 47], which are then processed further by reader models—primarily LLMs [76, 84]. Furthermore, several studies focus on fine-tuning reader LLMs by applying prompt-tuning with retrieved knowledge or using RAG API calls [67, 40, 2, 115, 101, 128, 60]. While RAG has seen considerable success in the NLP field, it has also been applied to tasks involving joint visual and text retrieval [138, 59, 58, 8, 124, 10], code retrieval [66, 133], audio retrieval [6, 31] and video retrieval [3, 100]. Although there have been applications of RAG on structured data such as KG-RAG for knowledge graphs [46, 43, 86, 92, 93, 38], these primarily leverage the text information of knowledge graph nodes to enhance language or graph models. In contrast, there are no significant studies utilizing RAG on structured graphs without text information to enhance pre-trained GNNs. Our work aims to extend this successful approach similarly to graph data, to enhance the capabilities of pre-trained GNNs, and can be adapted to various tasks and across different graphs without additional fine-tuning by integrating a plug-and-play RAG module.

2.2 Graph Prompt Learning

Inspired by the application of pre-training models [74, 75] and prompt learning [102, 133, 41] in NLP, recently, learning on the graph has been divided into pre-training models on large-scale graph data [33, 116, 34, 81, 73, 130, 104, 89, 119, 121, 122, 120, 123], with or without labels, followed by fine-tuning model parameters via prompts for diverse downstream tasks [65, 113, 73, 137, 89, 94]. The adoption of prompting mechanisms in graph learning represents a promising avenue to overcome the constraints of traditional graph representation methods, striking a balance between flexibility and expressiveness [91]. For instance, VNT [94] utilizes virtual nodes as prompts to refine the application of pre-trained graph models. GraphPrompt [65] introduces a task-specific readout mechanism to tailor models for various tasks, while GraphPro [113] implements spatial- and temporal-based gating mechanisms suited for dynamic recommendation systems. Furthermore, PRODIGY [73] constructs task graphs (prompts) and data graphs to enhance the model's ICL capabilities. Leveraging the successes in graph prompt learning, we aim to inject retrieved knowledge via prompt into pre-trained GNNs to support downstream tasks.

3 Preliminaries

In RAGRAPH, we focus on RAG on multi-level graph tasks. For consistency, we define the graphs as dynamic graphs, considering static graphs as the special cases within this framework. The subsequent definition provides a detailed description of toy graphs, including the definitions of keys and values utilized in RAGRAPH. Additionally, inspired by GraphPrompt [65], we have unified node-level, edge-level, and graph-level tasks into a cohesive framework, and employ query graphs to tackle downstream tasks with precision.

Definition 1. (Dynamic Graph) Let $\mathcal{G}=\{G_t\}_{t=1}^{T}$ denote a dynamic graph comprising a sequence of graph snapshots, each represented as a static graph $G_t=(V_t,E_t,X_t,A_t,Y_t)$. $\mathcal{V}=\bigcup_{t=1}^{T}V_t=\{v_1,...,v_n\}$ defines the combined set of nodes across all snapshots and $\mathcal{E}=\bigcup_{t=1}^{T}E_t\subseteq\mathcal{V}\times\mathcal{V}$ is the edge set, where V_t and E_t represent the nodes and edges of the t-th snapshot, respectively. Feature matrix $X_t=\{x_v\mid v\in\mathcal{V}\}\in\mathbb{R}^{n\times d}$ contains the feature vectors for the nodes in the t-th snapshot, where d is the feature dimension. A_t denotes the edge weight matrix at time t, where edge weight $A_t[i,j]\in(0,1]$ if $v_i,v_j\in V_t$ and $(v_i,v_j)\in E_t$, and 0 otherwise. Furthermore, Y_t represents the task-related labels associated with nodes, edges, or the graph at time t. Note that a graph is static if $T=1$ and for consistency in terminology, we unify static graphs as a particular instance of dynamic graphs.

Definition 2. (Toy Graph Vector Base) Let $\mathcal{G}^{\mathcal{R}}=\{G_t^{\mathcal{R}}\}_{t=1}^{T}$ denote a dynamic resource graph. We chunk $\mathcal{G}^{\mathcal{R}}$ into snapshots and take each node in $\mathcal{G}^{\mathcal{R}}$ as the master node v_m of the corresponding toy graph, and then store v_m with its neighbors within k hops as subgraphs. Data augmentation techniques [54, 132] such as node dropout, edge dropout, and random noise addition are employed on subgraphs to enhance the robustness and variability when generating each toy graph $G^{\mathcal{T}}$ (c.f. Section 4.1 for details). Each toy graph $G^{\mathcal{T}}\subseteq\mathcal{G}^{\mathcal{R}}$ is associated with a specific timestamp τ and master node $v_m\in\mathcal{V}$, with each toy graph's scale being considerably smaller in scale compared to their corresponding $\mathcal{G}^{\mathcal{R}}$. ❶ Toy graphs can be retrieved using **keys** that include the timestamp τ, the hidden embedding of the master node $h_m^{\tau}\in\mathbb{R}^{f_1}$ (*e.g.,* embedded by pre-trained GNNs in RAGRAPH), the environmental key (*e.g.,* the neighbors set $\mathcal{N}(v_m^{\tau})=\{v_i^{\tau}\mid A_{\tau}[m,i]>0,v_i^{\tau}\in G^{\mathcal{T}}\}$) and the structure-based position-aware code s_m^{τ} (cf. Appendix C.2 for details). ❷ By retrieving based on key similarity (c.f. Section 4.2 for details), we can obtain the required **values** of $G^{\mathcal{T}}$, *i.e.* task-specific output vector $\{o_i^{\tau}\in\mathbb{R}^{f_2}|v_i\in G^{\mathcal{T}}\}$ and hidden embeddings $\{h_i^{\tau}\in\mathbb{R}^{f_1}|v_i\in G^{\mathcal{T}}\}$ of the master node and its neighbors, where f_1 and f_2 represent the dimensions. Finally, we denote the **key-value** vector base for the toy graph as $\mathcal{G}^{\mathcal{T}}$.

Definition 3. (A Unified Graph Task Definition) Given a dynamic graph $\mathcal{G}$, it can be divided into training and testing subsets, *i.e.* $\mathcal{G}=\mathcal{G}_{\text{train}}\cup\mathcal{G}_{\text{test}}$ based on either snapshot or node set partitioning. The label y_i of a node v_i, edge (v_i,v_j) or subgraph G_i can be observed only if they belong to $\mathcal{G}_{\text{train}}$. The objective of label prediction is to predict test labels $Y_{\text{test}}\in\mathcal{G}_{\text{test}}$. Following GraphPrompt [65], we unify the three types of graph learning tasks (node-level, edge-level, and graph-level) into a single framework via similarity comparison $\texttt{sim}(\cdot,\cdot)$ of the task-specific output vector (abbreviated as O, where each entry is o) with the ground-truth (*i.e.,* the one-hot vector or the prototype embedding under few-shot setting). It's noted that o can be either low-dimensional (with the dimension equal to the number of predicted classes) under normal settings [48, 126], or high-dimensional under few-shot settings [65] or in link prediction tasks [113, 30]. In our experiment, ❶ for node-level and graph-level tasks, the downstream tasks are given in few-shot settings following [65]: For node / graph classification on a node / graph set, let $\mathcal{C}$ be the set of classes with $y_i\in\mathcal{C}$ denoting the class label of node / graph. For each node / graph class, the class prototypical output vector is calculated by the mean value of the κ-shot set $\mathcal{D}$: $\tilde{o}_c=\frac{1}{\kappa}\sum_{(i,y_i)\in\mathcal{D},y_i=c}o_i$. The class y_i of the node or graph is determined by calculating similarity with the class prototype as: $y_i=\text{argmax}_{c\in\mathcal{C}}\texttt{sim}(o_i,\tilde{o}_c)$. ❷ For edge-level tasks, to predict a link between nodes v_i and v_q, if $\exists v_j,(v_i,v_j)\in\mathcal{E}_{\text{train}}\in\mathcal{G}_{\text{train}}$ and $\texttt{sim}(o_i,o_q)\geq\texttt{sim}(o_i,o_j)+\epsilon$, we regard (v_i,v_q) as linked. Following PRODIGY [73] and GraphPrompt [65], we also apply a query graph $G^{\mathcal{Q}}$ that includes the center node and its neighbors within k hops. Specifically, for graph-level task, we apply a full-link virtual node as the center node inside the query graph $G^{\mathcal{Q}}$.

4 RAGRAPH Framework

In this section, we introduce RAGRAPH, a general and novel retrieval-augmented graph learning framework that can operate on arbitrary graphs with or without additional fine-tuning, as illustrated in Figure 2. Initially, in Section 4.1, we elucidate the methodology for constructing the Resource Toy Graphs. Subsequently, in Section 4.2 we detail the Toy Graphs Retrieval Process. Finally, the Training and Inference processes are elaborated in Section 4.3, which utilize retrieved toy graphs from two propagation views—intra and inter-propagation—and handle two types of information: hidden embeddings and task-specific output vectors in two techniques (noisy trainable approach or parameter-free approach). The main notations of RAGRAPH are summarized in Table 3, Appendix A. For enhanced clarity, the Toy Graph Construction is outlined in Algorithm 1 (cf. Appendix C.5) and the Training and Inference with Toy Graphs Retrieval are detailed in Algorithm 2 (cf. Appendix C.5). Moreover, in Appendix C.4, we theoretically prove the effectiveness of applying RAG on GNNs from the perspective of mutual information gain.

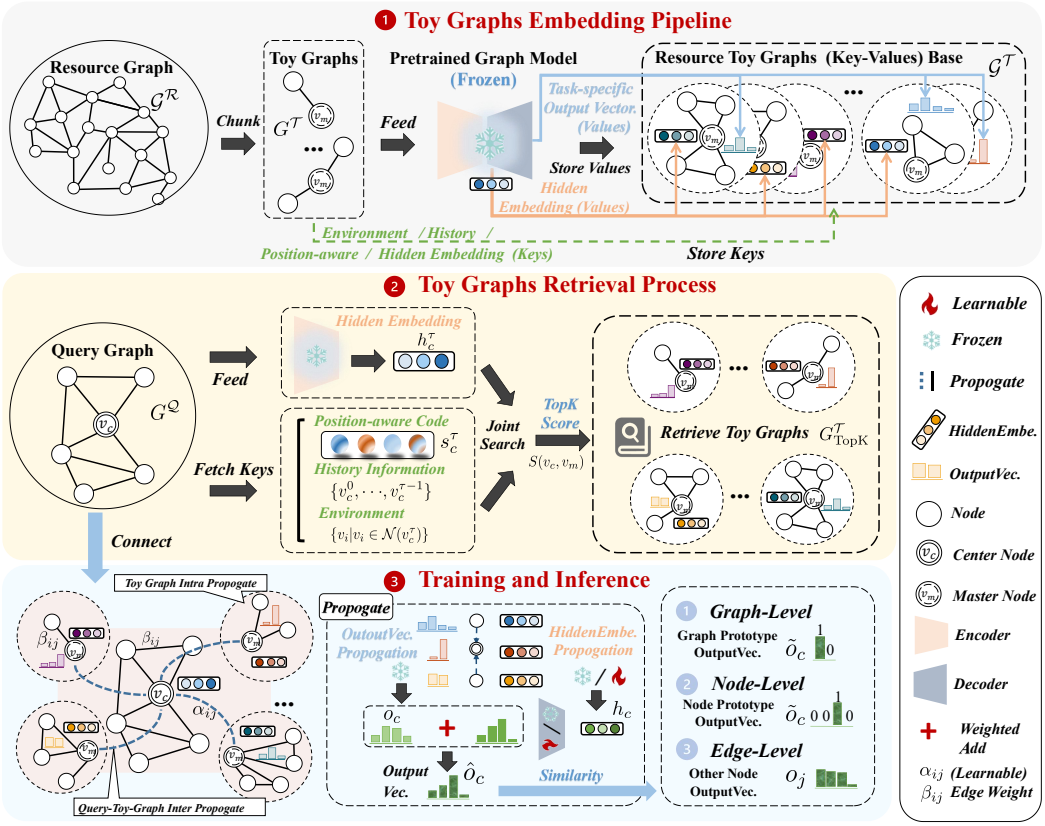


Figure 2: The overall framework of RAGRAPH. ❶ Given resource graph $\mathcal{G}^{\mathcal{R}}$, we chunk it and augment toy graphs $\{G^{\mathcal{T}}\}$, and feed them into pre-trained GNNs to generate hidden embeddings via the encoder and task-specific output vectors via decoder, which are stored as values. Keys such as environment, history, position-aware, and hidden embeddings are stored to form the key-value database of toy graphs $\mathcal{G}^{\mathcal{T}}$. ❷ For a given query graph $G^{\mathcal{Q}}$, the keys are fetched to retrieve the $topK$ toy graphs $G^{\mathcal{T}}_{\text{topK}}$ from the database. ❸ Leveraging $G^{\mathcal{T}}_{\text{topK}}$, intra- and inter-propagation are performed to propagate hidden embeddings and task-specific output vectors to pass retrieved knowledge to center node v_c. Through a weighted fusion, the aggregated output is used to perform graph-, node- and edge-level tasks.

4.1 Toy Graphs Embedding Pipeline

In graph-based learning, nodes with higher connectivity—typically with higher degrees—often hold more significance, meaning their information is more extensively learned during graph-pre-training processes. Conversely, less important nodes—those in the long tail—often have their features over-

looked. This issue is particularly pronounced in LLMs performing RAG, where the predominance of common knowledge overshadows the long-tail knowledge that RAG is meant to leverage. To tackle this, we construct toy graphs using an inverse importance sampling strategy, thereby countering this bias by preferentially sampling and augmenting toy graphs that accentuate the long-tail knowledge.

Inverse Importance Sampling Strategy. To achieve this, we calculate each node's importance $I(v)$ for node $v \in G_\tau^{\mathcal{R}}$ by combining PageRank $\text{PR}(v)$ and Degree Centrality $\text{DC}(v)$ using the formula $I(v) = \alpha\text{PR}(v) + (1-\alpha)\text{DC}(v)$, where $\alpha \in (0,1)$ is the balance weight. We reverse the node importance with $I'(v) = \frac{1}{I(v)+\epsilon}, \epsilon \to 0$, normalize it to obtain node v_i 's sampling probabilities $p_i = \frac{I'(v_i)}{\sum_{j=1}^{n} I'(v_j)}$, and perform weighted sampling function $\text{WEIGHTEDSAMPLING}(G_\tau^{\mathcal{R}}, p_i)$ to prioritize nodes with higher sampling probability (lower importance) according to p_i. When sampling, for each master node v_m, we generate its k-hop neighbors, termed an ego net $G_\tau^e(v_m)$. Given the constrained size of the resource graph, we adopt data augmentation techniques commonly used in contrastive learning [54, 117, 116] to enhance the representativeness and diversity of the resultant toy graphs.

Toy Graphs Augmentation Strategy. For augmentation, we first calculate the average reversed importance $\bar{I}'(G_\tau^e(v_m))$ of the nodes within an ego graph as $\bar{I}'(G_\tau^e(v_m)) = \frac{1}{|G_\tau^e(v_m)|} \sum_{v \in G_\tau^e(v_m)} I'(v)$, which then determines the number of augmentations $n_{\text{aug}}(G_\tau^e(v_m)) = \lfloor K \cdot \bar{I}'(G_\tau^e(v_m)) \rfloor$, where K is a scaling constant that adjusts the intensity of the augmentation. For node $v_i, v_j \in G_\tau^e(v_m)$, the augmentation techniques $\text{DATAAUGMENTATION}(G_\tau^e(v_m), n_{\text{aug}})$ employed include:

- ❶ **Node Dropout:** $v_i \in G_\tau^e(v_m)$ has a probability of being dropped: $p(v_i \text{ being dropped}) = 1 - p_i$.
- ❷ **Addition of Gaussian Noise:** we add gaussian noise to node features as augmentation $X'(v_i) = X(v_i) + \mathcal{N}(0, \sigma^2)$.
- ❸ **Node Interpolation:** a new node feature $X'(v_{new})$ is created by linearly combining the features of two existing nodes v_i and v_j, calculated as $X'(v_{new}) = \lambda X(v_i) + (1-\lambda)X(v_j), v_i, v_j \in G^{\mathcal{T}}$. And the edge weight between the new node v_{new} and node v_i is updated to $\lambda A[i,j]$ and node v_j is $(1-\lambda)A[i,j]$ accordingly [108].
- ❹ **Edge Rewriting:** we alter connections based on the average of the involved nodes' sampling probabilities, expressed as $p((v_i, v_j) \text{ being rewired}) = \frac{p_i + p_j}{2}$.

Key-Value Pairs Construction. After completing the sampling and augmentation procedures, the generated toy graphs are transformed into key-value pairs for storage [109]. Specifically, we collect each master node's v_m historical information (timestamps τ), environmental information (neighbors $\mathcal{N}(v_m^\tau)$), structural encodings s_m^τ (as described in the Appendix C.2), and the hidden embeddings h_m^τ (obtained by processing the toy graph through the frozen pre-trained GNNs) and store them as keys at the master node v_m of the toy graph. Additionally, we store task-specific output vectors $\{o_i^\tau | v_i \in G^{\mathcal{T}}\}$ and hidden embeddings $\{h_i^\tau | v_i \in G^{\mathcal{T}}\}$ as values at each node of the toy graph. For storage of these key-value pairs, we utilize the FAISS vector library [14] to facilitate accelerated retrieval and storage.

4.2 Toy Graphs Retrieval Process

After constructing the key-value toy graphs vector database, we proceed with the retrieval process for sub-tasks according to the four sub-similarities between the key values of the master node v_m in the toy graph and the center node v_c in the query graph, as detailed in Appendix C.3. The final similarity score is a weighted combination of these factors, and the $topK$ toy graphs are selected as the retrieval results:

$$S(v_c, v_m) = \mathbf{w} \times [S_{\text{time}}(v_c, v_m), S_{\text{structure}}(v_c, v_m), S_{\text{environment}}(v_c, v_m), S_{\text{semantic}}(v_c, v_m)]^{\text{T}}, \quad (1)$$

where $\mathbf{w} = [w_1, w_2, w_3, w_4]$ are the hyper-parameterized weights attributed to the time, structure, environment, and semantic similarities, respectively. Using this composite similarity, we rank and retrieve the $topK$ toy graphs:

$$G_{\text{TopK}}^{\mathcal{T}} = \text{TopK}_{G^\tau \in \mathcal{G}^\tau} S(v_c, v_m), \quad (2)$$

where $G_{\text{TopK}}^{\mathcal{T}}$ represents the subset of toy graphs that best match the query based on the combined criteria. This process ensures that we retrieve the most relevant toy graphs based on a comprehensive similarity measure, incorporating historical, structural, and environmental information.

4.3 Training and Inference

In Section 4.3.1, we detail the Knowledge Injection Propagation process, which includes two distinct propagation manners. Next, in Section 4.3.2, we present our approach for combining the retrieved hidden embeddings with the task-specific output vectors. Additionally, to enhance the robustness of RAGRAPH, a noise-based prompt tuning strategy is introduced in Section 4.3.3.

4.3.1 Knowledge Injection Propagation

After retrieving the $topK$ toy graphs $G^{\mathcal{T}}_{\text{TopK}}$, knowledge, specifically the task-specific output vectors O and hidden embeddings H, is propagated from these toy graphs to the master nodes (Toy Graph Intra Propagation) and then to the center node v_c (Query-Toy Graph Inter Propagation). This propagation utilizes message-passing mechanisms via GNNs (cf. Appendix C.1). Each master node v_m in the toy graphs is connected to the center node v_c of the query graph based on the similarity scores $S(v_c,v_m)$ computed in Eq.(1) and the connection weights dictate the influence of each toy graph, ensuring that graphs with higher similarity have a more substantial impact. This process can be implemented using either a parameter-free or a learnable approach. Moreover, it is worth noting that for learnable methods, the parameters of GNN are different.

❶ Toy Graph Intra Propagation Within each toy graph, information $\mathbf{z}$ is propagated from neighbors to the master node using pre-trained GNNs. The task-specific output vectors o and hidden embeddings h from the neighbors are aggregated and transmitted to the master node. For each node v_i in a toy graph $G^{\mathcal{T}}$, the GNN aggregates information from its neighbors $\mathcal{N}(v_i)$ to update the master node v_m:

$$\mathbf{z}_m = \text{GNN}(\{\mathbf{z}_i \,|\, v_i \in \mathcal{N}(v_m)\}), \tag{3}$$

where $\mathbf{z}_i$ and $\mathbf{z}_m$ represent the hidden embeddings h_i,h_m or task-specific output vectors o_i,o_m of the neighbor nodes and master node, respectively. For parameter-free situations, we can prepare $\mathbf{z}_m$ in advance when constructing the toy graph to improve inference efficiency.

❷ Query-Toy Graph Inter Propagation Next, information from the toy graphs is aggregated to the query graph. Specifically, during propagation, information $\mathbf{z}$ from the neighbors and master node of the toy graph is propagated to the center node using the same pre-trained GNNs. For a center node v_c in the query graph $G^{\mathcal{Q}}$, the GNN aggregates hidden embeddings H from its neighbors $\mathcal{N}(v_c)$ and the master node v_m from the toy graph:

$$h_c = \text{GNN}(\{h_i \,|\, v_i \in \mathcal{N}(v_c) \cup \{v_m\}\}). \tag{4}$$

When propagating the task-specific output vector O, only the master node's information is passed to the center node:

$$o_c = \text{GNN}(\{o_i \,|\, v_i \in \{v_m\}\}). \tag{5}$$

For scenarios where the propagation mechanism is learnable, attention mechanisms can be adapted on the edges. In parameter-free scenarios—where there are no learnable weights—the attention on the edges is determined based on the edge weights from the previous resource graph.

4.3.2 Knowledge Fusion Layer

Finally, at the data fusion layer, the aggregated hidden embeddings H of the center node v_c are processed through the pre-trained GNN's decoder $\text{DECODER}(\cdot)$ to obtain an output vector O. This output vector is then combined with the aggregated task-specific output vector in a weighted manner to produce the final output for downstream tasks as illustrated in Definition 3. The combined output is formulated as follows:

$$\hat{o}_c = \gamma o_c + (1-\gamma)\text{DECODER}(h_c), \tag{6}$$

where γ is a reweighting hyper-parameter. The resulting vector $\hat{o}_c$ is then utilized to perform node-, graph-, or edge-level tasks via a similarity function.

For the same task, the decoder can be directly used to generate outputs. For different tasks, the decoder can be masked, allowing the model to utilize pre-computed embeddings without additional training. Furthermore, the decoder can be fine-tuned to better meet the specific requirements of each task, providing both flexibility and optimized performance. This approach ensures that the model effectively integrates and leverages information from both the toy graphs and the query graph, enhancing its effectiveness in various downstream tasks through the use of the aggregated task-specific output vector.

4.3.3 Noise-based Graph Prompting Tuning

When prompt tuning, RAGRAPH employs the same prompt loss function $\mathcal{L}_{\text{prompt}}$ as the backbone model (e.g., GraphPro, GraphPrompt). However, to mitigate the challenge of noise retrieval—a common issue in traditional RAG where highly related but irrelevant data is often retrieved—we enhance the training process by incorporating noise data to bolster model robustness, motivated by [53]. Specifically, we implement two types of noise integration strategies:

- ❶ **Inner-Toy-Graph Noise:** This strategy involves artificially introducing irrelevant nodes ($v_j \notin G_\tau^e(v_m)$) into the toy graph during its construction, complementing other augmentation techniques.
- ❷ **Toy-Graph Noise:** Throughout the training phase, we not only retrieve the $topK$ toy graphs that are most relevant but also deliberately include the $bottomK$ toy graphs to incorporate noise knowledge.

The integration of these noise elements is intended to enhance the model's ability to distinguish relevant information from irrelevant information, significantly improving its robustness and overall performance in downstream tasks by noise training. However, during the inference stage, we do not incorporate the noise.

5 Experiments

In this section, we conduct a series of experiments to evaluate the performance of RAGRAPH against state-of-the-art baselines on three dynamic and five static datasets on three-level graph tasks. Further details and experiment results are provided in Appendix D.

5.1 Experimental Setup

Datasets. We use four static datasets *PROTEINS, COX2, ENZYMES* and *BZR* for graph classification and node classification, as well as three dynamic datasets *TAOBAO, KOUBEI* and *AMAZON* for link prediction. More details about these datasets can be found in Table 4 in Appendix D.1.

Methods and Baselines. We consider three versions of our proposed framework RAGRAPH: 1) RAGRAPH/NF, which indicates we utilize the plug-and-plag RAGRAPH without fine-tuning on the train set; 2) RAGRAPH/FT, which employs prompt tuning on the train set with RAG; and 3) RAGRAPH/NFT, which applies noise prompt tuning on the train set with RAG. For the baseline of the dynamic graph, we choose LightGCN [30], SGL [103], MixGCF [39], SimGCL [117], GraphPro [113] and GraphPro+PRODIGY [73]. For the static graph, we choose GCN [48], GraphSAGE [29],GAT [97], GIN [105], GraphPrompt [65], GraphPrompt+PRODIGY [73] as baselines. In addition, we denote '/NF' and '/FT' respectively to represent without fine-tuning and fine-tuning. A detailed description of baselines can be referred to in Appendix D.3.

Settings and Evaluation. We establish a training-resource split with the remainder of the data reserved as unseen during fine-tuning. For static graphs, the split is based on node partitioning with the ratio of 50%:30% [65], while for dynamic graphs, it is based on partitioning snapshots with the history snapshots as resource graph [73]. For fair comparisons, for methods employing PRODIGY and RAGRAPH, ❶ we fine-tune models using the training set while retrieving the resource graph to prevent information leakage and over-fitting; ❷ when testing, we retrieve the combined training and resource graphs. For other methods, fine-tuning was directly performed on the combined train and resource set for fairness. For the evaluation of static graphs, we refer GraphPrompt, utilizing pre-trained GNNs for both node- and graph-level tasks within a k-shot classification framework. For dynamic graphs, we follow GraphPro to employ pre-trained GNNs on a substantial dataset fraction, with fine-tuning and testing conducted on later snapshots. Moreover, we pre-train GraphPro and GraphPrompt unsupervised on other datasets within the similar domain following [65, 73] to avoid information leakage. For classification tasks, we utilize the accuracy as evaluation matric; For link prediction tasks, we use standard metrics Recall@k and nDCG@k at $k = 20$, in line with existing methodologies [30, 103, 117]. The metrics used in the experiment are detailed in Appendix D.2 and the implementation details of RAGRAPH and baselines are in Appendix D.4.

Table 1: Accuracy evaluation on node and graph classification. All tabular results (%) are in mean± standard deviation across five seeds run, with best **bolded** and runner-up underlined.

Methods	Node Classification		Graph Classification			
	PROTEINS (5-shot)	ENZYMES (5-shot)	PROTEINS (5-shot)	COX2 (5-shot)	ENZYMES (5-shot)	BZR (5-shot)
GCN	46.63±03.04	52.80±12.89	54.80±06.64	67.87±03.39	22.67±05.20	58.76±05.08
GraphSAGE	48.87±02.64	48.75±01.59	52.99±10.57	67.02±05.42	21.17±05.49	58.27±04.79
GAT	48.13±07.90	47.75±01.23	55.82±07.31	64.89±03.23	20.67±03.27	57.04±06.70
GIN	49.61±01.58	48.82±01.58	56.17±08.58	62.77±02.85	19.00±03.74	56.54±04.20
GraphPrompt+						
Vanilla/NF	44.88±13.17	48.81±01.88	56.68±03.63	53.04±04.13	36.50±03.31	68.77±03.44
Vanilla/FT	48.99±01.88	51.99±01.36	57.04±03.88	64.04±08.20	40.00±04.36	69.01±02.21
PRODIGY/NF	47.32±08.12	43.80±14.03	53.48±06.72	53.97±10.34	22.12±13.84	67.18±08.93
PRODIGY/FT	53.26±06.42	57.98±12.37	57.14±10.34	65.31±04.28	25.94±05.12	68.08±06.68
RAGRAPH/NF	56.12±04.11	75.92±01.72	58.48±03.93	55.32±04.15	38.17±03.39	**77.53**±05.26
RAGRAPH/FT	58.74±00.87	75.74±01.92	**62.33**±02.52	**76.60**±02.30	47.71±06.88	76.79±05.02
RAGRAPH/NFT	**59.83**±00.40	**76.23**±01.63	59.08±03.50	71.70±04.29	**49.17**±04.64	74.81±04.25

5.2 Retrieval-Augmented Graph Results

As discussed, we conduct experiments and report the results of the three graph tasks for static graph and dynamic graph, as illustrated in Table 1 and Table 2. From the reported accuracy, we can find the following observations:

Outperforming SOTA Methods. First, our proposed RAGRAPH outperforms almost all the baselines across the three graph tasks, demonstrating the effectiveness of RAGRAPH in transferring knowledge from the pre-training to downstream tasks compared to traditional GNNs *i.e.,* GCN and GraphSAGE. It achieves the highest average accuracy across almost all tasks on ENZYMES, with an improvement of at least 5.19% in the static graph, and up to 1.81‰ on the dynamic graph over the best baseline PRODIGY/FT. We argue that by virtue of the integration of hidden embedding and task-specific output vector, RAGRAPH is able to comprehend more knowledge than simply learns the paradigm from $X \rightarrow Y$. Second, compared with the models of PRODIGY/NF and RAGRAPH/NF, the introduction of noise training in noise prompt tuning also improves the robustness of the model, avoiding the influence of a large amount of noise on the information aggregation inside the query graph.

Strong Retrieval-Augmented Performance on Unseen Datasets. We observe that PRODIGY/NF and RAGRAPH/NF are better to Vanilla/NF, indicating that the retrieval knowledge truly works when testing on unseen datasets. Moreover, the difference between PRODIGY/NF and PRODIGY/FT is much greater than that of RAGRAPH, which also indicates that a simple learning paradigm for ICL is not enough and that RAGRAPH can achieve acceptable results even on unseen downstream datasets without the need for sophisticated fine-tuning.

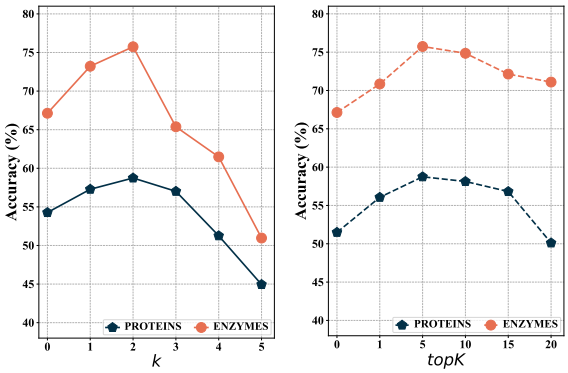


Figure 3: Hyper-parameter study with hops$k$ (**Left**) from 1 to 5 and$topk$ from 1 to 20 (**Right**) on node classification with PROTEINS, and ENZYMES datasets with the setting in Table 1.

Table 2: Performance evaluation (‰) on link prediction.

Method	TAOBAO		KOUBEI		AMAZON	
	Recall	nDCG	Recall	nDCG	Recall	nDCG
LightGCN	22.47±02.53	21.89±02.80	30.21±06.45	22.24±05.83	15.07±06.48	06.53±02.66
SGL	22.15±02.20	22.12±03.09	35.54±05.18	25.39±06.25	15.78±07.12	07.90±02.49
MixGCF	22.84±02.15	23.05±03.87	34.83±06.06	25.98±06.08	15.24±08.98	07.40±03.44
SimGCL	22.18±02.22	23.15±02.75	33.32±06.64	25.18±05.04	16.10±07.91	07.58±03.51
GraphPro+						
Vanilla/NF	20.10±02.07	20.15±02.45	28.42±04.21	20.09±03.26	12.09±07.72	05.45±03.36
Vanilla/FT	26.58±01.11	<u>26.22</u>±01.85	36.17±02.43	26.13±03.73	15.61±04.16	08.01±02.03
PRODIGY/NF	21.67±01.42	23.15±03.20	29.02±04.82	20.67±02.31	11.88±02.61	05.84±01.84
PRODIGY/FT	<u>27.05</u>±01.76	23.68±02.85	**38.83**±04.76	**27.68**±03.12	16.72±04.28	08.09±02.66
RAGRAPH/NF	20.31±01.60	20.45±01.44	29.24±01.45	21.60±02.91	12.40±07.40	06.16±03.81
RAGRAPH/FT	26.18±03.42	24.30±01.21	37.92±03.71	26.34±04.33	<u>17.68</u>±07.47	<u>08.76</u>±03.98
RAGRAPH/NFT	**27.53**±02.24	**26.47**±01.29	<u>37.98</u>±03.65	<u>27.13</u>±05.17	**18.53**±04.63	**09.02**±02.45

5.3 Hyper-parameter Study

In this section, we examine the impact of various hyper-parameters on RAGRAPH. We specifically analyze the effects of varying the number of hops k in toy graphs from the list $[1,2,3,4,5]$ and the number of linked toy graphs $topK$ from the list [1,5,10,15,30,50] to verify the sensitive:

Figure 3 (Left) illustrates relationships between accuracy and the toy graph hop k. We observe that as k increases, the volume of retrieved knowledge grows exponentially. However, an excessive accumulation of knowledge not only fails to enhance accuracy but also introduces increased irrelevant noise that burdens the GNNs. Notably, accuracy shows a trend of initial improvement followed by a decline as k is increased. This pattern suggests that at lower k values, the retrieved information tends to consist of isolated, less useful knowledge. In contrast, at higher k values, the GNNs struggle to process extensive reasoning chains, leading to the utilization of complex and abundant information that is less effective than even the baseline model's performance. Figure 3 (Right) shows effects on accuracy with different numbers of toy graphs $topK$. As with the previous figure, increasing $topK$ demonstrates that an excessive amount of knowledge can hinder the GNNs' comprehension capabilities. Conversely, smaller $topK$ results in insufficient knowledge to enhance performance on downstream tasks.

6 Conclusion

We introduced RAGRAPH, a novel and general framework that enhances Graph Neural Networks (GNNs) by integrating Retrieval-Augmented Generation (RAG) techniques. This plug-and-play approach improves GNNs' ability to generalize to unseen data by retrieving relevant information. Experimental results show that RAGRAPH outperforms state-of-the-art methods in various graph learning tasks, demonstrating its adaptability and robustness. While RAGRAPH is currently limited to retrieving subgraphs, future research could explore using more graph-structured data such as nodes, edges, and trees to further enhance its capabilities. In general, our work provides valuable insights and serves as a reference for future Large Graph Models.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No.U23A20468).

References

[1] A. Asai, S. Min, Z. Zhong, and D. Chen. Retrieval-based language models and applications. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 6: Tutorial Abstracts)*, pages 41–46, 2023.

[2] A. Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *ICLR*, 2024.

[3] S.-V. Bogolin, I. Croitoru, H. Jin, Y. Liu, and S. Albanie. Cross modal retrieval with querybank normalisation. In *CVPR*, 2022.

[4] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[5] H. Cai, V. W. Zheng, and K. C.-C. Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE TKDE*, 2018.

[6] D. M. Chan, S. Ghosh, A. Rastrow, and B. Hoffmeister. Using external off-policy speech-to-text mappings in contextual end-to-end automated speech recognition, 2023.

[7] M. Chen, W. Zhang, W. Zhang, Q. Chen, and H. Chen. Meta relational learning for few-shot link prediction in knowledge graphs. *arXiv preprint arXiv:1909.01515*, 2019.

[8] W. Chen, H. Hu, X. Chen, P. Verga, and W. W. Cohen. Murag: Multimodal retrieval-augmented generator for open question answering over images and text. In *EMNLP*, 2022.

[9] X. Chen, S. Zhang, Y. Xiong, X. Wu, J. Zhang, X. Sun, Y. Zhang, Y. Zhao, and Y. Kang. Prompt learning on temporal interaction graphs. *arXiv:2402.06326*, 2024.

[10] Z. Cheng, J. Zhang, X. Xu, G. Trajcevski, T. Zhong, and F. Zhou. Retrieval-augmented hypergraph for multimodal social media popularity prediction. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 445–455, New York, NY, USA, 2024. Association for Computing Machinery.

[11] X. Chu, Y. Jin, X. Wang, S. Zhang, Y. Wang, W. Zhu, and H. Mei. Wasserstein barycenter matching for graph size generalization of message passing neural networks. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[12] F. Cuconasu, G. Trappolini, F. Siciliano, S. Filice, C. Campagnano, Y. Maarek, N. Tonellotto, and F. Silvestri. The power of noise: Redefining retrieval for rag systems. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, volume 17 of *SIGIR 2024*, page 719–729. ACM, July 2024.

[13] H. Cui, Z. Lu, P. Li, and C. Yang. On positional and structural node features for graph neural networks on non-attributed graphs, 2021.

[14] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou. The faiss library, 2024.

[15] Y. Duan, G. Zhang, S. Wang, X. Peng, Z. Wang, J. Mao, H. Wu, X. Jiang, and K. Wang. Cat-gnn: Enhancing credit card fraud detection via causal temporal graph neural networks. *ArXiv*, abs/2402.14708, 2024.

[16] Y. Duan, J. Zhao, pengcheng, J. Mao, H. Wu, J. Xu, S. Wang, C. Ma, K. Wang, K. Wang, and X. Li. Causal deciphering and inpainting in spatio-temporal dynamics via diffusion model. 2024.

[17] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NeurIPS*, 2015.

[18] P. L. Eli Chien, Jianhao Peng and O. Milenkovic. Adaptive universal generalized pagerank graph neural network. In *ICLR*, 2021.

[19] F. Fang, Y. Bai, S. Ni, M. Yang, X. Chen, and R. Xu. Enhancing noise robustness of retrieval-augmented language models with adaptive adversarial training, 2024.

[20] T. Fang, Y. Zhang, Y. Yang, C. Wang, and L. Chen. Universal prompt tuning for graph neural networks. In *NeurIPS*, 2024.

[21] Y. Fang, Y. Qin, H. Luo, F. Zhao, B. Xu, L. Zeng, and C. Wang. When spatio-temporal meet wavelets: Disentangled traffic forecasting via efficient spectral graph attention networks. In *ICDE*, 2023.

[22] L. Gao, X. Ma, J. Lin, and J. Callan. Precise zero-shot dense retrieval without relevance labels, 2022.

[23] X. Gao, H. Chen, and J. Haworth. A spatiotemporal analysis of the impact of lockdown and coronavirus on london's bicycle hire scheme: from response to recovery to a new normal. *GIS*, 2023.

[24] X. Gao, J. Haworth, D. Zhuang, H. Chen, and X. Jiang. Uncertainty quantification in the road-level traffic risk prediction by spatial-temporal zero-inflated negative binomial graph neural network (stzinb-gnn). *GIScience 2023*, 2023.

[25] X. Gao, X. Jiang, J. Haworth, D. Zhuang, S. Wang, H. Chen, and S. Law. Uncertainty-aware probabilistic graph neural networks for road-level traffic crash prediction. *Accident Analysis & Prevention*, 208:107801, 2024.

[26] Z. Gao, X. Sun, Z. Liu, Y. Li, H. Cheng, and J. Li. Protein multimer structure prediction via PPI-guided prompt learning. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[27] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in facebook: A case study of unbiased sampling of osns. In *2010 Proceedings IEEE INFOCOM*, pages 1–9, 2010.

[28] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *SIGKDD*, 2016.

[29] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

[30] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *SIGIR*, 2020.

[31] Z. He, W. Hao, W.-T. Lu, C. Chen, K. Lerman, and X. Song. Alcap: Alignment-augmented music captioner. In *ICASSP*, 2023.

[32] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models, 2021.

[33] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

[34] Z. Hu, Y. Dong, K. Wang, K.-W. Chang, and Y. Sun. Gpt-gnn: Generative pre-training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1857–1867, 2020.

[35] J. Huang, G. Cai, J. Zhu, Z. Dong, R. Tang, W. Zhang, and Y. Yu. Recall-augmented ranking: Enhancing click-through rate prediction accuracy with cross-stage data. In *WWW*, 2024.

[36] K. Huang and M. Zitnik. Graph meta learning via local subgraphs, 2020.

[37] M. Huang, Y. Liu, X. Ao, K. Li, J. Chi, J. Feng, H. Yang, and Q. He. Auc-oriented graph neural network for fraud detection. In *WWW*, 2022.

[38] Q. Huang, H. Ren, and J. Leskovec. Few-shot relational reasoning via connection subgraph pretraining. In *NeurIPS*, 2022.

[39] T. Huang, Y. Dong, M. Ding, Z. Yang, W. Feng, X. Wang, and J. Tang. Mixgcf: An improved training method for graph neural network-based recommender systems. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 665–674, 2021.

[40] G. Izacard, P. Lewis, M. Lomeli, L. Hosseini, F. Petroni, T. Schick, J. Dwivedi-Yu, A. Joulin, S. Riedel, and E. Grave. Atlas: Few-shot learning with retrieval augmented language models, 2022.

[41] X. Jiang, Y. Fang, R. Qiu, H. Zhang, Y. Xu, H. Chen, W. Zhang, R. Zhang, Y. Fang, X. Chu, J. Zhao, and Y. Wang. Tc-rag:turing-complete rag's case study on medical llm systems. *ArXiv*, abs/2408.09199, 2024.

[42] X. Jiang, Z. Qin, J. Xu, and X. Ao. Incomplete graph learning via attribute-structure decoupled variational auto-encoder. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, WSDM '24, page 304–312, New York, NY, USA, 2024. Association for Computing Machinery.

[43] X. Jiang, R. Zhang, Y. Xu, R. Qiu, Y. Fang, Z. Wang, J. Tang, H. Ding, X. Chu, J. Zhao, and Y. Wang. Hykge: A hypothesis knowledge graph enhanced framework for accurate and reliable medical llms responses, 2024.

[44] X. Jiang, D. Zhuang, X. Zhang, H. Chen, J. Luo, and X. Gao. Uncertainty quantification via spatial-temporal tweedie model for zero-inflated and long-tail travel demand prediction. In *CIKM*, 2023.

[45] R. Y. Jiaxuan You and J. Leskovec. Position-aware graph neural networks. In *ICML*, 2019.

[46] B. Jin, Y. Zhang, Q. Zhu, and J. Han. Heterformer: Transformer-based deep node representation learning on heterogeneous text-rich networks. In *SIGKDD*, pages 1020–1031, 2023.

[47] J. Kim, S. M. Jaehyun Nam, J. Park, S.-W. Lee, M. Seo, J.-W. Ha, and J. Shin. Sure: Summarizing retrievals using answer candidates for open-domain qa of llms. In *ICLR*, 2024.

[48] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2016.

[49] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

[50] G. Li, M. Müller, A. Thabet, and B. Ghanem. Deepgcns: Can gcns go as deep as cnns? In *ICCV*, 2019.

[51] H. Li, X. Wang, Z. Zhang, and W. Zhu. Out-of-distribution generalization on graphs: A survey, 2022.

[52] K. Li, Y. Chen, Y. Liu, J. Wang, Q. He, M. Cheng, and X. Ao. Boosting the adversarial robustness of graph neural networks: An ood perspective. In *International Conference on Learning Representations*, 2024.

[53] K. Li, Y. Liu, X. Ao, J. Chi, J. Feng, H. Yang, and Q. He. Reliable representations make a stronger defender: Unsupervised structure refinement for robust gnn. In *SIGKDD*, 2022.

[54] R. Li, T. Zhong, X. Jiang, G. Trajcevski, J. Wu, and F. Zhou. Mining spatio-temporal relations via self-paced graph contrastive learning. In *SIGKDD*, 2022.

[55] S. Li, R. Xie, Y. Zhu, X. Ao, F. Zhuang, and Q. He. User-centric conversational recommendation with multi-aspect user modeling. In *SIGIR*, 2022.

[56] X. Li, D. Lian, Z. Lu, J. Bai, Z. Chen, and X. Wang. Graphadapter: Tuning vision-language models with dual knowledge graph. In *NeurIPS*, volume 36, 2024.

[57] X. Li, R. Zhao, Y. K. Chia, B. Ding, S. Joty, S. Poria, and L. Bing. Chain-of-knowledge: Grounding large language models via dynamic knowledge adapting over heterogeneous sources, 2023.

[58] W. Lin and B. Byrne. Retrieval augmented visual question answering with outside knowledge. In *EMNLP*, 2022.

[59] W. Lin, J. Mei, J. Chen, and B. Byrne. Preflmr: Scaling up fine-grained late-interaction multi-modal retrievers, 2024.

[60] X. V. Lin, X. Chen, M. Chen, W. Shi, M. Lomeli, R. James, P. Rodriguez, J. Kahn, G. Szilvasy, M. Lewis, L. Zettlemoyer, and S. Yih. Ra-dit: Retrieval-augmented dual instruction tuning. In *ICLR*, 2024.

[61] Y. Liu, X. Ao, F. Feng, and Q. He. Ud-gnn: Uncertainty-aware debiased training on semi-homophilous graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1131–1140, 2022.

[62] Y. Liu, X. Ao, F. Feng, Y. Ma, K. Li, T.-S. Chua, and Q. He. Flood: A flexible invariant learning framework for out-of-distribution generalization on graphs. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1548–1558, 2023.

[63] Y. Liu, X. Ao, Z. Qin, J. Chi, J. Feng, H. Yang, and Q. He. Pick and choose: a gnn-based imbalanced learning approach for fraud detection. In *Proceedings of the Web Conference 2021*, pages 3168–3177, 2021.

[64] Y. Liu, K. Zhang, Y. Li, Z. Yan, C. Gao, R. Chen, Z. Yuan, Y. Huang, H. Sun, J. Gao, L. He, and L. Sun. Sora: A review on background, technology, limitations, and opportunities of large vision models, 2024.

[65] Z. Liu, X. Yu, Y. Fang, and X. Zhang. Graphprompt: Unifying pre-training and downstream tasks for graph neural networks. In *WWW*, 2023.

[66] S. Lu, N. Duan, H. Han, D. Guo, S. won Hwang, and A. Svyatkovskiy. Reacc: A retrieval-augmented code completion framework. In *ACL*, 2022.

[67] H. Luo, Y.-S. Chuang, Y. Gong, T. Zhang, Y. Kim, X. Wu, D. Fox, H. Meng, and J. Glass. Sail: Search-augmented instruction learning, 2023.

[68] J. Luo, W. Zhang, Y. Fang, X. Gao, D. Zhuang, H. Chen, and X. Jiang. Timeseries suppliers allocation risk optimization via deep black litterman model. *ArXiv*, abs/2401.17350, 2024.

[69] K. Ma, H. Cheng, Y. Zhang, X. Liu, E. Nyberg, and J. Gao. Chain-of-skills: A configurable model for open-domain question answering, 2023.

[70] H. Mao, X. Chen, Q. Fu, L. Du, S. Han, and D. Zhang. Neuron campaign for initialization guided by information bottleneck theory. In *CIKM*, 2021.

[71] S. Maskey, R. Levie, Y. Lee, and G. Kutyniok. Generalization analysis of message passing neural networks on large random graphs. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 4805–4817. Curran Associates, Inc., 2022.

[72] S. Matsugu, Y. Fujiwara, and H. Shiokawa. Uncovering the largest community in social networks at scale. In *IJCAI*, 2023.

[73] K. Mishchenko and A. Defazio. Prodigy: An expeditiously adaptive parameter-free learner. In *NeurIPS*, 2023.

[74] OpenAI. Introducing chatgpt. https://openai.com/blog/chatgpt, 2022.

[75] OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023.

[76] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[77] S. Parashar, Z. Lin, T. Liu, X. Dong, Y. Li, D. Ramanan, J. Caverlee, and S. Kong. The neglected tails of vision-language models. In *CVPR*, 2024.

[78] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[79] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *SIGKDD*, 2014.

[80] Y. Qin, Y. Fang, H. Luo, F. Zhao, and C. Wang. Next point-of-interest recommendation with auto-correlation enhanced multi-modal transformer network. In *SIGIR*, 2022.

[81] J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1150–1160, 2020.

[82] Y. Qu, Y. Ding, J. Liu, K. Liu, R. Ren, W. X. Zhao, D. Dong, H. Wu, and H. Wang. Rocketqa: An optimized training approach to dense passage retrieval for open-domain question answering, 2021.

[83] R. A. Rossi and N. K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI Conference on Artificial Intelligence*, pages 4292–4293, 2015.

[84] P. Sarthi, S. Abdullah, A. Tuli, S. Khanna, A. Goldie, and C. D. Manning. Raptor: Recursive abstractive processing for tree-organized retrieval. In *ICLR*, 2024.

[85] S. Sharma and R. Sharma. Identifying possible rumor spreaders on twitter: A weak supervised learning approach. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021.

[86] K. Soman, P. W. Rose, J. H. Morris, R. E. Akbas, B. Smith, B. Peetoom, C. Villouta-Reyes, G. Cerono, Y. Shi, A. Rizk-Jackson, S. Israni, C. A. Nelson, S. Huang, and S. E. Baranzini. Biomedical knowledge graph-enhanced prompt generation for large language models, 2023.

[87] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009, Jan. 2009.

[88] J. Sun, Y. Zhou, and C. Zong. One-shot relation learning for knowledge graphs via neighborhood aggregation and paths encoding. *Transactions on Asian and Low-Resource Language Information Processing*, 21(3):1–19, 2021.

[89] M. Sun, K. Zhou, X. He, Y. Wang, and X. Wang. Gppt: Graph pre-training and prompt tuning to generalize graph neural networks. In *SIGKDD*, 2022.

[90] X. Sun, H. Cheng, J. Li, B. Liu, and J. Guan. All in one: Multi-task prompting for graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining (KDD'23)*, page 2120–2131, 2023.

[91] X. Sun, J. Zhang, X. Wu, H. Cheng, Y. Xiong, and J. Li. Graph prompt learning: A comprehensive survey and beyond. *arXiv:2311.16534*, 2023.

[92] Y. Tan, H. Lv, X. Huang, J. Zhang, S. Wang, and C. Yang. Musegraph: Graph-oriented instruction tuning of large language models for generic graph mining, 2024.

[93] Y. Tan, Z. Zhou, H. Lv, W. Liu, and C. Yang. Walklm: A uniform language model fine-tuning framework for attributed graph embedding. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *NeurIPS*, volume 36, pages 13308–13325. Curran Associates, Inc., 2023.

[94] Z. Tan, R. Guo, K. Ding, and H. Liu. Virtual node tuning for few-shot node classification. In *SIGKDD*, 2023.

[95] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *WWW*, 2015.

[96] B. Teji, S. Roy, D. S. Dhami, D. Bhandari, and P. H. Guzzi. Graph embedding techniques for predicting missing links in biological networks: An empirical evaluation. *IEEE Transactions on Emerging Topics in Computing*, 12(1):190–201, 2024.

[97] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *ICLR*, 2018.

[98] S. Wang, K. Ding, C. Zhang, C. Chen, and J. Li. Task-adaptive few-shot node classification. *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022.

[99] S. Wang, Y. Dong, X. Huang, C. Chen, and J. Li. FAITH: Few-shot graph classification with hierarchical task graphs. In *International Joint Conference on Artificial Intelligence*, 2022.

[100] X. Wang, L. Zhu, and Y. Yang. T2vlad: Global-local sequence alignment for text-video retrieval. In *CVPR*, 2021.

[101] Y. Wang, R. Ren, J. Li, W. X. Zhao, J. Liu, and J. Wen. Rear: A relevance aware retrieval augmented framework for open-domain question answering, 2024.

[102] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[103] J. Wu, X. Wang, F. Feng, X. He, L. Chen, J. Lian, and X. Xie. Self-supervised graph learning for recommendation. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*, pages 726–735, 2021.

[104] L. Xia, B. Kao, and C. Huang. Opengraph: Towards open graph foundation models, 2024.

[105] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[106] P. Xu, W. Ping, X. Wu, L. McAfee, C. Zhu, Z. Liu, S. Subramanian, E. Bakhturina, M. Shoeybi, and B. Catanzaro. Retrieval meets long context large language models. In *ICLR*, 2024.

[107] Y. Xu, X. Chu, K. Yang, Z. Wang, P. Zou, H. Ding, J. Zhao, Y. Wang, and B. Xie. Seqcare: Sequential training with external medical knowledge graph for diagnosis prediction in healthcare data. In *Proceedings of the ACM Web Conference 2023*, pages 2819–2830, 2023.

[108] Y. Xu, X. Jiang, X. Chu, Y. Xiao, C. Zhang, H. Ding, J. Zhao, Y. Wang, and B. Xie. Protomix: Augmenting health status representation learning via prototype-based mixup. In *Knowledge Discovery and Data Mining*, 2024.

[109] Y. Xu, K. Yang, C. Zhang, P. Zou, Z. Wang, H. Ding, J. Zhao, Y. Wang, and B. Xie. Vecocare: Visit sequences-clinical notes joint learning for diagnosis prediction in healthcare data. In *IJCAI*, volume 23, pages 4921–4929, 2023.

[110] Y. Xu*, R. Zhang*, X. Jiang*, Y. Feng, Y. Xiao, X. Ma, R. Zhu, X. Chu, J. Zhao, and Y. Wang. Parenting: Optimizing knowledge selection of retrieval-augmented language models with parameter decoupling and tailored tuning. *arXiv preprint arXiv:2410.10360*, 2024.

[111] K. Yang, Y. Xu, P. Zou, H. Ding, J. Zhao, Y. Wang, and B. Xie. Kerprint: local-global knowledge graph enhanced diagnosis prediction for retrospective and prospective interpretations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 5357–5365, 2023.

[112] S. Yang, X. Jiang, H. Zhao, W. Zeng, H. Liu, and Y. Jia. Faima: Feature-aware in-context learning for multi-domain aspect-based sentiment analysis. In *COLING*, 2024.

[113] Y. Yang, L. Xia, D. Luo, K. Lin, and C. Huang. Graphpro: Graph pre-training and prompt learning for recommendation. In *WWW*, 2024.

[114] J. Yoo, N. Ahn, and K.-A. Sohn. Rethinking data augmentation for image super-resolution: A comprehensive analysis and a new strategy. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8372–8381, 2020.

[115] O. Yoran, T. Wolfson, O. Ram, and J. Berant. Making retrieval-augmented language models robust to irrelevant context. In *ICLR*, 2024.

[116] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen. Graph contrastive learning with augmentations. *Advances in neural information processing systems*, 33:5812–5823, 2020.

[117] J. Yu, H. Yin, X. Xia, T. Chen, L. Cui, and Q. V. H. Nguyen. Are graph augmentations necessary? simple graph contrastive learning for recommendation. In *Proceedings of the 45th international ACM SIGIR conference on research and development in information retrieval*, pages 1294–1303, 2022.

[118] W. Yu, H. Zhang, X. Pan, K. Ma, H. Wang, and D. Yu. Chain-of-note: Enhancing robustness in retrieval-augmented language models, 2023.

[119] X. Yu, Y. Fang, Z. Liu, Y. Wu, Z. Wen, J. Bo, X. Zhang, and S. C. Hoi. Few-shot learning on graphs: from meta-learning to pre-training and prompting. *arXiv preprint arXiv:2402.01440*, 2024.

[120] X. Yu, Z. Liu, Y. Fang, Z. Liu, S. Chen, and X. Zhang. Generalized graph prompt: Toward a unification of pre-training and downstream tasks on graphs. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[121] X. Yu, Z. Liu, Y. Fang, and X. Zhang. Dygprompt: Learning feature and time prompts on dynamic graphs. *arXiv preprint arXiv:2405.13937*, 2024.

[122] X. Yu, J. Zhang, Y. Fang, and R. Jiang. Non-homophilic graph pre-training and prompt learning. *arXiv preprint arXiv:2408.12594*, 2024.

[123] X. Yu, C. Zhou, Y. Fang, and X. Zhang. Multigprompt for multi-task pre-training and prompting on graphs. In *WWW*, 2024.

[124] Z. Yuan, Q. Jin, C. Tan, Z. Zhao, H. Yuan, F. Huang, and S. Huang. Ramm: Retrieval-augmented biomedical visual question answering with multi-modal pre-training. In *CVPR*, 2023.

[125] C. Zhang, H. Yao, C. Huang, M. Jiang, Z. Li, and N. V. Chawla. Few-shot knowledge graph completion. In *AAAI*, 2020.

[126] R. Zhang, X. Jiang, Y. Fang, J. Luo, Y. Xu, Y. Zhu, X. Chu, J. Zhao, and Y. Wang. Infinite-horizon graph filters: Leveraging power series to enhance sparse information aggregation. *ArXiv*, abs/2401.09943, 2024.

[127] R. Zhang, Y. Xu, Y. Xiao, R. Zhu, X. Jiang, X. Chu, J. Zhao, and Y. Wang. Kapo: Knowledge-aware preference optimization for controllable knowledge selection in retrieval-augmented language models. *arXiv preprint arXiv:2408.03297*, 2024.

[128] Y. Zhang, Z. Chen, Y. Fang, L. Cheng, Y. Lu, F. Li, W. Zhang, and H. Chen. Knowledgeable preference alignment for llms in domain-specific question answering, 2023.

[129] Z. Zhang, H. Li, Z. Zhang, Y. Qin, X. Wang, and W. Zhu. Graph meets llms: Towards large graph models, 2023.

[130] H. Zhao, A. Chen, X. Sun, H. Cheng, and J. Li. All in one and one for all: A simple yet effective method towards cross-domain graph pretraining, 2024.

[131] R. Zhao, H. Chen, W. Wang, F. Jiao, X. L. Do, C. Qin, B. Ding, X. Guo, M. Li, X. Li, and S. Joty. Retrieving multimodal information for augmented generation: A survey, 2023.

[132] J. Zhou, C. Xie, Z. Wen, X. Zhao, and Q. Xuan. Data augmentation on graphs: A technical survey, 2023.

[133] S. Zhou, U. Alon, F. F. Xu, Z. Wang, Z. Jiang, and G. Neubig. Docprompting: Generating code by retrieving the docs. In *ICLR*, 2023.

[134] X. Zhou, R. Lumbantoruan, Y. Ren, L. Chen, X. Yang, and J. Shao. Dynamic bi-layer graph learning for context-aware sequential recommendation. *ACM Trans. Recomm. Syst.*, 2(2), apr 2024.

[135] F. Zhu, W. Lei, C. Wang, J. Zheng, S. Poria, and T.-S. Chua. Retrieving and reading: A comprehensive survey on open-domain question answering, 2021.

[136] K. Zhu, X. Feng, X. Du, Y. Gu, W. Yu, H. Wang, Q. Chen, Z. Chu, J. Chen, and B. Qin. An information bottleneck perspective for effective noise filtering on retrieval-augmented generation. In L.-W. Ku, A. Martins, and V. Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1044–1069, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics.

[137] Y. Zhu, J. Guo, and S. Tang. Sgl-pt: A strong graph learner with graph prompt tuning, 2023.

[138] Y. Zhu, Z. Ou, X. Mou, and J. Tang. Retrieval-augmented embodied agents. In *CVPR*, 2024.

A Notations

The notations in this paper are summarized in Table 3.

Table 3: Notations Tables in RAGRAPH

Notation	Definition
$\mathcal{G}$ / $\mathcal{V}$ / $\mathcal{E}$	The dynamic graph / node / edge set with G_t/ V_t / E_t as its entry
$\mathcal{G}^{\mathcal{R}}$	The resource graph with $G_t^{\mathcal{R}}$ as its entry
$G^{\mathcal{Q}}$	The query graph with v_c as its center node
$\mathcal{G}^{\mathcal{T}}$	The toy graph database
$G^{\mathcal{T}}_{\text{TopK}}$	The topK retrieved toy graphs
$G^{\mathcal{T}}$	The toy graph
$G^e_\tau(v_m)$	k-hop ego net for node v_m
X_t / A_t / Y_t	The feature / edge weight / label matrix at time t
$\mathcal{C}$	The set of label classes
$\mathcal{N}(v)$	The neighbors of node v
H / O	The hidden embeddings / task-specific output vector with h_i / o_i as its v_i vector
$\mathcal{D}$	The κ-shot labeled set
t/τ	Timestamp
n	Number of nodes
d	The dimension of node feature
v_i	The i-th node
v_m/v_c	The master node of toy graph / The center node of toy graph
f_1/f_2	The dimension of hidden embedding / task-specific output vector
ϵ	The minimum value
k	k hop
$I(v)$	The node importance for node $v \in G^{\mathcal{R}}$
PR$(\cdot)$/DC$(\cdot)$	The PageRank / Degree Centrality value
p_i	The sampling probability of node v_i
K	The scaling constant
$n_{\text{aug}}(G^{\mathcal{T}})$	The number of augmentations of toy graph $G^{\mathcal{T}}$
$S(v_c,v_m)$	The weighted similarity between query node v_c and master node v_m
l	The layer of a GNN
α	The balance weight with $\alpha \in (0,1)$
λ	The weight of mixup
w_i	The weights of the time, structure, semantic, and environment similarities
γ	The reweight hyper-parameter

B More Motivation Details

B.1 Why Toy Graph Augmentation is needed

The reasons for toy graph augmentation:

- Expanding toy graph base, enriching the scale of the knowledge repository [114].
- Simulating Real-World Scenarios: Real-world graphs often encounter challenges such as missing nodes [42], noisy attributes [52], and unexplored connections [96]. We introduce node dropout, noise injection, and edge removal to simulate these scenarios accurately.
- Addressing Graph Domain Shift: To mitigate domain shift between the graph knowledge base and testing graphs, our augmentations employ Mixup techniques such as Node Interpolation and Edge Rewiring. These techniques interpolate between training samples to generate synthetic samples, effectively smoothing decision boundaries in embedding and reducing the model's sensitivity to minor variations in input data, thereby stabilizing predictions on domain shift testing samples [108].

B.2 Why Noise-based Graph Prompt Tuning is needed

To address inherent challenges in toy graph quality, we introduce Noise-based Graph Prompting Tuning (c.f. Section 4.3.3). This method involves fine-tuning the model with artificially introduced noisy toy graphs (Inner-Toy-Graph Noise & Toy-Graph Noise), inspired by noise-tuning techniques in NLP [19, 12, 115]. Our approach enhances the model's robustness against real-world retrieval noise, as evidenced by superior performance compared to traditional tuning methods (in Main Text Tables 1 and 2). This approach reduces the stringent requirement for an exceptionally high-quality graph vector base, thereby ensuring robust performance across various tasks within our RAGRAPH, and significantly mitigating data quality impacts.

B.3 Difficulty to construct and maintain high-quality and diverse graph vector base

In RAGRAPH, the toy graph base largely leverages significant prior research datasets in pre-trained GNNs [65, 104, 73, 123], which are trained on meticulously curated graph datasets and cover diverse domains, such as biology, chemistry, medicine recommendation tasks, etc. For example, the PROTEINS dataset [4], derived from cryo-electron microscopy and X-ray crystallography, and the ENZYMES dataset [99], based on EC enzyme classification, are meticulously annotated by medical experts.

B.4 Why Inverse Importance Sampling Strategy is needed

The adoption of the Inverse Importance Sampling strategy is crucial. In RAGRAPH, subgraphs are sampled as toy graphs, where nodes with higher degrees (non-long-tail knowledge, extensively learned and embedded into GNN parameters) are more frequently included in subgraphs due to their extensive connections with neighbors, resulting in higher frequency in toy graph base [27]. Conversely, nodes with low degrees (long-tail knowledge), are more important but ignored. To mitigate this issue, we propose this by prioritizing nodes with lower degrees to capture long-tail knowledge when sampling.

B.5 Why Four Similarities are needed

In practical applications, the four similarities all contribute to performance improvement and we state the significance as follows:

- Time information is crucial to predict future states or trends [113] via node history, i.e. in social networks, analyzing historical user interaction aids in predicting future behaviors.
- Structure pertains to how nodes are interconnected and overall graph topology, vital for capturing similar graph structure patterns [13, 42, 112]. In transportation networks, factories are always located on the outer ring of the city, sharing similar structural connectivity, aiding in the discovery of spatiotemporal patterns [54, 16].
- Sharing similar neighborhoods is essential for evaluating node similarity and correlation. In recommendations, shared purchase histories between users and products indicate potential interests, akin to collaborative filtering [87].
- Semantic information measures similarity based on features [73]. In knowledge graphs, identifying relevant subgraphs to query nodes enhances retrieval accuracy based on semantic similarity.

B.6 Why Knowledge Fusion is needed

Fusion and decoder here represent one of the core contributions of RAGRAPH:

- Overall Task Perspective: For the same tasks, the decoder can be directly employed to obtain outputs. For different tasks, the decoder can be masked and utilize pre-computed embeddings without training or be tuned to better adapt. This underscores our primary contribution, where the decoder functions as a versatile "plug-and-play" and "tune-free" component.
- Integral Fusion Strategy: Fusion Strategy facilitates concurrent information propagation from toy graphs X (hidden embeddings) and Y (task-specific output vector) to query graph, aligning with our secondary contribution.

B.7 How RAGRAPH works on par with RAG in NLP and CV

In NLP, RAG enhances the generation of LLM by retrieving relevant information via prompts. Similarly, in RAGRAPH, we enhance downstream graph learning by integrating information from retrieved toy graphs. Using these toy graphs with shared patterns assists the model inference. In our framework, the "generation" involves the retrieval-enhanced Graph Prompt: Toy Graph Intra Propagate & Query-Toy-Graph Inter Propagate to propagate retrieved knowledge (X and Y) into the query graph. To illustrate, we analyze this from both experiment and theory.

1. **Experiment 1:** We perform a case study to illustrate how "generation" works by displaying specific instances of node vectors in Appendix D.6.
2. **Experiment 2:** In traditional GNN tasks, GCN, GAT, and GIN typically expand their receptive fields through stacked message-passing layers or neighborhood subgraph sampling for inference. Patterns learned in these contexts are often localized within the constrained receptive field. In contrast, in RAGRAPH, we observe that subgraphs sharing similar patterns often exhibit properties more aligned with downstream tasks. These subgraphs provide richer information for inference compared to simply enlarging receptive fields. As shown in Main Text Tables 1 and 2, Figure 3, RAGRAPH's strategy of incorporating toy graphs significantly outperforms baselines.
3. **Theory 1:** Furthermore, we provide a theoretical justification of retrieval augmentation in GNNs (see Appendix B.4). From an information-theoretic perspective, introducing RAG knowledge into GNNs enhances the mutual information between input features X and output labels Y, such that:

$$I(X,RAG;Y) \geq I(X;Y),$$

   thereby improving the performance of downstream tasks. This is aligned with the information theory of RAG in NLP [71].
4. **Theory 2:** Recent studies [11, 136] also suggest the generalization error diminishes with an increase in the node number of the graph in Theorem 1.1 [136]: the generalization error between the expected loss $R_{\text{exp}}(\Theta) = \mathbb{E}_{(x,y)\sim\mu_G}[\mathcal{L}(\Theta(x),y)]$ and empirical loss $R_{\text{emp}}(\Theta) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\Theta(x^i),y^i)$ are super bounded:

$$|R_{\text{exp}}(\Theta) - R_{\text{emp}}(\Theta)| \leq \sqrt{\frac{C}{m}q(n)},$$

   where C represents the model complexity (e.g., parameters), m denotes the training set size, and $q(n) = \mathbb{E}_{n\sim\nu_G}[n^{-\frac{1}{D+1}}]$ depends on the average graph size (node number) with ν as the graph size distribution and D is the metric-measure space dimension. In RAGRAPH, retrieving similar toy graphs significantly increases the number of graph nodes (via Query-Toy-Graph Inter Propagate, linking toy graph nodes to query graph), significantly augmenting n while reducing $q(n)$. Consequently, the upper bound of generalization error decreases, promoting smoother graph learning convergence and enhancing pattern learning.

C Further Methods Details

C.1 Revisiting Graph Neural Networks

The goal of a GNN is to learn node embeddings based on an iterative aggregation of messages from the local network neighborhood. We use embedding matrix $\{\mathbf{z}_v^{(L)}\}_{v\in\mathcal{V}}$ to denote the embedding for all the nodes after applying an L-layer GNN. The l-th layer of a GNN, $\{\mathbf{z}_v^{(L)}\} = \text{GNN}^{(l)}(\{\mathbf{z}_v^{(l-1)}\})$, can be written as:

$$\begin{aligned}\mathbf{m}_{u\to v}^{(l)} &= \text{MSG}^{(l)}(\mathbf{z}_u^{(l-1)}, \mathbf{z}_v^{(l-1)}),\\ \mathbf{z}_v^{(l)} &= \text{AGG}^{(l)}\big(\{\mathbf{m}_{u\to v}^{(l)} \,|\, u\in\mathcal{N}(v)\}, \mathbf{z}_v^{(l-1)}\big),\end{aligned} \tag{7}$$

where $\mathbf{z}_v^{(l)}$ is the embedding for $v\in V$ after passing through l layers, $\mathbf{z}_v^{(0)} = x_v$ or h_v or o_v, $\mathbf{m}_{u\to v}^{(l)}$ is the message embedding, and $\mathcal{N}(v)$ is the set of direct neighbors of v. Different GNNs can have various definitions of message-passing functions $\text{MSG}^{(l)}(\cdot)$ and aggregation functions $\text{AGG}^{(l)}(\cdot)$ and these two functions could be parameter-free.

C.2 Key Construction of Position-aware Code

Given a randomly selected node anchor set $\mathcal{V}_S \subset \mathcal{V}$, we calculate the minimal distances, *a.k.a.* hops between the two node sets. Suppose $v_u \in \mathcal{V}, v_w \in \mathcal{V}_S$, the distance similarity between node v_u and v_w can be depicted as $dis(v_u,v_w)$. By normalizing the similarity to [0, 1], distance-to-centroid $d2c(v_u,v_w)$:

$$d2c(v_u,v_w) = \begin{cases} \dfrac{1}{dis(v_u,v_w)+1}, & \text{if } dis(v_u,v_w) < dis_q \\ 0, & \text{otherwise} \end{cases}, \tag{8}$$

here hyperparameter dis_q is the maximum hops, the distance beyond this boundary is considered invalid. The structure feature of node v_u is $d2c(v_u,\mathcal{V}_S)$. By collecting all distances with anchor-set $\mathcal{V}_S$, the structure $S \in R^{n\times|\mathcal{V}_S|}$ is written as follows:

$$\begin{aligned} &d2c(v_u,\mathcal{V}_S) = [d2c(v_u,v_w)|v_w \in \mathcal{V}_S \subset \mathcal{V}], \\ &S = [d2c(v_u,\mathcal{V}_S)|v_u \in \mathcal{V}], \end{aligned} \tag{9}$$

where $[\cdot]$ means the concatenation operation. The distance-to-centroid feature converts the non-Euclidean structure to the Euclidean structure. $d2c$ dramatically reduces the size of the matrix and meanwhile contains more structure information instead of identifier information, the size of the anchor set is $\log_2 n$ follows P-GNNs[45, 42].

C.3 Similarity Functions

For the history key, we adopt an exponential decay function to measure the time similarity values. We smooth the impact of time differences and provide a controlled decay coefficient $\eta > 0$. The time similarity, S_{time}, between the same node v_c and v_m with different timestamp $t(v_m)$,$t(v_c)$, is defined as:

$$S_{\text{time}}(v_c,v_m) = e^{-\eta|t(v_c)-t(v_m)|}, \tag{10}$$

where η is a positive parameter that controls the rate of exponential decay.

For the environment key, we match the environment of node v using Jaccard similarity to compare the sets of neighbors $\mathcal{N}(v_c)$ in the query graph and $\mathcal{N}(v_m)$ in the toy graph:

$$S_{\text{environment}}(v_c,v_m) = \frac{|\mathcal{N}(v_c) \cap \mathcal{N}(v_m)|}{|\mathcal{N}(v_c) \cup \mathcal{N}(v_m)|}. \tag{11}$$

For the hidden embedding key, we input the query graph into pre-trained GNNs to obtain the hidden embedding for the query node, with the similarity defined as:

$$h_c = \text{GNN}(X_{G^Q}), \quad S_{\texttt{semantic}}(v_c,v_m) = \texttt{cosine}(h_c,h_m). \tag{12}$$

For the position-aware code, we denote s_c,s_m as the position-aware code of node v_c,v_m, and utilize cosine similarity as before, defined as $S_{\text{structure}}(v_c,v_m) = \texttt{cosine}(s_c,s_m)$.

C.4 Proof of the Effectiveness of RAG

In this section, we will theoretically prove that introducing RAG knowledge can significantly improve the predictive performance of the model.

Assume X represents the input features, Y represents the target output labels, and RAG represents external knowledge related to the input features (or even the output labels). We analyze from the mutual information view, where $I(X;Y)$ quantifies the dependency between X and Y, which reflects the performance of the model, the larger the value, the better the performance of the model [54, 70]. By introducing RAG knowledge RAG into GNNs, we can effectively increase the mutual information between the input features X and the output labels Y as $I(X,RAG;Y) \geq I(X;Y)$, thereby improve

the model's downstream task performance. The derivation is as follows:

$$
\begin{aligned}
& I(X,RAG;\mathbf{Y}) - I(X;\mathbf{Y}) \\
&= \sum_{X,RAG,\mathbf{Y}} p(X,RAG,\mathbf{Y}) \log \frac{p(X,RAG,\mathbf{Y})}{p(X,RAG)p(\mathbf{Y})} - \sum_{X,\mathbf{Y}} p(X,\mathbf{Y}) \log \frac{p(X,\mathbf{Y})}{p(X)p(\mathbf{Y})} \\
&= \sum_{X,RAG,\mathbf{Y}} p(X,RAG,\mathbf{Y}) \log \frac{p(X,RAG,\mathbf{Y})}{p(X,RAG)p(\mathbf{Y})} - \sum_{X,RAG,\mathbf{Y}} p(X,RAG,\mathbf{Y}) \log \frac{p(X,\mathbf{Y})}{p(X)p(\mathbf{Y})} \\
&= \sum_{X,RAG,\mathbf{Y}} p(X,RAG,\mathbf{Y}) \log \left(\frac{p(X,RAG,\mathbf{Y})}{p(X,RAG)} \cdot \frac{p(X)}{p(X,\mathbf{Y})} \right) \\
&= \sum_{X,RAG,\mathbf{Y}} p(X,RAG,\mathbf{Y}) \log \frac{p(X,RAG,\mathbf{Y})}{p(RAG \mid X)p(X,\mathbf{Y})} \\
&= \sum_{X,RAG,\mathbf{Y}} p(RAG,\mathbf{Y} \mid X)p(X) \log \frac{p(RAG,\mathbf{Y} \mid X)}{p(RAG \mid X)p(\mathbf{Y} \mid X)} \\
&= I(RAG;\mathbf{Y} \mid X) \geq 0,
\end{aligned} \tag{13}
$$

where $\sum_{X,RAG,\mathbf{Y}} = \sum_X \sum_{RAG} \sum_{\mathbf{Y}}$. Moreover, $I(RAG;\mathbf{Y} \mid X)$ measures that how much additional knowledge RAG provides to assist in predicting Y based on X, this term will approach zero when the RAG is noise to the prediction task. In summary, the integration of RAG knowledge can enhance the mutual information between X and Y, thereby improving the performance and accuracy of downstream tasks.

C.5 Algorithms

In this section, we will provide a detailed description of the algorithms of Toy Graph Construction (Algorithm 1) and Training and Inference with Toy Graphs Retrieval (Algorithm 2).

Algorithm 1. We outline the process for constructing toy graphs in Algorithm. 1. In line 1, the toy graph database $\mathcal{G}^{\mathcal{T}}$ is initialized.

Lines 2-15 describe the steps to construct toy graphs by iterating through each snapshot of the dynamic resource graph $\mathcal{G}^{\mathcal{R}}$.

In more detail, lines 3-5 details calculate the importance and reverse importance of each node within the snapshot. Following this, lines 6-7 involve normalizing the sampling probabilities according to the reverse importance values. The selection of a master node and the generation of its k-hop ego network are carried out in lines 8-9. Subsequently, line 10 involves augmenting the toy graph through specific data augmentation techniques.

Lines 11-16 detail the generation of key-value pairs for each toy graph. This includes saving the timestamp as the history key, the neighbors of the master node as the environment key, the structural encoding as the structure key, the hidden embedding as the semantic key, and the task-specific output vectors as the value. Each toy graph is then stored in the database $\mathcal{G}^{\mathcal{T}}$.

Ultimately, in line 18, the algorithm returns the toy graph database $\mathcal{G}^{\mathcal{T}}$.

Algorithm 2. We introduce the algorithm for training and inference with toy graph retrieval in Algorithm. 2. Initially, in line 1, we define the required inputs, including the testing graph $\mathcal{G}_{\text{test}}$, the toy graph database $\mathcal{G}^{\mathcal{T}}$, and other relevant parameters. The final output is the aggregated result $\tilde{o}_c$.

The RETRIEVETOYGRAPHS function, detailed in lines 3-11, initializes an empty similarity list and iterates over each toy graph in the database. Lines 5-6 compute various similarity metrics, and the overall similarity is determined in line 7. This similarity score is then added to the list. After sorting by similarity, the $topK$ toy graphs are retrieved and returned in line 11.

Within the PROPAGATION function (lines 12-17), each retrieved toy graph undergoes intra-propagation in line 14. The intra-propagation step follows in line 16, ultimately returning the propagated results $\mathbf{z}_c$.

The KNOWLEDGEFUSION function, found in lines 18-21, combines the outputs from previous steps. The final combined outputs $\tilde{o}_c$ are generated by the decoder and returned in line 21.

Algorithm 1 Toy Graph Construction

Require: Dynamic Resource Graph $\mathcal{G}^{\mathcal{R}}$, node importance balance weight α, toy graph scaling constant K, maximum hop k
Ensure: Toy graph embedding key-value database $\mathcal{G}^{\mathcal{T}}$
1: Initialize toy graph database $\mathcal{G}^{\mathcal{T}} \leftarrow \emptyset$
2: **for** each snapshot $G_\tau^{\mathcal{R}} \in \mathcal{G}^{\mathcal{R}}$ **do** ▷ **Construct Toy Graphs**
3: **for** each node $v \in G_\tau^{\mathcal{R}}$ **do**
4: Calculate importance $I(v) \leftarrow \alpha \mathrm{PR}(v) + (1-\alpha)\mathrm{DC}(v)$
5: Reverse node importance $I'(v) \leftarrow \frac{1}{I(v)+\epsilon}$
6: **end for**
7: **for** each node $v \in G_\tau^{\mathcal{R}}$ **do**
8: Normalize sampling probabilities $p_i \leftarrow \frac{I'(v_i)}{\sum_{j=1}^{n} I'(v_j)}$
9: **end for**
10: Sample master node $v_m \leftarrow \textsc{WeightedSampling}(G_\tau^{\mathcal{R}}, p_i)$ based on probability p_i
11: Generate k-hop ego net $G_\tau^e(v_m)$ for node v_m
12: Augment toy graph $\{G^{\mathcal{T}}\} \leftarrow \textsc{DataAugmentation}(G_\tau^e(v_m), n_{\text{aug}})$ with $n_{\text{aug}}(G_\tau^e(v_m)) = \lfloor K \cdot \bar{I}'(G_\tau^e(v_m)) \rfloor$
13: **for** each $G^{\mathcal{T}} \in \{G^{\mathcal{T}}\}$ **do** ▷ **Generate keys-values pair**
14: Save timestamp τ as history key
15: Save neighbors $\mathcal{N}(v_m^\tau)$ of master node v_m as environment key
16: Save structural encoding s_m^τ of node v_m via Eq. (9) as structure key
17: Save hidden embedding h_m^τ by feeding $G^{\mathcal{T}}$ into pre-trained GNNs as semantic key
18: Save the hidden embedding $\{h_i^\tau | v_i \in G^{\mathcal{T}}\}$ as value
19: Save task-specific output vectors $\{o_i^\tau | v_i \in G^{\mathcal{T}}\}$ by feeding $\{h_i^\tau | v_i \in G^{\mathcal{T}}\}$ into decoder as value
20: Store toy graph $G^{\mathcal{T}}$ into database $\mathcal{G}^{\mathcal{T}}$
21: **end for**
22: **end for**
23: **return** Toy graph database $\mathcal{G}^{\mathcal{T}}$

The main algorithm begins in line 22. If fine-tuning is enabled and the prompt loss has not converged, lines 23-34 detail the process of toy graph retrieval and propagation for each query graph. This includes the optional addition of noise in lines 26-29. The hidden embedding and task-specific output vectors are propagated in lines 30-31, and the aggregated outputs are fused in line 32. The prompt loss is computed, and fine-tuned parameters are updated in lines 33-34.

If fine-tuning is not required, lines 35-39 describe a similar process of toy graph retrieval and propagation, without the fine-tuning steps. The aggregated outputs are computed directly.

D Further Experiment Details

D.1 Datasets Statics

We employ eight benchmark datasets for evaluation including four public static classification datasets for node- and graph-level tasks.

(1) PROTEINS [4] is a collection of protein graphs, including the amino acid sequence, conformation, structure, and features such as active sites of the proteins. Each node represents a secondary structure, while each edge illustrates the neighboring relationship either within the amino acid sequence or in 3D space. The nodes are divided into three categories, and the graphs are classified into two distinct classes. This dataset is used for node and graph classification tasks, containing 1,113 graphs with an average of 39.06 nodes and 72.82 edges per graph, with a density of 4.8e-2.

(2) COX2 [83] is a dataset of molecular structures, including 467 cyclooxygenase-2 inhibitors. Each node represents an atom, and each edge signifies a chemical bond between atoms, such as single, double, triple, or aromatic bonds. All the molecules belong to two categories. This dataset is used for

Algorithm 2 Training and Inference with Toy Graphs Retrieval

Require: Testing graph $\mathcal{G}_{\text{test}}$, toy graph database $\mathcal{G}^{\mathcal{T}}$, pre-trained GNN model $\text{GNN}_{\Theta_0}(\cdot)$, number of $TopK$ toy graphs to retrieve, similarity weights w_1,w_2,w_3,w_4, fine-tuning flag $fine\_tune$, noise prompt-tuning flag add_noise

Ensure: Aggregated output $\tilde{o}_c$

function RETRIEVETOYGRAPHS($G^{\mathcal{Q}}$, $\mathcal{G}^{\mathcal{T}}$, $TopK$)
  Initialize similarity list $\{S\} \leftarrow \emptyset$
  for each toy graph $G^{\mathcal{T}} \in \mathcal{G}^{\mathcal{T}}$ **do**
    Calculate time similarity S_{time}, environment similarity $S_{\text{environment}}$, structure similarity $S_{\text{structure}}$ and semantic similarity S_{semantic}
    Compute similarity $S \leftarrow w_1 \cdot S_{\text{time}} + w_2 \cdot S_{\text{structure}} + w_3 \cdot S_{\text{environment}} + w_4 \cdot S_{\text{semantic}}$
    Add $(G^{\mathcal{T}}, S)$ to $\{S\}$
  end for
  Sort $\{S\}$ by similarity in descending order
  Retrieve $topK$ toy graphs $G^{\mathcal{T}}_{\text{TopK}} \leftarrow \{G^{\mathcal{T}} \in \{S\}$ with $topK$ similarities$\}$
  return Retrieved toy graphs $G^{\mathcal{T}}_{\text{TopK}}$
end function
function PROPAGATION($G^{\mathcal{Q}}$, $G^{\mathcal{T}}_{\text{TopK}}$)
  for each toy graph $G^{\mathcal{T}} \in G^{\mathcal{T}}_{\text{TopK}}$ **do**
    Perform Intra Propagation $\mathbf{z}_m \leftarrow$ INTRAPROPAGATION($G^{\mathcal{T}}$)
  end for
  $\mathbf{z}_c \leftarrow$ INTERPROPAGATION($G^{\mathcal{Q}}$, $G^{\mathcal{T}}_{\text{TopK}}$)
  return $\mathbf{z}_c$
end function
function KNOWLEDGEFUSION(h_c,o_c)
  Combined output $\hat{o}_c \leftarrow \gamma o_c + (1-\gamma)$DECODER$(h_c)$
  return Combined outputs $\tilde{o}_c$
end function
if $fine\_tune$ & $\mathcal{L}_{\text{prompt}}$ not converged **then**
  for each query graph $G^{\mathcal{Q}} \in \mathcal{G}_{\text{test}}$ **do** ▷ Toy Graph Retrieval and Propagation
    $G^{\mathcal{T}}_{\text{TopK}} \leftarrow$ RETRIEVETOYGRAPHS($G^{\mathcal{Q}}$, $\mathcal{G}^{\mathcal{T}}$, $TopK$)
    if add_noise **then**
      for each toy graph $G^{\mathcal{T}} \in G^{\mathcal{T}}_{\text{TopK}}$ **do**
        Introduce noise $G^{\mathcal{T}} \leftarrow$ ADDNOISE($G^{\mathcal{T}}$) ▷ Inner-Toy-Graph Noise
      end for
      Add $bottomK$ toy graphs to $G^{\mathcal{T}}_{\text{TopK}}$ ▷ Toy-Graph Noise
    end if
    $h_c \leftarrow$ PROPAGATION($G^{\mathcal{Q}}$, $G^{\mathcal{T}}_{\text{TopK}}$) ▷ Propogate hidden embedding
    $o_c \leftarrow$ PROPAGATION($G^{\mathcal{Q}}$, $G^{\mathcal{T}}_{\text{TopK}}$) ▷ Propogate task-specific output vector
    Aggregated outputs $\hat{o}_c \leftarrow$ KNOWLEDGEFUSION(h_c,o_c)
    Compute loss $\mathcal{L}_{\text{prompt}}$ via $\tilde{o}_c$ and $\hat{o}_c$ ▷ Based on task-specific loss function
    Update fine-tuned parameters Θ by minimizing $\mathcal{L}_{\text{prompt}}$
  end for
else
  for each query graph $G^{\mathcal{Q}} \in \mathcal{G}_{\text{test}}$ **do** ▷ Toy Graph Retrieval and Propagation
    $G^{\mathcal{T}}_{\text{TopK}} \leftarrow$ RETRIEVETOYGRAPHS($G^{\mathcal{Q}}$, $\mathcal{G}^{\mathcal{T}}$, $TopK$)
    $h_c \leftarrow$ PROPAGATION($G^{\mathcal{Q}}$, $G^{\mathcal{T}}_{\text{TopK}}$) ▷ Propogate hidden embedding
    $o_c \leftarrow$ PROPAGATION($G^{\mathcal{Q}}$, $G^{\mathcal{T}}_{\text{TopK}}$) ▷ Propogate task-specific output vector
    Aggregated outputs $\hat{o}_c \leftarrow$ KNOWLEDGEFUSION(h_c,o_c)
  end for
end if
return Aggregated outputs $\tilde{o}_c$

Table 4: Statistics of the experimental datasets and summary of datasets.

Statistics	TAOBAO	KOUBEI	AMAZON	PROTEINS	COX2	ENZYMES	BZR
# Nodes per Graph	204,168	221,366	238,735	39.06	41.22	32.63	35.75
# Edges per Graph	8,795,404	3,986,609	876,237	72.82	43.45	62.14	38.36
# Density	8.6e-4	3.3e-4	6.2e-5	4.8e-2	2.6e-2	5.9e-2	3.0e-2
# Graphs	1	1	1	1,113	467	600	405
# Graph Classes	/	/	/	2	2	6	2
# Node Features	1	1	1	1	3	18	3
# Node Classes	/	/	/	3	/	3	/
Snapshot Granularity	daily	weekly	weekly	/	/	/	/
Task	Edge	Edge	Edge	Node, Graph	Graph	Node, Graph	Graph
Type	Dynamic	Dynamic	Dynamic	Static	Static	Static	Static
Dataset Partition	Snapshot	Snapshot	Snapshot	Node, Graph	Graph	Node, Graph	Graph

graph classification tasks, with each graph having an average of 41.22 nodes and 43.45 edges and a density of 2.6e-2.

(3) ENZYMES [99] is a dataset of 600 enzymes collected from the BRENDA enzyme database. These enzymes are labeled into 6 categories according to their top-level EC enzyme classification. This dataset is used for node and graph classification tasks, with each graph having an average of 32.63 nodes and 62.14 edges and a density of 5.9e-2.

(4) BZR [83] is a collection of 405 ligands for the benzodiazepine receptor. Each ligand is represented by a graph, and all ligands are categorized into two groups. This dataset is used for graph classification tasks, with each graph having an average of 35.75 nodes and 38.36 edges and a density of 3.0e-2.

Additionally, we leverage three publicly available datasets encompassing a wide array of real-world scenarios in dynamic recommendation (link prediction):

(5) The TAOBAO dataset captures implicit feedback data from Taobao.com, a prominent Chinese e-commerce platform, collected over a span of 10 days. This dataset is used for edge classification tasks, containing 204,168 nodes and 8,795,404 edges, with a density of 8.6e-4.

(6) The KOUBEI dataset records 9 weeks of user interactions with nearby stores on Koubei, a platform integrated within Alipay. This dataset is used for edge classification tasks, containing 221,366 nodes and 3,986,609 edges, with a density of 3.3e-4.

(7) The AMAZON dataset comprises a collection of product reviews sourced from Amazon, spanning a duration of 13 weeks. This dataset is used for edge classification tasks, containing 238,735 nodes and 876,237 edges, with a density of 6.2e-5.

These datasets' detailed statistics are summarized in Table 4. The "Task" column provides information about the type of downstream task conducted on each dataset: "Node" denotes node classification tasks, "Graph" signifies graph classification tasks, and "Edge" indicates tasks related to link prediction. The "Type" column indicates the type of graph dataset: "Dynamic" for dynamic dataset $t \geq 1$, and "Static" for static dataset $t = 1$. For dynamic datasets, the "Snapshot Granularity" denotes the time granularity for each dataset. In our experimental setup, for dataset partition, dynamic graphs are partitioned according to snapshots, while static graphs are partitioned either by node or by the entire graph.

D.2 Evaluation Matrices

Node and Graph classification evaluation. For the node classification, we use the prediction accuracy to measure the model.

Link prediction evaluation. For the link prediction, we evaluate the recall and ranking quality of the effects of recommendation following previous studies [117, 30]. We use Recall@k and NDCG@k as metrics. Note that this task should be a binary task. We denote the $topk$ largest value as $rel_{ij}, j \in [1,k]$ for node v_i.

Recall@k measures the ratio of true positive links contained in the top k predicted links for each node:

$$\text{Recall@}k = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k}\frac{rel_{ij}}{\sum\mathbb{I}(A[i:]>0)}, \tag{14}$$

where $rel_{ij}=1$ if the j-th predicted link for node v_i exists, otherwise 0. $\mathbb{I}(\cdot)$ is the indicator function, and if $A[i:]>0$ then $\mathbb{I}(A[i:]>0)=1$.

NDCG@k (Normalized Discounted Cumulative Gain) is computed by normalizing DCG@k (Discounted Cumulative Gain) which accounts for the position of correctly predicted links. DCG@k is defined as:

$$\text{DCG@}k = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k}\frac{rel_{ij}}{\log_2(j+1)}. \tag{15}$$

D.3 Baseline Details

In this section, we present the details of baselines.

Table 5: Baseline Code URLs of Github Repository

Baseline	Type	Code Repo URL
GCN	Static	https://github.com/tkipf/gcn
GraphSAGE	Static	https://github.com/williamleif/GraphSAGE
GAT	Static	https://github.com/PetarV-/GAT
GIN	Static	https://github.com/weihua916/powerful-gnns
LightGCN	Dynamic	https://github.com/kuandeng/LightGCN
SGL	Dynamic	https://github.com/wujcan/SGL-Torch
MixGCF	Dynamic	https://github.com/Wu-Xi/SimGCL-MixGCF
SimGCL	Dynamic	https://github.com/Wu-Xi/SimGCL-MixGCF
GraphPro	Dynamic	https://github.com/HKUDS/GraphPro
GraphPrompt	Dynamic, Static	https://github.com/Starlien95/GraphPrompt
PRODIGY	Dynamic, Static	https://github.com/snap-stanford/prodigy

- **GCN** [48]: GCN is an end-to-end learning framework for graph-structured data. It utilizes neighborhood aggregation to integrate structural information, which is particularly effective in node classification and graph classification tasks.

- **GraphSAGE** [29]: GraphSAGE, is a general and inductive framework that leverages node feature information (e.g., text attributes) to efficiently generate node embeddings for previously unseen data.

- **GAT** [97]: GAT is a spatial domain method, which aggregates information through the attention-learned edge weights.

- **GIN** [105]: GIN utilizes a multi-layer perceptron to sum the results of GNN and learns a parameter to control residual connection.

- **LightGCN** [30]: LightGCN learns user and item embeddings by linearly propagating them on the user-item interaction graph, and uses the weighted sum of the embeddings learned at all layers as the final embedding.

- **SGL** [103]: SGL is to supplement the classical supervised task of recommendation with an auxiliary self-supervised task, which reinforces node representation learning via self-discrimination.

- **MixGCF** [39]: MixGCF generates the synthetic negative by aggregating embeddings from different layers of raw negatives' neighborhoods to perform collaborative filtering.

- **SimGCL** [117]: SimGCL applies unsupervised contrastive learning to enhance representation learning, making it suitable for link prediction tasks. It is applied to dynamic graphs to test its adaptability and performance.
- **GraphPro** [113]: GraphPro extends GraphPrompt by introducing spatial and temporal prompts tailored for dynamic graph learning, enhancing the ability to capture both structural and temporal relationships within graph data.
- **GraphPrompt** [65]: GraphPrompt integrates pre-training and downstream tasks using a unified template approach and employs task-specific prompts to enhance sub-task learning, applicable to both dynamic and static graph contexts.
- **PRODIGY** [73]: PRODIGY focuses on facilitating downstream tasks through in-context examples and learning from the $X \rightarrow Y$ paradigm. It is implemented to enhance learning in both dynamic and static graphs by leveraging contextual learning strategies.

D.4 Implementation Details

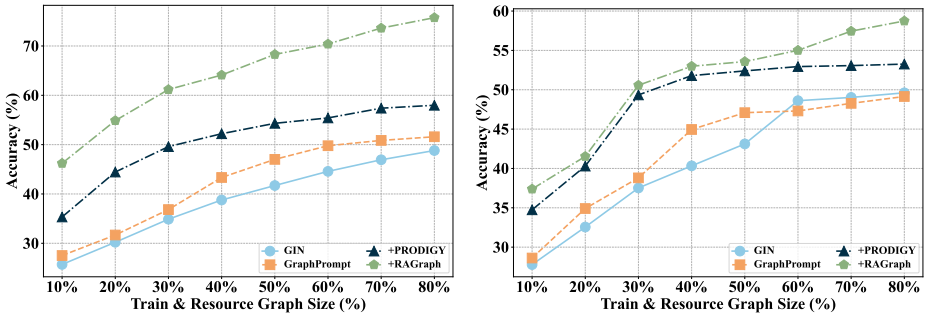
Implementations are done using the PyTorch 2.3.0 framework [78] in Python 3.11, on an Ubuntu server equipped with 1 V100 GPU and an Intel(R) Xeon(R) CPU.

In node and graph classification tasks: For baseline GCN [48], we employ a 2-layer architecture and set the hidden dimension as 256. For GraphSAGE [29], we utilize the mean aggregator and employ a 2-layer architecture. The hidden dimension is also set to 256. For GAT [97], we employ a 2-layer architecture and set the hidden dimension as 256. Besides, we apply 8 attention heads in the first GAT layer. Similarly, for GIN [105], we also employ a 2-layer architecture and set the hidden dimension as 256. For GraphPrompt, we follow [65] to employ a 2-layer GCN as the backbone and set the hidden dimensions as 256.

In the link prediction task: For LightGCN, SGL, MixGCF, SimGCL and GraphPro, we employ a 3-layer GNN architecture and set the hidden dimension as 64 with Low-Rank Adaptation (LoRA) [32] rank equals to 16. For GraphPro, the backbone graph encoder is SimGCL.

Moreover, for all three tasks, the hyper-parameters of baselines are based on the recommended values provided in the paper. In PRODIGY and RAGRAPH, k is set to 2, $topK$ is set to 5, γ is set to 0.8 for PROTEINS and 0.5 for ENZYMES in node level, γ is set to 0.5 for PROTEINS, 0.6 for COX2, 0.8 for ENZYMES and 0.5 for BZR in graph level, $\alpha = \lambda = 0.5, K = 3, w_1 = w_2 = w_3 = 0.05, w_4 = 0.85$.

D.5 Resource Graph Scalability Study



(a) Node Classification on ENZYMES dataset (b) Node Classification on PROTEINS dataset

Figure 4: Performance comparisons of RAGRAPH and several baselines with different proportions of training and resource data.

We assess the impact of varying amounts of training and resource data on model performance. As illustrated in Figure 4, we vary the proportion of train and resource graph size from 10% to 80%, with increments of 10%, and conduct experiments on node classification tasks using the ENZYMES and PROTEINS datasets, respectively. For comparative analysis, we select GIN, GraphPrompt, and PRODIGY as baseline models. To ensure fairness in our experiments, we maintain a consistent ratio of train to resource data at 3:5 during fine-tuning, utilizing the sum of these as a retrieval database.

As shown in Figure 4, there is a clear trend where the accuracy of the model improves as the proportion of the dataset increases. However, the rate of accuracy improvement starts to plateau once a certain dataset proportion is reached (*i.e.*, 30% in PROTEINS for PRODIGY and RAGRAPH, 40% in PROTEINS for GraphPrompt). Among the evaluated models, GIN and GraphPrompt show the slowest convergence rates, whereas PRODIGY converges at a moderate pace, and RAGRAPH converges the fastest. This rapid convergence in PRODIGY and RAGRAPH is attributed to its ability to engage in effective knowledge retrieval, significantly enhancing the model's comprehension abilities. Remarkably, both PRODIGY and RAGRAPH can achieve commendable results in downstream tasks even with a small proportion of the dataset. Compared to PRODIGY, RAGRAPH exhibits superior performance because while PRODIGY primarily learns a mapping from X to Y, RAGRAPH not only learns this mapping but also integrates additional knowledge into GNNs more effectively. This integration becomes increasingly beneficial as the dataset proportion grows, allowing RAGRAPH to outperform other models, particularly at higher data volumes where it can better leverage its knowledge integration capabilities.

D.6 Qualitative Analyses of Toy Graphs Retrieving

In this section, we conduct qualitative analyses of the toy graphs retrieving experiment. For the sake of understanding, we conduct experiments under normal settings where the dimensionality of the task-specific output vector is equal to the number of classes.

On the ENZYMES dataset, for a 3-class node classification task, regarding node "13984", which belongs to class 3, if we only use the GraphPrompt Backbone, the resulting one-hot encoding is $[0.28,0.34,0.38]$.

However, since the node is of class 3, we expect the one-hot encoding to be as close as possible to $[0,0,1]$. In RAGRAPH retrieval, taking the top 3 retrieved graphs as examples, the connection weights for these 3 toy graphs to query graphs are 0.5, 0.7, and 0.1, respectively, and their corresponding label one-hot encodings are $[0,0,1]$, $[0,0,1]$, and $[0,1,0]$. Therefore, the result obtained by propagating the task-specific output vector through toy graphs is: $[0,0.1,1.2]$, and after normalization, the result is $[0,0.08,0.92]$.

Meanwhile, the vector obtained by propagating toy graphs hidden embedding and via decoder is $[0.37,0.32,0.66]$. The retrieval of toy graphs notably enhances performance at both the task-specific output vector and hidden embedding levels. The final vector is obtained through a weighted sum with $\gamma = 0.5$ in Eq(6) is $[0.185,0.20,0.79]$, after normalization the result is $[0.157,0.170,0.673]$, which greatly enhances the model's discriminative ability compared to GraphPrompt $[0.28,0.34,0.38]$.

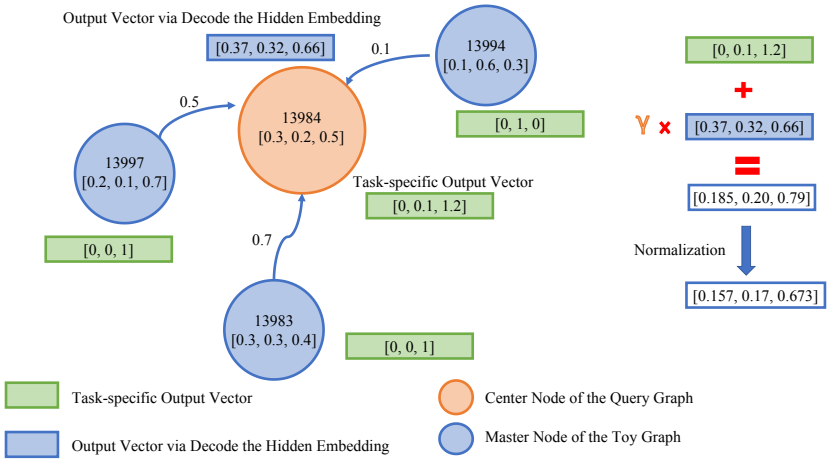


Figure 5: Qualitative analyses of toy graphs retrieving – how "generation" works.

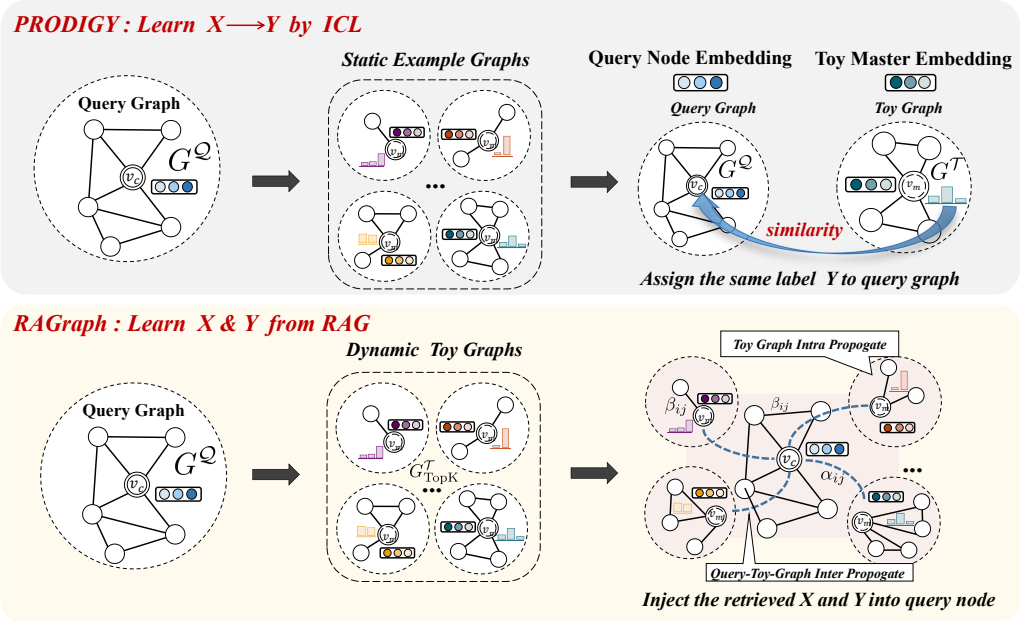


Figure 6: Difference Illustration between PRODIGY and RAGRAPH.

E Difference between ICL (PRODIGY) and RAG (RAGRAPH)

In this section, we explore the distinctions between PRODIGY and RAGRAPH from several critical perspectives, as illustrated in Figure 6:

- **PRODIGY:** This approach utilizes fixed examples as rules, which may not be optimal for dynamic and evolving scenarios. PRODIGY primarily focuses on learning direct mappings from X to Y through in-context learning. However, it encounters challenges in integrating external information that is more pertinent to the query node. This is particularly problematic when the distribution of each node belonging to the same label class varies, and simply learning the mapping based on the prototype node will somehow be misleading.
- **RAGRAPH:** In contrast, RAGRAPH is designed to handle non-static, streaming knowledge, making it well-suited to dynamic graph structures and evolving tasks. It actively retrieves relevant knowledge on-demand, effectively incorporating information about both X and Y from external sources into GNNs. Moreover, RAGRAPH can operate without the need for model fine-tuning, providing substantial flexibility. This adaptability enables RAGRAPH to excel in tasks that require continuous adaptation to changing conditions and the integration of external, relevant information.

In summary, we argue that a qualified Retrieval-Augmented Generation (RAG) system for Graph Learning should fulfill several essential criteria to effectively support complex reasoning tasks: 1) It should retrieve ample feature and task-related label information, analogous to how attributes are gathered in the NLP domain to stimulate the reasoning capabilities of LLMs; 2) The system should adapt to new tasks or unseen datasets without requiring fine-tuning of model parameters; 3) Knowledge within the system must be dynamically updated and stored, ensuring current and relevant data utilization.

F Broader Impacts

Our work builds on the widespread application of Retrieval-Augmented Generation (RAG) in large language models (LLMs) and aims to extend its success to graph data, thereby constructing graph foundation models. This approach allows models to transfer rapidly without requiring learnable parameters, avoiding potential performance degradation from fine-tuning pre-trained models. As a result, RAG is particularly effective in domains with scarce and long-tail data, such as network anomaly detection, rare disease diagnosis/treatment, supply chain disruption, and new user recommendations.

Additionally, our model establishes an excellent paradigm by incorporating retrieved features and label information into the learning process, significantly enhancing the model's understanding capabilities. Our work provides valuable insights and serves as a reference for future Large Graph Models.

G Data Ethics Statement

To evaluate the efficacy of this work, we conducted experiments that only use publicly available datasets, namely, PROTEINS, COX2, ENZYMES, BZR[3], TAOBAO, KOUBEI and AMAZON in accordance to their usage terms and conditions if any. We further declare that no personally identifiable information was used, and no human or animal subject was involved in this research.

[3] https://chrsmrrs.github.io/datasets/

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: In the introduction section, we delineate the problems addressed by this work and outline our contributions.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: In the conclusion section, we highlight the limitations of the current work and suggest directions for future research.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: We provide the complete theoretical proofs in Appendix C.4.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide detailed experiment settings in Appendix D.4. Besides, code is anonymously available at `https://anonymous.4open.science/r/GLM-RAG-049D/`.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Code is anonymously available at https://anonymous.4open.science/r/GLM-RAG-049D/.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: We provide experiment settings in Section 5.1 and Appendix D.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: For each experiment, we conducted 5 repeated experiments and reported the standard deviation in Section 5.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide sufficient information on the computer resources in Appendix D.4.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: I have read the NeurIPS Code of Ethics and I confirm our research in the paper conforms with Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We have discussed the potential impacts in Appendix F.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The graph neural network framework proposed in our paper does not extend to application domains requiring safeguards. Additionally, the datasets used are widely-used node classification datasets, thus eliminating the need for specific safeguards.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We provide source links for all datasets and baselines in Appendix D, and we have cited all referenced works.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We release our code anonymously at `https://anonymous.4open.science/r/GLM-RAG-049D/`.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.