

AI-Enabled KG Platform: Incorporating LLMs and GNNs into KG Engines

Hussein Abdallah

**A Thesis
in The Department
of
Computer Science and Software Engineering**

**Presented in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy (Computer Science)
at Concordia University
Montréal, Québec, Canada**

December 2025

© Hussein Abdallah, 2026

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis

prepared By: **Mr. Hussein Abdallah**

Entitled: **AI-Enabled KG Platform:
Incorporating LLMs and GNNs into KG Engines**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

Dr. Adel Hanna Chair

Dr. Arijit Khan External Examiner

Dr. Jun Yan Arms-Length Examiner

Dr. Yann-Gaël Guéhéneuc Examiner

Dr. Abdelhak Bentaleb Examiner

Dr. Essam Mansour Supervisor

Approved by _____
Paquet, Joey, Chair
Department of Computer Science and Software Engineering

_____ 2025

MOURAD DEBBABI, Dean
Faculty of Engineering and Computer Science

Abstract

AI-Enabled KG Platform: Incorporating LLMs and GNNs into KG Engines

Hussein Abdallah, Ph.D.

Concordia University, 2026

Heterogeneous graphs (HGs) are complex networks with diverse node and edge types, capturing intricate interconnections. Knowledge graphs (KGs) are a specialized type of HGs that store real-world entities and their relationships as triples. Major companies, such as Meta, Google and LinkedIn, utilize KGs for multi-domain knowledge management and information retrieval. Heterogeneous graph neural networks (HGNNs) have emerged as an effective method for learning the structural semantics within graphs and generating graph-level or entities-level embeddings as features for downstream ML tasks such as node classification and link predictions. GNN embeddings can also support knowledge graph question answering (KGQA) by integrating structural embeddings with the reasoning capabilities of large language models through a Retrieval-Augmented Generation (RAG) pipeline. In this setting, the LLM is augmented with relevant knowledge graph information to address unseen questions, thereby enhancing reasoning, reducing hallucinations, and eliminating the need for costly domain-specific fine-tuning.

Training GNN models on top of existing KG engines for large-scale KGs remains a challenging task. Firstly, *KG data migration is inefficient*. Existing graph ML libraries, like DGL and PYG do not support training HGNN models directly on top of existing KG engines. They require exporting the whole KG data, which is impractical for large-scale KGs and fails to capture frequent KG updates. Also, it requires deep experience in graph ML to write complex data transformation, model training, and model inference pipelines. Secondly, *existing HGNN training methods are task-agnostic* as they are trained on the entire KG data, including task-irrelevant data. This adds

noise, increases training time and memory usage, and results in complex multi-layered GNN models with costly inference time. Moreover, task-agnostic training does not guarantee optimal performance for specific tasks. Thirdly, *existing KGQA methods are not designed for question answering over incomplete KGs; the common scenario in production*. Existing KGQA methods combined with graph-RAG pipelines struggle when the question answer is missing from the KG and incur high retrieval and prompting costs in large-scale KGs .

My thesis addresses these challenges through KGNET, an AI-enabled KG platform designed to overcome the scalability limitations of HGNN models on large-scale KGs. KGNET tackles three key challenges: (1) The seamless integration of HGNNs, LLMs, and KG engines through our novel system design that extends SPARQL language to support graph ML tasks and thus automates the graph ML training/inference pipelines directly on KG engines with a lightweight task-relevant data extraction and a basic knowledge of graph ML. (2) KGTOSA, a task-oriented sampling method that enables scalable and efficient HGNN training on large-scale knowledge graphs, ensuring high performance in node classification and link prediction tasks without compromising accuracy. (3) Incorporating a graph RAG pipeline with HGNNs that enables effective KGQA over incomplete KGs.

Acknowledgments

It feels as though I began my Ph.D. only yesterday, back in 2020. I still vividly recall the mixture of excitement and anxiety upon my arrival at the Concordia campus Five years ago. As I now conclude this journey, I can confidently state that I have completed my Ph.D. without regret, a milestone made possible through the support and encouragement of many individuals along the way.

First and foremost, I would like to express my deepest gratitude to my advisor, Essam Mansour, for his guidance, mentorship, expertise, and patience. Thanks to my co-authors, Panos Kalins and Ashraf Aboulmaga. They taught me to think critically, work smartly, and their commitment to excellence inspired me throughout this journey. I am grateful for my committee members Yann-Gaël Guéhéneuc, Jun Yan, Abdelhak Bentaleb, and Arijit Khan for serving on my committee and for providing essential feedback throughout the 5 years. In addition, I would like to thank the amazing research students I have worked with, and especially the members who contributed to this work: Walid Afandi and Yasaman Sabbagh, and the members of CoDs.ai Lab: Ahmed, Mossad, and Reham; my friends and colleagues.

I am very grateful to my parents, The mercy of ALLAH up on them, who supported me throughout my life and provided the foundation upon which I have built my education. Their love, patience, and great sacrifices have made this thesis possible. A special thanks to my brothers; especially Khalid and Hassan, and my sisters for always cheering me on for their invaluable support.

Last but not least, I extend my deepest gratitude to my wife, Samar, and my children, Mustasem and Moaaz, for their unwavering support, companionship, advice, joy, and love throughout this truly challenging Ph.D. journey. The past five years have been both memorable and demanding, and this thesis would not have been possible without you.

Contents

List of Figures	x
List of Tables	xvi
1 Introduction	1
1.1 Motivation	1
1.2 Thesis Contributions & Outline	4
1.3 Statement Regarding Authorship and Contributions	5
1.4 Publications List	5
1.5 Notation	6
2 Preliminary	7
2.1 Knowledge Graph (KG)	7
2.2 Knowledge Graph Querying	8
2.3 Knowledge Graph Embedding (KGE)	9
2.4 Graph Convolutional Neural Networks (GNN)	9
2.5 Heterogeneous Graph Convolutional Neural Networks (HGNN)	10
3 Towards a GML-Enabled Knowledge Graph Platform	12
3.1 Introduction	12
3.2 Background: ML pipelines for KGs	16
3.3 Challenges of GML-enabled KG Engine	17
3.3.1 Automatic Training: Method Selection and Graph-sampling	17

3.3.2	Seamless Integration Between GML Models and KGs	19
3.3.3	Optimizing SPARQL ^{ML} Queries and Benchmarks	19
3.4	The KGNET Platform	20
3.4.1	GML as a Service (GMLaaS)	20
3.4.2	The SPARQL ^{ML} as a Service	22
3.5	Experimental Evaluation	26
3.5.1	Evaluation Setup	26
3.5.2	GML Experiments With Real KGs	27
3.6	Related Work	28
3.7	Discussion	30

4 Task-Oriented GNNs Training on Large Knowledge Graphs for Accurate and Efficient

Modeling		31
4.1	Introduction	31
4.2	Background	35
4.2.1	HGNNs on KGs	35
4.2.2	Sampling Techniques Adapted in HGNNs	36
4.2.3	GNN Tasks on KGs	37
4.3	The KGTOSA Extraction Approach	38
4.3.1	Characteristics of Effective HGNNs Training	39
4.3.2	The KGTOSA’s Graph Pattern	41
4.4	Our Graph Pattern in Practice: Algorithms	42
4.4.1	Biased Random Walk Sampling	43
4.4.2	Influenced-based Sampling	44
4.4.3	A SPARQL-based Extraction Method	46
4.5	EXPERIMENTAL EVALUATION	48
4.5.1	Evaluation Setup	48
4.5.2	The KGTOSA Impact on HGNN Methods	51
4.5.3	Analyzing KGTOSA	54

4.6	Related Work	58
4.7	Discussion	59
4.7.1	Limitations	59
5	Leveraging LLM-GNN Integration for Open-World Question Answering over Knowl- edge Graphs	60
5.1	Introduction	60
5.2	Related Work	62
5.3	The GLOW Approach	64
5.3.1	GLOW Pipeline Overview	64
5.3.2	GLOW Pipeline Variants	66
5.4	An Open-World Benchmark for KGQA	67
5.5	Experiments	68
5.5.1	Experimental Setup	68
5.5.2	Experimental Results	69
5.5.3	Effect of Domain, Graph, and Question	72
5.5.4	GNN Models and Answer Selection	73
5.5.5	System Efficiency Analysis	73
5.6	Discussion	75
5.7	Limitations	75
6	Conclusion	76
	Bibliography	78
	Appendix A Chapter Appendices	98
A.1	KGTOA	98
A.1.1	The KG-Ben Benchmark: datasets and GNN tasks	98
A.1.2	Node Classification Datasets:	100
A.1.3	Link Prediction Benchmark datasets:	101
A.2	Node classification Results	102

A.3	Confusion Matrix and Classes Distribution	104
A.3.1	The MAG-PV labels distribution and normalized confusion matrix	104
A.3.2	The DBLP-PV labels distribution and normalized confusion matrix	105
A.4	Training Convergence Rate Results	106
A.5	Illustrative example subgraphs sampled using KGTOSA, URW, BRW, and IBS.	106
A.6	GLOWQA	108
A.6.1	GLOW Algorithm	108
A.6.2	GLOW Prompt Examples	108
A.6.3	LLM as-a-Judge Prompt	113
A.6.4	Full Details of Our GLOW-Bench	114
A.6.5	The GNN Training/Inference Technical Details.	114
A.6.6	The OWA-QA Detailed Results:	116

List of Figures

Figure 2.1	An example SPARQL query.	8
Figure 3.1	A KG with nodes/edges in red, which could be predicted by classification and link prediction models on the fly.	13
Figure 3.2	$\text{SPARQL}_{pv}^{\text{ML}}$: a $\text{SPARQL}^{\text{ML}}$ query uses a node classification model to predict a paper’s venue by querying and inferencing over the KG shown in Figure 3.1.	14
Figure 3.3	The KGNET architecture, which provides an interface language ($\text{SPARQL}^{\text{ML}}$) and enables AI applications and data scientists to automatically train GML models on top of KGs for querying and inferencing KGs based on the trained models.	15
Figure 3.4	A traditional GML pipeline [1] using a GML framework. The pipeline extracts the KG data, followed by data transformation into sparse matrices to train models for a GML task. Finally, the inference step is ready to predict results in isolation from the graph databases.	16
Figure 3.5	A taxonomy of methods for training GML models.	18
Figure 3.6	The automation of training pipeline and inference in our GML-as-a-service (GMLaaS). GMLaaS interacts with the KGMeta Manager to train a model for a specific task with limited budget. The automated pipeline opt to the near-optimal GML method for training a model within a limited budget. GMLaaS supports task inference through RestAPI that is called by a UDF.	19

Figure 3.7	A KGMeta graph of two trained models for node classification and link prediction tasks. The white nodes are nodes from the original data KG. The dashed nodes/edges are metadata collected per trained model.	21
Figure 3.8	A SPARQL ^{ML} insert query that trains a paper-venue classifier on DBLP. The TrainGML function is a UDF that is implemented inside the RDF engine. . . .	22
Figure 3.9	A SPARQL ^{ML} delete query that deletes a trained model and its meta-data. .	22
Figure 3.10	A SPARQL ^{ML} query predicting author affiliation link (edge) on DBLP KG.	23
Figure 3.11	A candidate SPARQL for SPARQL ^{ML} _{pv}	24
Figure 3.12	A candidate SPARQL for SPARQL ^{ML} _{pv}	25
Figure 3.13	(a) Accuracy, (B) Training Time, and (C) Training Memory for DBLP KG Paper-Venue node classification task. The KGNET task-oriented sampled subgraph (KG') significantly improves accuracy, training time, and memory.	27
Figure 3.14	(a) Accuracy, (B) Training Time, and (C) Training Memory for YAGO-4 KG Place-Country node classification task. The KGNET task-oriented sampled subgraph (KG') significantly improves accuracy, training time, and memory. . . .	28
Figure 3.15	(a) Accuracy, (B) Training Time, and (C) Training Memory for the DBLP Author Affiliation link prediction task. The KGNET task-oriented edge sampled subgraph (KG') significantly improves the Hits@10 MRR score, training time, and training memory.	29
Figure 4.1	(A) Accuracy (higher is better), (B) Training-Time (lower is better), (C) Training-Memory (lower is better). Training a node classification task (PV) to predict the paper venue using ShaDowSAINT [2] and SeHGNN [3] on a MAG graph with 42M vertices (MAG-42M). The handcrafted task-oriented subgraph (OGBN-MAG) from MAG-42M trades accuracy to reduce time and memory usage. Our KGTOSA _{d1h1} task-oriented subgraph is extracted automatically from MAG-42M for PV to reduce time and memory consumption while improving accuracy.	33

Figure 4.2	Examples of subgraphs generated by the uniform random walk (URW) sampler in GraphSAINT for different GNN tasks. The black vertices indicate the target vertices. The same colour means the same vertex or edge type. This sampling method does not guarantee to (i) include enough representation of target vertices ($\mathcal{V}_T$) and (ii) exclude vertices disconnected from $\mathcal{V}_T$ as these vertices do not contribute to the embeddings of $\mathcal{V}_T$	35
Figure 4.3	The TOSG’s generic graph pattern is based on two parameters: (i) the direction (outgoing and incoming) predicates, and (i) the number of hops.	40
Figure 4.4	The KGTOSA generic workflow. The TOSG (KG') of Task $\mathcal{A}$ is extracted and transformed into an adjacency matrix. Then, the HGNN training performs either mini-batch or full-batch training. The size of KG' is proportional to $\mathcal{V}_T$, the average degree of $\mathcal{V}_T$, and the average distance to $\mathcal{V}_T$. The smaller KG' , the faster the training pipeline.	41
Figure 4.5	The subgraphs (a, b, and c) are generated by our biased random walk sampler. The black vertices indicate the target vertices. Node and edge types are colour-coded. Our approach leads to a higher ratio of target vertices w.r.t. RW in Figure 4.2 while the non-target vertices of different node/edge types are reachable to at least one vertex in $\mathcal{V}_T$	42
Figure 4.6	Performance across NC tasks is based on three metrics: (A) Accuracy (higher is better), (B) Training-Time (lower is better), and (C) Memory (lower is better). The top and middle sections illustrate the results for the Paper-Venue task on MAG and DBLP, respectively. The bottom figures present the outcomes for the place-country task. With KGTOSA, all methods experience improvements in accuracy in most cases while reducing memory and time, even with KGTOSA’s preprocessing time in yellow.	50
Figure 4.7	Performance of RGCN, MorsE, and LHGNN in the LP tasks. (A) Hits@10 (higher is better), (B) Training-Time, (C) Training-Memory. RGCN exceeded the 3TB RAM in VM^{TB}	53

Figure 4.8 (A) Accuracy, (B) Extraction, Transformation, and Training Time, (C) Memory. GraphSAINT+BRW on FG v.s. Our SPARQL-based method with different parameters (KGTOSA_{dihj}). PV/MAG-42M at the top, PV/DBLP-15M in the middle, and PC/YAGO-30M at the bottom. KGTOSA_{dihj} achieves comparable accuracy to BRW and IB while enabling HGNN methods to reduce training time and memory usage.	56
Figure 4.9 Convergence rate analysis. GraphSAINT while training the six NC tasks using the full graph (FG) and KG' extracted by KGTOSA. KGTOSA enables the GNN method to generalize faster with comparable accuracy.	57
Figure 5.1 Overview of GLOW’s four stages: question understanding, retrieval, augmentation, and generation. GLOW builds GNN-guided prompts using KG context and/or GNN predictions, with 3 variants: (a) KG facts (GLOW-G), (b) GNN predictions (GLOW-N), and (c) both (GLOW-GN).	65
Figure 5.2 Average Exact and Hierarchical Match Accuracy (%) across datasets using different methods and LLMs. Dark bars show Exact Match; lighter segments show added Hierarchical Match gains. GCR and AskGNN are inapplicable to commercial LLMs.	70
Figure 5.3 Effect of GLOW-Bench characteristics on the answer accuracy (%) of GLOW (with Qwen3-8B). The effects are grounded by A) Knowledge Domain (KD), B) Knowledge Graph (KG) , and C) Multiple Choice Count (MCC).	71
Figure 5.4 Study of error propagation: overall system accuracy in the presence of weaker GNN models where GNN prediction’s accuracy $< 50\%$. GLOW boosts accuracy by up to 12% over weak GNNs using textual cues. AskGNN closely follows GNN performance.	72
Figure 5.5 Impact of GNN top-K answer count on GLOW-GN’s average accuracy (%) across different LLMs. GLOW-GN achieves the best score at top-K=3, while higher K misleads the LLMs.	73

Figure A.1	Performance of RGCN, Graph-SAINT, ShaDow-SAINT, and SeHGNN in the NC tasks. (A) Accuracy (higher is better), (B) Training-Time (lower is better), (C) Training-Memory (lower is better). The figures on top show the results for the paper-venue classification task on DBLP. The figures at the bottom show the results for the Author-Affalition_Country classification task. KGTOSA enables all methods to reduce memory and time while improving accuracy or keeping comparable scores, even with KG-TOSA's preprocessing time in yellow.	102
Figure A.2	Performance of RGCN, Graph-SAINT, ShaDow-SAINT, and SeHGNN in the NC tasks. (A) Accuracy (higher is better), (B) Training-Time (lower is better), (C) Training-Memory (lower is better). The figures on top show the results for the place-country classification task on YAGO. The figures at the bottom show the results for the CreativeWork-Genere classification task. KGTOSA enables all methods to reduce memory and time while improving accuracy , even with KG-TOSA's preprocessing time in yellow.	103
Figure A.3	Performance of RGCN, Graph-SAINT, ShaDow-SAINT, and SeHGNN in the NC tasks. (A) Accuracy (higher is better), (B) Training-Time (lower is better), (C) Training-Memory (lower is better). The figures on top show the results for the paper-venue classification task on MAG. The figures at the bottom show the results for the Paper-Discipline classification task on MAG. KGTOSA enables all methods to reduce memory, time and accuracy, even with KG-TOSA's preprocessing time in yellow	103
Figure A.4	The top 50 frequent labels distribution of the MAG-PV dataset.	104
Figure A.5	The normalized confusion matrix of KGTOSA _{d1h1} on top-50 MAG-PV labels using Graph-SAINT. The model accuracy is 81% and the F1(Macro) score is 0.63.	104
Figure A.6	The label distribution of the DBLP-PV dataset.	105
Figure A.7	The normalized confusion matrix of KGTOSA _{d1h1} on DBLP-PV using Graph-SAINT. The model accuracy is 90% and the F1(Macro) score is 0.66. . . .	105

Figure A.8 Convergence rate analysis. RGCN while training the six NC tasks using the full graph (FG) and KG' extracted by KGTOSA. KGTOSA enables the GNN method to generalize faster with comparable accuracy.	106
Figure A.9 Generated subgraphs using URW and our TOSG sampling approaches are presented. Rows 1 represent URW Each row includes subfigures sampled from YAGO-30M-CW, MAG-42-PV, and DBLP-15M-PV datasets, where the percentage represents the task accuracy score. Black vertices indicate target vertices $v_t \in \mathcal{V}_T$. Node and edge types are color-coded. The sampling parameters are $bs=20$ and $h=2$. URW sampling doesn't ensure sufficient representation of target vertices ($\mathcal{V}_T$) and may include target disconnected vertices.	106
Figure A.10 Generated subgraphs using our TOSG sampling approaches are presented. Rows 1 to 3 represent BRW, IBS, and KGTOSA subgraphs, respectively. Each row includes subfigures sampled from YAGO-30M-CW, MAG-42-PV, and DBLP-15M-PV datasets where the percentage represents the task accuracy score. Black vertices indicate target vertices $v_t \in \mathcal{V}_T$. Node and edge types are color-coded. The sampling parameters are $bs=20$ and $h=2$. The baselines BRW and IBS, and KGTOSA, achieve a higher ratio of target vertices connected to non-target vertices, leading to increased diversity in node/edge types. URW sampling, however, doesn't ensure sufficient representation of target vertices ($\mathcal{V}_T$) and may include target disconnected vertices.	107

List of Tables

Table 3.1	Statistics of the used KGs and GNN tasks. We used four times larger KGs (DBLP and Yago) than the ones reported in OGB [1].	26
Table 4.1	Our Benchmark Statistics. The number of nodes and edges (RDF triples) is in millions. The number of node types (n-type) and edge types (e-type) is tens to thousands.	48
Table 4.2	A summary of our GNN tasks: Task Types (TT) are single-label node classification (NC) or missing entity link prediction (LP).	49
Table 4.3	Statistics of extracted subgraphs using URW, BRW, IBS, and KGTOSA _{d₁h₁} on Yago-4, DBLP, and MAG KGs. The walk length h (no. of hops) is = 3 and the size of the initial set of target nodes is 20k. Our SPARQL-based method using _{d₁h₁} achieves similar statistics and comparable accuracy to BRW and IBS. However, BRW and IBS consume excessive computing resources w.r.t the SPARQL-based method as shown in Figure 4.8.	54
Table 4.4	A breakdown of the cost involved in training a GNN task: (i) KGTOSA _{d₁h₁} to extract KG' from FG (optional), (ii) transforming the RDF triples into adjacency matrices (mandatory), and (iii) training the task using GraphSaint.	55
Table 5.1	Average accuracy (%) on OW-QA benchmarks by reasoning depth. GLOW-GN outperforms all baselines on both 1-hop and 2-hop questions. GCR drops sharply under OWA. All use Qwen3-8B as the LLM. Scores for existing datasets are averaged over arxiv2023, ogbn-arxiv, and ogbn-products.	61

Table 5.2	Summary of Open-World Question Template (OW-QT) across four KG-based features: (1) Knowledge Domain (KD): Domain-Specific (DS), Entertainment (E), or Generic (G); (2) Target Entity Type (ET); (3) Reasoning Hops (RH); and (4) Multiple Choice Count (MCC).	67
Table 5.3	Exact Match Accuracy (%) of GLOW vs. baselines across four OWA-QA datasets. GLOW-GN consistently outperforms AskGNN, GraphSAINT, GCR, and LLM-only setups across all LLMs and datasets, showing strong generalization and effective use of both textual and structural signals, even with smaller LLMs. . . .	69
Table 5.4	The average training time in Hours, Answer tokens count per question in (K-Tokens) and answer time per question in Seconds using Qwen3-8B LLM. . . .	74
Table A.1	Our Benchmark Statistics. The number of nodes and edges (RDF triples) is in millions. The number of node types (n-type) and edge types (e-type) is tens to thousands.	99
Table A.2	A summary of our GNN tasks: Task Types (TT) are single-label node classification (NC) or missing entity link prediction (LP).	99
Table A.3	Our GLOW-Bench benchmark characterizes each Open-World Question Template (OW-QT) across four KG-based features: (1) Knowledge Domain (KD): Domain-Specific (DS), Entertainment (E), or Generic (G); (2) Target Entity Type (ET); (3) Reasoning Hops (RH); and (4) Multiple Choice Count (MCC).	114
Table A.4	The detailed accuracy (%) of GLOW compared to baseline models on existing question answering datasets from [4] and our developed benchmark GLOW-Bench. All results below use Qwen3-8B as the underlying LLM. Best results are marked in bold	116
Table A.5	The detailed accuracy (%) of GLOW compared to baseline models on existing question answering datasets from [4] and our developed benchmark GLOW-Bench. All results below use GPT-4o-Mini [5] as the underlying LLM. Best results are marked in bold	117

Table A.6	The detailed accuracy (%) of GLOW compared to baseline models on existing question answering datasets from [4] and our developed benchmark GLOW-Bench. All results below use DeepSeek-V3 [6] as the underlying LLM. Best results are marked in bold .	118
Table A.7	The detailed accuracy (%) of GLOW compared to baseline models on existing question answering datasets from [4] and our developed benchmark GLOW-Bench. All results below use DeepSeek-R1-Distill-Qwen:7B [7] as the underlying LLM. Best results are marked in bold .	119
Table A.8	The detailed accuracy (%) of GLOW compared to baseline models on existing question answering datasets from [4] and our developed benchmark GLOW-Bench. All results below use IBM Granite-3.3-8B-instruct [8] as the underlying LLM. Best results are marked in bold .	120

Chapter 1

Introduction

1.1 Motivation

Graphs and Their Applications: A graph is an abstract mathematical construct comprising a set of nodes interconnected by edges. It provides a concise yet powerful framework for modeling elements and the relationships among them in real-world systems. Numerous real-world entities and their interactions can be naturally represented as graphs. This representation can fit a homogeneous graph, which contains a single type of node and edge, or heterogeneous graphs, which comprise multiple types of nodes and edges within a single graph that represents much broader world scenarios. Various prediction tasks can be formulated at different levels of a graph, including individual nodes, edges (i.e., pairs of nodes), or the entire graph. For example, categorizing papers within a citation network can be posed as a node-level classification problem [1], whereas recommending new friendships in a social network constitutes an edge-level prediction task [1]. At the graph level, a prominent application involves predicting molecular properties, such as toxicity and solubility, which play a critical role in drug discovery [9].

The Evolution of Graph Machine Learning: A range of machine learning methods has been developed to enable accurate predictions on graphs. Before the emergence of deep learning, the dominant approach relied on manually engineering graph features, followed by training relatively simple models—such as linear models, kernel machines, or decision trees—on these features to perform predictions [10, 11]. However, the biggest limitation of these methods is that node/graph

features need to be carefully hand-engineered for each task of interest, which requires tremendous efforts by human experts. To eliminate the need for manual feature engineering, early efforts focused on learning node or graph embeddings [12–14], drawing inspiration from word embedding techniques [15]. However, these approaches learn embeddings in an unsupervised manner, and thus the resulting representations are not directly optimized for the specific task of interest. Furthermore, the embeddings are derived solely from the graph structure and do not incorporate input node or edge features commonly available in real-world graphs. As a result, these methods typically provide only limited improvements in predictive performance.

Recently, Graph Neural Networks (GNNs) have emerged as a powerful class of models for graph-structured data. By recursively aggregating and transforming the representations of neighboring nodes, GNNs naturally handle irregular graph structures and are insensitive to node ordering [16, 17]. This enables strong predictive performance across graph-based tasks without the need for manual feature engineering [18, 19]. GNN excels at learning from complex graph-structured data to predict missing nodes or edges. This includes node classification tasks, such as recommender systems [20] and entity alignment [21], as well as link prediction tasks, such as drug discovery [22] and fraud detection [23]. Despite their success, GNN design has largely relied on empirical intuition and heuristics, with limited theoretical understanding of the properties and limitations of different GNN variants and KG structures. Additionally, most existing GNN models have been developed and evaluated on small, unrealistic homogeneous graph benchmarks [1, 9] with a limited support for heterogeneous graphs.

Heterogeneous Graph: **Heterogeneous Graph (HG)** is a graph composed of different types of nodes and/or edges, also known as heterogeneous information networks [24, 25]. Different node/edge types have different traits, and their features may follow different data formats. HGs have been recognized as a powerful model that is able to handle rich structural information in real-world networks, ranging from social networks, academic networks, biomedical networks, and product co-purchasing networks to knowledge graphs [25]. **A knowledge graph (KG)** is an HG that accumulates and stores knowledge of real-world entities and abstracts their complex relationships in multi-domains [26]. KG includes nodes representing different entities of $|C|$ classes and edges representing $|R|$ relations between these entities. C and R are the sets of node and

edge types in KG , respectively. The fundamental unit of a knowledge graph is the triple (subject, predicate, object) that represents a real fact ,i.e., (Bill Gates, Founded, Microsoft). The relation could be asymmetric, hence, the direction of a link matters. Real-world KGs, such as YAGO [27], Wikidata [28], and others, have hundreds to thousands of node/edge types. KGs have become crucial components of numerous daily applications, i.e., recommender systems [29], biomedical research [22], fraud detection [30], question answering [31], and others [32, 33].

GNN Limitations on KGs: [Heterogeneous Graph Neural Network \(HGNN\)](#) have been used recently to support ML on HGs such as KGs by considering the two views of graph heterogeneity. The first view considers the node types to construct task-based meta-paths and convert them into homogeneous graphs. The second view considers dealing with each relation type as a separate layer with a dedicated learnable weight matrix. The main limitation emerges when scaling to real-world knowledge graphs (KGs) that consist of hundreds to thousands of node and edge types, hundreds of millions of nodes, and billions of edges, such as in YAGO4 [27], Wikidata [28], DBLP [34], and MAG [35]. When incorporating far-away neighbour nodes, the features aggregation during the training phase introduces a scalability challenge to HGNN methods [36]. The complexity further increases in heterogeneous graphs, as the number of GNN layers scales with the number of edge types, and the aggregation process requires handling adjacency matrices for each node–edge type combination, making computation costly [3, 37, 38].

Question Answering Limitations over Incomplete KGs: Knowledge graphs (KGs) are dynamic in nature, continuously evolving as new nodes and edges are introduced over time. existing KG question answering (KGQA) pipelines assume the question answer exists inside the KG and employ deep search techniques to retrieve it [39]. At any given time, the KG remains inherently incomplete, causing existing KGQA methods to struggle when answers involve missing nodes or edges.

These challenges hinder the development of powerful HGNN models for real-world prediction tasks over existing KG engines. This thesis aims to address these limitations by building an AI-enabled KG engine that automates the HGNN training and inference pipelines, optimizes HGNN training, and enables KGQA over incomplete knowledge graphs (IKGs).

1.2 Thesis Contributions & Outline

The focus of this dissertation is to develop a comprehensive analysis of the challenges faced by developing an AI-enabled knowledge graph platform using KG engines and existing shallow GNNs, and propose effective KG sampling techniques based on empirically and theoretically motivated methods. The organization and contributions of this dissertation are as follows.

In Chapter 2, we provide the background knowledge required to understand the rest of the thesis, including some important definitions, a brief introduction to knowledge graphs, and graph neural networks and their graph machine learning applications.

In Chapter 3, we introduce the KGNET, a novel system design, and its system components integration challenges. These challenges exhibit query writing, storage management, training/inference automation, and metadata storage. On several node classification and link prediction tasks, the proposed architectures achieve state-of-the-art (SOTA) performance.

In Chapter 4, we study the heterogeneous graph sampling and its scalability challenges using shallow GNNs. Though capturing neighborhood information and being effective for various tasks, we show that the aggregation step is potentially a problematic factor underlying all GNNs, as it forces the node representations to be similar, making the nodes gradually lose their identity and become indistinguishable. To counter this, we introduce a task-oriented sampling operation that allows node distinctiveness and preserves identity. Such sampling replaces the random walk sampling with task-oriented sampling, which theoretically enriches the node representations. In practice, the proposed sampling can be readily applied to both deep and shallow baseline GNNs to train task-relevant models. In the experiments, we observe

In Chapter 5, we adopt KGNET for question answering over incomplete KG (IKGQA). We interface KGNET with an API that incorporates KG, GNN, and LLM to answer multi-hop open-world questions accurately via a carefully designed Graph-RAG pipeline. We introduced the first multi-hop OW question answering dataset on top of existing KGs of various sizes and domains. We study the limitations of existing baselines and datasets. Then introduce a novel solution with empirical/theoretical justifications.

Finally, the thesis conclusion and further research directions are presented in Chapter 6

1.3 Statement Regarding Authorship and Contributions

Significant portions of the chapters of this dissertation (excluding the introduction, preliminary, related work, conclusion, and future directions) are composed of material from previously published work for which Hussein Abdallah is a primary author. According to the standard publication agreements of these venues, Hussein Abdallah has the right to reproduce these works in this dissertation. Walid Afandi, as a second author, helped with developing the KGNET inference pipeline [40] and technical writing in KGTOSA [41]; Yasman helped with experimental evaluation and writing in KGTOSA [41]. My supervisor, Essam Mansour, helped with paper research, detailed discussion, mentoring, proofreading, and checked all the algorithms and definitions. My co-supervisor, Panos Kalins, helped with writing and proofreading for KGTOSA [41] and GLOW [42].

1.4 Publications List

Conference

- **Hussein Abdallah, Essam Mansour.** [Towards a GML-Enabled Knowledge Graph Platform.](#) 39th International Conference on Data Engineering (ICDE) (**ICDE 2023**).
- **Hussein Abdallah, Waleed Afandi, Panos Kalnis, Essam Mansour.** [Task-Oriented Graph Neural Network Training on Large Knowledge Graphs for Accurate and Efficient Modeling.](#) 40th International Conference on Data Engineering (ICDE) (**ICDE 2024**).

Demos

- **Hussein Abdallah, Waleed Afandi, Essam Mansour.** [Demonstration of SPARQL^{ML}: An Interfacing Language for Supporting Graph Machine Learning in the KGNet Platform.](#) 49th International Conference on Very Large Data Bases (**VLDB 2023**).
- **Hussein Abdallah, The K.Nguyen, D.Nguyen, Essam Mansour.** [Demonstration of KGNet: a Cognitive KG Platform.](#) International Semantic Web Conference (**ISWC 2021**).

Under Submission

- **Hussein Abdallah**, Ibrahim AbdelAziz, Panos Kalnis, **Essam Mansour**. Leveraging LLM-GNN Integration for Open-World Question Answering over Knowledge Graphs [42].
- Walid Affendi, **Hussein Abdallah**, Ashraf Aboulnaga, **Essam Mansour**. An LLM-Guided Query-Aware Inference System for GNN Models on Large Knowledge Graphs [43].

1.5 Notation

In this section, we introduce some notation and terms to be used in this thesis. We use $G := (V, R)$ to denote a graph with the vertex (node) set V and the relation set R . Unless specifically mentioned, all graphs used are heterogeneous and directed graphs extracted from real-world knowledge graphs KG s.

We denote sets by upper cases letters e.g. V for nodes, R for relations (edges), A for adjacency matrix and their instances by normal type letters (e.g. v for a single node, and r_{ij} , and a_{ij} for a relation instance between nodes i and j). We denote target sets of nodes or edges by the T symbol. e.g., V_T and E_T represent the target node and edge sets, respectively, while v_t and r_t represent the target node and edge.

We use $\mathbb{R}$ and $\mathbb{Z}$ to denote the sets of real scalars and integer scalars, respectively, $\mathbb{R}^n$ and $\mathbb{Z}^n$ to denote the set of n -dimensional real vectors and integer vectors respectively, and $\mathbb{R}^{m \times n}$ and $\mathbb{Z}^{m \times n}$ to denote $m \times n$ real matrices and integer matrices, respectively

For an embedding vector $x \in \mathbb{R}^n$, $x_{i:j}$ denotes the subvector composed of elements of x with indices from i to j , x_i or $x(i)$ denotes the i^{th} element of x .

For a matrix $A \in \mathbb{R}^{m \times n}$, $A_{i,:}$ denotes the i^{th} row of A , and $A_{:,j}$ denotes the j^{th} column of A . The (i,j) element of A is denoted by $A_{i,j}$ or $A(i,j)$.

For a set S , $|S|$ denotes the number of elements S has.

Chapter 2

Preliminary

2.1 Knowledge Graph (KG)

A knowledge base is a dataset that represents real-world facts and semantic relations in the form of triplets. When these triplets are structured as a graph, with nodes denoting entities and edges denoting relations, the result is a knowledge graph (KG). In most contexts, the terms knowledge base and knowledge graph are used interchangeably. [32,44]

The schema of a knowledge graph is typically defined as an ontology, which specifies the properties of a particular domain and the relations among them. Consequently, ontology construction is a fundamental stage in building a knowledge graph and understanding its relational data structure. [44]. The concept of the modern knowledge graph gained prominence in 2012, when Google introduced the Google Knowledge Graph as part of its knowledge base [45]. Since then, numerous large-scale knowledge graphs have been developed and widely adopted, including: Wikidata [28]: a cross-lingual, document-oriented knowledge graph supporting Wikipedia and other platforms. YAGO [27]: a large-scale knowledge base that combines entities and relations extracted from resources such as Wikipedia. DBpedia [46]: extracts semantically meaningful information from Wikipedia and converts it into a well-structured, ontological knowledge base. Freebase [47]: integrates information from multiple sources to provide a structured, global resource.

A knowledge graph (KG) is a directed graph consisting of nodes and edges, where nodes represent entities (either real-world objects or abstract concepts), and edges denote semantic relations

between entities [26, 48]. The fundamental unit of a KG is the triple—represented as (subject, predicate, object) or equivalently (head, relation, tail)—for example, (Bill Gates, founderOf, Microsoft). Since relations are not necessarily symmetric, edge directionality is essential: head entities point to tail entities through the corresponding relation. Two common data models for representing and managing KGs are the Resource Description Framework (RDF) [49] and Labeled Property Graphs (LPGs) [32].

The KG includes nodes representing different entities ($\mathcal{V}$) of $|\mathcal{C}|$ classes, e.g., *Paper* and *Author*, and edge types representing $|\mathcal{R}|$ relations, e.g., *publishedIn* and *write*, between these entities. $\mathcal{C}$ and $\mathcal{R}$ are the sets of node and edge types in the KG, respectively. Real-world KGs, such as Yago [27] and Wikidata [28], have hundreds to thousands of node/edge types.

2.2 Knowledge Graph Querying

Knowledge graph engines store and index a large amount of facts to be retrieved later for information retrieval, enabling structured reasoning over complex domains. Querying these graphs typically relies on SPARQL [50], a standardized query language designed for RDF-based data [49]. SPARQL allows users to specify graph patterns, filter conditions, and logical constraints to retrieve precise information from large, heterogeneous datasets. Its expressive power makes it suitable for tasks such as entity lookup, relationship exploration, and multi-hop reasoning, while maintaining a declarative style similar to SQL [51]. For example, to retrieve the authors of books published after 2015 from an academic network KG, an equivalent SPARQL Query could be written as:

```

1 | PREFIX pref: <http://example.org/schema/>
2 | from <Named-KG>
3 | SELECT ?author ?book
4 | WHERE {
5 |   ?book a pref:Book .
6 |   ?book pref:publishYear ?year .
7 |   ?book pref:author ?author .
8 |   FILTER (?year > 2015).
9 | }
```

Figure 2.1: An example SPARQL query.

2.3 Knowledge Graph Embedding (KGE)

Knowledge graph embedding (KGE) seeks to map entities and relations in a knowledge graph into a low-dimensional vector space, while preserving its semantic and structural properties [52]. The resulting vector representations can then be effectively leveraged by machine learning models for downstream tasks. Broadly, triplet-based KGE methods can be categorized into three main families: (a) tensor factorization-based, (b) translation-based, and (c) neural network-based approaches. Overall, tensor factorization-based methods offer strong theoretical foundations and can capture complex interactions but suffer from scalability issues on large graphs. Translation-based models, such as TransE [53], are highly efficient and intuitive, yet they struggle to represent complex relational patterns. Neural network-based approaches provide the greatest expressive power and flexibility, enabling the integration of diverse signals, though at the cost of higher computational demands and reduced interpretability. Thus, the choice of method typically involves a trade-off between scalability, expressiveness, and efficiency. In this thesis, we are not incorporating KGEs for KGs embeddings as they do not support inductive settings, i.e., generating embeddings for evolving graphs nodes or edges [54, 55].

2.4 Graph Convolutional Neural Networks (GNN)

Let $G = (V, E)$ denote a homogeneous graph, where V is the set of nodes and $E \subseteq V \times V$ is the set of edges. Each node $v \in V$ is associated with an input feature vector $\mathbf{x}_v \in \mathbb{R}^d$, where d is the feature dimension. For example, in a citation network, paper nodes may have textual features. Given a graph G , GNNs perform multiple layers of neighborhood aggregation to compute embeddings (i.e., representation vectors) of nodes. At each layer, the embedding of a node is updated by aggregating and transforming the embeddings of its neighbors from the previous layer.

Let $\mathbf{h}_v^{(\ell)}$ denote the embedding of node v at the ℓ -th layer of a GNN. Formally, the node embeddings are initialized with the input features at layer $\ell = 0$: $\mathbf{h}_v^{(0)} \leftarrow \mathbf{x}_v$. For subsequent layers $\ell = 1, \dots, L$, the embedding of node v is updated as:

$$\mathbf{a}_v^{(\ell)} = \text{AGGREGATE}^{(\ell)}\left(\{\mathbf{h}_u^{(\ell-1)} : u \in \mathcal{N}(v)\}\right) \quad (1)$$

$$\mathbf{h}_v^{(\ell)} = \text{COMBINE}^{(\ell)}(\mathbf{h}_v^{(\ell-1)}, \mathbf{a}_v^{(\ell)}). \quad (2)$$

where $\mathcal{N}(v)$ denotes the set of neighbors of node v , and $\text{AGGREGATE}^{(\ell)}(\cdot)$ and $\text{COMBINE}^{(\ell)}(\cdot)$ are functions modeled using neural networks. The $\text{COMBINE}^{(\ell)}(\cdot)$ function is conceptually simple: it is a neural network that combines and transforms the previous embedding $\mathbf{h}_v^{(\ell-1)}$ and the aggregated message $\mathbf{a}_v^{(\ell)}$ into a new embedding [18].

On the other hand, $\text{AGGREGATE}^{(\ell)}(\cdot)$ is more non-trivial, as it operates on a multiset of neighboring node embeddings and produces a fixed-dimensional output. To achieve this, a GNN employs a permutation-invariant neural network. In other words, the output of $\text{AGGREGATE}^{(\ell)}(\cdot)$ does not depend on the ordering of its inputs, a desirable property for predictive models on graphs. The model’s predictions should be invariant to node ordering.

2.5 Heterogeneous Graph Convolutional Neural Networks (HGNN)

HGNN methods build upon the GNN by incorporating multi-layer GCNs to support the heterogeneity of KG node and edge types during the message passing and aggregation steps. In HGNNs, the number of GCN layers is associated with the number of metapaths in the heterogeneous graph. A metapath in KG captures distinct semantic relationships (edge types) between node types. For example, the metapath Author-(write)-Paper-(publishedIn)-Venue (APV) describes the semantic relationships of an author who published a paper in a particular venue. Metapaths can range from a single edge type and one-hop to paths with multiple hops and edge types [3, 56]. For a $KG = (\mathcal{V}, \mathcal{C}, \mathcal{L}, \mathcal{R}, \mathcal{T})$, a metapath is defined as a sequence of relations across node types in $\mathcal{C}$ in the form of $c_1 \xrightarrow{r_1} c_2 \xrightarrow{r_2} \dots \xrightarrow{r_h} c_{h+1}$. This leads to composite relations (metapaths), which construct all possible combinations of $\mathcal{C}$ and $\mathcal{R}$ for hop h based on the KG schemas.

Relational Graph Convolutional Network (RGCN) [57] is based on a metapath of one hop and edge type. RGCN uses a stack of GCNs, where each GCN is dedicated to a metapath involving one specific edge type. In both categories, the initial node embeddings are iteratively aggregated with embeddings received from neighboring nodes connected to a specific relation until the embeddings of all nodes converge. For instance, in RGCN-based methods, the final embedding of a node is obtained through two aggregations: an outer aggregation over each relation type and an

inner aggregation over neighbouring nodes of a specific relation, and is defined by RGCN [57] as follows:

$$h_i^{(l+1)} = \sigma \left(\sum_{r \in \mathcal{R}} \sum_{j \in N_i^r} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)} \right) \quad (3)$$

where l is an RGCN layer, $h_j^{(l+1)}$ is the hidden embedding of node j at layer $l+1$, σ is element-wise activation function, N_i^r denotes the set of neighbour indices of node i under relation $r \in \mathcal{R}$, $c_{i,r}$ is a problem-specific normalization constant that can either be learned or chosen in advance (such as $c_{i,r} = |N_i^r|$), $W_r^{(l)}$ is the weight matrix for relation r at layer l , and $W_0^{(l)}$ is the initial matrix at layer l .

These node embeddings can be used to make predictions at different tasks on the KG's individual nodes or links (edges)

- **Node property prediction.** For each node $v \in V$, a prediction can be made by applying a prediction head F_{pred} over the node embedding $\mathbf{h}_v^{(L)}$, i.e., $\hat{y}_v = F_{\text{pred}}(\mathbf{h}_v^{(L)})$. Common choices for F_{pred} include a Multi-Layer Perceptron (MLP) or a linear model.
- **Link property prediction.** For a link between nodes $(u, v) \in V \times V$, a prediction can be made by applying a prediction head over the pair of node embeddings, i.e., $\hat{y}_{uv} = F_{\text{pred}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})$. Common choices for F_{pred} include the inner product or an MLP over the concatenated embeddings $[\mathbf{h}_u^{(L)} \parallel \mathbf{h}_v^{(L)}]$.

Lastly, the parameters of an HGNN and the prediction head F_{pred} can be combined in an end-to-end pipeline for a given task, eliminating the need for manual feature engineering on graphs.

In the following chapters, we introduce the development of our AI-enabled KG engine designed to integrate heterogeneous graph neural network (HGNN) models into existing RDF engines. The system enables:

- Automated training of lightweight HGNN models tailored to specific tasks.
- Efficient storage and indexing of trained HGNN models for on-demand inference retrieval.
- Seamless prediction of missing nodes and edges within the underlying KG through a structured SPARQL-like query interface.
- Open-world question answering over incomplete KGs.

Chapter 3

Towards a GML-Enabled Knowledge Graph Platform

3.1 Introduction

Knowledge graphs (KGs) are constructed based on semantics captured from heterogeneous datasets using various Artificial Intelligence (AI) techniques, such as representation learning and classification models [58]. Graph machine learning (GML) techniques, such as graph representation learning and graph neural networks (GNNs), are powerful tools widely used to solve real-world problems by defining them as prediction tasks on KGs. For instance, node classification tasks for problems, such as recommendations [20] and entity alignment [21], can be solved using GML techniques. Similarly, drug discovery [22] and fraud detection [23, 30] problems are tackled as link prediction tasks using GML techniques.

Data scientists often work with KGs, which are typically stored in RDF engines. They are responsible for developing GML pipelines using frameworks, such as PyG [59] and DGL [60], to train models on these KGs. However, there is often a gap between the GML frameworks and RDF engines. This necessitates an initial step of transforming the entire KG from RDF triple format into adjacency matrices in a traditional GML pipeline. Afterward, the data scientist needs to select a suitable GML method from a wide range of KG embedding (KGE) or GNN methods [61, 62] to train the model. For the average user, this responsibility is time-consuming.

Furthermore, the trained models are isolated from the RDF engine, where the KG is stored. Therefore, automating the training of GML models on KGs and providing accessibility to the trained models via a SPARQL-like query is essential. We refer to this query as a $\text{SPARQL}^{\text{ML}}$ query.

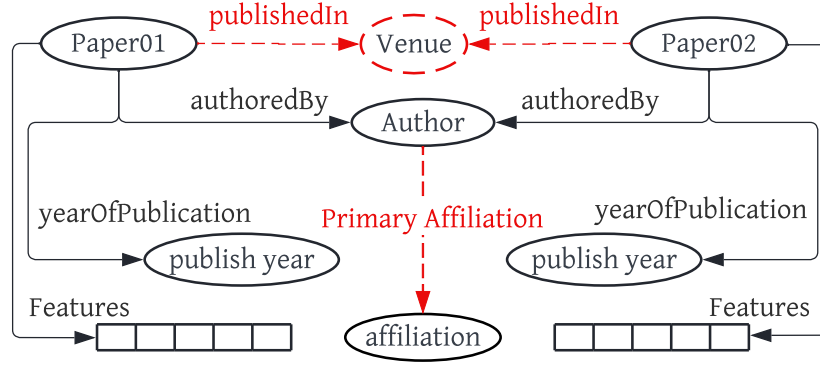


Figure 3.1: A KG with nodes/edges in red, which could be predicted by classification and link prediction models on the fly.

The KG shown in Figure 3.1 contains information about published papers in DBLP [34]. However, the traditional SPARQL query language cannot be used to apply GML models on top of a KG, such as predicting a node’s class or a missing affiliation link for an author. For instance, the venue node in Figure 3.1 is a virtual node that could be predicted using a node classification (NC) model. It would be fascinating to query this KG using a GML model for NC through a SPARQL-like query to obtain the paper-venue node, as shown in the $\text{SPARQL}^{\text{ML}}_{pv}$ in Figure 3.2.

This query uses a model of type *kgnet:NodeClassifier* to predict a venue for each paper. The $\text{SPARQL}^{\text{ML}}$ triple patterns in lines 8-10 will retrieve all models of type *kgnet:NodeClassifier* that predict a class of type *dblp:venue*. In the triple pattern $\langle ?paper, ?NodeClassifier, ?venue \rangle$, we refer to *?NodeClassifier* as a user-defined predicate. Enabling queries like $\text{SPARQL}^{\text{ML}}_{pv}$, shown in Figure 3.2, presents several challenges. These include: (i) automatically training GML models for various tasks, (ii) optimizing $\text{SPARQL}^{\text{ML}}$ for GML model selection based on accuracy and inference time, and (iii) efficiently interacting with the selected model during query execution. Additionally, seamless integration of GML models into RDF engines is necessary. As a result, users should be able to express their $\text{SPARQL}^{\text{ML}}$ queries easily by following the SPARQL logic of pattern matching, avoiding the explicit use of user-defined functions (UDFs).

There is a growing adoption of integrating GML with existing graph databases, such as

```

1 | prefix dblp: <https://www.dblp.org/>
2 | prefix kgnet: <https://www.kgnet.com/>
3 | select ?title ?venue
4 | where {
5 |   ?paper a dblp:Publication.
6 |   ?paper dblp:title ?title.
7 |   ?paper ?NodeClassifier ?venue.
8 |   ?NodeClassifier a kgnet:NodeClassifier.
9 |   ?NodeClassifier kgnet:TargetNode dblp:Publication.
10 | ?NodeClassifier kgnet:NodeLabel dblp:venue.}

```

Figure 3.2: SPARQL_{pv}^{ML}: a SPARQL^{ML} query uses a node classification model to predict a paper’s venue by querying and inferencing over the KG shown in Figure 3.1.

Neo4j [63] or Stardog [64]. However, while these databases offer some machine learning primitive methods, such as PageRank and shortest-path using the *Cypher* language, they do not address the challenges of integrating GML models with RDF engines. For example, Neo4j Graph Data Science [65] supports limited graph embedding methods in a beta version, such as FastRP [66], Node2Vec [12], and Graph-SAGE [67]. However, a user must train the models separately.

To address these challenges, there is a need to bring GML to data stored in RDF engines instead of getting data to machine learning pipelines. This would encourage the development of KG data science libraries powered by the expressiveness of SPARQL, enabling better analysis and insight discovery based on KG structure and semantics. These libraries would empower data scientists with a full breadth of KG machine learning services on top of KGs stored in RDF engines. This work introduces KGNET, an on-demand GML-as-a-service on top of RDF engines to support SPARQL^{ML} queries, as illustrated in Figure 3.3. KGNET extends existing RDF engines with two main components GML-as-a-service (GMLaaS) and SPARQL^{ML} as a Service.

KGNET automatically trains a GML model on a KG for tasks, such as node classification or like prediction, and maintains metadata of the trained model as an RDF graph called *KGMeta*. To reduce training time and memory usage while improving accuracy on a specific task $\mathcal{A}$, KGNET performs task-oriented sampling to identify a task-specific subgraph KG' of the larger KG that preserves essential characteristics relevant to $\mathcal{A}$. This enables KGNET to scale on large KGs.

GMLaaS is in charge of: (i) selecting the near-optimal GML method for training $\mathcal{A}$ using KG' based on a given time or memory budget, and (ii) communicating with RDF engines via HTTP calls

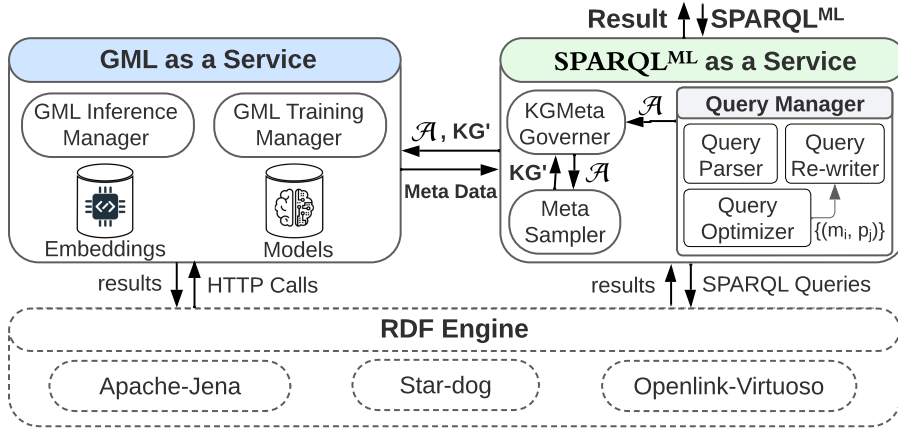


Figure 3.3: The KGNET architecture, which provides an interface language (SPARQL^{ML}) and enables AI applications and data scientists to automatically train GML models on top of KGs for querying and inferencing KGs based on the trained models.

requesting inferencing of a specific trained model, (iii) storing the trained models and embeddings related to KGs. The SPARQL^{ML} service transparently: (i) maintains and interlinks the KGMeta with associated KGs, (ii) optimizes the GML model selection for a user-defined predicate, and (iii) finally rewrites the SPARQL^{ML} query as a SPARQL query.

In summary, the contributions of this visionary system are:

- A fully-fledged GML-enabled KG platform¹ on top of existing RDF engines.
- GML-as-a-service to provide automatic training of GML models based on a given memory or time budget. This automatic training utilizes task-specific subgraphs extracted using our task-oriented sampling approach.
- SPARQL^{ML} as a Service to perform task-oriented sampling, maintain training meta-data in KGMeta, and optimize the GML model selection, i.e., opt for the near-optimal model based on constraints on accuracy and inference time.
- A comprehensive evaluation with different GML methods using three GML tasks on real KGs. Our experiments show that KGNET achieved comparable or improved accuracy compared to training on the entire KG, while significantly reducing training time and memory usage.

¹<https://github.com/CoDS-GCS/KGNET>

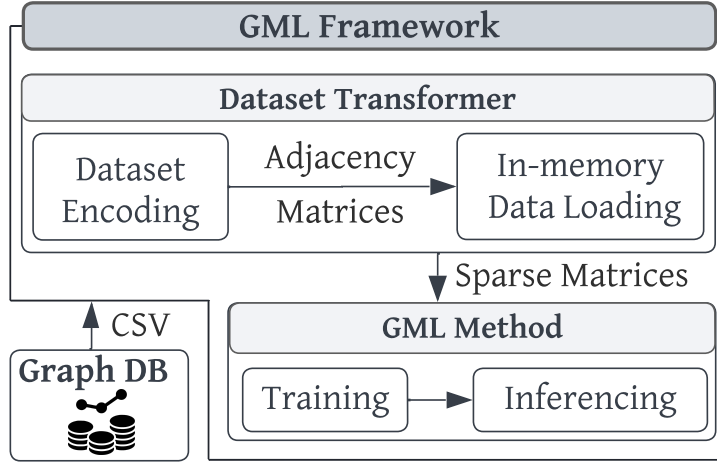


Figure 3.4: A traditional GML pipeline [1] using a GML framework. The pipeline extracts the KG data, followed by data transformation into sparse matrices to train models for a GML task. Finally, the inference step is ready to predict results in isolation from the graph databases.

3.2 Background: ML pipelines for KGs

ML pipelines developed to train models on a KG can be grouped into three main categories: (i) traditional ML on KG data in tabular format, (ii) traditional ML on KG embeddings, and (iii) graph neural networks (GNNs) trained directly on the KG. In the traditional ML approach using KG data in tabular format, data from the KG is transformed into in-memory data frames, and classical ML classifiers are trained using feature engineering techniques and libraries, such as Scikit-Learn or SparkMLlib. In contrast, traditional ML on KG embeddings avoids the feature engineering process and generates embeddings for nodes and edges. Apple Saga [68] is an example of this approach, which uses graph ML libraries like DGL-KE [60] to generate KG embeddings. Data scientists have the flexibility to choose the ML method for training.

GNNs have gained significant popularity in recent years. Hence, data scientists frequently utilize them to perform GML tasks. The Open Graph Benchmark (OGB) [1] standardized the GNN training pipeline, emphasizing the best practices for tackling GML tasks and building a GNN training pipeline. Figure 3.4 summarizes this pipeline, which involves encoding KG nodes and edges, generating adjacency matrices, loading them into memory, and training the GNN methods.

Various GML frameworks, such as DGL [60] and PyG [59], offer multiple implementations of GNN methods. These frameworks support data transformation by loading graphs into memory as

graph data structures and applying transformations. However, existing GML frameworks require significant memory and processing time for large KGs and a deep understanding of various GNN methods. In comparison, the OGB pipeline is simple, but it is a semi-automated process that necessitates human intervention and ML expertise to construct an effective pipeline and select an appropriate GNN method. Data scientists may choose the most appropriate GNN method based on various constraints, such as time or memory limitations. Furthermore, as depicted in Figure 3.4, the separation of the trained models from the KG engines adds an extra layer of complexity for data scientists to apply their models when inferring the KG.

3.3 Challenges of GML-enabled KG Engine

This section highlights the open research challenges and opportunities raised by developing GML-enabled KG Engine.

3.3.1 Automatic Training: Method Selection and Graph-sampling

There are numerous methods for training models for GML tasks, as summarized in Figure 3.5. These methods could be classified mainly into two categories KG embeddings (KGE) or graph neural network (GNN) methods. Examples of KGE methods are TransE, RotatE, ComplEx, and DistMult [62]. Some GNN methods support sampling on full graph, such as Graph-SAINT [69], Shadow-SAINT [2], and MorsE [70]. Examples of GNN full-batch training (without sampling) methods are RGCN [57] and GAT [71]. Our taxonomy has more categories, as shown in Figure 3.5.

GML methods vary significantly in terms of their accuracy, training time, and memory requirements. Furthermore, the complexity of each GML task may differ depending on various factors, such as the size of KGs and the number of node/edge types related to the task. For example, link prediction can be more resource-intensive than node classification. Different GML methods may perform differently under the same budget constraints, and selecting the best method can depend on several factors. Hence, automating a training pipeline for a specific GML task based on a user’s budget for time and memory is challenging. For instance, some GML methods perform full-batch

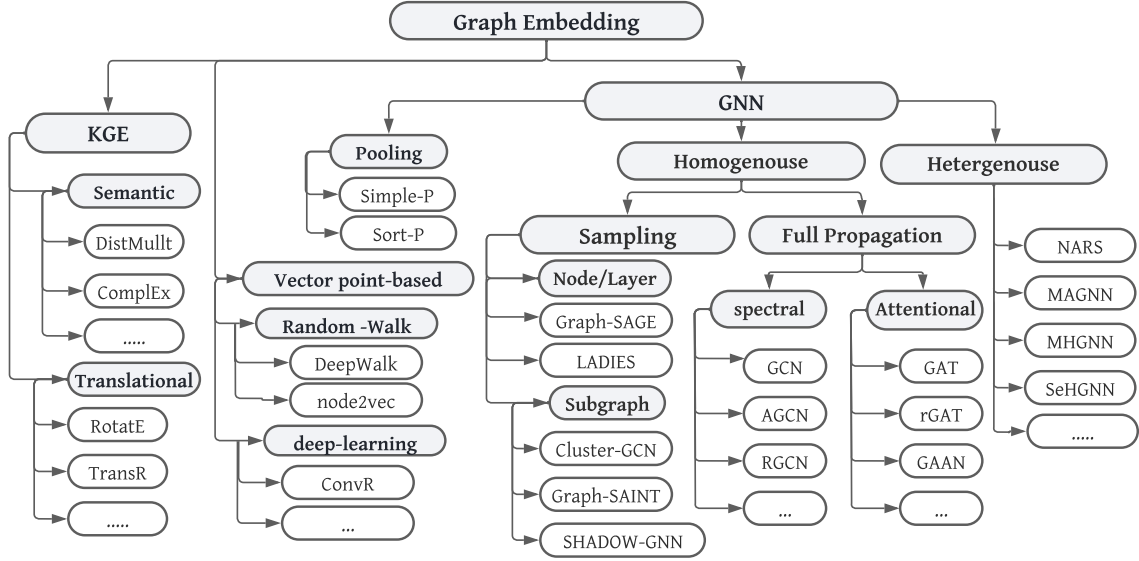


Figure 3.5: A taxonomy of methods for training GML models.

training, which requires more memory budget. These methods require huge memory to train models on large KGs. Some other GNN methods may suffer from over-smoothing, which can cause accuracy degradation. Sampling-based GNN (mini-batch training) methods use different types of sampling, which vary in avoiding these limitations. Therefore, automating the selection of GML methods for a specific task based on a given time or memory budget is challenging.

Real KGs can contain millions to billions of triples, such as DBLP [34] and MAG [35]. However, training GML models on these large KGs requires colossal computing resources that exceed the capabilities of a single machine. As a result, there is a need to identify a smaller training dataset of the KG that is specific to the task at hand. This process is known as task-oriented graph sampling. It has been proposed in various application domains, including computer vision [72, 73] and speech recognition [74], to extract a training dataset that is tailored to the given task.

In the context of GML, efficient graph sampling presents an opportunity to optimize training models on large KGs by selecting a representative sub-graph that is relevant to the task. This approach can help reduce time and memory requirements without sacrificing accuracy. Therefore, exploring the potential benefits of using task-oriented sampling in training GML models to extract task-specific subgraphs is crucial. By doing so, we can improve the efficiency and effectiveness of GML methods on large-scale KGs. This raises a research opportunity to explore different task-oriented graph sampling approaches for GML methods on large knowledge graphs (KGs).

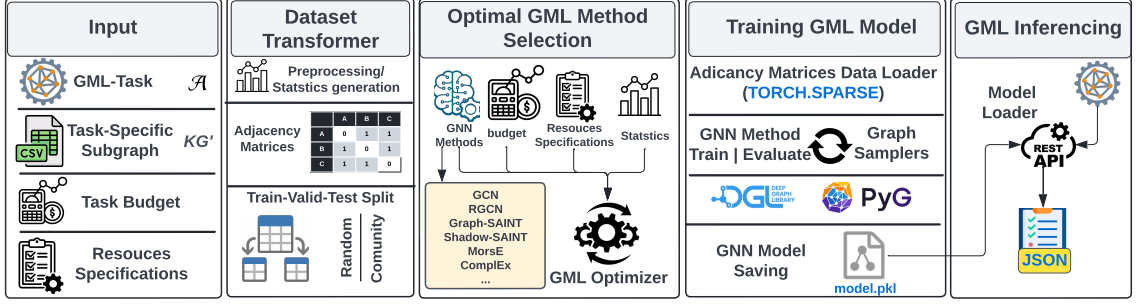


Figure 3.6: The automation of training pipeline and inference in our GML-as-a-service (GMLaaS). GMLaaS interacts with the KGMeta Manager to train a model for a specific task with limited budget. The automated pipeline opt to the near-optimal GML method for training a model within a limited budget. GMLaaS supports task inference through RestAPI that is called by a UDF.

3.3.2 Seamless Integration Between GML Models and KGs

Enabling GML on top of RDF engines poses significant challenges, mainly interfacing between the trained models and the underlying data management engine. One common approach is to use user-defined functions (UDFs) to implement this interface [75–77]. However, this comes with a cost for query optimizations in data systems [78]. The existence of an extensive catalog of UDFs can limit the expressiveness of ML-based queries. For instance, a large catalog of UDFs makes it difficult for users to choose between UDFs and find the right one for their needs. Most existing query optimizers do not have models estimating the cost of these UDFs. Hence, automating the query optimization of SPARQL^{ML} queries is challenging. There is a research opportunity for seamless integration between trained GML models and RDF. To address these challenges, we proposed KGMeta as a graph representation of metadata of trained models interlinked with the KGs.

3.3.3 Optimizing SPARQL^{ML} Queries and Benchmarks

User-defined predicates were first proposed for SQL [79]. In SPARQL^{ML}, a user-defined predicate is used to get a prediction from one of the trained models associated with a specific node in the graph. Estimating the cost of evaluating a user-defined predicate is more complex than estimating the cost of a traditional RDF predicate. While cardinality estimation is used to optimize only the execution time for the latter, a user-defined predicate in a SPARQL^{ML} query can be inferred by multiple models, each with varying accuracy and inference time. RDF engines are unaware of this information, leading to the problem of selecting the best model for inference.

For a SPARQL^{ML} query, the inference step in an RDF engine using a chosen model is a challenging task that requires optimization, specifically for rank-ordering the inference process. The challenge lies in deciding whether to perform the inference in a single call to a UDF or per instance, which may result in an extensive number of UDF calls. Additionally, each model has a unique cardinality, i.e., the total number of predictions it can make. This makes predicting rank-ordering complex as RDF engines lack accurate estimation of UDF costs.

To address these challenges, there are research opportunities for developing benchmarks to evaluate optimization approaches for SPARQL^{ML} queries. These benchmarks should consider various models for different user-defined predicates and be designed to work with large datasets. Furthermore, each SPARQL^{ML} query should vary in the number of user-defined predicates and be associated with variables of different cardinalities. This will enable a comprehensive evaluation of the performance and scalability of varying optimization approaches for SPARQL^{ML} queries.

3.4 The KGNET Platform

KGNET provides two main services, namely GML as a Service (GMLaaS) and SPARQL^{ML} as a Service on top of existing RDF engines, as shown in Figure 3.3.

3.4.1 GML as a Service (GMLaaS)

KGNET is a platform that offers end-to-end automation of GML training on KGs, as depicted in Figure 3.6. The platform provides *GMLaaS*, a Restful service that manages GML models in terms of automatic training and interactive inferencing. Additionally, it utilizes an embedding store to facilitate entity similarity search tasks by computing the similarity between embedding vectors. The *GML training manager* automates the training pipeline per task. However, the automation of GML training on KGs is challenging due to the complexity and size of KGs. Therefore, KGNET leverages our meta-sampling approach to optimize the training process by selecting a task-specific subgraph (KG') that is specific to the given task. This step helps reduce the time and memory required without trading accuracy. The pipeline takes as input a task-specific subgraph (KG'), the GML task, the task budget, and the available resources within the ML environment.

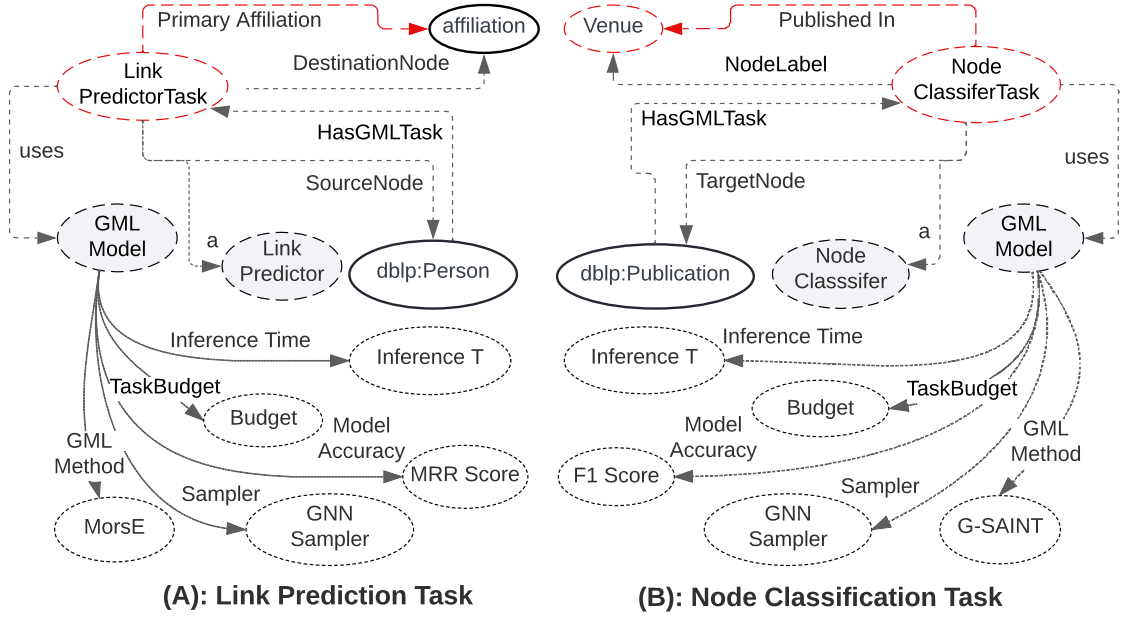


Figure 3.7: A KGMeta graph of two trained models for node classification and link prediction tasks. The white nodes are nodes from the original data KG. The dashed nodes/edges are metadata collected per trained model.

The *Data Transformer* step converts the subgraph into a sparse-matrix optimized for in-memory matrix operations. This format is compatible with popular graph ML data loaders, such as PyG and DGL, and is ideal for sparse KGs. Our pipeline ensures data consistency by validating node/edge types counts, removing literal data and target class edges, and generating graph statistics. We also perform a train-validation-test split using different strategies like random and community-based. KGNET automates this transformation, making ad-hoc GML training queries possible.

The *Optimal GML Method Selection* step selects the best GML method for a given task. KGNET supports various GNN methods, including GCN, RGCN, Graph-SAINT, Shadow-SAINT, Morse, and KGE methods such as ComplEx. We estimate the required memory for each method based on the size and the number of generated sparse-matrices, as well as the training time based on the matrix dimensions and feature aggregation approach. Moreover, we estimate the training time based on the dimension of the sparse-matrices and GNN neighbour nodes features aggregation approach adopted by each method. For GNN sampling-based methods, the sampling cost basically depends on the sampling heuristic used [80]. Thus, we are working on a more advanced estimation method based on sampling the sparse-matrices and running a few epochs on them.

```

1 | prefix dblp:<https://www.dblp.org/>
2 | prefix kgnet:<https://www.kgnet.com/>
3 | Insert into <kgnet> { ?s ?p ?o }
4 | where {select * from kgnet.TrainGML(
5 | {Name: 'MAG_Paper-Venue_Classifier',
6 |   GML-Task:{ TaskType:kgnet:NodeClassifier,
7 |   TargetNode: dblp:publication,
8 |   NodeLabel: dblp:venue},
9 |   Task Budget:{ MaxMemory:50GB, MaxTime:1h,
10 |     Priority:ModelScore} } )};

```

Figure 3.8: A SPARQL^{ML} insert query that trains a paper-venue classifier on DBLP. The TrainGML function is a UDF that is implemented inside the RDF engine.

```

1 | prefix dblp:<https://www.dblp.org/>
2 | prefix kgnet:<https://www.kgnet.com/>
3 | delete {?NodeClassifier ?p ?o}
4 | where {
5 |   ?NodeClassifier a kgnet:NodeClassifier.
6 |   ?NodeClassifier kgnet:TargetNode dblp:Publication.
7 |   ?NodeClassifier kgnet:NodeLabel dblp:venue.}

```

Figure 3.9: A SPARQL^{ML} delete query that deletes a trained model and its meta-data.

KGNET’s GML-optimizer determines the necessary resources for each method and optimizes the training settings, ensuring scalability in distributed environments. The automated pipeline trains a model and collects evaluation metrics and inference time statistics. A URI is generated for the trained model to distinguish it from other models used for inference tasks. The model meta-data is returned to the KGMeta Manager to update the KGMeta graph. Figure 3.7.a and b show the generated meta-data for link prediction and node classification models, respectively. The *Embedding Store* sub-component, shown in Figure 3.3, is used for fast similarity search by storing, indexing, and searching embeddings. The *GML Inferencing* receives HTTP calls for inference, serializes the result into a JSON Restful-API response, and sends it back to the RDF engine, as shown in Figure 3.3.

3.4.2 The SPARQL^{ML} as a Service

We offer a SPARQL^{ML} as a Service, which comprises three main components: Query Manager, KGMeta Governor, and Meta-sampler. In addition, we provide an interfacing language called SPARQL^{ML} that enables users to express SPARQL-like queries for INSERT, DELETE, or

```

1 | prefix dblp: <https://www.dblp.com/>
2 | prefix kgnet: <https://www.kgnet.com/>
3 | select ?author ?affiliation
4 | where { ?author a dblp:person.
5 | ?author ?LinkPredictor ?affiliation.
6 | ?LinkPredictor a kgnet:LinkPredictor.
7 | ?LinkPredictor kgnet:SourceNode dblp:person.
8 | ?LinkPredictor kgnet:DestinationNode dblp:affiliation.
9 | ?LinkPredictor kgnet:TopK-Links 10.}

```

Figure 3.10: A SPARQL^{ML} query predicting author affiliation link (edge) on DBLP KG.

SELECT operations, such that: (i) a SPARQL^{ML} *INSERT* query is used to train a GML model and maintain its metadata in KGMeta (as shown in Figure 3.8), (ii) a SPARQL^{ML} *DELETE* query is used to delete trained model files and associated embeddings from the GML-aaS component and then deletes its metadata from the KGMeta (as in Figure 3.9), (iii) a SPARQL^{ML} *SELECT* query is for querying and inferencing the KG, e.g., the query in Figure 3.10. When a SPARQL^{ML} query is received, the Query Manager parses it. An INSERT or DELETE query is sent to the KGMeta Governor. If it is a SELECT query, it is optimized and rewritten as a SPARQL query.

KGMeta Governor

The KGMeta Governor maintains a KGMeta graph for each KG, using statistics and metadata collected from trained GML models specific to that KG. The INSERT query is a request to train a task on a certain KG. The parsed information includes the task type (such as node classification or link prediction), the task inputs (such as the target nodes and classification labels (Y classes) for a classification task), and a budget (such as memory and time budget). Experienced ML users can provide additional information, such as hyperparameters or a specific GML method. This information is encapsulated as a JSON object, as shown in Figure 3.8. At line 4, the *TrainGML* is a UDF that takes as input a JSON object that encapsulates all required information to train a GML model. The KGMeta Governor sends the task to the meta-sampler to obtain a task-specific subgraph (KG') for the given task. Then governor interacts with the GML Training Manager to automate the training pipeline for this task. Once training is complete, the KGMeta Governor receives the trained model's metadata, including accuracy and inference time, to maintain the KGMeta, as illustrated in Figure 3.7.

```

1 | prefix dblp: <https://www.dblp.org/>
2 | prefix kgnet: <https://www.kgnet.com/>
3 | select ?title
4 |         sql:UDFS.getNodeClass($m,?paper) as ?venue
5 | where {
6 |     ?paper a dblp:Publication.
7 |     ?paper dblp:title ?title.
8 | }

```

Figure 3.11: A candidate SPARQL for $\text{SPARQL}_{pv}^{\text{ML}}$

Task Oriented Sampler

Our Task Oriented Sampler aims to identify a task-specific subgraph (KG') for training a GML task. Each GML task targets nodes of a specific type, such as `dblp:Publication` in $\text{SPARQL}_{pv}^{\text{ML}}$. Our meta-sampler extracts a task-specific subgraph (KG'), which comprises a set of triples with representative triples associated with the target nodes. Based on the KG schema structure the size of KG' is much smaller than the size of KG. This smaller size will optimize training time and require less memory for training the GML task $\mathcal{A}$. However, the KG may contain triples that are not reachable from a target node set v^T or connected via more than three hops from v^T . These triples do not assist the model in generalizing and may lead to over-smoothing problems [81, 82].

Our SPARQL-based task-oriented sampling method determines the scope of the extracted subgraph based on two parameters: (i) the direction d , where $d = 1$ for outgoing and $d = 2$ for bidirectional (i.e., both outgoing and incoming), and (ii) the number of hops h . We evaluated the performance of our method using four combinations of $d \in \{1, 2\}$ and $h \in \{1, 2\}$. Our task-oriented sampling approach achieved better results with $d = 1$ and $h = 1$ for node classification, whereas for link prediction, our meta-sampling method performed better with $d = 2$ and $h = 1$.

The Query Manager

The Query Manager is responsible for optimizing $\text{SPARQL}^{\text{ML}}$ queries for model selection and rank-ordering to evaluate user-defined predicates. In the case of $\text{SPARQL}_{pv}^{\text{ML}}$ shown in Figure 3.2, the query optimizer fetches all URIs of the models satisfying the conditions associated with the user-defined predicate `?NodeClassifier`. The KGMeta is an RDF graph containing optimizer statistics, such as model accuracy, inference time, and model cardinality. Therefore, we use a SPARQL query to obtain the models' URIs, accuracy, inference time, and cardinality. The query optimizer

```

1 | prefix dblp: <https://www.dblp.org/>
2 | prefix kgnet: <https://www.kgnet.com/>
3 | select ?title
4 |   sql:UDFS.getKeyValue(?venues_dic,?paper) as ?venue
5 | where {
6 |   ?paper a dblp:Publication.
7 |   ?paper dblp:title ?title.
8 | {select sql:UDFS.getNodeClass($m,dblp:Publication)
9 |   as ?venues_dic where { } }}

```

Figure 3.12: A candidate SPARQL for $\text{SPARQL}_{pv}^{\text{ML}}$

selects the near-optimal GML model that achieves high accuracy and low inference time. We define this problem as an integer programming optimization problem to minimize total execution time or maximize inference accuracy.

The $\text{SPARQL}^{\text{ML}}$ *Query Re-writer* uses the near-optimal GML model with URI m to generate a candidate SPARQL query. KGNED currently supports two possible execution plans, whose query templates are shown in Figures 3.11 and 3.12. The core idea is to map a user-defined predicate into a user-defined function (UDF), such as `sql:UDFS.getNodeClass`, to send HTTP calls during the execution time to the GML Inference Manager in our GMLaaS to get inference based on the pre-trained model m . The number of HTTP calls may dominates the query execution cost. For example, $\text{SPARQL}_{pv}^{\text{ML}}$ predicts the venue of all papers, whose size is $|\text{?papers}|$.

The query template shown in Figure 3.11 will generate $|\text{?papers}|$ HTTP calls. However, the query template shown in Figure 3.12 reduces the number of HTTP calls to one by enforcing an inner select query constructing a dictionary of all papers and their predicted venues. Then, `sql:UDFS.getKeyValue` is used to look up the venue of each paper. Our query optimizer decomposes the triple patterns related querying the KG triples in the $\text{SPARQL}^{\text{ML}}$ query into sets per variable associated with a user-defined predicate. For example, in the $\text{SPARQL}_{pv}^{\text{ML}}$ query shown in Figure 3.2, our optimizer identifies two triple patterns that match the variable `?paper` and one triple pattern that matches the variable `?venue`. We use a SPARQL query to get the cardinality of each set, which is the number of distinct values of the variable in the dataset. We formulate this problem as another integer programming optimization problem [83] that minimizes the total number of HTTP calls or minimizes the constructed dictionary size, which is based on the model cardinality. For instance, in the query shown in Figure 3.12, our optimizer generates a dictionary of papers and their venues and

Table 3.1: Statistics of the used KGs and GNN tasks. We used four times larger KGs (DBLP and Yago) than the ones reported in OGB [1].

Knowledge Graph	DBLP	YAGO4
#Triples	252M	400M
#Targets	50 Venue 51K Affiliations 1.2M papers	200 Country
#Edge Types	48	98
#Node Types	42	104
Tasks	Node Classification Link Prediction Entity Similarity Search	Node Classification

retrieves the venue of each paper using the UDF *sql:UDFS.getKeyValue*.

3.5 Experimental Evaluation

This section analyzes the ability of KGNET in automating pipelines to train a model for a specific task with less time and memory w.r.t traditional pipelines on full graphs.

3.5.1 Evaluation Setup

Compared Methods: We used RGCN [57] as a full-batch training method, and GraphSAINT (G-SAINT) [69], ShaDowGNN (SH-SAINT) [2] as mini-batch sampling-based methods for node classification, and MorsE [70] as an edge sampling-based method for link prediction. The OGB [1] default configurations are used in both sampling and training. Node features are initialized randomly using Xavier weight initialization in all experiments.

Computing Infrastructure: All experiments are conducted on Ubuntu server virtual machine that is equipped with dual 64-core Intel Xeon 2.4 GHZ (Skylake, IBRS) CPUs, 256 GB of main memory and 1TB of disk storage.

Real KGs: We mainly focus on two benchmark KGs distinguishing in graph size, graph data domain, task type, and connection density including (DBLP [34] and Yago-4 [27]). We conducted two node classification tasks and one link prediction. We followed the tasks used in OGB [1]. Statistics about used KG and tasks are provided in Table 3.1.

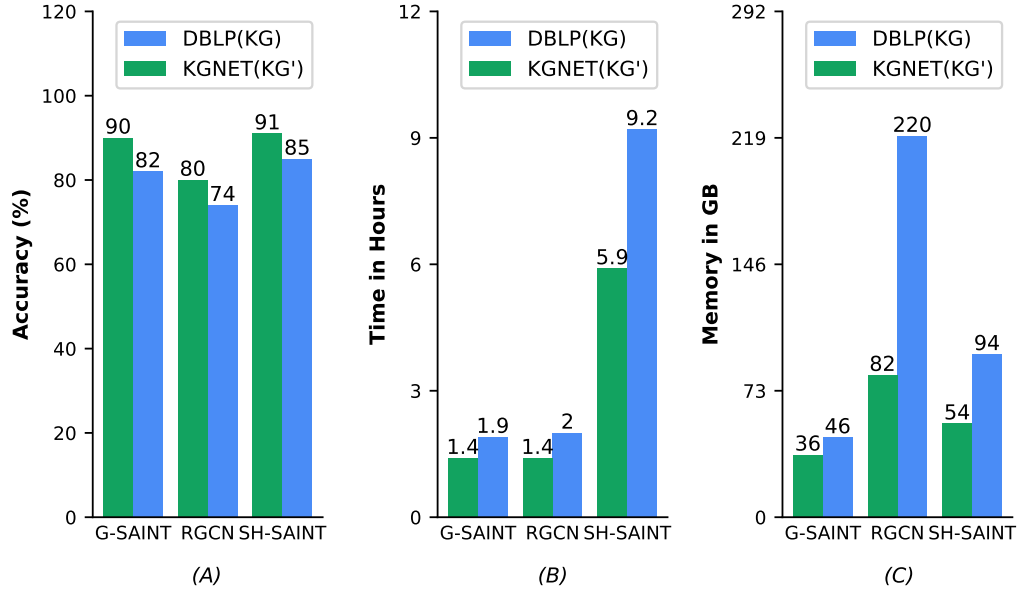


Figure 3.13: (a) Accuracy, (B) Training Time, and (C) Training Memory for DBLP KG Paper-Venue node classification task. The KGNET task-oriented sampled subgraph (KG') significantly improves accuracy, training time, and memory.

Endpoints: We use Virtuoso 07.20.3229 as a SPARQL endpoint, as it is widely adopted as an endpoint for large KGs, such as DBLP. The standard, unmodified installation of the Virtuoso engine was run at the endpoints and used in all experiments.

3.5.2 GML Experiments With Real KGs

Three GML tasks are conducted to evaluate the KGNET automated GML pipeline. For **Node classification task**, GNN methods are used to train node classifiers to predict a venue for each DBLP paper. The KG is loaded into the Virtuoso RDF engine. KGNET performs meta-sampling using *d1h1* to extract the task-specific subgraph (KG') to train RGCN, Graph-SAIN, and Shadow-SAIN methods. The task results in Figures 3.13 and 3.14 show that our KGNet training pipeline using (KG') outperforms the traditional pipeline on the full KG in all methods with up to 11% accuracy score. The automated training pipeline of KGNET has successfully enabled GNN methods to achieve significant reductions in memory consumption and training time. Specifically, KGNET has demonstrated a reduction of at least 22% in memory consumption and 27% in training time. These results demonstrate that KGNET can effectively discover task-specific subgraphs for each task.

The Link prediction task aims to predict an author's affiliation link based on their publications

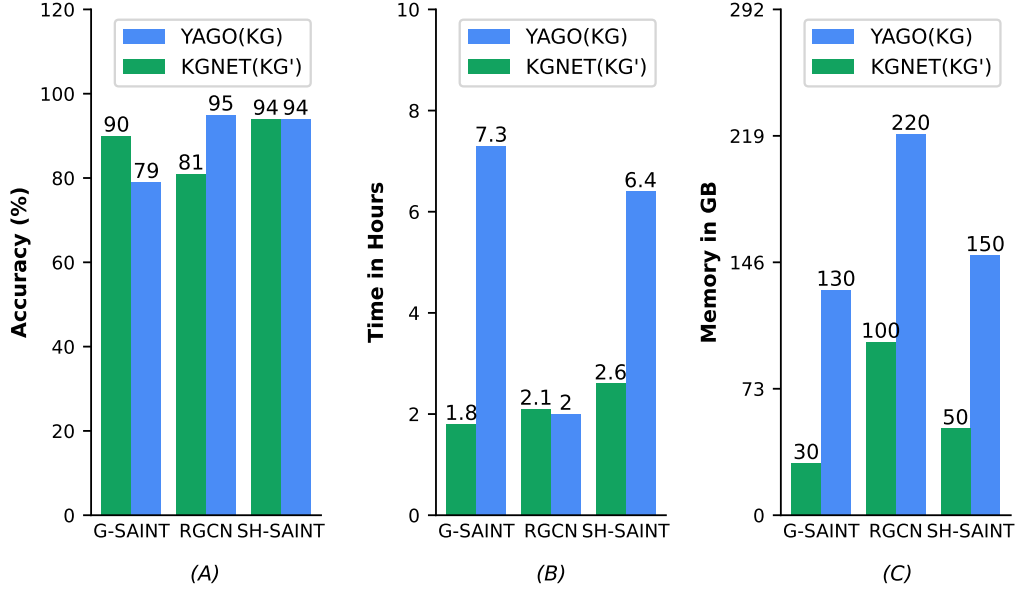


Figure 3.14: (a) Accuracy, (B) Training Time, and (C) Training Memory for YAGO-4 KG Place-Country node classification task. The KGNET task-oriented sampled subgraph (KG') significantly improves accuracy, training time, and memory.

and affiliations history on the DBLP knowledge graph. MorsE [70] is the state-of-the-art link-prediction sampling-based method. We use the MorsE in the traditional pipeline with the full KG. In the KGNET pipeline, our meta-sampling first extracts the task-specific subgraph (KG') using $d2h1$ to train MorsE. The results, shown in Figure 3.15, demonstrate that the KGNET automated pipeline outperforms the pipeline trained on the full KG in terms of Hits@10 MRR score. KGNET achieves a significant reduction in memory usage and training time, with a reduction of 94% compared to the pipeline trained on the full KG.

3.6 Related Work

The adoption of combining AI and database systems has been growing rapidly, with two main approaches emerging: AI models incorporated in DB systems (AI4DB) and database techniques optimized for AI systems for better scalability (DB4AI) [84]. In KGNET, we classify SPARQL^{ML} as part of the AI4DB approach since we have extended the KG engine to query and perform inference on KGs using GML models. However, we classify GMLaaS as part of the DB4AI approach since we have optimized the training pipeline using our meta-sampling approach, which queries a

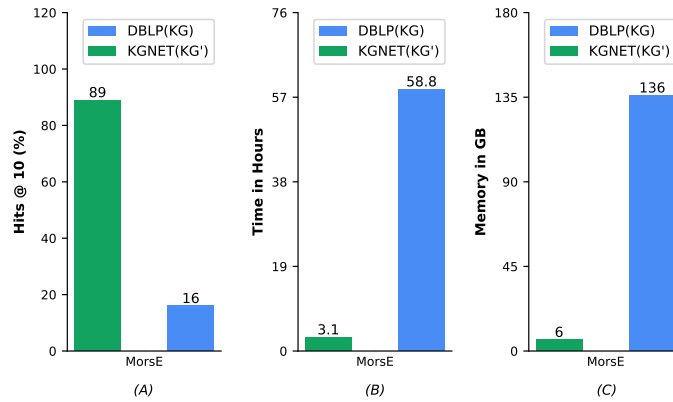


Figure 3.15: (a) Accuracy, (B) Training Time, and (C) Training Memory for the DBLP Author Affiliation link prediction task. The KGNET task-oriented edge sampled subgraph (KG') significantly improves the Hits@10 MRR score, training time, and training memory.

KG to extract a task-specific subgraph. Works RDFFrames [85], DistRDF2ML [86], and Apple Saga [68] aim to bridge the gap between ML and RDF systems by enabling the user to extract data from heterogeneous graph engines in a standard tabular format to apply traditional ML tasks such as classification, regression, and clustering or use KGE methods to generate node/edge embeddings for similarity search applications.

Yuo Lu et.al. addressed the problem of AI-enabled query optimization for SQL in [75] and introduced the probabilistic predicates (PPs) method that can be trained without any knowledge of the inference models. In Learned B+tree [87], the B+tree index is optimized based on AI models that map each query key to its page. Hasan et al [88] allow fast join queries by utilizing auto-regressive densities model to represent the joint data distribution among columns. ITLCS [89] introduced an index selection ML-based method that uses a classifier model as well as a genetic algorithm that selects the accurate index. Stardog [64] supports supervised learning to build predictive analytics models. Stardog enables users to write SPARQL queries that collect the ML training features set in a tabular format and apply classical ML, i.e., classification, clustering, and regression that can be used for inference queries.

Google's BigQuery ML [90] provides user-friendly tools to support AI models in SQL statements by introducing a hybrid language model that contains both AI and DB operations, which executes AI operations on AI platforms such as TensorFlow and Keras. SQL4ML [91] translates ML operators implemented in SQL into a TensorFlow pipeline for efficient training. To enable

ad-hoc GML pipelines using SPARQL, RDF engines require this level of support.

Bordawekar et al. [92] built a cognitive relation database engine that queries database records utilizing word similarity using word2vec embeddings and extends results with external data sources. The cognitive DB represents a step towards linking representation learning with DB using text embedding techniques. EmbDI [93] automatically learns local relation embeddings with high quality from relational datasets using a word embedding to support datasets schema matching. Unlike all the above-mentioned systems, KGNET proposed a platform combining DB4AI and AI4DB approaches to bridge the gap between GML frameworks and RDF engines.

3.7 Discussion

The lack of integration between GML frameworks and RDF engines necessitates that data scientists manually optimize GML pipelines to retrieve KGs stored in RDF engines and select appropriate GML methods that align with their computing resources. Furthermore, the trained models cannot be directly used for querying and inference over KGs, which impedes systems' scalability, particularly as KGs grow in size and require excessive computing resources. Additionally, these limitations impact the system's flexibility, as descriptive query languages are incapable of incorporating GML models. To overcome these limitations, this work proposed KGNET, an on-demand GML-as-a-service (GMLaaS) platform on top of RDF engines to support GML-enabled SPARQL queries (SPARQL^{ML}). KGNET uses meta-sampling to extract a task-specific subgraph (KG') as a search query against a KG for a specific task. Our GMLaaS automates a cost-effective pipeline using KG' to train a model within a given time or memory budget. KGNET maintains the metadata and statistics of trained models as an RDF graph called KGMeta, which is stored alongside associated KGs. KGMeta leads to a seamless integration between GML models and RDF engines, allowing users to easily express their SPARQL^{ML} queries based on the SPARQL logic of pattern matching. Moreover, KGMeta enables KGNET to optimize SPARQL^{ML} queries for model selection and rank-ordering for the inferencing process. KGNET raises research opportunities spanning across data management and AI.

Chapter 4

Task-Oriented GNNs Training on Large Knowledge Graphs for Accurate and Efficient Modeling

4.1 Introduction

A knowledge graph (KG) is a heterogeneous graph that includes nodes representing different entities ($\mathcal{V}$) of $|\mathcal{C}|$ classes, e.g., *Paper* and *Author* and edge types representing $|\mathcal{R}|$ relations, e.g., *publishedIn* and *write*, between these entities. $\mathcal{C}$ and $\mathcal{R}$ are the sets of node and edge types in the KG, respectively. Real-world KGs, such as Yago [27] and Wikidata [28], have hundreds to thousands of node/edge types. Heterogeneous graph neural networks (HGNNs) have emerged as a powerful tool for analyzing KGs by defining real-world problems as node classification, such as recommender systems [20], and entity alignment [21], or link prediction, such as drug discovery [22] and fraud detection [23] tasks.

HGNN methods adopt mechanisms from GNNs designed for homogeneous graphs, such as multi-layer graph convolutional networks (GCNs) and sampling in mini-batch training [2, 56, 69, 70, 94, 95]. Unlike GNNs on homogeneous graphs, the number of GCN layers in HGNNs is associated with the number of edge types or metapaths in the heterogeneous graph to model the distinct

relationships effectively. A metapath captures distinct semantic relationships (edge types) between node types. For example, the metapath Author-(write)-Paper-(publishedIn)-Venue (APV) describes the semantic relationships of an author who published a paper in a particular venue. In mini-batch HGNN training, the incorporated sampling mechanisms aim to obtain a subgraph (a subset of nodes and edges) that captures the representative structure of the original KG in general. The sampling is performed without considering node and edge types, where some specific types may have negligible or no effect in training a particular task. For example, a DBPedia ¹ subgraph, including instances of APV does not contribute to training a model predicting a movie genre. Hence, HGNN methods are not optimized for identifying subgraphs of a smaller subset of types to reduce the number of GCN layers in training a specific task.

The space complexity of training HGNNs includes the memory required to store the graph structures, associated features, and multi-layer GCNs. Additionally, the time complexity includes the sampling process, in the case of mini-batch training, and the message-passing iterations needed to aggregate the embeddings of neighbouring nodes in each GCN. That leads to computationally expensive memory usage and training time with large KGs, where the complexity ² is influenced by the KG’s size, density, $|\mathcal{R}|$, and $|\mathcal{C}|$ [2, 3, 69].

To alleviate the excessive computation on a large KG, AI practitioners manually handcraft a subgraph of a subset of $\mathcal{R}$ and $\mathcal{C}$ from a given KG for training HGNNs [1, 96]. We refer to this subgraph as a task-oriented subgraph (TOSG). Training the task using TOSG instead of the original KG helps HGNN methods reduce training time and memory usage. For example, the open graph benchmark (OGB) [1, 9] provided OGBN-MAG, a subgraph of 21M triples out of 8.4B triples $\approx 0.2\%$ that is extracted from the MAG KG [35] and is relevant to the node classification task for predicting a paper venue (PV). OGBN-MAG contains nodes and edges of only four types.

To demonstrate the effectiveness of this methodology, we trained the PV task on a superset of OGBN-MAG that contains 166M triples $\approx 2\%$ of MAG KG with 42M vertices, 58 node

¹The DBPedia KG contains information about movies and academic work.

²The actual complexity varies depending on the specific implementation.

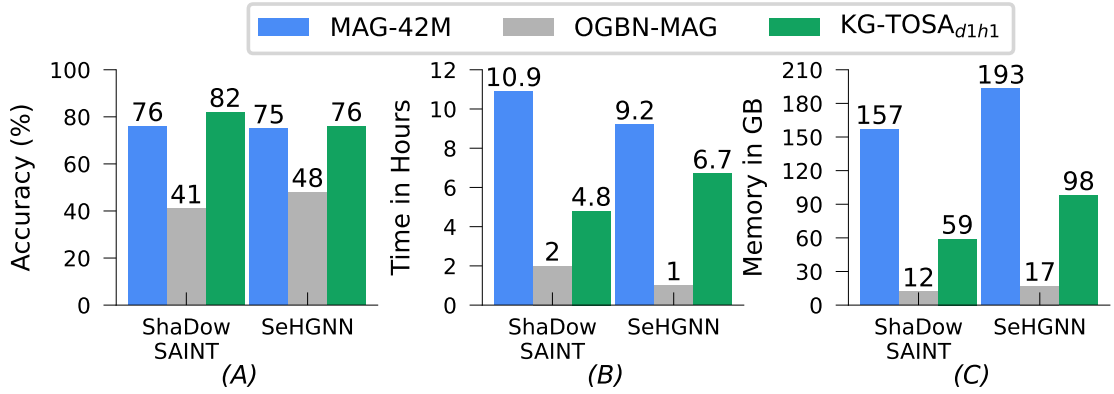


Figure 4.1: (A) Accuracy (higher is better), (B) Training-Time (lower is better), (C) Training-Memory (lower is better). Training a node classification task (*PV*) to predict the paper venue using ShaDowSAINT [2] and SeHGNN [3] on a MAG graph with 42M vertices (*MAG-42M*). The handcrafted task-oriented subgraph (*OGBN-MAG*) from *MAG-42M* trades accuracy to reduce time and memory usage. Our KGTOSA_{d1h1} task-oriented subgraph is extracted automatically from *MAG-42M* for *PV* to reduce time and memory consumption while improving accuracy.

types and 62 edge types; we refer to it as (*MAG - 42M*). The *OGBN-MAG* [1] is a subgraph of *MAG-42M* relevant to *PV*, i.e., task-oriented subgraph for *PV*. We used the state-of-the-art GNN methods, namely ShaDowSAINT [2] and SeHGNN [3]. As shown in Figure 4.1, *OGBN-MAG* trades the accuracy to help both methods reduce time and memory usage in training the task. Furthermore, crafting the TOSG demands a deep understanding of the KG’s structure and the task’s objectives. Hence, the TOSG extraction is a challenging and time-consuming process and does not guarantee to improve the model performance.

This work proposes KGTOSA³, an approach to automate the TOSG extraction for task-oriented HGNN training on a large KG. In KGTOSA, we define a generic graph pattern that effectively captures both the local and global structure of a KG for a specific task. Our definition is grounded in the theoretical foundation of training HGNNs and considering crucial factors, such as data sufficiency and graph topology. The objective of our graph pattern is to maximize the diversity of neighbour node types while minimizing the average depth between non-target and target vertices. We preserve the local context by identifying an initial set of nodes targeted by the task. Then, we expand our selection to include neighbouring nodes within a specified distance of

³KGTOSA stands for KG Task Oriented Sampling. It is available at <https://github.com/CoDS-GCS/KGTOSA>

h hops. Our graph pattern interlinks the local context around each target node to generate a larger subgraph of node and edge types globally related to the task. This approach allows us to effectively capture the intricate relationships and dependencies between different entities in the KG to enhance the representation of the task-specific knowledge captured in the TOSG.

We explore different techniques to extract subgraphs matching our graph pattern. Two techniques sample around targeted nodes using biased random walk or influence scores. We also propose a SPARQL-based extraction method leveraging RDF engines’ built-in indices. Our biased random walk sampling performs random walks starting from nodes that match the target vertices in the task. In our influence-based sampling approach, we utilize a Personalized PageRank (PPR) score to measure the relevance or importance of nodes in the KG to the task. However, it is essential to note that these sampling methods can incur a significant extraction overhead, particularly with a large KG. This excessive overhead may overshadow the potential time savings in training using the TOSG generated by both sampling methods instead of the original KG. To mitigate this overhead, we propose a SPARQL-based method that leverages built-in indices in RDF engines. Hence, our SPARQL-based method automates the TOSG extraction with negligible preprocessing overhead. Figure 4.1 motivates the need for our KGTOSA approach, which helps both ShaDowSAINT and SeHGNN (i) reduce the overall maintenance cost of the multi-layer GCNs, and (ii) improve the modelling performance, including accuracy and inference time.

For evaluating GNN methods on node classification and link prediction tasks, existing benchmarks, such as OGB [1], OGB-LSC [9], and Open-HGNN [97], include KGs of few node/edge types. Unlike these benchmarks, we develop a benchmark of real KGs of large sizes, up to hundred millions of triples and hundreds of node and edge types with various tasks for node classification and link prediction. Hence, our benchmark presents challenging settings for the state-of-the-art (SOTA) HGNN methods. Our benchmark encompasses KGs from the academic domain, such as MAG [35] and DBLP [34], as well as two versions of YAGO as a general-purpose KG of different sizes, namely YAGO-4 [27] and YAGO-3 [98]. We integrated KGTOSA with SOTA GNN methods, namely SeHGNN [3], GraphSAINT [69], ShaDowSAINT [2], RGCN [57], MorsE [70], and LHGNN [56]. Our comprehensive evaluation using large real KGs and various ML tasks shows that KGTOSA enables GNN methods to converge faster and use less memory for training. Moreover,

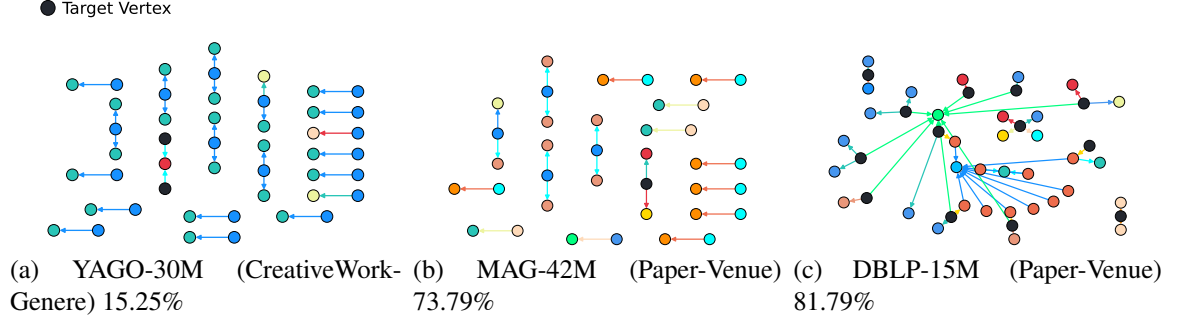


Figure 4.2: Examples of subgraphs generated by the uniform random walk (URW) sampler in GraphSAINT for different GNN tasks. The black vertices indicate the target vertices. The same colour means the same vertex or edge type. This sampling method does not guarantee to (i) include enough representation of target vertices ($\mathcal{V}_T$) and (ii) exclude vertices disconnected from $\mathcal{V}_T$ as these vertices do not contribute to the embeddings of $\mathcal{V}_T$.

KGTOA empowers GNN methods to enhance and attain comparable levels of accuracy.

In summary, this work contributions are:

- the first approach (KGTOA) for task-oriented HGNN training based on our generic graph pattern identifying a KG’s local and global context related to a specific task, Section 4.3.
- three mechanisms for automating the TOSG extraction using our generic graph pattern based on task-oriented sampling and SPARQL-based query extraction. Our SPARQL-based method outperforms the task-oriented sampling in time and memory usage, Section 4.4.
- a comprehensive evaluation using our KG benchmark and six SOTA GNN methods. Our KG benchmark includes four real KGs, six node classification tasks, and three link-prediction tasks. The KGs have up to 42 million nodes and 400 million edges. KGTOA achieves up to 70% memory and time reduction while improving the model performance for better accuracy, smaller model size, and less inference time, Section 4.5,

4.2 Background

4.2.1 HGNNs on KGs

Existing HGNN methods build upon GNN techniques designed for homogeneous graphs by incorporating multi-layer GCNs and employing sampling in mini-batch training [2, 69, 70, 94, 95]. In HGNNs, the number of GCN layers is associated with the number of metapaths in the heterogeneous graph. Metapaths can range from a single edge type and one-hop to paths with multiple hops and

edge types [3, 56]. For a $KG = (\mathcal{V}, \mathcal{C}, \mathcal{L}, \mathcal{R}, \mathcal{T})$, a metapath is defined as a sequence of relations across node types in $\mathcal{C}$ in the form of $c_1 \xrightarrow{r_1} c_2 \xrightarrow{r_2} \dots \xrightarrow{r_h} c_{h+1}$. This leads to composite relations (metapaths), which construct all possible combinations of $\mathcal{C}$ and $\mathcal{R}$ for hop h based on the KG schemas.

Relational Graph Convolutional Network (RGCN) [57] is based on a metapath of one hop and edge type. RGCN uses a stack of GCNs, where each GCN is dedicated to a metapath involving one specific edge type. Various GNN methods, such as GraphSAINT [69], Shadow-GNN [2], MorseE [70], and FastGCN [94], have adapted RGCN to support heterogeneous graphs. We refer to this category of methods as RGCN-based HGNN methods. Another category of HGNN methods supports metapaths involving multiple hops of different edge types, e.g., SeHGNN [3], MHGCN [95], and FastGTN [99]. We refer to this category as metapath-based methods [100].

In both categories, the initial node embeddings are iteratively aggregated with embeddings received from neighboring nodes connected to a specific relation until the embeddings of all nodes converge. For instance, in RGCN-based methods, the final embedding of a node is obtained through two aggregations: an outer aggregation over each relation type and an inner aggregation over neighbouring nodes of a specific relation and defined by RGCN [57] as follows:

$$h_i^{(l+1)} = \sigma \left(\sum_{r \in \mathcal{R}} \sum_{j \in N_i^r} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)} \right) \quad (4)$$

where l is an RGCN layer, $h_j^{(l+1)}$ is the hidden embedding of node j at layer $l + 1$, σ is element-wise activation function, N_i^r denotes the set of neighbour indices of node i under relation $r \in \mathcal{R}$, $c_{i,r}$ is a problem-specific normalization constant that can either be learned or chosen in advance (such as $c_{i,r} = |N_i^r|$), $W_r^{(l)}$ is the weight matrix for relation r at layer l , and W_0^l is the initial weight matrix at layer l .

4.2.2 Sampling Techniques Adapted in HGNNs

HGNN mini-batch sampling methods perform sampling method on a KG then train on a set of sampled subgraphs [101].

Sampling in RGCN-based Methods: Several RGCN-based GNN Methods in [2, 69, 70], adopt

different sampling techniques. These techniques are originally designed for homogeneous graphs to reduce only the number of nodes. Hence, they do not guarantee to reduce node and edge types, which is required in extracting the TOSG. These sampling techniques can be classified into three main groups: node-wise, layer-wise, and recently subgraph sampling. As Node-wise and Layer-wise sampling techniques suffer from scalability to large KGs [101]. Recent GNN methods, such as GraphSAINT [69], Shadow-GNN [2], utilize subgraph-based sampling techniques to sample a subgraph from the original graph G in each epoch. The sampling method ensures that the sampled subgraph G' preserves the general structure of G . GraphSAINT subgraph sampler uses a uniform random-walk sampler (URW) by default to randomly select a set of initial root nodes and performs a random walk of length h from each root node to its neighbours. URW collects all the nodes and edges that are reached in the whole process that forms a sample graph $G'(\mathcal{V}', \mathcal{R}')$. Through URW the high degree nodes have higher probability to be visited without consideration for node/edge types [102]. GraphSAINT further applies normalization techniques during the training to prevent the bias in the induced sub-graphs.

Sampling for Metapaths: The idea is to keep balanced nodes count per node type c while generating the metapaths instances w.r.t the high variability of KG nodes/edges types. Importance-sampling strategy as in [103] is used to reduce the variance in metapaths instances and reduce the number of sampled nodes/edges. Constructing subgraphs for all possible metapaths is a time-intensive process [104]. Additionally, aggregating these metapaths into a metapath graph also incurs a significant amount of memory overhead [95]. SeHGNN [3] optimizes the metapath sampling and features aggregation cost by performing the GNN neighbour aggregation only once in the pre-processing stage.

4.2.3 GNN Tasks on KGs

GNNs are particularly well adapted in real-world applications for node classification (NC) and link prediction (LP) tasks [16]. Examples of NC tasks are predicting the community to which a node belongs in a social network [105] or a venue for publishing a paper [1].

Definition 4.2.1 (Node Classification) *For a given $KG = (\mathcal{V}, \mathcal{C}, \mathcal{L}, \mathcal{R}, \mathcal{T})$, a node classification task $NC(KG, \mathcal{V}_T, c_T)$ aims to predict labels for each target vertex $v_t \in \mathcal{V}_T$, where v_t has the*

node type c_T . The predicted labels can be either a single-label or a multi-label, depending on the problem setting.

A single-label NC task predicts only one label from a set of mutually exclusive labels. The task objective is to predict the most appropriate label for each node, e.g., a venue of a paper is predicted from a set of venues. In contrast, a multi-label NC task predicts multiple labels simultaneously. The task aims to predict the presence or absence of multiple labels for each node [105], e.g., predicting keywords of a paper.

LP tasks predict the potential or missing link between existing nodes in a graph, e.g., affiliations of an author. LP is formalized either as predicting the missing entities (vertices) or predicting the missing relations [106]. In this work, we consider the missing entities task where we predict the correct vertex that completes $\langle v_t, p, ? \rangle$ or $\langle ?, p, v_t \rangle$. We refer to the known vertex as the target vertex and perform link prediction as follows:

Definition 4.2.2 (Link Prediction) *For a given $KG = (\mathcal{V}, \mathcal{C}, \mathcal{L}, \mathcal{R}, \mathcal{T})$ and for every predicate (edge type) $p_T \in \mathcal{R}$, a link prediction task $LP(KG, \mathcal{V}_T, C_T, p_T)$ aims to predict a set of potential vertices that could be connected to the vertex $v_t \in \mathcal{V}_T$ via the predicate p_T , where v_t belongs to the vertex set $\mathcal{V}_T$ and has the type $c_T \in C_T$.*

Existing benchmarks for HGNN methods, such as [1, 9, 96, 107], use heterogeneous graphs of a few node/edge types. For example, the subgraph MAG240M in OGB [9] includes three node and edge types. These benchmarks defined different NC and LP tasks to be modelled on these graphs. Real KGs contain hundreds to thousands of node and edge types such as Wikidata [28] containing 1,301 and 10,012 node and edge types and Yago [27] containing 8,902 and 156 node and edge types, respectively. There is a need to benchmark using heterogeneous graphs with up to hundreds of types.

4.3 The KGTOSA Extraction Approach

The core contribution of our approach is based on our comprehensive study of data sufficiency and graph topology on HGNNs training. Our findings enable us to discover and design a generic

graph pattern for extracting a TOSG from a KG for a specific task. This section highlights these findings and the graph pattern.

4.3.1 Characteristics of Effective HGNNs Training

Our study explored the factors contributing to good performance and strong generalization in training HGNNs for a specific task. We focused on understanding the characteristics of the graph structure that enable HGNNs to generate distinguished embeddings for vertices targeted by a graph-related task. The learned embeddings are then utilized in a downstream task, such as node classification or link prediction. Our study included two main characteristics, namely data sufficiency and graph topology. In our study, we used GraphSAINT and its Uniform Random Walk (URW) sampler to investigate the quality of the sampled subgraph using three different tasks in three different KGs, as shown in Figure 4.2.(a,b,c). URW samples these subgraphs with a walk-length (no. of hops) of $h = 2$ and an initial set of nodes of 20 for training a GNN task.

Data Sufficiency: Adequate and representative data is essential for training HGNNs effectively [108, 109]. Sufficient data ensures that the model has access to diverse graph structures and informative node/edge attributes relevant to the task at hand. This allows the training process to capture meaningful patterns and relationships. Existing HGNN methods consider the whole KG in training or sample uniformly a representative graph structure. Suppose the training data includes an insufficient number or low ratio of target vertices. This leads to training batches (subgraphs) with a few target vertices (black vertices shown in Figure 4.2). The subgraphs in Figure 4.2.(a) and Figure 4.2.(b) are examples of this case. This limited representation of target vertices adversely impacts the model performance with a slower convergence rate, i.e., increases the training time. Figure 4.2 shows that these methods do not guarantee a diverse graph structure around the target vertices. For example, the target vertices are connected to a limited number of vertices of other types.

Graph Topology: We also investigate the graph topology. The arrangement and connectivity patterns of nodes and edges profoundly impact the quality of embeddings and the efficiency of generating them in HGNNs. The graph topology determines how information propagates and diffuses across the graph during message passing. It influences the reachability of nodes, the diversity of neighbour node types, and the local/global context available for each vertex. By considering the

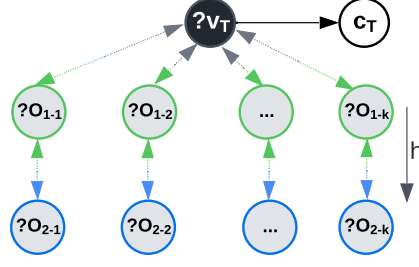


Figure 4.3: The TOSG’s generic graph pattern is based on two parameters: (i) the direction (outgoing and incoming) predicates, and (i) the number of hops.

graph topology, HGNNs can effectively capture and leverage the structural dependencies to generate embeddings that encapsulate the relevant information for the given task.

Suppose the training data includes vertices disconnected from the target vertices. Then, a GNN method will consume unnecessary aggregation iterations to calculate embeddings that will not affect the final embeddings of the target vertices. For example, subgraphs in Figure 4.2.(a) and (b) include several vertices that are disconnected from target vertices. Moreover, the training data may include vertices that are far from the target vertices. For example, the subgraph in Figure 4.2.(c) includes distanced vertices, e.g., up to 4 hops, from a target vertex. This leads to the over-smoothing problem [81, 110], i.e., generating indistinguishable embeddings. Irrelevant or less related vertices have a diminished impact on model generalization and potentially hinder performance. Furthermore, including node/edge types unrelated to the task’s target nodes can introduce unnecessary complexity to the GNN model layers. This increased complexity contributes to more model parameters and prolongs the training and inference time.

Diverse node neighborhoods help improve the efficacy of GNNs [111]. To assess graph structural diversity, we employ Shannon Entropy [112] to measure the variability in the number of neighbor node types per node, as follows:

$$H(\mathcal{N}_t) = - \sum_{i=1}^n \mathbb{P}(\mathcal{N}_t(i)) \cdot \log_2(\mathbb{P}(\mathcal{N}_t(i))) \quad (5)$$

where $\mathbb{P}(\mathcal{N}_t(i))$ represents the probability of the count of neighbour node types for a node i in a subgraph with n nodes. A higher $H(\mathcal{N}_t)$ indicates higher diversity in neighbouring node types. This diversity enhances the graph’s structure to enable more effective learning of node representations.

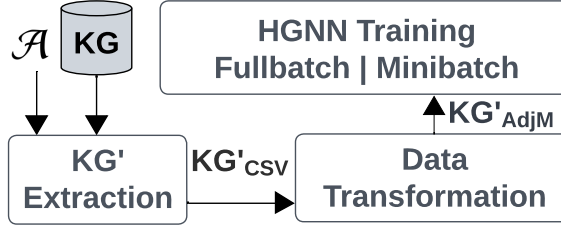


Figure 4.4: The KGTOSA generic workflow. The TOSG (KG') of Task $\mathcal{A}$ is extracted and transformed into an adjacency matrix. Then, the HGNN training performs either mini-batch or full-batch training. The size of KG' is proportional to $\mathcal{V}_T$, the average degree of $\mathcal{V}_T$, and the average distance to $\mathcal{V}_T$. The smaller KG' , the faster the training pipeline.

Even methods that utilize full-batch approaches, such as RGCN, train on the whole KG and treat all vertices equally in the training process. These methods do not exploit the potential benefits of data sufficiency and the inherent structure of the graph topology. Consequently, these methods allocate resources toward vertices that have no or little impact on the desired task. That leads to wasted computational time and memory usage. Hence, training using the TOSG is crucial for optimizing model complexity, and improving training and inference efficiency for both mini- and full- batch training methods.

4.3.2 The KGTOSA's Graph Pattern

We introduce a graph pattern for a TOSG that is carefully designed based on our study of data sufficiency and graph topology. We define the TOSG as follows:

Definition 4.3.1 (Task-oriented Subgraph for Training HGNNs) *Given a knowledge graph $KG = (\mathcal{V}, \mathcal{C}, \mathcal{L}, \mathcal{R}, \mathcal{T})$, and a GNN task $\mathcal{A}$ targeting the set of vertices $\mathcal{V}_T$, there is a task-oriented subgraph of KG , denoted as $KG' = (\mathcal{V}', \mathcal{C}', \mathcal{L}', \mathcal{R}', \mathcal{T}')$. KG' is a compact subset preserving the local and global graph structure of KG relevant to $\mathcal{A}$. KG' aims to maximize the neighbour node type entropy, where every non-target vertex is reachable to a vertex in $\mathcal{V}_T$, and minimize the average depth between non-target to target vertices.*

In our graph pattern, we expand each vertex in $\mathcal{V}_T$ to its neighbouring vertices connected via incoming or outgoing predicates in a range of h hops. Figure 4.3 illustrates our graph pattern. This graph pattern helps to guarantee a high quality of the final embeddings generated for $\mathcal{V}_T$.

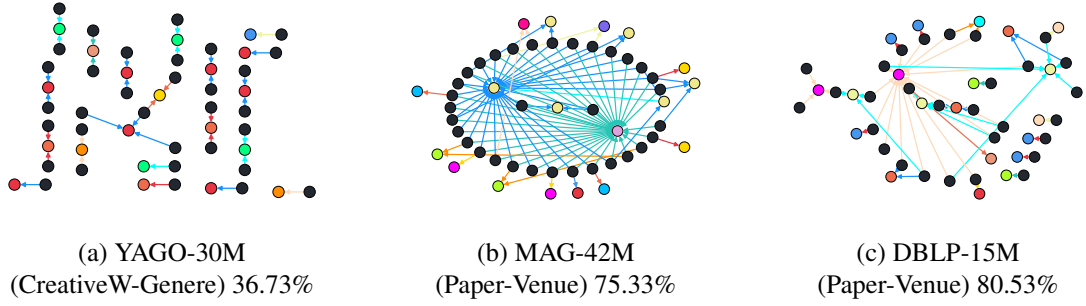


Figure 4.5: The subgraphs (a, b, and c) are generated by our biased random walk sampler. The black vertices indicate the target vertices. Node and edge types are colour-coded. Our approach leads to a higher ratio of target vertices w.r.t. RW in Figure 4.2 while the non-target vertices of different node/edge types are reachable to at least one vertex in $\mathcal{V}_T$.

These embeddings are obtained by aggregating the initial embeddings of $\mathcal{V}_T$ with those of its neighbouring vertices up to a specified hop distance h , where each iteration of aggregation reflects embeddings of neighbouring vertices at hop $h + 1$ (e.g., neighbours of neighbours).

Our graph pattern preserves the local structure for target nodes by considering the reachable incoming or outgoing neighbours to a certain depth h . Merging the $\mathcal{V}_T$ neighbours to construct the KG' will preserve the global structure for $\mathcal{V}_T$ by including the links between their neighbours, which results in a deeper graph with node/edge distribution close to the original KG depth. Figure 4.4 shows an end-to-end workflow of TOSG (KG') extraction, transformation, and HGNN training. Figure A.10 illustrates examples of the subgraphs extracted using our graph pattern for the same tasks and KGs used in Figure 4.2. In these subgraphs, all non-target vertices are reachable to at least one target vertex, and the target vertices are surrounded by more diverse sets of vertices and edges of different types.

4.4 Our Graph Pattern in Practice: Algorithms

This section introduces three techniques for extracting subgraphs matching our graph pattern: biased random walk (BRW), influenced-based sampling (IBS), and a SPARQL-based method. Each technique achieves this goal through different mechanisms (random walks, influence-based sampling, and SPARQL Basic Graph Pattern (BGP) matching), contributing to the overall objective outlined in Definition 3.1. We adapted our graph pattern to extend the recent approaches applied

to homogeneous graphs to develop our two main baselines: BRW and IBS. These baselines play a crucial role in evaluating the effectiveness of our SPARQL-based method.

4.4.1 Biased Random Walk Sampling

We developed the BRW technique, whose pseudocode is illustrated in Algorithm 1. Our technique biases the random walk expansion toward graph regions centered around the target vertices within a given walk length. We adopt the widely studied URW [69] to choose an initial set of vertices randomly from the set of $\mathcal{V}_T$. This adoption maintains task-relevant nodes by excluding the task targets' disconnected nodes, as shown in Figure A.10.(a,b, and c). It also increases the number of target nodes to allow data sufficiency. The randomness in the walk, when biased towards the target vertices, contributes to maximizing neighbor node type entropy and ensuring reachability, i.e., better graph topology.

In Algorithm 1, the function `getInitialVertices` randomly selects bs vertices from $\mathcal{V}_T$, as shown in line 2. Then, a random walk sampler expands these initial vertices $\mathcal{V}_{initial}$ in h hops to finally select a set of nodes $\mathcal{V}_s$ as shown in lines 3-6. The KG' subgraph is constructed in line 7 by including all edges between $\mathcal{V}_s$ nodes in the original KG. The BRW sampler preserves the local graph structure relevant to the task by considering connected vertices to $\mathcal{V}_T$ in performing the random walk. The subgraph extraction in line 7 interconnects all edges between these nodes to construct KG' . This helps BRW interlink between different vertices located across the KG and related to the task.

Two factors dominate the complexity of BRW. First, the random walk complexity $\mathcal{O}(h|\mathcal{V}_T|)$ in lines 4-6, where h is the walk length that controls how far neighbour nodes to include in the sampled subgraph. Second, the subgraph extraction in line 7 with complexity $\mathcal{O}(d|\mathcal{V}_s|)$, where d is the average degree of nodes in $\mathcal{V}_s$ that depends on the structure of the graph. As d decreases, $\mathcal{V}_s$ generally increases in size, leading to a higher extraction cost for KG' . Larger $\mathcal{V}_T$ and h as well contributes to larger $|\mathcal{V}_s|$. Random walks are performed per v_t . So it could be easily parallelized. However, the dominating factor of Algorithm 1 is the subgraph extraction at line 7. The main advantage of BRW is the lightweight sampling complexity.

Algorithm 1 Biased Random Walk Sampling

Input: Knowledge graph KG ; GML task $\mathcal{A}$; Walk length h ; batch size bs .

Output: Knowledge graph KG' .

```
1: function BRW_MS( $KG, \mathcal{A}, h, bs$ )
2:    $\mathcal{V}_{initial} \leftarrow \text{getInitialVertices}(bs, \mathcal{A}.\mathcal{V}_T)$ 
3:    $\mathcal{V}_s \leftarrow \mathcal{V}_{initial}$ 
4:   for  $v \in \mathcal{V}_{initial}$  do
5:      $\mathcal{V}_s \leftarrow \mathcal{V}_s \cup \text{randomWalkSampler}(KG, v, h)$ 
6:   end for
7:    $KG' \leftarrow \text{extractSubgraph}(\mathcal{V}_s, KG)$ 
8: end function
```

4.4.2 Influenced-based Sampling

We also developed an influenced-based sampling (IBS) technique that expands from target vertices to neighbouring nodes based on an influenced-based score. This score is to reflect the importance of a node to a particular target vertex. Our IBS technique adapts the PPR [113] to approximate an influence score between the two nodes, where the influence score determines the local scope sensitivity of node v on node u as follows:

$$I(v, u) = \sum_i \sum_j \left| \frac{\partial h_{ui}^{(L)}}{\partial X_{vj}} \right| \quad (6)$$

The equation (6) represents the influence score between nodes v and u in a neural network model. In this equation, $h_{ui}^{(L)}$ refers to the i -th element in the embedding of node u at the last layer L , while X_{vj} represents the j -th feature of node v . The partial derivative in equation (6) measures the contribution of node v to the embedding of node u in the final layer L . A higher value indicates that the update from node v has a more significant impact on node u . This influence measurement is crucial for filtering out irrelevant neighbour nodes in a given task. The influence score function can be implemented using various methods, such as a weighted random walk function with l steps or the shortest path distance. IBS potentially increases neighbour node type entropy by strategically selecting nodes based on their influence and ensures reachability by expanding from target vertices to the most influencing neighbour nodes.

Algorithm 2 is the pseudocode of IBS. The influence score between each target node in $\mathcal{V}_T$ and all neighbour nodes is calculated using the approximate PPR in lines 2-3 on a homogeneous graph. The top- k pairs of target neighbour nodes are selected to construct a graph partition of bs target

Algorithm 2 Influence-based Sampling

Input: Knowledge graph KG ; GML task $\mathcal{A}$; batch size bs ; top- k k ;

Output: Knowledge graph KG' .

```
1: function IBS( $KG, \mathcal{A}, pn, k$ ):  
2:   Inf-mat  $\leftarrow$  getInfluenceScore( $KG, \mathcal{A}.\mathcal{V}_T$ )  
3:    $topk\_pairs \leftarrow$  SelectTopK-Nodes( $\mathcal{A}.\mathcal{V}_T$ , Inf-mat,  $k$ )  
4:    $gPartition \leftarrow$  getPartition( $KG, topk\_pairs, bs$ )  
5:    $KG' \leftarrow$  extractSubgraph( $KG, gPartition$ )  
6: end function
```

nodes in line 4. The functions at lines 2 to 4 are parallelized using multi-threading. The graph partition contains the neighbour nodes with high overlap between the bs nodes for efficient batches. The graph partition nodes and their in-between edges in the original KG are used to construct the KG' subgraphs. The IBS sampler preserves the local graph structure of the task by selecting nodes connected to $\mathcal{V}_T$ with the highest influence scores. Simultaneously, it takes into account all edges among the chosen nodes to construct KG' , thereby preserving the global graph structure of the task. After the graph partition selection in line 4, we transform back the homogeneous subgraph into a heterogeneous subgraph by adding the node/edge types. The GCN layers are replaced with RGCN layers to learn the heterogeneous graph semantics effectively.

Two factors dominate the Complexity of IBS. Firstly the PPR to expand for neighbour nodes with complexity $\mathcal{O}(\frac{N_{out}}{\epsilon\alpha})$ where ϵ is the error value that goes to zero and α is the PPR teleport probability which is very small value in dense graphs (between 0.1-0.25). This complexity is dependent on the number of non-target nodes and the graph density. Secondly, the induced subgraph extraction in line 5 with complexity $\mathcal{O}(d|\mathcal{V}_s|)$, where d is the average degree of nodes in $gPartition$ that depends on the structure of the graph. The large k and bs lead to a large subgraph size that requires larger training memory and time. Tuning these parameters helps achieve good balance based on the graph structure and density. However, the user should be aware of these factors for a given KG . Our IBS samples the neighbouring nodes based on their importance. Hence, it may lead to a better model's performance. However, its complexity will lead to excessive computation to extract the TOSG from a dense KG with a large number of vertices. Extracting the TOSG aims to reduce the overall training time and memory usage. Hence, it is necessary to optimize the cost of extracting the TOSG.

Algorithm 3 SPARQL-based TOSG Extraction

Input: KG_{ID} ; GML task $\mathcal{A}$; Number of Direction d hops h ; SPARQL Endpoint SP ; batch size: bs , number of threads: $\mathcal{P}$

Output: Knowledge graph KG' .

```
1: function SPARQL_MS( $KG_{ID}, \mathcal{A}, d, h, SP, bs, \mathcal{P}$ ):  
2:    $BGP \leftarrow \text{getBGP}(\mathcal{A}, d, h)$ .  
3:    $count \leftarrow \text{getGraphSize}(KG_{ID}, SP, BGP)$   
4:    $\mathcal{QB} \leftarrow \text{executionPlanner}(KG_{ID}, SP, BGP, count, bs)$   
5:    $index = 0$  and initializeWorkers( $\mathcal{P}$ )  
6:   Worker [RequestHandler]:  
7:   while  $i < \mathcal{QB}.size()$  do ▷ loop over the query batches  
8:      $Triples + = \text{sparqlExec}(KG_{ID}, SP, \mathcal{QB}[index++])$   
9:   end while  
10:   $KG'_{csv} \leftarrow \text{dropDuplicates}(Triples)$   
11: end function
```

4.4.3 A SPARQL-based Extraction Method

The SPARQL-based method is designed to offload the extraction of subgraphs matching our generic graph pattern to an RDF engine. This facilitates the integration of GNN training systems onto existing RDF engines, which are widely used to store KGs [40, 114]. Our generic graph pattern for KG' , as illustrated in Figure 4.3, could be formalized as a basic graph pattern (BGP), which could be performed as a SPARQL query. Hence, the SPARQL-based method efficiently leverages the built-in indices, which RDF engines construct by default. Moreover, unlike BRW and IBS, the SPARQL-based method eliminates the need for a complete migration of the KG outside the RDF engine. This full migration proves to be computationally expensive with large KGs [40, 114]. The SPARQL-based method preserves: (i) the local graph structure by considering connected nodes to $\mathcal{V}_T$, and (ii) the global structure by merging the generated triples to construct KG' .

Our SPARQL-based method could be tuned to generate query variations by adjusting two main parameters: (i) predicate direction d and (ii) the number of hops h . These parameters control the average distance of nodes from the target nodes and incorporate various node/edge types. In KGTOSA, the BGP query will include by default the outgoing predicates, i.e., $d = 1$, and one hop, $h = 1$ ($KGTOSA_{d1h1}$). TOSG can include three more variations: (i) ($KGTOSA_{d2h1}$) outgoing and incoming predicates, i.e., $d = 2$, and one hop, $h = 1$, (ii) ($KGTOSA_{d1h2}$) outgoing predicates, i.e., $d = 1$, and two hop, $h = 2$, and (iii) ($KGTOSA_{d2h2}$) outgoing and incoming predicates, i.e., $d = 2$, and two hop, $h = 2$. A SPARQL query (Q^{d2h1}) for $KGTOSA_{d2h1}$ could be formulated as:

```

1  select ?s ?p ?o {
2    select ?v as ?s ?p ?o
3    where { ?v a <Node_Type_URI>.
4             ?v ?p ?o.}
5    union select ?s ?p ?v as ?o
6    where { ?v a <Node_Type_URI>.
7            ?s ?p ?v.} }

```

KGTOSA supports all these variations. These four variations are used for node classification tasks. In the case of a link prediction task targeting vertices of two different types, we add the triple pattern $\langle ?v_{Ti}, p_T, ?v_{Tj} \rangle$ between the two subgraphs extracted for $?v_{Ti}$ and $?v_{Tj}$. Algorithm 3 is the pseudocode of our parallel SPARQL-based TOSG extraction. It formalizes a BGP for a given task based on a certain d direction and h hops in line 2. However, this query may target a large number of vertices, which can cause issues, e.g., network congestion or low bandwidth. To address this, KGTOSA uses compression and pagination optimization techniques when dealing with query results of a large number of triples. Most RDF engines support compression. Hence, KGTOSA sends an HTTP request to the SPARQL endpoint with a compression flag.

Existing RDF engines support pagination using LIMIT and OFFSET. By dividing the results into k mini-batches, most RDF engines will execute the query k times. This is inefficient as our queries are formulated with UNION, such as line 5 in Q^{d1h1} . Repeating the UNION query k time is time-consuming due to duplicate elimination. Thus, Algorithm 3 paginates each subquery independently. Each subquery will benefit from the RDF built-in indices, as the query targets a vertex of a known type. Existing RDF engines support six indexing schemes for traditional lookup on any subject, predicate or object in a SPARQL query [115]. Thus, our SPARQL-based queries are executed efficiently by leveraging the indices existing in RDF engines. Algorithm 3 collects in $\mathcal{P}$ parallel threads the set of triples in a Pandas DataFrame (DF) and finally uses DF to eliminate duplicates. The algorithm gets as input the batch size per HTTP request. These optimizations are performed in lines 3 to 10. The complexity of KGTOSA is dominated by QB size in line 7 and the average execution time per query. Also, the drop duplicates step at line 10 is performed in $\mathcal{O}(|KG'|)$, which is dominated by the small size of $|KG'|$. The d and h parameters contribute to the average query execution time and the size of KG' .

Table 4.1: Our Benchmark Statistics. The number of nodes and edges (RDF triples) is in millions. The number of node types (n-type) and edge types (e-type) is tens to thousands.

KG-Dataset	#nodes	#edges	#n-type	#e-type
MAG-42M	42.4M	166M	58	62
YAGO-30M	30.7M	400M	104	98
DBLP-15M	15.6M	252M	42	48
ogbl-wikikg2	2.5M	17M	9.3K	535
YAGO3-10	123K	1.1M	23	37

The merging process at line 8 enables KGTOSA to maintain longer metapaths. For example, for the metapath Author-Writes-Paper-Cites-Paper-PublishedIn-Venue in MAG, the subgraphs of common vertices will be interconnected. This process forms a larger sub-graph (TOSG) with longer metapaths while still maintaining a smaller number of hops (h) from the target vertices (in this case, vertices of type "Publication"). By using longer metapaths, the SPARQL-based method enables HGNNs to achieve better semantic attention. It also allows for constructing multi-layered GCNs with a limited number of hops. Compared to BRW and IBS, the SPARQL-based method has lower complexity since it does not require sampling on the entire graph and utilizes the RDF engine’s built-in indices. Hence, The SPARQL-based method ensures a decrease in both training time and memory usage while extracting a TOSG of comparable graph quality to those obtained by BRW and IBS in terms of data sufficiency and graph topology.

4.5 EXPERIMENTAL EVALUATION

4.5.1 Evaluation Setup

Our Benchmark (KG datasets and GNN tasks)

Our KG datasets include real KGs, such as MAG, DBLP, YAGO, and WikiKG. The datasets are extracted from real KGs of different application domains: general-fact KGs (YAGO4 and ogbl-wikikg2), and academic KGs (MAG and DBLP). These KGs contain up to 42.4 million vertices, 400 million triples, and tens to thousands of node/edge types. Existing HGNN methods for LP tasks on large KGs require excessive computing resources. Hence, we also used YAGO3-10, a smaller version of YAGO, and ogbl-wikikg2 [1], a dataset extracted from Wikidata. It contains

Table 4.2: A summary of our GNN tasks: Task Types (TT) are single-label node classification (NC) or missing entity link prediction (LP).

TT	Name	KG	Split	Ratio	Metric
NC	PV	MAG-42M	Time	84/9/7	Accuracy
NC	PD	MAG-42M	Time	87/8/5	Accuracy
NC	PC	YAGO-30M	Random	80/10/10	Accuracy
NC	CG	YAGO-30M	Random	80/10/10	Accuracy
NC	PV	DBLP-15M	Time	79/10/11	Accuracy
NC	AC	DBLP-15M	Time	80/10/10	Accuracy
LP	AA	DBLP-15M	Time	99/0.7/0.3	Hits@10
LP	PO	ogbl-wikikg2	Time	94/2.5/3.5	Hits@10
LP	CA	YAGO3-10	Random	99/0.5/0.5	Hits@10

2.5M entities and 535 edge types. Tables A.1 and A.2 summarize the details of the used KGs and our defined NC and LP tasks⁴, respectively. Our benchmark is available online for further study.

Our benchmark includes NC and LP tasks. In KGs, an NC task is categorized into single- or multi-label classifications, as defined in 4.2.1. Our benchmark followed existing datasets in [1–3] and defined single-label NC tasks. We choose the accuracy metric for these NC tasks to evaluate the performance. The train-valid-test splits are either split using a logical predicate that depends on the task or stratified random split with 80% for training, 10% for validation and 10% for testing. Table A.2 summarizes the details of the split schema and ratio. Our benchmark includes two NC tasks per KG with balanced class distribution per task. shorten

For LP tasks, KGs contain many edge types that vary in importance to a specific task. For example, a GNN task predicting the academic affiliation will benefit from the predicates (edge type) *discipline* and *affiliation* in Wikidata but not from predicates related to *movies* and *films*. Moreover, training HGNNs for an LP task on a large KG of many edge types is computationally expensive. Hence, we define our LP tasks for a specific predicate to enable the HGNN methods to train the models with our available computing resources: a Linux machine with 32 cores and 3TB RAM. We choose the Hits@10 metric to evaluate the predictions ranking performance following SOTA methods [1, 2, 70]. The train-valid-test splits are either split using three versions of the KG based on time or randomly. The details of the LP tasks are summarized in Table A.2.

⁴Detailed explanation for our benchmark datasets/tasks are available [here](#).

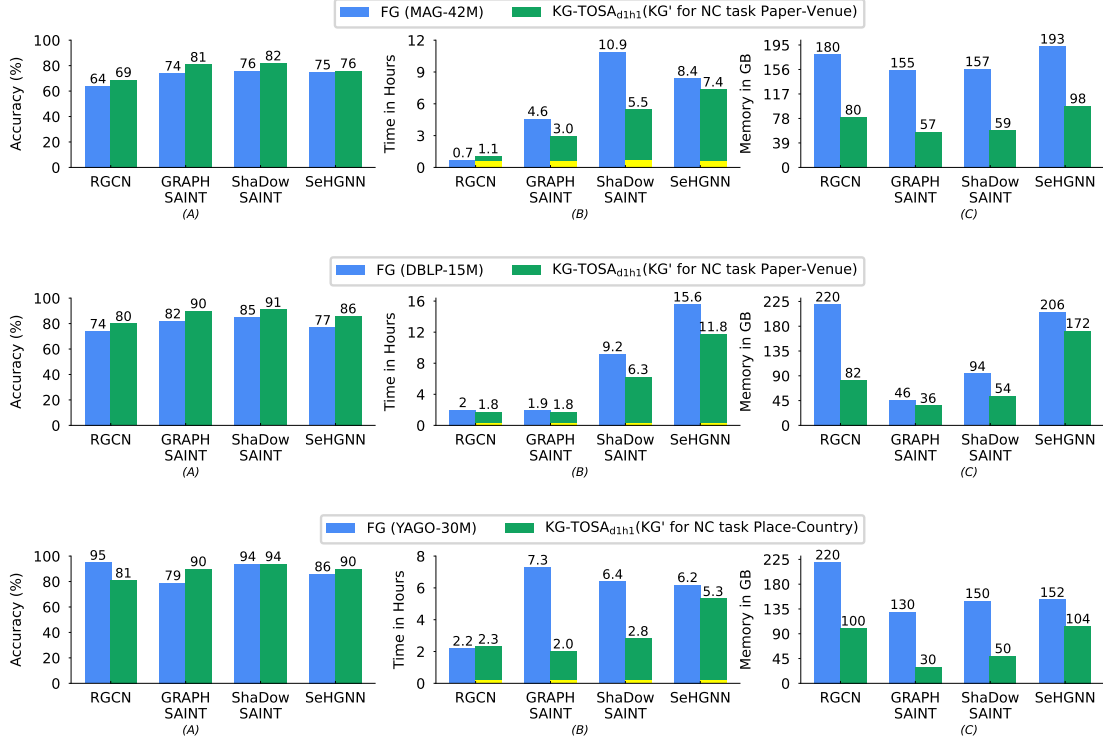


Figure 4.6: Performance across NC tasks is based on three metrics: (A) Accuracy (higher is better), (B) Training-Time (lower is better), and (C) Memory (lower is better). The top and middle sections illustrate the results for the Paper-Venue task on MAG and DBLP, respectively. The bottom figures present the outcomes for the place-country task. With KGTOSA, all methods experience improvements in accuracy in most cases while reducing memory and time, even with KGTOSA’s preprocessing time in yellow.

Computing Infrastructure

Our experiments used two different settings: (i) VM^{GB} : two Linux machines, each with dual 64-core Intel Xeon 2.4 GHz (Skylake, IBRS) CPUs and 256GB RAM, and (ii) VM^{TB} : one Linux machine with 32 cores and 3TB RAM. We use the standard, unmodified installation of Virtuoso 07.20.3229. On VM^{GB} , we installed one Virtuoso instance per KG. We run each method in VM^{TB} for the NC task. We use VM^{GB} for the LP task on Yago3-10 and VM^{TB} for the LP task on DBLP and ogbl-wikikg2.

Compared GNN Methods

We evaluate KGTOSA performance against our developed baselines BRW and IBS using the state-of-the-art (SOTA) GNN methods as follows: (i) RGCN-based methods, such as ShaDowSAINT [2] and GraphSAINT [69], a metapath-based method, namely SeHGNN [3]. These SOTA methods are based on sampling and mini-batch training, (ii) RGCN⁵ [57], full-batch training (no sampling), and (iii) MorsE [70] and LHGNN [56]. We used the code available in the GitHub repositories provided by the authors of each method. The code of SeHGNN, ShaDowSAINT, and GraphSAINT support only node classification tasks. MorsE and LHGNN are the SOTA methods in link prediction, and their code does not support node classification tasks. The RGCN code supports both tasks. We used the default tuned training parameters provided by the authors of each method as indicated in their available code and paper and initialized node embeddings randomly using Xavier weight. MorsE-TransE version is used in the LP tasks. KGTOSA utilizes by default the SPARQL-based method to extract KG' from KG for a specific task. Then, each HGNN method is tested twice: (i) using the original KG denoted as a full graph (FG), and (ii) using the KG' . We perform the three runs of each method on the exact VM and report the metric average value.

4.5.2 The KGTOSA Impact on HGNN Methods

We evaluate the performance of each method in terms of the model performance, training time, and memory consumption for each task. Below, we present our results by task type. $KGTOSA_{dihj}$ has two primary parameters: d (1 or 2) and h (number of hops). For NC tasks, we utilized KGTOSA with only outgoing predicates ($d = 1$) and one hop ($h = 1$), which we refer to as $KGTOSA_{d1h1}$. For LP tasks, we utilized KGTOSA with bidirectional predicates ($d = 2$) and one hop ($h = 1$), which we refer to as $KGTOSA_{d2h1}$.

Node Classification Tasks

We utilized $KGTOSA_{d1h1}$ to extract KG' for each NC task/KG. Figure 4.6 presents the performance of each GNN method using the full KG (FG) and KG' . The preprocessing overhead

⁵ RGCN+ implementation used for NC and RGCN-PYG implementation used for LP.

associated with extracting and transforming the KG' , denoted in yellow in Figure 4.6.(B), is negligible compared to the overall savings. Due to space constraints, we only show the results for three NC tasks. The remaining tasks and confusion matrix results are available in the appendix A.2 and A.3. In general, KGTOSA enables all SOTA methods for the six NC tasks to decrease training time and memory usage while improving accuracy in most cases. The improvement achieved by KGTOSA varies based on different factors, such as the sampling support and the size of KG' compared to FG. The latter depends on the diversity of FG and the number of target vertices in a task. For instance, MAG-42M and YAGO-30M are diverse KGs with around 60 to 100 node/edge types, while YAGO-30M and DBLP-15M are the largest in terms of triples, with 400M and 252M, respectively, as reported in Table A.1.

RGCN is a full-batch GNN method without performing any sampling, unlike other methods, such as GraphSAINT, ShaDowSAINT, and SeHGNN. Hence, RGCN has the shortest training time, but it consumes excessive memory. As a result, KGTOSA enables RGCN to significantly decrease its memory consumption, up to 63%, with DBLP-15M for the paper-venue task. RGCN is the method that benefits the least from KGTOSA in terms of training time. However, GraphSAINT, ShaDowSAINT, and SeHGNN significantly benefit from the extracted KG' by KGTOSA, which empowers them to reduce their time and memory usage across all tasks and KGs with an improvement in accuracy by up to 11%.

Link Prediction Tasks

We utilized $KGTOSA_{d2h1}$ to extract KG' for each LP) task. Figure 4.7 shows the performance of RGCN, MorsE and LHGNN using the full KG (FG) and our KG' . The preprocessing overhead (denoted in yellow) of $KGTOSA_{d2h1}$ is reported in Figure 4.7.(B). This preprocessing is very small w.r.t the training time. Hence, it is hard to be recognized. For the DBLP LP task, our preprocessing is 14 minutes, while training time is 10 hours. KGTOSA enables RGCN, LHGNN, and MorsE to reduce the training time and memory consumption while improving the model’s performance (higher Hits@10) in most cases, as shown in figure 4.7. LHGNN achieved the highest score and consumed excessive time and memory. Hence, LHGNN did not finish training for other LP tasks on the larger KGs, i.e., wikikg2 and DBLP-15MK.

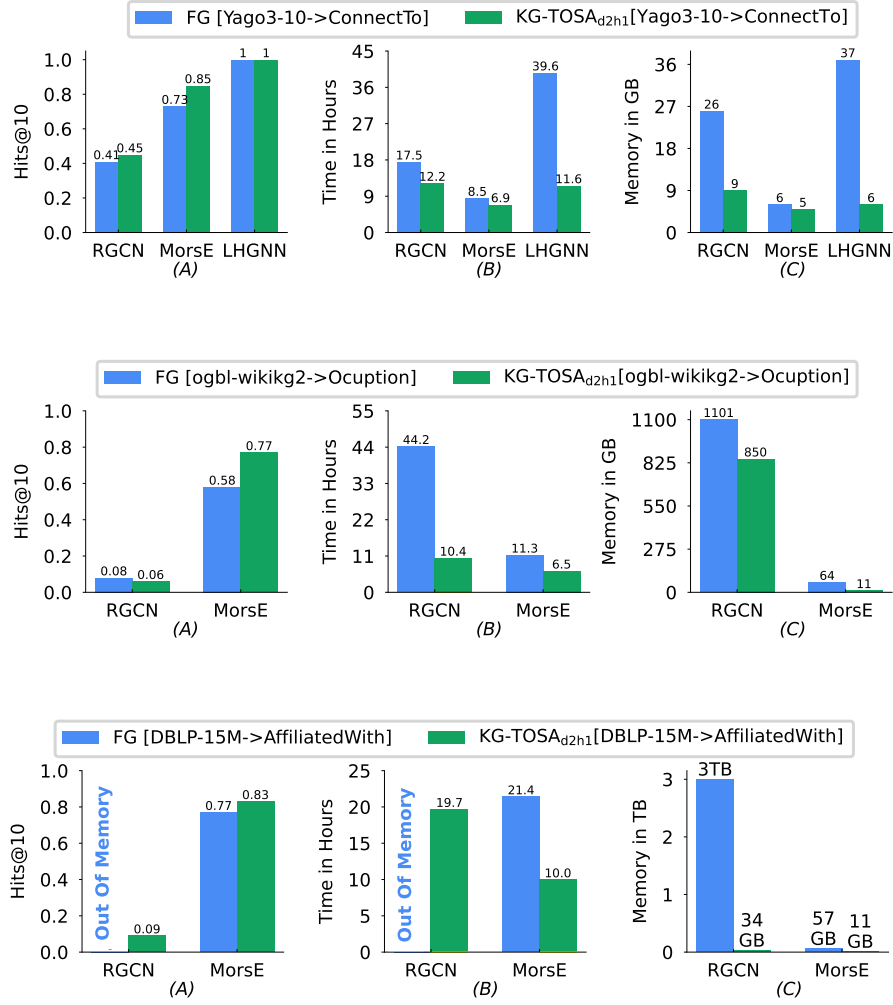


Figure 4.7: Performance of RGCN, MorsE, and LHGNN in the LP tasks. (A) Hits@10 (higher is better), (B) Training-Time, (C) Training-Memory. RGCN exceeded the 3TB RAM in VM^{TB}.

RGCN and MorsE consume more memory and time with large KGs, such as DBLP-15M, w.r.t smaller ones, e.g., YAGO3-10 and ogbl-wikikg2. RGCN failed to train the task on DBLP-15M using VM^{TB} with 3TB RAM. However, KGTOSA enabled RGCN to finish this task with only 35GB RAM. This is a tremendous memory savings of more than 98%. Moreover, KGTOSA empowered MorsE to improve the accuracy of DBLP from 0.77 to 0.83 and ogbl-wikikg2 from 0.58 to 0.77. This improvement demonstrates that as tasks' complexity and the size of KGs increase, KGTOSA enables GNN methods to achieve more performance improvements (better Hits@10) while reducing time and memory usage.

Table 4.3: Statistics of extracted subgraphs using URW, BRW, IBS, and KGTOSA_{d₁h₁} on Yago-4, DBLP, and MAG KGs. The walk length h (no. of hops) is = 3 and the size of the initial set of target nodes is 20k. Our SPARQL-based method using d_1h_1 achieves similar statistics and comparable accuracy to BRW and IBS. However, BRW and IBS consume excessive computing resources w.r.t the SPARQL-based method as shown in Figure 4.8.

Indicator	KG'				Data Sufficiency											
Task	V				V _T (%)				C'				R'			
	RW	BRW	IBS	d ₁ h ₁	RW	BRW	IBS	d ₁ h ₁	RW	BRW	IBS	d ₁ h ₁	RW	BRW	IBS	d ₁ h ₁
CG/YAGO	55720	40936	78562	50429	1.1	61.2	82.4	35.7	69	36	33	34	64	31	33	33
PC/YAGO	55299	42016	79830	50419	11.4	55.3	74.6	35	85	27	41	25	80	22	39	24
PV/DBLP	40618	44886	128026	51833	29.9	65.4	19.4	40.2	34	33	32	24	33	34	33	25
PV/MAG	36149	44956	92103	56744	4.9	78	26.5	36.2	48	20	23	26	34	22	29	25
Indicator	Graph Topology												Other Statistics			
Task	Target-Discon.(%)				Avg.Dist.Target				Avg.Entropy(Eq.5)				Accuracy			
	RW	BRW	IBS	d ₁ h ₁	RW	BRW	IBS	d ₁ h ₁	RW	BRW	IBS	d ₁ h ₁	RW	BRW	IBS	d ₁ h ₁
CG/YAGO	76.7	0	0	0	7.1	4.23	4.7	4.18	1.27	2.68	3.02	2.34	15.25	36.73	42	36.72
PC/YAGO	14.8	0	0	0	7.46	4.12	5.2	4.62	1.27	2.67	2.96	2.40	79.28	96.1	97.2	89.52
PV/DBLP	0	0	0	0	4.23	3.71	3.95	3.1	1.77	2.75	1.64	2.18	81.79	80.53	85.4	89.52
PV/MAG	89.3	0	0	0	3.1	2.9	3.2	3.00	1.49	4.44	2.36	3.18	73.79	75.33	75.4	81.08

Our defined LP tasks are based on one given predicate. Performing LP for all edge types, i.e., KG completion, consumes excessive computing resources. Hence, training HGNNs on large KGs might not be feasible due to a lack of resources. For example, performing KG completion using MorsE on DBLP-15M consumed 330GB of memory and 124 training hours compared with 11 GB and 9.8 training hours using the KG' of KGTOSA for *affiliatedWith* edge type only. This huge saving in memory and time (one order of magnitude less time and memory) enables users to perform LP on predicates of their interest. Moreover, we can efficiently train LP tasks on a set of individual predicates in parallel.

4.5.3 Analyzing KGTOSA

Analyzing our Extraction Methods in Overall Performance

These experiments compare our SPARQL-based method against our developed baseline methods (BRW and IBS) for extracting the TOSG. Our BRW implementation adopts the GraphSAINT subgraph sampler [69]. We replaced the RW subgraph sampler with our implemented BRW sampler. The parameters of BRW are the same for all tasks: $bs = 20000$, $h = 3$ and RGCN embedding-dim=128. For IBS, the parameters are the same for all tasks as follows: $bs = 20000$ and $top - k = 16$. The training parameters are $\alpha = 0.25$, $\epsilon = 0.0002$, and RGCN embedding-dim=128. Our SPARQL-based method is based on two parameters, $d \in \{1, 2\}$ for edge direction and $h \in \{1, 2, \dots, n\}$ for the number of hops. We evaluated four variations of KGTOSA where d and h ranged from 1 to 2, as shown in Figure 4.8.

Table 4.4: A breakdown of the cost involved in training a GNN task: (i) KGTOSA_{d1h1} to extract KG' from FG (optional), (ii) transforming the RDF triples into adjacency matrices (mandatory), and (iii) training the task using GraphSaint.

	PV/MAG-42M		PD/MAG-42M		PV/DBLP-15M		AC/DBLP-15M		PC/YAGO-30M		CG/YAGO-30M	
Step	FG	KG'	FG	KG'	FG	KG'	FG	KG'	FG	KG'	FG	KG'
KG Extraction Time (mins)	-	18	-	16	-	19	-	1	-	22	-	3
Transformation Time (mins)	46	22	41	19	30	11	9	1	52	10	60	3
GNN Training Time (mins)	274	135	290	129	112	85	170	13	439	105	292	23
Total Time (mins)	320	175	331	164	142	115	179	15	491	137	352	29
Accuracy(%)	74	81	67	74	82	90	81	79	79	90	15	37
Model Size (#Params (M))	5349	1415	5348	1408	3301	1477	3306	96	3656	1085	3933	1038
Inference Time (sec)	89	52	87	52	678	454	1003	28	1265	368	1283	32
Training Memory (GB)	155	57	139	57	47	36	80	3	130	30	90	3

We used KGTOSA_{dihj} to extract KG' and compared its performance against BRW and IBS (our baselines). The parameters in KGTOSA_{dihj} determine the TOSG sampling scope around a target vertex, which affects the size of KG' depending on the KG structure. Among the six NC tasks, KGTOSA_{d1h1} met all characteristics discussed in Section 4.3.1 while consuming the least time and memory with comparable accuracy. KGTOSA based on the SPARQL-based variations managed to improve the accuracy in PV/MAG-42M and PV/DBLP-15M and maintain comparable accuracy on PC/YAGO-30M while reducing time and memory w.r.t BRW and IBS. YAGO-30 showcases diversity with 104 node types and 98 edge types (see Table A.1). BRW and IBS employ aggressive sampling that improves the identification of critical features (nodes and edges). Our three methods extract TOSGs with varying ratios of $\mathcal{V}_T$ and $\mathcal{V}_s$, as shown in Table III. IBS, for YAGO-30 PC, yields a KG' with an average of 74.6% target vertices (25.4% $\mathcal{V}_s$), while BRW produces a KG' with an average of 55.3% target vertices (54.7% $\mathcal{V}_s$). Hence, IBS is faster on YAGO-30 PC compared to BRW, whereas BRW demonstrates faster performance than IBS on MAG and DBLP. On the one hand, BRW and IBS achieved better accuracy in the YAGO-30 PC task, but the sampling cost of both methods introduced excessive training time and memory consumption. On the other hand, KGTOSA_{d1h1} introduces the best balance by achieving comparable or better accuracy in all tasks with negligible preprocessing cost. We break down the KGTOSA preprocessing cost in table 4.4. We analyzed the sensitivity of these parameters on the LP tasks, where KGTOSA_{d2h1} fulfils all our evaluation criteria.

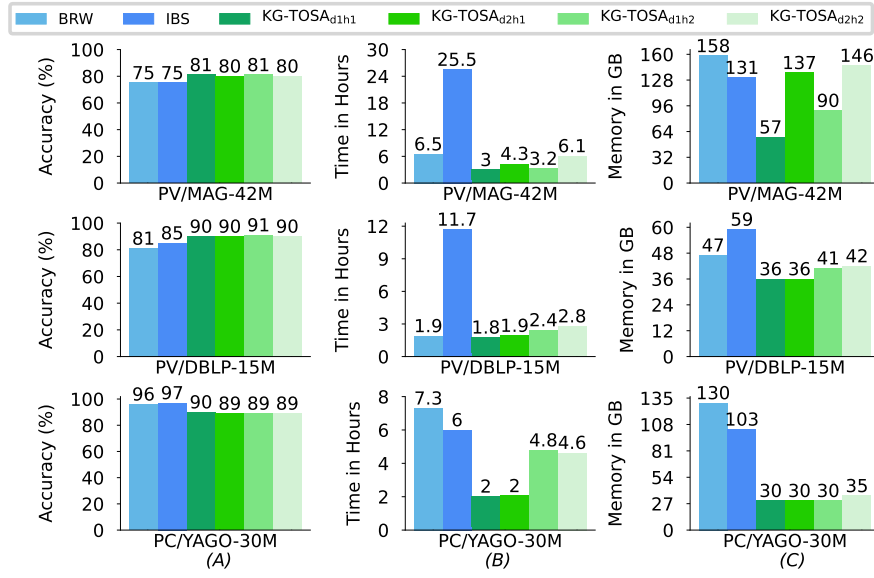


Figure 4.8: (A) Accuracy, (B) Extraction, Transformation, and Training Time, (C) Memory. GraphSAINT+BRW on FG v.s. Our SPARQL-based method with different parameters (KG-TOSA_{dihj}). PV/MAG-42M at the top, PV/DBLP-15M in the middle, and PC/YAGO-30M at the bottom. KG-TOSA_{dihj} achieves comparable accuracy to BRW and IB while enabling HGNN methods to reduce training time and memory usage.

Analyzing our Extraction Methods in terms of Data Sufficiency and Graph Topology

We analyzed the quality indicators of subgraphs generated using our sampling and SPARQL-based methods against the URW for four different tasks on three KGs, as shown in Table 4.3. The BRW, IBS, and KG-TOSA_{d1h1} subgraphs statistics consistently show better quality indicators, compared to URW. For data sufficiency, the percentage of target nodes is deficient in URW samples, which range from 1% to 30%. Our methods increased this ratio in most cases. KG-TOSA_{d1h1} maintained the highest balance between 35% and 40% in all cases. Our generic graph pattern helps the three methods to extract relevant node/edge types. However, URW may include irrelevant node/edge types. Hence, the number of $|\mathcal{C}'|$ and $|\mathcal{R}'|$ is decreased in most cases by our three methods.

For graph topology, our methods excluded disconnected target nodes, reduced the average distance between non-target and target nodes, and enhanced the average entropy of neighbour node types as calculated by Equation 5. This results in a subgraph with fewer hops and a greater diversity

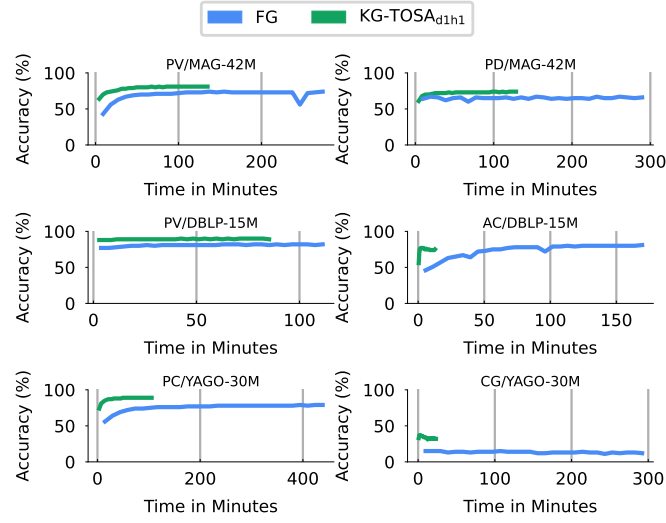


Figure 4.9: Convergence rate analysis. GraphSAINT while training the six NC tasks using the full graph (FG) and KG' extracted by KGTOSA. KGTOSA enables the GNN method to generalize faster with comparable accuracy.

of node types compared to URW. Thus, training models using the KG' helps reduce GNN over-smoothing [110]. Overall, our three methods outperform GraphSAINT, which uses URW by default, with up to 27% better accuracy. These results demonstrate that our methods allow the extraction of higher-quality training subgraphs (mini-batches). However, IBS and BRW incur an excessive extraction overhead, particularly with a large KG that may overshadow the potential time savings.

Task-oriented sampling and Model Convergence Rate

To analyze the time reduction achieved by KGTOSA, we tracked the accuracy achieved by GraphSAINT as a function of the number of training epochs while using FG and KG' . We run with 30 epochs each. However, the average time per epoch with KG' is much shorter. Thus, we report the time instead of the number of epochs in Figure 4.9. Across all tasks, KGTOSA enables GraphSAINT to achieve a faster convergence rate. KGTOSA identifies a smaller subset of the KG, i.e., KG' , that helps GraphSAINT generate mini-batches of high-quality sub-graphs, as shown in Table 4.3. Hence, the model attains optimal performance in fewer and faster iterations, which helps reduce training times and computational costs. We got similar results ⁶ with other GNN methods

⁶Results are shown in the [supplementary materials](#) due to the lack of space.

with an even better convergence rate.

The KGTOSA Cost Breakdown in GNN Training

GNN methods require a KG to be provided as adjacency matrices. Thus, traditional GNN pipelines involve data transformation from RDF triples to adjacency matrices. For extracting KG' , KGTOSA_{d1h1} used VM^{TB} with $\mathcal{P} = 64$ and a batch size bs of 1M triples, see Algorithm 3. For the six NC tasks, Table 4.4 summarizes the cost breakdown, accuracy, model size, inference time, and memory usage for traditional GNN pipelines (FG) and the pipeline with KGTOSA_{d1h1} (KG') for our NC tasks. KGTOSA reduces training and data transformation time, which empowers GNN methods to achieve performance gains that outweigh its negligible preprocessing overhead. KGTOSA identifies a subgraph, KG' , of smaller size and structure. That helps models trained using KGTOSA exhibit a remarkable reduction in size, up to 34 times smaller, while demonstrating significantly faster inference times, up to 40 times faster, as shown in Table 4.4.

4.6 Related Work

We developed KGTOSA as part of our vision towards [40, 114] an on-demand graph ML (GML) as a service on top of KG engines to support GML-enabled SPARQL queries. The KGTOSA task-oriented sampling aims to prune the search space by identifying the minimal subgraph sg in a KG, where sg is related to the task at hand. Unlike our approach, IBMB [102] introduced Personalized Pagerank (PPR) to calculate the influenced scores in homogeneous graphs to sample a representative graph that could be used to train different tasks. IBMB does not support KGs and HGNNs.

There is a growing effort to develop systems for training GNNs on large graphs at scale. These systems aim to (i) distribute the training across multiple GPUs or CPUs in a single machine (scale-up) [116, 117], or (ii) distribute the training across a cluster of machines with multiple GPUs or CPUs per machine (scale-out) [60, 118–121]. KGTOSA can be used with scale-out and scale-up systems to optimize the use of existing resources. All the used GNN methods in our evaluation are based on scale-up frameworks, such as the PYG [59]. NeuGraph [116] leverages multi-GPU in single-machine to improve training performance. Marius [122] optimizes graph embedding learning

by using partition caching and buffer-aware data ordering. There are several scale-out GNN frameworks, as follows. DistDGL [60] uses partitioning with load balancing to achieve high scalability. SANCUS [119] reduces communication overhead using a powerful parallel algorithm and abstracting GNN processing as sequential matrix multiplication. AliGraph [120] also uses static cache but only supports CPU servers. DGL-Dist [60] supports RGCN and different sampling techniques.

4.7 Discussion

Handcrafted task-oriented subgraphs (TOSG) help GNN methods reduce training and memory usage by trading accuracy for performance. Extracting a TOSG from an arbitrary KG is time-consuming and challenging for AI practitioners. In this work, we conducted a comprehensive study to analyze these challenges. Based on our analysis, we discovered a generic graph pattern that captures local and global KG structures relevant to training a specific task. This work proposes two task-oriented sampling techniques based on biased random walks and Personalized PageRank scores for extracting subgraphs matching our generic graph pattern. To address the computational overhead of the task-oriented sampling techniques, we propose a SPARQL-based method leveraging RDF engines for TOSG extraction. This method also avoids the full KG migration while achieving comparable or better modelling performance. This work benchmarks our approach on large KGs, such as MAG, DBLP, YAGO, and Wikidata, with diverse node classification and link prediction tasks. Our comprehensive evaluation includes six state-of-the-art GNN methods. Our results demonstrate that our approach significantly reduces training, inference time and memory footprint while improving performance metrics (e.g., accuracy, Hits@10). Our approach emerges as a cost-effective HGNN training approach, particularly beneficial for large KGs. This is a step forward to developing sustainable approaches and optimizations for data science.

4.7.1 Limitations

- Our task-oriented sampler is tailored for large-scale graphs with high node and edge type heterogeneity; homogeneous or low-heterogeneity graphs are unlikely to show benefit.
- The KG must be loaded and indexed in a KG engine that supports compression and parallel SPARQL execution in order to realize subgraph extraction savings.

Chapter 5

Leveraging LLM-GNN Integration for Open-World Question Answering over Knowledge Graphs

5.1 Introduction

Open-World Question Answering (OW-QA) over knowledge graphs (KGs) aims to answer questions when relevant facts are missing or the KG is incomplete. This challenge arises in real-world domains like biomedicine, scientific research, and finance, where knowledge is often evolving, implicit, or incomplete [39, 123]. For example, systems may need to infer undocumented drug interactions, latent collaborations, or causal relationships. In such cases, answers are not explicitly stored in the KG and must be predicted based on graph structure, entity semantics, and cues in the question.

Traditional KGQA methods assume a *closed-world* setting, where all facts are known and retrievable. This limits their use in dynamic or incomplete environments. Open-world QA instead requires reasoning over both observed and missing information. Unlike standard retrieval or link prediction, it must integrate symbolic language with structural graph signals. This motivates hybrid approaches that combine the semantic flexibility of Large Language Models (LLMs) with the relational reasoning of Graph Neural Networks (GNNs).

Table 5.1: **Average accuracy (%)** on OW-QA benchmarks by reasoning depth. GLOW-GN outperforms all baselines on both 1-hop and 2-hop questions. GCR drops sharply under OWA. All use **Qwen3-8B** as the LLM. Scores for existing datasets are averaged over arxiv2023, ogbn-arxiv, and ogbn-products.

Method	Existing Datasets	GLOW-Bench (ours)	
	1-Hop	1-Hop	2-Hop
LLM _{Only}	25.7	30.0	18.9
GCR	7.8	34.1	15.3
GoG	44.6	48.3	29.5
AskGNN	53.7	79.4	34.3
GLOW-GN	71.7	83.3	42.4

Most existing KGQA systems rely on structured or dense retrieval to find explicit paths between question and answer entities. Methods like G-Retriever [124], GNN-RAG [125], RoG [126], and ToG [127] retrieve graph fragments using semantic similarity. GCR [128] improves scalability but still assumes a complete graph. These methods often fail under open-world conditions, where answer paths are missing or poorly aligned with the question. This results in noisy retrieval and low accuracy. OW-QA instead requires *predictive reasoning* to infer plausible answers beyond observed facts. AskGNN [4] addresses this by using GNNs to guide LLM inference, but relies heavily on structural embeddings and lacks semantic grounding. This limits its ability to handle complex or multi-hop questions.

To address this challenge, we propose GLOW, a novel hybrid approach that synergistically combines the strengths of both LLMs and GNNs. Our method uses a pre-trained GNN to predict the top-k possible answers based on the graph topology. These answers, along with a serialized subgraph of KG facts relevant to the question, are then incorporated into the LLM prompt using a controlled and structured format (e.g., relational triples, top-ranked candidate answers), enabling the LLM to reason jointly over language and structure. Rather than relying on retrieval or pretraining alone, our model dynamically bridges gaps in the KG by augmenting the LLM with on-the-fly, GNN-driven prompts.

We evaluate GLOW on the AskGNN [4] open-world QA benchmark, which is based on ogbn-arxiv, ogbn-products, and arxiv2023 datasets. AskGNN used these datasets to reflect realistic

KG incompleteness and avoid the closed-world assumption in existing benchmarks, such as WebQSP or CWQ [128]. To test generalization, we introduce GLOW-BENCH, a new benchmark of 1,000 natural language questions across diverse domains. Each question requires reasoning over incomplete KGs, with the correct answer deliberately removed. Unlike AskGNN’s dataset, which covers only single-hop questions in less diverse domains, GLOW-BENCH includes 1- and 2-hop questions across multiple KGs.

We compare our method against state-of-the-art baselines, including AskGNN [4], GCR [128], a KG-grounded QA pipeline, GoG [129], KGQA over incomplete KG, and LLMs such as GPT-4o-mini [5] and Qwen3-8B [130]. As shown in Table 5.1, LLMs perform poorly on deeper reasoning tasks, often failing to retrieve or organize relevant knowledge. AskGNN performs better but is limited by its structural focus and lack of semantic flexibility. GCR struggles under incomplete KGs, often hallucinating answer paths or returning incorrect answers. GoG hallucinates answer path generation as it overlooks the underlying KG schema and structural constraints, resulting in semantically inconsistent or invalid paths. Our benchmark highlights these issues and shows the need for models that can reason jointly over symbolic and structural signals.

In summary, our contributions are:

- We propose GLOW, a novel OW-KGQA system that combines a GNN and an LLM to jointly reason over structured and unstructured knowledge.
- We present GLOW-BENCH, a 1,000-question benchmark for open-world KGQA with multi-hop reasoning over incomplete, cross-domain KGs.
- We demonstrate that GLOW outperforms the SOTA LLM–GNN QA and KGQA systems across standard benchmarks and GLOW-BENCH, with up to 53.3% and an average of 38% improvement.

5.2 Related Work

Our work connects to research in KG completion, LLM-based QA, GNN–LLM hybrid models, and open-world KGQA.

Knowledge Graph Completion. Embedding-based methods, such as TransE, RotatE, and ComplEx [52], and recent semantic matching models, like StructurE, HopfE, and DensE [54], aim

to infer missing links using latent representations. Rule-based systems like AMIE3 [131], AnyBURL [132], and SAFRAN [133] generalize KG patterns via logical rules. While useful for KG augmentation, these methods operate independently of question answering. CBR-iKB [134] introduced a case-based reasoning approach based using KGEs which it is computationally expensive on large-scale KGs and operates under transductive settings only. In contrast, GLOW integrates KG completion directly into QA via GNN-guided prompting.

LLMs for KGQA. Fine-tuning LLMs on KG triples can improve domain adaptation [135–137], but requires extensive training and risks catastrophic forgetting [138, 139]. Our approach avoids fine-tuning by injecting both graph-structured data and GNN outputs into LLM prompts, enabling semantic and structural reasoning without modifying LLM weights.

GNN-LLM Hybrid Models. GNN-based QA systems [41, 140, 141] support multi-hop reasoning, but often ignore language semantics. Retrieval-augmented methods, such as GNN-RAG [125], G-Retriever [124], STaRK [142], and RoG [126], embed graph fragments for LLMs to reason over. However, they assume complete graphs and rely on retrieving full answer paths, which breaks under missing links [143]. Our method differs by using a GNN to predict candidate answers and relevant subgraphs, which are serialized into prompts for LLM reasoning.

Open-World QA with In-Context Learning. AskGNN [4] enhances retrieval using GNN-based Structure-Enhanced Retrieval (SE-Retriever) to select in-context examples. However, it relies on joint LLM-GNN training using open-weight LLMs and scales poorly with the model and graph size. It may also bias predictions toward dominant classes. GLOW avoids these issues by using lightweight GNNs for candidate generation and prompt construction, without requiring fine-tuning, and generalizing to various LLMs. GoG [129] addresses KGQA over incomplete KGs by deliberately removing randomly answer path predicates. While effective, this approach cannot guarantee the complete elimination of all answer paths and their associated edges, and generates on the fly LLM-based triples that do not conform to the KG schema, hence causing answer hallucination.

Dense KGQA via Path Retrieval. Methods like G-Retriever, GNN-RAG, ToG [127], and RoG [126] retrieve semantically similar paths, assuming the answer exists in the KG. GCR [128] improves on RoG by fine-tuning LLMs to extract answers from retrieved paths. However, these systems fail in incomplete KG settings where the answer path is missing from the KG [143].

Unlike them, GLOW supports predictive reasoning by prompting the LLM with GNN-predicted candidates and structured facts, even when no complete path exists.

Benchmarks. Existing KGQA benchmarks (e.g., WebQuestionsSP [144], LC-QuAD [145], MetaQA [146], STaRK [142]) assume closed-world settings with guaranteed answer paths. AskGNN [4] introduced open-world benchmarks but is limited to single-hop reasoning in narrow domains. We introduce GLOW-BENCH, a benchmark of 1,000 OWQA requiring single and multi-hop reasoning steps, where gold answers are removed to test generalization under incompleteness.

OW-KGQA vs. GNN Node Classification. Our task fundamentally differs from GNN node classification (NC); it takes natural language questions as input and infers answer nodes using KG structure and semantics, whereas NC operates on a graph as inputs without linguistic reasoning.

5.3 The GLOW Approach

GLOW¹ is a hybrid approach for open-world QA on KGs that integrates graph-based reasoning with LLMs. It leverages *In-Context Learning* (ICL), but instead of relying solely on text, GLOW follows a complementary paradigm: *In-Structure Learning*, where both textual and structured KG signals guide LLM reasoning. GLOW uses a pre-trained GNN to predict top- k candidate answers and retrieve a relevant KG subgraph based on the question’s entity type. These elements are serialized into a structured prompt containing relational triples and GNN predictions. This GNN-guided prompt enables the LLM to reason jointly over linguistic and structural cues, supporting open-world QA without requiring additional fine-tuning.

5.3.1 GLOW Pipeline Overview

We develop three GLOW variants to explore different ways of integrating structured signals: GLOW-G (graph context), GLOW-N (GNN predictions), and GLOW-GN (combined). The full architecture is shown in Figure 5.1. The pipeline proceeds in four stages: QA & Linking, Retrieval, Augmentation, and Generation. Each contributes to building a GNN-guided LLM prompt.

¹Graph-LLM for Open-World Question Answering (QA)

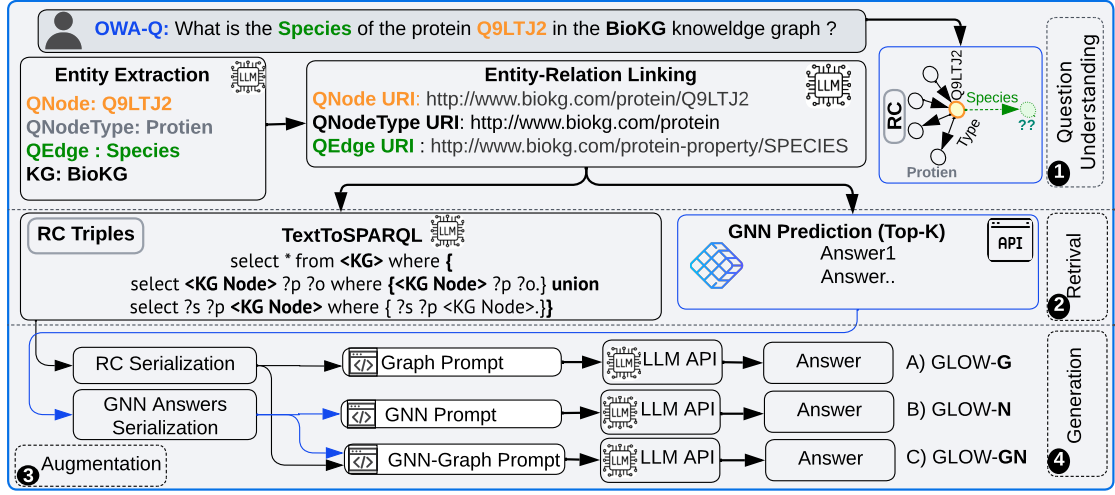


Figure 5.1: Overview of GLOW’s four stages: question understanding, retrieval, augmentation, and generation. GLOW builds GNN-guided prompts using KG context and/or GNN predictions, with 3 variants: (a) KG facts (GLOW-G), (b) GNN predictions (GLOW-N), and (c) both (GLOW-GN).

Question Understanding & Linking: Given a question (e.g., "What is the **species** of the protein **Q9LTJ2** from **BioKG**"?), we extract the question main entity node (Q_n), its type, the target relation (Q_e), and the KG name. The linking step maps these elements to KG schema types using an LLM-based prompt (see Appendix A.6) over RDF metadata. For example, species maps to predicate URI `http://www.biokg.com/property/SPECIES`, denoted e_t , and protein maps to a schema node type URI. We resolve the question entity to node v_t via a SPARQL query over name or label fields. Unlike exact matching, this step uses semantic signals from attributes (name, label, description, URI) and schema-aware prompts to identify the node and relation types robustly. For two-hop questions, the second edge is parsed and appended to the SPARQL query. GLOW generalizes to multi-hop reasoning through extended semantic parsing.

KG Retrieval: LLMs often lack domain-specific facts (e.g., in BioKG) to answer questions about v_t . However, neighboring KG nodes provide rich textual and structural context ($\mathcal{RC}$), which aids in inferring the missing label along e_t . The possible labels L for edge e_t are retrieved via SPARQL and provided to constrain generation. We retrieve 1-hop triples connected to v_t using a SPARQL query generated via *Text-To-SPARQL* that accounts for schema and namespace details.

Instead of retrieving top-K similar entities, as in AskGNN [4], which biases toward dominant

GNN classes, we train a GNN NC model per question pattern (e.g., Protein→Species). These GNNs are trained independently of the LLM and queried via an API at inference time to return top- k candidate answers. See the GNN technical training/inference details in appendix A.6.5. These predictions guide the LLM to correlate graph-derived candidates with the textual semantics in $\mathcal{RC}$, improving robustness. Unlike AskGNN, if the GNN unperformed due to poor graph structure, the LLM relies on $\mathcal{RC}$ textual semantics. This decouples model performance from GNN reliability.

Augmentation: Each prompt contains the question Q , node v_t , edge e_t , and possible answers L , optionally augmented with $\mathcal{RC}$ and GNN predictions. The prompt takes the form:

$$\hat{Y} = f(Q, v_t, e_t, L, \mathcal{RC}, GNN_{Ans}) \quad (7)$$

Variant’s Examples are provided in Appendix A.6.2.

Answer Generation: The LLM predicts a label for each node. Predictions are evaluated using an LLM-as-a-judge module described in §5.4.

5.3.2 GLOW Pipeline Variants

We study how structured prompts influence QA using three *In-Structure Learning* variants.

Graph-Context Prompt(GLOW-G): This variant adds the neighborhood $\mathcal{RC}$ of v_t , serialized into text via verbalization strategies [147] as shown in Figure 5.1.A. While this provides grounded semantic context, it may inflate prompt size if neighborhoods are large. See Figure 5.1 *Text-To-SPARQL* query. AskGNN $\mathcal{RC}$ comprises the top- k similar question nodes as ICL examples, whereas GLOW-G $\mathcal{RC}$ includes a subgraph of attributes and neighboring nodes connected to the question node and serialized to offer contextual grounding to the LLM for effective reasoning.

A GNN-Guided Prompt (GLOW-N): As shown in Figure 5.1.B, this variant injects top- k GNN predictions for node v_t as soft guidance. It provides structural signals without needing full KG serialization, but depends on GNN accuracy. We train a GNN model for each question pattern using

Table 5.2: Summary of Open-World Question Template (OW-QT) across four KG-based features: (1) Knowledge Domain (KD): Domain-Specific (DS), Entertainment (E), or Generic (G); (2) Target Entity Type (ET); (3) Reasoning Hops (RH); and (4) Multiple Choice Count (MCC).

KG	KD	#OW-QT	Target ET(s)	RH	#Class	MCC	Label Types
YAGO4	G	10	Person & CW	1–2	3–102	2–32+	Nationality, Publisher, Occupation
BioKG	DS	6	Drug & Protein	1–2	2–29	2–32	Kingdom, Class, SPECIES, R.Keyword
LinkedMDB	E	5	Film	1–2	7–39	4–32+	Language, Producer, Genre
CrunchBase	DS	4	Investor	1	6–14	4–16	Country, InvestRegion, Company

GraphSAINT. The training subgraphs are extracted using KGTOSA [41], excluding benchmark nodes. This improves scalability and yields diverse, task-specific subgraphs. At inference time, the GNN model returns top- k candidates via API. See GNN details in the appendix.

Hybrid Graph-GNN Prompt (GLOW-GN): This variant combines GLOW-G and GLOW-N by injecting both $\mathcal{RC}$ and GNN predictions. As shown in Figure 5.1.C, it enables the LLM to reason jointly over semantic and structural cues. If GNN confidence is low, the LLM can still rely on the verbalized KG context.

In summary, the LLM is augmented with KG triples in (GLOW-G), GNN predictions in (GLOW-N), or both semantics in (GLOW-GN), leveraging both structural and semantic signals

5.4 An Open-World Benchmark for KGQA

We present GLOW-Bench, a benchmark for multi-hop reasoning across diverse domains. It includes 25 open-world question templates based on four real-world KGs. Each template spans one of four dimensions: reasoning depth, knowledge domain, target entity type, and multiple-choice answer count. Table 5.2 summarizes the templates by (see Appendix A.6.4 for details): 1) *Reasoning Hops (RH)*: 1–2 steps from target to answer; 2) *Target Entity Type*: ranging from general (e.g., people, works) to domain-specific (e.g., drugs, proteins); 3) *Knowledge Domain (KD)*: Generic (G, YAGO4 [27]), Entertainment (E, LinkedMDB [148]), and Domain-Specific (DS, BioKG [149], CrunchBase [150]); 4) *Multiple Choice Count (MCC)*: 2–32+ candidates, one correct.

While GLOW-Bench builds on existing KGs, it introduces a new benchmark for OW-QA and multi-hop reasoning, which current KGQA datasets lack. All questions are designed with answers

absent from the KG, enabling realistic evaluation under incompleteness, and are grounded in real KGs but formulated for OW-QA.

Task Formulation: Each template defines a node classification task. For example, template #1 classifies drugs by structure (Organic vs. Non-Organic) using BioKG. Given a target node v_t , the KG context is retrieved while excluding the gold answer (if present) from $\mathcal{RC}$. The model selects the correct answer from the candidate set.

Answer Evaluation: Evaluating LLM predictions is challenging due to linguistic variation, and exact matching is often insufficient. We use the LLM-as-a-Judge framework [151], where an auxiliary LLM compares outputs to gold answers. Two match types are used for evaluation: 1) *Exact Match (EM)*: Identical or semantically equivalent terms (e.g., *Actor* vs. *Film Star*); 2) *Hierarchical Match (HM)*: Synonyms or subtypes (e.g., *Athlete* and *Player*). We use GPT-4o-mini as the evaluation judge [152].

5.5 Experiments

5.5.1 Experimental Setup

Datasets and Metrics: We evaluated GLOW on four benchmarks; three existing datasets [4] in addition to our GLOW-Bench benchmark. For existing benchmarks, we used arxiv2023, ogbn-arxiv, ogbn-products. These datasets are originally node classification datasets, but AskGNN [4] adopted it for OW-QA, where each dataset is converted into a QA dataset using predefined question templates. In all experiments, we report the average across two runs.

Baselines: We compare KGNET against representative SOTA approaches in 3 categories: *LLM-Only*, *Open-world KGQA*, and *closed-world KGQA*.

LLM: we use commercial models like GPT-4o-Mini [5] and DeepSeek-V3 [153], as well as open-weight models including Qwen3-8B [130], DeepSeek-R1-Distill-Qwen-7B [7], and IBM Granite 3.3-8B-Instruct [8].

Open-world KGQA: 1- *AskGNN* [4] an approach that integrates LLMs and GNNs for QA over homogeneous graph datasets. We adapt *AskGNN* to the OW-QA setting over KGs and evaluate its performance on more challenging benchmark datasets. 2. *GoG* [129] a KGQA system designed for

incomplete KGs where the answer path is intentionally partially removed.

Closed-world KGQA: *GCR* [128] a method designed for scalable KGQA over large KGs. It outperformed *RoG* [126] via fine-tuning LLMs to extract answers from retrieved answer paths more effectively.

Evaluation Setup: GNN training was performed on an Ubuntu VM with dual 32-core Intel Xeon 2.4GHz CPU, 250GB RAM, and V100D-8C 16G vGPU. All models were trained for the node classification tasks using GraphSAINT [69] and ShaDowGNN [2] with the task-oriented sampling method in [41]. KGs were hosted on Virtuoso 07.20.32 with default settings. The GCR 8B models are fine-tuned using Colab A100 GPUs with 40G of VRAM.

5.5.2 Experimental Results

Table 5.3: **Exact Match Accuracy (%) of GLOW vs. baselines across four OWA-QA datasets.** GLOW-GN consistently outperforms AskGNN, GraphSAINT, GCR, and LLM-only setups across all LLMs and datasets, showing strong generalization and effective use of both textual and structural signals, even with smaller LLMs.

	LLM-Model	Dataset	RH	LLM-Only	AskGNN	GCR	GoG	GLOW-G	GLOW-N	GLOW-GN
Open-Weight LLMs	Qwen3-8B [130]	Arxiv2023	1	31	62	4	53	64	66	81
		ogbn-arxiv	1	12	70	14	57	71	60	77
		ogbn-product	1	34	29	5	44	45	51	57
		GLOW-Bench	1	30	79	34	48	60	80	83
		GLOW-Bench	2	19	34	15	29	44	37	42
	DeepSeek-R1-Distill-Qwen-7B [7]	Arxiv2023	1	24	48	7	28	55	59	67
		ogbn-arxiv	1	13	50	15	35	30	61	63
		ogbn-product	1	27	19	12	44	38	43	51
		GLOW-Bench	1	21	75	36	20	58	78	81
		GLOW-Bench	2	12	31	16	13	27	32	39
	Granite-3.3-8B-Instruct [8]	Arxiv2023	1	14	63	14	53	32	63	77
		ogbn-arxiv	1	5	66	21	56	31	55	73
		ogbn-product	1	22	30	13	47	45	52	54
		GLOW-Bench	1	24	79	47	42	58	80	82
		GLOW-Bench	2	14	35	19	27	39	32	42
Commercial LLMs	GPT-4o-mini [5]	Arxiv2023	1	35	N/A	N/A	41	59	48	62
		ogbn-arxiv	1	50	N/A	N/A	58	63	55	67
		ogbn-product	1	37	N/A	N/A	48	43	51	59
		GLOW-Bench	1	29	N/A	N/A	57	68	78	82
		GLOW-Bench	2	28	N/A	N/A	31	53	35	49
	DeepSeek-V3 [6]	Arxiv2023	1	17	N/A	N/A	44	57	61	73
		ogbn-arxiv	1	58	N/A	N/A	61	62	63	65
		ogbn-product	1	36	N/A	N/A	51	45	50	57
		GLOW-Bench	1	34	N/A	N/A	60	69	78	84
		GLOW-Bench	2	30	N/A	N/A	47	54	34	53

Benchmark Results: We evaluate GLOW on three standard OW-QA datasets and our GLOW-Bench benchmark. As shown in Table 5.3, GLOW consistently outperforms all baselines

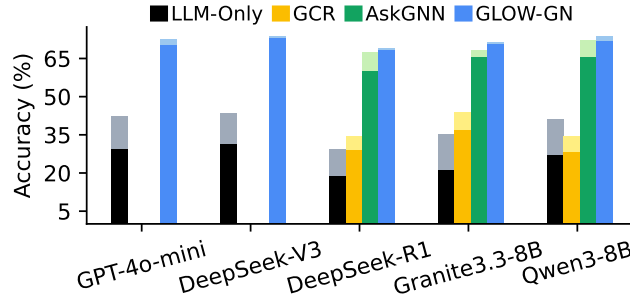


Figure 5.2: Average Exact and Hierarchical Match Accuracy (%) across datasets using different methods and LLMs. Dark bars show Exact Match; lighter segments show added Hierarchical Match gains. GCR and AskGNN are inapplicable to commercial LLMs.

across LLMs and datasets, showing strong generalization and effective use of textual and structural cues. Unlike AskGNN, which depends on GNN-based ICL examples, GLOW retrieves the $\mathcal{RC}$ and combines GNN predictions with neighborhood text for better performance. For example, using Qwen3-8B, GLOW-GN outperforms AskGNN by 18% on average across the AskGNN datasets, and by 4% and 8% on 1-hop and 2-hop GLOW-Bench questions, respectively. It also surpasses GCR by (64%, 49%, and 27%) and GoG by (20%, 35%, and 13%). Similar trends hold across other open-weight (e.g., DeepSeek-R1-Distill-Qwen-7B, Granite-3.3-8B-Instruct) and commercial (e.g., GPT-4o-mini, DeepSeek-V3) LLMs.

AskGNN’s performance varies and declines when either the GNN or LLM underperforms, showing its reliance on both components. GCR depends entirely on the presence of correct answer paths; when missing, it generates incorrect paths and answers. GoG relies entirely on generating missing paths, which are not constrained by the underlying KG schema, often leading to the creation of misleading or semantically invalid answer paths. In contrast, GLOW remains robust even with smaller LLMs, thanks to effective retrieval and contextualization. More detailed results per template and LLM are shown in Appendix A.5. For generic questions like Person $\rightarrow$ Nationality, textual attributes (e.g., name, residence, education) often suffice for LLMs to infer the answer. But for domain-specific tasks (e.g., Protein $\rightarrow$ Family or Protein $\rightarrow$ Species), textual cues are limited. Structurally similar nodes help the GNN yield better predictions.

Combining both inputs in GLOW-GN consistently boosts accuracy—up to 26 points on ogbn-arxiv and ogbn-products, and 6 points on GLOW-Bench templates like creative-work $\rightarrow$ country.

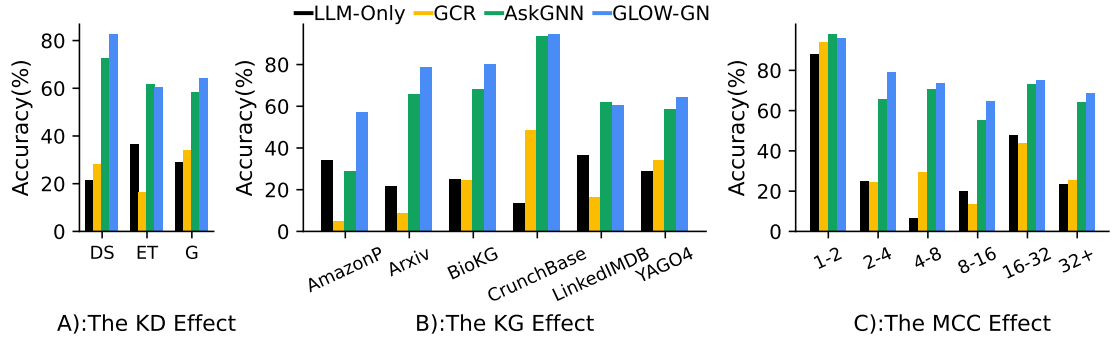


Figure 5.3: Effect of GLOW-Bench characteristics on the answer accuracy (%) of GLOW (with Qwen3-8B). The effects are grounded by A) Knowledge Domain (KD), B) Knowledge Graph (KG), and C) Multiple Choice Count (MCC).

GLOW-G performs best on 2-hop GLOW-Bench with GPT-4o-mini and DeepSeek-V3 due to weaker GNN results on dense graphs. For example, in creative-work→genre, the GNN reached only 22% vs. 33% by GLOW-G. In contrast, for drug→class, GNN accuracy was higher (68% vs. 45%).

Exact vs. Hierarchical Match Accuracy: LLMs paraphrase answers or return semantically related concepts rather than producing exact matches. For instance, the occupation "Singer" may be returned in place of "Artist", its superclass—potentially acceptable in some contexts. Figure 5.2 analyzes this phenomenon by comparing Hierarchical-Match and Exact-Match accuracies, where GPT-4o-mini is used as a judge for the Hierarchical-Match.

The LLM_{Only} pipeline, which relies purely on pretraining without structural grounding, is particularly prone to generating such approximate answers. On average, its Hierarchical-Match accuracy exceeds Exact-Match accuracy by 12.5%, reflecting this tendency. This gap narrows to 3.3% with AskGNN, which supplements the LLM with contextual graph signals but still lacks fine-grained control over output specificity. In contrast, GLOW-GN demonstrates minimal reliance on hierarchical leeway—showing only a 1.1% gain—indicating its robustness in steering the LLM toward precise answers. The integration of textual and structural semantics helps the LLM disambiguate fine-grained targets, improving accuracy and consistency across datasets and LLMs.

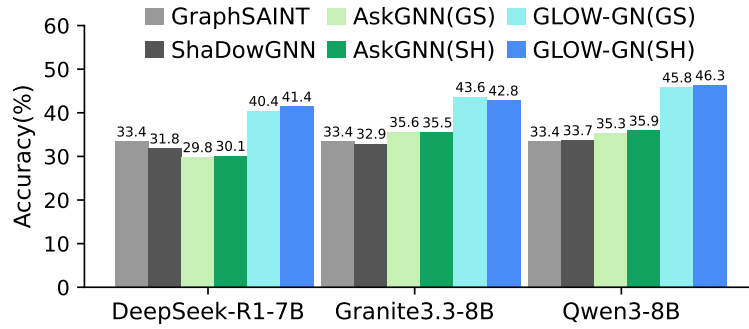


Figure 5.4: Study of error propagation: overall system accuracy in the presence of weaker GNN models where GNN prediction’s accuracy < 50%. GLOW boosts accuracy by up to 12% over weak GNNs using textual cues. AskGNN closely follows GNN performance.

5.5.3 Effect of Domain, Graph, and Question

Knowledge Domain (KD): We analyze model performance across question domains using GLOW-Bench. As shown in Figure 5.3.A, GLOW-GN consistently outperforms all baselines. Gains are most notable in domain-specific (DS) areas such as pharmaceuticals and proteins, where GLOW-GN significantly exceeds models like Qwen3-8B. These results reflect LLMs’ difficulty in handling fine-grained, specialized knowledge. By integrating textual and structural signals, GLOW-GN enables stronger generalization. Even in general domains like Generic (G) and Entertainment (ET), where entities are likely seen during pretraining, GLOW-GN maintains an edge—often rivaling AskGNN without requiring additional fine-tuning.

KG Structure: We assess the effect of KG structure by comparing performance across KGs used in Amazon-Product, Arxiv, and four GLOW-Bench KGs (BioKG, CrunchBase, LinkedIMDB, YAGO4). Figure 5.3.B shows that GLOW-GN outperforms baselines on all domain-specific KGs and performs well on YAGO4, a generic KG. The gap narrows only on LinkedIMDB, likely due to high entity overlap with LLM training corpora.

Answer Choices (MCQ Format): We also examine model robustness under varying multiple-choice (MCQ) settings. Prior work [154] shows LLM performance degrades with more answer choices. Figure 5.3.C confirms this trend. Yet GLOW-GN maintains its advantage, especially when choices exceed two, highlighting the benefit of graph-based disambiguation.

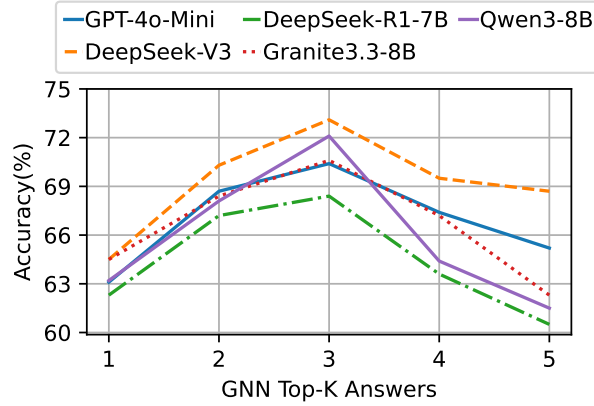


Figure 5.5: Impact of GNN top-K answer count on GLOW-GN’s average accuracy (%) across different LLMs. GLOW-GN achieves the best score at top-K=3, while higher K misleads the LLMs.

5.5.4 GNN Models and Answer Selection

GNN Error Propagation Analysis: AskGNN’s performance depends heavily on GNN quality. When the GNN is weak, AskGNN provides little to no gain—and may even underperformed the GNN itself. Figure 5.4 shows overall QA accuracy on OWA questions on which their corresponding GraphSAINT and ShaDowGNN models (used by AskGNN) scored below 50%, across several LLMs. AskGNN typically matches or lags behind the GNN baseline. In contrast, GLOW-GN combines textual semantics and retrieval-based reasoning to outperform weak GNNs, with gains up to 12%. For example, on Amazon-product with Qwen3-8B, GraphSAINT scored 45%, AskGNN dropped to 29%, while GLOW-GN reached 51%. On Protein-Keyword, AskGNN achieved 35%, GraphSAINT 42%, and GLOW-GN improved to 57%.

Varying GNN Top-K Answers: Increasing the number of top-K GNN answers fed to the LLM does not improve performance; instead, it often confuses the LLM, leading to lower accuracy. The experiment in Figure 5.5 shows that GLOW-GN achieves the highest score at top-K=3 across five LLMs, while increasing it to 4 or 5 reduces overall performance. In particular, large closed-weight LLMs appear less sensitive to higher top K values compared to open-weight models.

5.5.5 System Efficiency Analysis

Training Time: AskGNN’s joint GNN–LLM training grows with graph size, structural complexity (e.g., node/edge types), and LLM scale, averaging 23.4 hours across tasks. GCR, fine-tuned on WebQSP, takes about 6 GPU hours for 500 epochs. contrast, GLOW-N and GLOW-GN decouple

Table 5.4: The average training time in Hours, Answer tokens count per question in (K-Tokens) and answer time per question in Seconds using Qwen3-8B LLM.

Method	Training Time (H)	Tokens Count (K)	Answer Time (Sec)
AskGNN	23.4	0.79	11.2
GCR	6	0.84	13.4
GoG	N/A	1.5	15.7
GLOW-G	N/A	0.78	12.6
GLOW-N	1.7	0.43	8.2
GLOW-GN	1.7	0.65	11.5

GNN and LLM training, requiring just 1.7 hours for GNN training (Table 5.4). This modular design enhances scalability and avoids dependence on LLM size, easing adaptation to new domains.

Token Count: We analyze average token consumption per question across all pipelines. GLOW-N incurs the lowest token usage at 0.43K tokens per prompt, as its retrieved context ($\mathcal{RC}$) consists solely of GNN-predicted answers and candidate labels, mirroring AskGNN’s structure but with reduced verbosity. GCR incurs the higher token cost due to XML-formatted triples. GoG incurs the highest token cost due to performing an agentic chain-of-thought rounds for missing triples generation. In AskGNN averages 0.79K tokens, driven by 20 ICL examples; fewer examples significantly reduce performance. GLOW-G and GLOW-GN offer a balanced cost, with prompt length shaped by KG density and the number of neighbors connected to the question node.

Question Answering Time: GLOW-N provides the fastest answer time, averaging 8.2 seconds per question due to its compact prompt. GCR is slower, as it is fine-tuned to infer answers from retrieved paths, which are often missing in OW-QA, leading to hallucinations and longer processing. GoG is the slowest due to performing an agentic chain-of-thought rounds for missing triples generation. AskGNN follows with 11.2 seconds, slowed by reasoning over ICL examples. GLOW-GN has moderate latency due to combining GNN outputs with subgraph context. All experiments use vGPUs, not high-end A100/H100 GPUs; faster hardware would likely reduce the runtime cost.

Human Based Evaluation: To evaluate our LLM-as-a-Judge prompts, we manually validated the Qwen3:8B GLOW-GN answers across the GLOW-Bench question patterns, using 5 randomly selected questions per pattern. Exact-match accuracy reached 66.6%, Hierarchical match 78.5%, and Manual verification is 82.4%. Notably, Hierarchical matching recovered 95.2% of manually verified correct answers, demonstrating the robustness of our prompts. Some responses were marked

as non-matches hierarchically despite being valid (e.g., (CEO, Co-Founder), (20th Century Studios, Paramount Pictures), (Catalan, Spanish)), while a few cases were incorrectly flagged as non-exact matches (e.g., (American English, English)).

5.6 Discussion

This work introduced GLOW, a system for open-world QA on KGs that integrates LLMs with GNNs. GLOW fuses GNN predictions and relevant subgraphs as structured context to enhance multi-hop reasoning over incomplete KGs. By combining structural and textual semantics, GLOW overcomes key limitations of closed-world and retrieval-based KGQA. It consistently achieves strong performance across question types, domains, and LLMs, even when component quality varies. On standard open-world benchmarks and our new GLOW-Bench dataset, GLOW shows significant improvements in exact and semantic accuracy. These results highlight the need for hybrid approaches tailored to open-world settings and confirm GLOW’s robustness and generalizability.

5.7 Limitations

This work has three main limitations: First, data quality issues, such as sparse KGs with limited node and edge descriptions, impair both textual and structural semantics, reducing performance, especially for large LLMs. Second, our approach depends on high-performing GNN models for effectiveness. Third, all questions are currently framed as node classification tasks, though some may be better suited to link prediction, requiring prior evaluation and task-specific formulation.

Chapter 6

Conclusion

Graphs are ubiquitous, ranging from large-scale social networks and recommender systems to biological networks and small molecules. Many important prediction tasks can be formulated over graphs, such as item recommendation and molecular property prediction. However, designing effective machine learning models for graph-structured data has remained a challenging open problem. Existing approaches often rely on carefully hand-engineered features for each task, which is human demanding and frequently leads to sub-optimal performance.

Deep learning provides a promising paradigm to overcome these limitations. It takes the raw data as input, processes it through deep neural networks, and produces the desired output. These models can be trained end-to-end to directly solve the prediction task of interest. Deep learning has achieved remarkable success in domains such as computer vision, natural language processing, and speech processing. However, most existing models are designed for regularly structured data such as images or sequences and are not directly applicable to irregularly structured graph data. Consequently, a new class of models is required to perform machine learning tasks effectively on graphs.

Graph Neural Networks (GNNs) have emerged as a promising class of deep learning models for predictions over graphs. GNNs follow a *neighborhood aggregation* scheme, in which the embedding of a node is computed by recursively aggregating and transforming the embeddings of its neighbors. GNNs aggregate information from neighboring nodes [18] and have demonstrated promising predictive performance on graph machine learning tasks without requiring manual feature engineering. In this thesis, we aim to address the challenges of developing a powerful AI-enabled KG engine

capable of making accurate predictions over real-world KG data. To this end, we provide comprehensive contributions that span the system design, performance optimization and implementation choices, performance analysis, and benchmarking of HGNN predictive capabilities on various KGs.

In chapter 3, we introduced the seamless integration between the RDF engines and the trained HGNN models to enable predictive question answering for ML non-expert users. Firstly, *KG data migration is inefficient*. Existing graph ML libraries, like DGL and PYG train HGNN models in isolation of the RDF engines. They require exporting the whole KG data, which is impractical for large-scale KGs and fails to capture frequent KG updates. Also, it requires deep experience in graph ML to write complex data transformation, model training, and model inference pipelines.

In Chapter 4 we studied the scalability limitation of training HGNN models on large-scale KGs. *Existing HGNN training methods are task-agnostic*. Existing HGNN methods train on the entire KG data, including task-irrelevant data. This adds noise, increases training time and memory usage, and results in complex multi-layered models with costly information aggregation. Moreover, task-agnostic training does not guarantee optimal performance for specific tasks.

In Chapter 5, we introduced a QA pipeline over incomplete KGs via a carefully designed *Graph-RAG pipeline*. Existing KGQA methods struggle when the question answer is missing from the external resources, such as KG, and incur high retrieval and prompting costs in large-scale KGs .

Altogether, we are excited by the significant progress in GNN research over the years, driven in part by our contributions to a scalable HGNN training on heterogeneous graphs, which reduces the computational cost of training HGNNs while maintaining predictive performance that is critical for enabling HGNNs across industrial-scale prediction tasks. A promising approach is to combine the power of HGNNs, graph transformers [155], and LLMs for Agentic multi-hop KGQA at scale.

Bibliography

- [1] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, “Open graph benchmark: Datasets for machine learning on graphs,” in *Advances in Neural Information Processing Systems 33: NeurIPS*, 2020. [Online]. Available: <https://arxiv.org/abs/2005.00687>
- [2] H. Zeng, M. Zhang, Y. Xia, and et.al., “Decoupling the depth and scope of graph neural networks,” *CoRR*, vol. abs/2201.07858, 2022. [Online]. Available: <https://arxiv.org/abs/2201.07858>
- [3] X. Yang, M. Yan, S. Pan, X. Ye, and D. Fan, “Simple and efficient heterogeneous graph neural network,” *AAAI*, vol. abs/2207.02547, 2023, gitHub Code: <https://github.com/ICT-GIMLab/SeHGNN>. [Online]. Available: <https://doi.org/10.48550/arXiv.2207.02547>
- [4] Z. Hu, Y. Li, Z. Chen, J. Wang, H. Liu, K. Lee, and K. Ding, “Let’s ask GNN: empowering large language model for graph in-context learning,” in *Findings of the Association for Computational Linguistics: EMNLP*, 2024, pp. 1396–1409. [Online]. Available: <https://aclanthology.org/2024.findings-emnlp.75>
- [5] OpenAI, J. Achiam, and S. A. et.al., “Gpt-4 technical report,” 2024. [Online]. Available: <https://arxiv.org/abs/2303.08774>
- [6] DeepSeek-AI, “Deepseek llm: Scaling open-source language models with longtermism,” *arXiv preprint arXiv:2401.02954*, 2024. [Online]. Available: <https://github.com/deepseek-ai/DeepSeek-LLM>

- [7] DeepSeek-AI, D. Guo, D. Yang, and et.al., “Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning,” 2025. [Online]. Available: <https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-7B>
- [8] G. Saon, A. Dekel, and et.al., “Granite-speech: open-source speech-aware llms with strong english asr capabilities,” 2025. [Online]. Available: <https://huggingface.co/ibm-granite/granite-3.3-8b-instruct>
- [9] W. Hu, M. Fey, H. Ren, and et.al., “OGB-LSC: A large-scale challenge for machine learning on graphs,” in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS*, 2021. [Online]. Available: <https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/db8e1af0cb3aca1ae2d0018624204529-Abstract-round2.html>
- [10] D. Liben-Nowell and J. M. Kleinberg, “The link prediction problem for social networks,” in *Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management, New Orleans, Louisiana, USA, November 2-8, 2003*. ACM, 2003, pp. 556–559. [Online]. Available: <https://doi.org/10.1145/956863.956972>
- [11] B. Schölkopf and A. J. Smola, *“Learning with Kernels: support vector machines, regularization, optimization, and beyond”*, ser. Adaptive computation and machine learning series. MIT Press, 2002. [Online]. Available: <https://www.worldcat.org/oclc/48970254>
- [12] A. Grover and J. Leskovec, “node2vec: Scalable feature learning for networks,” in *Proceedings of the 22nd ACM SIGKDD*, 2016, pp. 855–864. [Online]. Available: <https://doi.org/10.1145/2939672.2939754>
- [13] P. Ristoski, J. Rosati, T. D. Noia, R. D. Leone, and H. Paulheim, “Rdf2vec: RDF graph embeddings and their applications,” *Semantic Web*, vol. 10, no. 4, pp. 721–752, 2019. [Online]. Available: <https://doi.org/10.3233/SW-180317>
- [14] J. Portisch, M. Hladik, and et.al., “Kgvec2go - knowledge graph embeddings as a service,” in *LREC(12)*, 2020, pp. 5641–5647. [Online]. Available: <https://aclanthology.org/2020.lrec-1.692.pdf>

- [15] Y. Goldberg and O. Levy, “word2vec explained: deriving mikolov et al.’s negative-sampling word-embedding method,” *arXiv preprint arXiv:1402.3722*, 2014.
- [16] L. Waikhom and R. Patgiri, “Graph neural networks: Methods, applications, and opportunities,” *CoRR*, vol. abs/2108.10733, 2021. [Online]. Available: <https://arxiv.org/abs/2108.10733>
- [17] D. Flam-Shepherd, T. C. Wu, P. Friederich, and A. Aspuru-Guzik, “Neural message passing on high order paths,” *CoRR*, vol. abs/2002.10413, 2020. [Online]. Available: <https://arxiv.org/abs/2002.10413>
- [18] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in *ICLR*. OpenReview.net, 2017. [Online]. Available: <https://openreview.net/forum?id=SJU4ayYgl>
- [19] W. L. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in *Advances in Neural Information Processing Systems 30: NeurIPS , USA*, 2017, pp. 1024–1034. [Online]. Available: <https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7e9bea9-Abstract.html>
- [20] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, “Graph neural networks in recommender systems: A survey,” *ACM Comput. Surv.*, vol. 55, no. 5, pp. 97:1–97:37, 2023. [Online]. Available: <https://doi.org/10.1145/3535101>
- [21] Z. Wang, Q. Lv, X. Lan, and Y. Zhang, “Cross-lingual knowledge graph alignment via graph convolutional networks,” in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018, pp. 349–357. [Online]. Available: <https://aclanthology.org/D18-1032>
- [22] X. Lin, Z. Quan, Z. Wang, T. Ma, and X. Zeng, “Kggn: Knowledge graph neural network for drug-drug interaction prediction,” in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2020, pp. 2739–2745. [Online]. Available: <https://doi.org/10.24963/ijcai.2020/380>

- [23] Y. Dou, Z. Liu, L. Sun, Y. Deng, H. Peng, and P. S. Yu, “Enhancing graph neural network-based fraud detectors against camouflaged fraudsters,” in *The ACM International Conference on Information and Knowledge Management (CIKM)*, 2020, pp. 315–324. [Online]. Available: <https://doi.org/10.1145/3340531.3411903>
- [24] Y. Sun and J. Han, “Mining heterogeneous information networks: a structural analysis approach,” *SIGKDD Explor.*, vol. 14, no. 2, pp. 20–28, 2012. [Online]. Available: <https://doi.org/10.1145/2481244.2481248>
- [25] e. Xiao Wang, Deyu Bo, “A survey on heterogeneous graph embedding: Methods, techniques, applications and sources,” *CoRR*, vol. abs/2011.14867, 2020. [Online]. Available: <https://arxiv.org/abs/2011.14867>
- [26] K. D. Bollacker, C. Evans, and et.al., “Freebase: a collaboratively created graph database for structuring human knowledge,” in *Proceedings of the ACM International Conference on Management of Data, SIGMOD 2008*. ACM, 2008, pp. 1247–1250. [Online]. Available: <https://doi.org/10.1145/1376616.1376746>
- [27] T. P. Tanon, G. Weikum, and F. M. Suchanek, “YAGO 4: A reasonable knowledge base,” in *ESWC*, vol. 12123, 2020, pp. 583–596. [Online]. Available: https://doi.org/10.1007/978-3-030-49461-2_34
- [28] D. Vrandečić and M. Krötzsch, “Wikidata: a free collaborative knowledge base,” *Commun. ACM*, vol. 57, no. 10, pp. 78–85, 2014. [Online]. Available: <https://doi.org/10.1145/2629489>
- [29] M. Wang, Y. Lin, G. Lin, K. Yang, and X. Wu, “M2GRL: A multi-task multi-view graph representation learning framework for web-scale recommender systems,” in *KDD ’20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, CA, USA*. ACM, 2020, pp. 2349–2358. [Online]. Available: <https://doi.org/10.1145/3394486.3403284>
- [30] Y. Dou, Z. Liu, L. Sun, Y. Deng, H. Peng, and P. S. Yu, “Enhancing graph neural network-based fraud detectors against camouflaged fraudsters,” in *CIKM ’20: The 29th ACM International Conference on Information and Knowledge Management, Ireland*. ACM, 2020, pp. 315–324. [Online]. Available: <https://doi.org/10.1145/3340531.3411903>

- [31] M. Yasunaga, H. Ren, A. Bosselut, P. Liang, and J. Leskovec, “QA-GNN: reasoning with language models and knowledge graphs for question answering,” in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021*. Association for Computational Linguistics, 2021, pp. 535–546. [Online]. Available: <https://doi.org/10.18653/v1/2021.naacl-main.45>
- [32] C. Peng, F. Xia, M. Naseriparsa, and F. Osborne, “Knowledge graphs: Opportunities and challenges,” *Artificial Intelligence Review*, Apr 2023. [Online]. Available: <https://doi.org/10.1007/s10462-023-10465-9>
- [33] C. Shi, X. Wang, and P. S. Yu, *Heterogeneous Graph Representation Learning and Applications*, ser. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer, 2022. [Online]. Available: <https://doi.org/10.1007/978-981-16-6166-2>
- [34] M. R. Ackermann. (2022) dblp in rdf. [Online]. Available: <https://blog.dblp.org/2022/03/02/dblp-in-rdf/>
- [35] M. Färber, “The microsoft academic knowledge graph: A linked data source with 8 billion triples of scholarly data,” in *ISWC*, vol. 11779, 2019, pp. 113–129. [Online]. Available: https://doi.org/10.1007/978-3-030-30796-7_8
- [36] L. Wu, P. Cui, J. Pei, and L. Zhao, *Chapter 6: Graph Neural Networks: Scalability*. Singapore: Springer Singapore, 2022.
- [37] H. Zhang, Z. Yu, G. Dai, and et al., “Understanding GNN computational graph: A coordinated computation, io, and memory perspective,” in *Proceedings of Machine Learning and Systems 2022, MLSys 2022, Santa Clara, CA, USA, August 29 - September 1, 2022*. mlsys.org, 2022. [Online]. Available: <https://proceedings.mlsys.org/paper/2022/hash/9a1158154dfa42caddbd0694a4e9bdc8-Abstract.html>
- [38] T. Thanapalasingam, L. van Berkel, P. Bloem, and P. Groth, “Relational graph convolutional networks: A closer look,” *CoRR*, vol. abs/2107.10015, 2021. [Online]. Available: <https://arxiv.org/abs/2107.10015>

- [39] J. Lu and C. Yang, “Open-world taxonomy and knowledge graph co-learning,” in *AKBC*, 2022. [Online]. Available: https://akbc.ws/2022/papers/11_open_world_taxonomy_and_knowledge
- [40] H. Abdallah and E. Mansour, “Towards a GML-Enabled Knowledge Graph Platform,” in *ICDE*. IEEE, 2023, pp. 2946–2954. [Online]. Available: <https://doi.org/10.1109/ICDE55515.2023.00225>
- [41] H. Abdallah, W. Afandi, P. Kalnis, and E. Mansour, “Task-Oriented GNNs Training on Large Knowledge Graphs for Accurate and Efficient Modeling,” in *ICDE*. IEEE, 2024, pp. 1833–1846. [Online]. Available: <https://doi.org/10.1109/ICDE60146.2024.00148>
- [42] H. Abdallah, I. Abdelaziz, P. Kalnis, and E. Mansour, “Leveraging LLM-GNN Integration for Open-World Question Answering over Knowledge Graphs,” Committed to EACL 2026.
- [43] H. Abdallah, W. Afandi, A. Aboulmaga, and E. Mansour, “An LLM-Guided Query-Aware Inference System for GNN Models on Large Knowledge Graphs,” 2025, Under review at ICDE 2026.
- [44] A. Hogan, E. Blomqvist, M. Cochez, C. D’amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A.-C. N. Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, and A. Zimmermann, “Knowledge graphs,” *ACM Comput. Surv.*, vol. 54, no. 4, Jul. 2021. [Online]. Available: <https://doi.org/10.1145/3447772>
- [45] X. L. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmman, S. Sun, and W. Zhang, “Knowledge vault: A web-scale approach to probabilistic knowledge fusion,” in *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, 2014, pp. 601–610, evgeniy Gabrilovich Wilko Horn Ni Lao Kevin Murphy Thomas Strohmman Shaohua Sun Wei Zhang Jeremy Heitz. [Online]. Available: <http://www.cs.cmu.edu/~nlao/publication/2014.kdd.pdf>
- [46] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann, “Dbpedia - A crystallization point for the web of data,” *J. Web Semant.*, vol. 7, no. 3, pp. 154–165, 2009. [Online]. Available: <https://doi.org/10.1016/j.websem.2009.07.002>

- [47] K. Bollacker, P. Tufts, T. Pierce, and R. Cook, “A platform for scalable, collaborative, structured information integration,” in *Intl. Workshop on Information Integration on the Web (II-Web’07)*, 2007, pp. 22–27.
- [48] L. Ehrlinger and W. Wöß, “Towards a definition of knowledge graphs,” in *Joint Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems - SEMANTiCS 2016 and the 1st International Workshop on Semantic Change & Evolving Semantics (SuCCESS’16) (SEMANTiCS)*, ser. CEUR Workshop Proceedings, vol. 1695. CEUR-WS.org, 2016. [Online]. Available: <http://ceur-ws.org/Vol-1695/paper4.pdf>
- [49] O. Lassila, R. R. Swick, W. Wide, and W. Consortium, “Resource description framework (rdf) model and syntax specification,” 1998. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.44.6030>
- [50] J. Pérez, M. Arenas, and C. Gutierrez, “Semantics and complexity of SPARQL,” *ACM Trans. Database Syst.*, vol. 34, no. 3, pp. 16:1–16:45, 2009. [Online]. Available: <https://doi.org/10.1145/1567274.1567278>
- [51] W. Ali, M. Saleem, B. Yao, A. Hogan, and A. N. Ngomo, “A survey of RDF stores & SPARQL engines for querying knowledge graphs,” *VLDB J.*, vol. 31, no. 3, pp. 1–26, 2022. [Online]. Available: <https://doi.org/10.1007/s00778-021-00711-3>
- [52] A. Rossi, D. Barbosa, and et.al., “Knowledge graph embedding for link prediction: A comparative analysis,” *ACM Trans. Knowl. Discov. Data*, vol. 15, no. 2, pp. 14:1–14:49, 2021. [Online]. Available: <https://doi.org/10.1145/3424672>
- [53] Z. Wang, J. Zhang, J. Feng, and Z. Chen, “Knowledge graph embedding by translating on hyperplanes,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 28, no. 1, 2014.
- [54] X. Ge, Y. C. Wang, B. Wang, C.-C. J. Kuo *et al.*, “Knowledge graph embedding: An overview,” *APSIPA Transactions on Signal and Information Processing*, vol. 13, no. 1, 2024.

- [55] M. Galkin, M. Berrendorf, and C. T. Hoyt, “An open challenge for inductive link prediction on knowledge graphs,” *CoRR*, vol. abs/2203.01520, 2022. [Online]. Available: <https://doi.org/10.48550/arXiv.2203.01520>
- [56] T. Nguyen, Z. Liu, and Y. Fang, “Link prediction on latent heterogeneous graphs,” in *Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*. ACM, 2023, pp. 263–273. [Online]. Available: <https://doi.org/10.1145/3543507.3583284>
- [57] M. S. Schlichtkrull, T. N. Kipf, and e. a. Peter Bloem, “Modeling relational data with graph convolutional networks,” in *ESWC*, vol. 10843. Springer, 2018, pp. 593–607. [Online]. Available: https://doi.org/10.1007/978-3-319-93417-4_38
- [58] H. Aidan, B. Eva, and et.al., “Knowledge graphs,” *ACM Comput. Surv.*, vol. 54, no. 4, 2021. [Online]. Available: <https://doi.org/10.1145/3447772>
- [59] PyG. (2022) Torch geometric documentation. [Online]. Available: <https://pytorch-geometric.readthedocs.io/en/latest/index.html>
- [60] D. Zheng, C. Ma, M. Wang, and et.al., “DistDGL: Distributed Graph Neural Network Training for Billion-Scale Graphs,” in *IEEE/ACM, 10th*. IEEE, 2020.
- [61] Z. Wu, S. Pan, F. Chen, and et.al., “A comprehensive survey on graph neural networks,” *IEEE Trans. Neural Networks Learn. Syst.*, vol. 32, no. 1, pp. 4–24, 2021. [Online]. Available: <https://doi.org/10.1109/TNNLS.2020.2978386>
- [62] Q. Wang, Z. Mao, B. Wang, and L. Guo, “Knowledge graph embedding: A survey of approaches and applications,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 12, pp. 2724–2743, 2017.
- [63] D. Gordon. (2021) Fulltext search in neo4j. [Online]. Available: <https://neo4j.com/developer/kb/fulltext-search-in-neo4j/>
- [64] docs@stardog.com. Stardog documentation-machine learning. [Online]. Available: <https://docs.stardog.com/machine-learning#machine-learning>

- [65] Neo4j.Inc. (2022) Neo4j graph data science for machine learning. [Online]. Available: <https://neo4j.com/docs/graph-data-science/current/machine-learning/machine-learning/>
- [66] H. Chen, S. F. Sultan, and et.al., “Fast and accurate network embeddings via very sparse random projection,” in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, ser. CIKM ’19, 2019, p. 399–408. [Online]. Available: <https://doi.org/10.1145/3357384.3357879>
- [67] W. L. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in *NeurIPS*, 2017, pp. 1024–1034. [Online]. Available: <https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7e9bea9-Abstract.html>
- [68] I. F. Ilyas, T. Rekatsinas, V. Konda, J. Pound, X. Qi, and M. A. Soliman, “Saga: A platform for continuous construction and serving of knowledge at scale,” in *SIGMOD*, 2022, pp. 2259–2272. [Online]. Available: <https://doi.org/10.1145/3514221.3526049>
- [69] H. Zeng, H. Zhou, and et.al., “Graphsaint: Graph sampling based inductive learning method,” in *ICLR*, 2020, , GitHub Code: https://github.com/snap-stanford/ogb/blob/master/examples/nodeproppred/mag/graph_saint.py. [Online]. Available: <https://openreview.net/forum?id=BJe8pkHFwS>
- [70] M. Chen, W. Zhang, Y. Zhu, H. Zhou, Z. Yuan, C. Xu, and H. Chen, “Meta-knowledge transfer for inductive knowledge graph embedding,” in *Proceedings of the 45th International ACM SIGIR Conference*, ser. SIGIR ’22. Association for Computing Machinery, 2022, p. 927–937, , GitHub Code: <https://github.com/zjukg/MorsE>. [Online]. Available: <https://doi.org/10.1145/3477495.3531757>
- [71] M. Chen, Y. Zhang, X. Kou, and et.al., “r-gat: Relational graph attention network for multi-relational graphs,” *CoRR*, vol. abs/2109.05922, 2021. [Online]. Available: <https://arxiv.org/abs/2109.05922>
- [72] T. Y. Cheng, Q. Hu, Q. Xie, N. Trigoni, and A. Markham, “Meta-sampler: Almost-universal yet task-oriented sampling for point clouds,” *ECCV*, 2022. [Online]. Available: <https://doi.org/10.48550/arXiv.2203.16001>

- [73] O. Dovrat, I. Lang, and S. Avidan, “Learning to sample,” in *IEEE CVPR*, 2019, pp. 2760–2769. [Online]. Available: http://openaccess.thecvf.com/content_CVPR_2019/html/Dovrat_Learning_to_Sample_CVPR_2019_paper.html
- [74] Y. Xiao, K. Gong, P. Zhou, G. Zheng, X. Liang, and L. Lin, “Adversarial meta sampling for multilingual low-resource speech recognition,” in *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*. AAAI Press, 2021, pp. 14 112–14 120. [Online]. Available: <https://ojs.aaai.org/index.php/AAAI/article/view/17661>
- [75] Y. Lu, A. Chowdhery, S. Kandula, and S. Chaudhuri, “Accelerating machine learning inference with probabilistic predicates,” in *SIGMOD*, ser. SIGMOD ’18, 2018, p. 1493–1508. [Online]. Available: <https://doi.org/10.1145/3183713.3183751>
- [76] R. Bordawekar and O. Shmueli, “Enabling cognitive intelligence queries in relational databases using low-dimensional word embeddings,” *CoRR*, vol. abs/1603.07185, 2016. [Online]. Available: <http://arxiv.org/abs/1603.07185>
- [77] M. Schule, H. Lang, M. Springer, A. Kemper, T. Neumann, and S. Gunnemann, “In-database machine learning with sql on gpus,” in *SSDBM*, 2021. [Online]. Available: <https://doi.org/10.1145/3468791.3468840>
- [78] K. Awada, M. Y. Eltabakh, C. Tang, M. Al-Kateb, S. Nair, and G. Au, “Cost estimation across heterogeneous sql-based big data infrastructures in teradata intellisphere,” in *EDBT*, 2020.
- [79] S. Chaudhuri and K. Shim, “Optimization of queries with user-defined predicates,” *ACM Trans. Database Syst.*, vol. 24, no. 2, pp. 177–228, 1999. [Online]. Available: <https://doi.org/10.1145/320248.320249>
- [80] M. Serafini, “Scalable graph neural network training: The case for sampling,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 55, no. 1, pp. 68–76, 2021. [Online]. Available: <https://doi.org/10.1145/3469379.3469387>

- [81] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, “Measuring and relieving the over-smoothing problem for graph neural networks from the topological view,” in *IAAI*, 2020, pp. 3438–3445.
- [82] J. Liu, K. Kawaguchi, B. Hooi, Y. Wang, and X. Xiao, “Eiggn: Efficient infinite-depth graph neural networks,” in *NIPS*, vol. 34, 2021, pp. 18 762–18 773. [Online]. Available: <https://proceedings.neurips.cc/paper/2021/file/9bd5ee6fe55aeb673025dbcb8f939c1-Paper.pdf>
- [83] S. Bradley, A. Hax, A. Hax, and T. Magnanti, *Chapter 9: Integer Programming*. Addison-Wesley, 1977.
- [84] X. Zhou, C. Chai, G. Li, and J. Sun, “Database meets artificial intelligence: A survey,” *IEEE TKDE*, vol. 34, no. 3, pp. 1096–1116, 2022.
- [85] A. Mohamed, G. Abuoda, and et.al., “Rdframes: Knowledge graph access for machine learning tools,” *PVLDB*, vol. 13, no. 12, 2020. [Online]. Available: <http://www.vldb.org/pvldb/vol13/p2889-mohamed.pdf>
- [86] C. F. Draschner, C. Stadler, F. Bakhshandegan Moghaddam, J. Lehmann, and H. Jabeen, “Distrdf2ml-scalable distributed in-memory machine learning pipelines for rdf knowledge graphs,” in *Proceedings of the 30th ACM*, 2021, p. 4465–4474. [Online]. Available: <https://doi.org/10.1145/3459637.3481999>
- [87] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, “The case for learned index structures,” in *SIGMOD*. ACM, 2018, pp. 489–504. [Online]. Available: <https://doi.org/10.1145/3183713.3196909>
- [88] S. Hasan, S. Thirumuruganathan, J. Augustine, N. Koudas, and G. Das, “Multi-attribute selectivity estimation using deep learning,” *CoRR*, vol. abs/1903.09999, 2019. [Online]. Available: <http://arxiv.org/abs/1903.09999>
- [89] W. G. Pedrozo, J. C. Nievola, and D. C. Ribeiro, “An adaptive approach for index tuning with learning classifier systems on hybrid storage environments,” in *H AIS*, vol. 10870, 2018, pp. 716–729. [Online]. Available: https://doi.org/10.1007/978-3-319-92639-1_60

- [90] S. Fernandes and J. Bernardino, “What is bigquery?” *Proceedings of the 19th International Database Engineering & Applications Symposium*, 2015.
- [91] N. Makrynioti, R. Ley-Wild, and V. Vassalos, “sql4ml A declarative end-to-end workflow for machine learning,” *CoRR*, vol. abs/1907.12415, 2019. [Online]. Available: <http://arxiv.org/abs/1907.12415>
- [92] R. Bordawekar, B. Bandyopadhyay, and O. Shmueli, “Cognitive database: A step towards endowing relational databases with artificial intelligence capabilities,” vol. abs/1712.07199, 2017. [Online]. Available: <http://arxiv.org/abs/1712.07199>
- [93] C. Riccardo, P. Paolo, and T. Saravanan, “Creating embeddings of heterogeneous relational datasets for data integration tasks,” in *ACM SIGMOD*, USA, 2020, p. 1335–1349. [Online]. Available: <https://doi.org/10.1145/3318464.3389742>
- [94] J. Chen, T. Ma, and C. Xiao, “Fastgcn: Fast learning with graph convolutional networks via importance sampling,” in *6th International Conference on Learning Representations, ICLR*. OpenReview.net, 2018. [Online]. Available: <https://openreview.net/forum?id=rytstxWAW>
- [95] P. Yu, C. Fu, Y. Yu, C. Huang, Z. Zhao, and J. Dong, “Multiplex heterogeneous graph convolutional network,” in *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*. ACM, 2022, pp. 2377–2387. [Online]. Available: <https://doi.org/10.1145/3534678.3539482>
- [96] Q. Lv, M. Ding, and et. al., “Are we really making much progress?: Revisiting, benchmarking and refining heterogeneous graph neural networks,” in *KDD 21*. ACM, 2021, pp. 1150–1160. [Online]. Available: <https://doi.org/10.1145/3447548.3467350>
- [97] X. Wang, D. Bo, C. Shi, S. Fan, Y. Ye, and P. S. Yu, “A survey on heterogeneous graph embedding: Methods, techniques, applications and sources,” *IEEE Trans. Big Data*, vol. 9, no. 2, pp. 415–436, 2023. [Online]. Available: <https://doi.org/10.1109/TBDATA.2022.3177455>

- [98] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel, “Convolutional 2d knowledge graph embeddings,” in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18)*, 2018, pp. 1811–1818. [Online]. Available: <https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17366>
- [99] S. Yun, M. Jeong, S. Yoo, S. Lee, S. S. Yi, R. Kim, J. Kang, and H. J. Kim, “Graph transformer networks: Learning meta-path graphs to improve gnns,” *Neural Networks*, vol. 153, pp. 104–119, 2022. [Online]. Available: <https://doi.org/10.1016/j.neunet.2022.05.026>
- [100] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu, “Pathsim: Meta path-based top-k similarity search in heterogeneous information networks,” *Proc. VLDB Endow.*, vol. 4, no. 11, p. 992–1003, jun 2020. [Online]. Available: <https://doi.org/10.14778/3402707.3402736>
- [101] X. Liu, M. Yan, L. Deng, G. Li, X. Ye, and D. Fan, “Sampling methods for efficient training of graph convolutional networks: A survey,” *IEEE CAA J. Autom. Sinica*, vol. 9, no. 2, pp. 205–234, 2022. [Online]. Available: <https://doi.org/10.1109/JAS.2021.1004311>
- [102] J. Gasteiger, C. Qian, and S. Günnemann, “Influence-based mini-batching for graph neural networks,” in *Learning on Graphs Conference, LoG 2022, 9-12 December 2022, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 198, 2022, p. 9. [Online]. Available: <https://proceedings.mlr.press/v198/gasteiger22a.html>
- [103] Z. Hu, Y. Dong, K. Wang, and Y. Sun, “Heterogeneous graph transformer,” in *Proceedings of The Web Conference 2020*. New York, NY, USA: Association for Computing Machinery, 2020, p. 2704–2710. [Online]. Available: <https://doi.org/10.1145/3366423.3380027>
- [104] L. Yu, J. Shen, J. Li, and A. Lerer, “Scalable graph neural networks for heterogeneous graphs,” 2020. [Online]. Available: <https://arxiv.org/abs/2011.09679>
- [105] S. Xiao, S. Wang, Y. Dai, and W. Guo, “Graph neural networks in node classification: survey and evaluation,” *Machine Vision and Applications*, vol. 33, pp. 1–19, 2022. [Online]. Available: <https://link.springer.com/article/10.1007/s00138-021-01251-0>

- [106] M. Wang, L. Qiu, and X. Wang, “A survey on knowledge graph embeddings for link prediction,” *Symmetry*, vol. 13, no. 3, 2021. [Online]. Available: <https://www.mdpi.com/2073-8994/13/3/485>
- [107] H. Han, T. Zhao, and et.al., “Openhgnn: An open source toolkit for heterogeneous graph neural network,” in *CIKM*, 2022. [Online]. Available: <https://dl.acm.org/doi/pdf/10.1145/3511808.3557664>
- [108] J. Lin, A. Zhang, M. Lécuyer, J. Li, A. Panda, and S. Sen, “Measuring the effect of training data on deep learning predictions via randomized experiments,” in *Proceedings of the 39th International Conference on Machine Learning*, 2022, pp. 13 468–13 504. [Online]. Available: <https://proceedings.mlr.press/v162/lin22h/lin22h.pdf>
- [109] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, “Heterogeneous graph neural network,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*. ACM, 2019, pp. 793–803. [Online]. Available: <https://doi.org/10.1145/3292500.3330961>
- [110] Y. Rong, W. Huang, T. Xu, and J. Huang, “Dropege: Towards deep graph convolutional networks on node classification,” in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. [Online]. Available: <https://openreview.net/forum?id=Hkx1qkrKPr>
- [111] X. Bingbing, H. Shen, Q. Cao, Y. Liu, K. Cen, and X. Cheng, “Towards powerful graph neural networks: Diversity matters,” 2021. [Online]. Available: <https://openreview.net/references/pdf?id=5SP0-4MqUr>
- [112] Y. M. Omar and P. Plapper, “A survey of information entropy metrics for complex networks,” *Entropy*, vol. 22, no. 12, 2020. [Online]. Available: <https://www.mdpi.com/1099-4300/22/12/1417>
- [113] R. Andersen, F. R. K. Chung, and K. J. Lang, “Local graph partitioning using pagerank vectors,” in *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*

- 2006), 21-24 October 2006, Berkeley, California, USA, *Proceedings*. IEEE Computer Society, 2006, pp. 475–486. [Online]. Available: <https://doi.org/10.1109/FOCS.2006.44>
- [114] H. Abdallah, W. Afandi, and E. Mansour, “Demonstration of sparqlml: An interfacing language for supporting graph machine learning for rdf graphs,” *Proc. VLDB Endow.*, vol. 16, no. 12, p. 3974–3977, aug 2023. [Online]. Available: <https://doi.org/10.14778/3611540.3611599>
- [115] C. Weiss, P. Karras, and A. Bernstein, “Hexastore: sextuple indexing for semantic web data management,” *Proceedings of VLDB Endowment, (PVLDB)*, vol. 1, no. 1, pp. 1008–1019, 2008. [Online]. Available: <http://www.vldb.org/pvldb/vol1/1453965.pdf>
- [116] L. Ma, Z. Yang, Y. Miao, J. Xue, M. Wu, L. Zhou, and Y. Dai, “Neugraph: Parallel deep neural network computation on large graphs,” in *2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019*. USENIX Association, 2019, pp. 443–458. [Online]. Available: <https://www.usenix.org/conference/atc19/presentation/ma>
- [117] H. Liu, S. Lu, X. Chen, and B. He, “G3: When graph neural networks meet parallel graph processing systems on gpus,” *PVLDB (Proceedings of the VLDB Endowment), demo paper*, 2020. [Online]. Available: <https://dl.acm.org/doi/10.14778/3415478.3415482>
- [118] Z. Cai, Yan, Xiao, and et.al., “Dgcl: An efficient communication library for distributed gnn training,” in *Proceedings of the Sixteenth European Conference on Computer Systems*. New York, NY, USA: Association for Computing Machinery, 2021, p. 130–144. [Online]. Available: <https://doi.org/10.1145/3447786.3456233>
- [119] J. Peng, Z. Chen, and et.al., “Sancus: Staleness-aware communication-avoiding full-graph decentralized training in large-scale graph neural networks,” *Proc. VLDB Endow.*, vol. 15, no. 9, p. 1937–1950, 2022. [Online]. Available: <https://doi.org/10.14778/3538598.3538614>
- [120] R. Zhu, K. Zhao, H. Yang, W. Lin, C. Zhou, B. Ai, Y. Li, and J. Zhou, “Aligraph: a comprehensive graph neural network platform,” *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 2094–2105, 2019. [Online]. Available: <https://arxiv.org/abs/1902.08730>

- [121] O. Ferludin, A. Eigenwillig, M. Blais, and et.al., “TF-GNN: graph neural networks in tensorflow,” *CoRR*, vol. abs/2207.03522, 2022. [Online]. Available: <http://arxiv.org/abs/2207.03522>
- [122] J. Mohoney, R. Waleffe, H. Xu, T. Rekatsinas, and S. Venkataraman, “Marius: Learning massive graph embeddings on a single machine,” in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, 2021, pp. 533–549. [Online]. Available: <https://www.usenix.org/conference/osdi21/presentation/mohoney>
- [123] C. Wang, Q. Long, and et.al, “Biorag: A RAG-LLM framework for biological question reasoning,” *CoRR*, vol. abs/2408.01107, 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2408.01107>
- [124] X. He, Y. Tian, Y. Sun, N. V. Chawla, T. Laurent, Y. LeCun, X. Bresson, and B. Hooi, “G-retriever: Retrieval-augmented generation for textual graph understanding and question answering,” in *NeurIPS*, 2024. [Online]. Available: http://papers.nips.cc/paper_files/paper/2024/hash/efaf1c9726648c8ba363a5c927440529-Abstract-Conference.html
- [125] C. Mavromatis and G. Karypis, “GNN-RAG: Graph Neural Retrieval for Large Language Model Reasoning,” *arXiv preprint arXiv:2405.20139*, 2024.
- [126] L. Luo, Y.-F. Li, G. Haffari, and S. Pan, “Reasoning on graphs: Faithful and interpretable large language model reasoning,” in *International Conference on Learning Representations*, 2024.
- [127] J. Sun, C. Xu, L. Tang, S. Wang, C. Lin, Y. Gong, L. M. Ni, H. Shum, and J. Guo, “Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph,” in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. [Online]. Available: <https://openreview.net/forum?id=nnVO1PvbTv>
- [128] L. Luo, Z. Zhao, C. Gong, G. Haffari, and S. Pan, “Graph-constrained reasoning: Faithful reasoning on knowledge graphs with large language models,” in *Forty-second International Conference on Machine Learning*, 2025.

- [129] Y. Xu, S. He, J. Chen, Z. Wang, Y. Song, H. Tong, G. Liu, J. Zhao, and K. Liu, “Generate-on-graph: Treat LLM as both agent and KG for incomplete knowledge graph question answering,” in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Nov. 2024, pp. 18 410–18 430. [Online]. Available: <https://aclanthology.org/2024.emnlp-main.1023/>
- [130] A. Yang, A. Li, and et.al., “Qwen3 technical report,” *arXiv preprint arXiv:2505.09388*, 2025. [Online]. Available: <https://huggingface.co/Qwen/Qwen3-8B>
- [131] J. Lajus, L. Galárraga, and F. M. Suchanek, “Fast and exact rule mining with AMIE 3,” in *ESWC*, ser. Lecture Notes in Computer Science, vol. 12123. Springer, 2020, pp. 36–52. [Online]. Available: https://doi.org/10.1007/978-3-030-49461-2_3
- [132] A. Rossi, D. Barbosa, D. Firmani, A. Matinata, and P. Merialdo, “Knowledge graph embedding for link prediction: A comparative analysis,” *ACM Trans. Knowl. Discov. Data*, vol. 15, no. 2, pp. 14:1–14:49, 2021. [Online]. Available: <https://doi.org/10.1145/3424672>
- [133] S. Ott, C. Meilicke, and M. Samwald, “SAFRAN: an interpretable, rule-based link prediction method outperforming embedding models,” in *AKBC*, 2021. [Online]. Available: <https://doi.org/10.24432/C5MK57>
- [134] D. Thai, S. Ravishankar, I. Abdelaziz, and et.al., “CBR-iKB: A Case-Based Reasoning Approach for Question Answering over Incomplete Knowledge Bases,” *CoRR*, vol. abs/2204.08554, 2022. [Online]. Available: <https://doi.org/10.48550/arXiv.2204.08554>
- [135] B. Wang, T. Shen, G. Long, T. Zhou, Y. Wang, and Y. Chang, “Structure-augmented text representation learning for efficient knowledge graph completion,” in *WWW. ACM / IW3C2*, 2021, pp. 1737–1748. [Online]. Available: <https://doi.org/10.1145/3442381.3450043>
- [136] D. Shu, T. Chen, M. Jin, C. Zhang, M. Du, and Y. Zhang, “Knowledge graph large language model (KG-LLM) for link prediction,” in *ACML*, ser. Proceedings of Machine Learning Research, V. Nguyen and H. Lin, Eds., vol. 260. PMLR, 2024, pp. 143–158. [Online]. Available: <https://proceedings.mlr.press/v260/shu25a.html>

- [137] P. Jiang, L. Cao, and et.al., “KG-FIT: Knowledge Graph Fine-Tuning Upon Open-World Knowledge,” in *NeurIPS*, 2024. [Online]. Available: <https://arxiv.org/abs/2405.16412>
- [138] Y. Xia, J. Kim, and et.al., “Understanding the performance and estimating the cost of LLM fine-tuning,” in *IISWC*. IEEE, 2024, pp. 210–223.
- [139] Z. Zhao, E. Wallace, S. Feng, D. Klein, and S. Singh, “Calibrate before use: Improving few-shot performance of language models,” in *ICML*, ser. Proceedings of Machine Learning Research, vol. 139. PMLR, 2021, pp. 12 697–12 706. [Online]. Available: <http://proceedings.mlr.press/v139/zhao21c.html>
- [140] M. Yasunaga, H. Ren, A. Bosselut, P. Liang, and J. Leskovec, “QA-GNN: Reasoning with Language Models and Knowledge Graphs for Question Answering,” in *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2021.
- [141] Z. Zhu, X. Yuan, M. Galkin, L. A. C. Xhonneux, M. Zhang, M. Gazeau, and J. Tang, “A*Net: A Scalable Path-based Reasoning Approach for Knowledge Graphs,” in *NeurIPS*, 2023. [Online]. Available: <https://arxiv.org/abs/2206.04798>
- [142] S. Wu, S. Zhao, M. Yasunaga, K. Huang, K. Cao, Q. Huang, V. N. Ioannidis, K. Subbian, J. Zou, and J. Leskovec, “STaRK: Benchmarking LLM Retrieval on Textual and Relational Knowledge Bases,” in *Advances in Neural Information Processing Systems*, vol. 37. Curran Associates, Inc., 2024, pp. 127 129–127 153. [Online]. Available: <https://arxiv.org/abs/2404.13207>
- [143] D. Zhou, Y. Zhu, Y. He, J. Chen, E. Kharlamov, and S. Staab, “Evaluating knowledge graph based retrieval augmented generation methods under knowledge incompleteness,” 2025. [Online]. Available: <https://arxiv.org/abs/2504.05163>
- [144] W.-t. Yih, M. Richardson, C. Meek, M.-W. Chang, and J. Suh, “The value of semantic parse labeling for knowledge base question answering,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 201–206. [Online]. Available: <https://aclanthology.org/P16-2033/>

- [145] P. Trivedi, G. Maheshwari, M. Dubey, and J. Lehmann, “Lc-quad: A corpus for complex question answering over knowledge graphs,” in *International Semantic Web Conference*. Springer, 2017, pp. 210–218.
- [146] Y. Zhang, H. Dai, A. J. Smola, and L. Song, “Variational reasoning for question answering with knowledge graph,” in *AAAI*, 2018.
- [147] J. Baek, A. F. Aji, and A. Saffari, “Knowledge-augmented language model prompting for zero-shot knowledge graph question answering,” *CoRR*, vol. abs/2306.04136, 2023. [Online]. Available: <https://doi.org/10.48550/arXiv.2306.04136>
- [148] O. Hassanzadeh and M. P. Consens, “Linked Movie Data Base,” in *WWW-2009 Workshop on Linked Data on the Web*, ser. CEUR Workshop Proceedings, vol. 538. CEUR-WS.org, 2009. [Online]. Available: https://ceur-ws.org/Vol-538/ldow2009_paper12.pdf
- [149] B. Walsh, S. K. Mohamed, and V. Nováček, “BioKG: A Knowledge Graph for Relational Learning On Biological Data,” in *Proceedings of the 29th ACM Conference on Information Knowledge Management*. New York, NY, USA: Association for Computing Machinery, 2020, p. 3173–3180. [Online]. Available: <https://doi.org/10.1145/3340531.3412776>
- [150] M. Färber, C. Menne, and A. Harth, “A linked data wrapper for crunchbase,” *Semantic Web*, vol. 9, no. 4, pp. 505–515, 2018. [Online]. Available: <https://doi.org/10.3233/SW-170278>
- [151] J. Gu, X. Jiang, and et.al., “A survey on llm-as-a-judge,” *CoRR*, vol. abs/2411.15594, 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2411.15594>
- [152] S. Tan, S. Zhuang, and et.al, “JudgeBench: A Benchmark for Evaluating LLM-Based Judges,” in *ICLR*. OpenReview.net, 2025. [Online]. Available: <https://openreview.net/forum?id=G0dksFayVq>
- [153] DeepSeek-AI, A. Liu, and et.al, “Deepseek-v3 technical report,” 2025. [Online]. Available: <https://arxiv.org/abs/2412.19437>

- [154] C. Zheng, H. Zhou, and et.al., “Large language models are not robust multiple choice selectors,” in *ICLR*. OpenReview.net, 2024. [Online]. Available: <https://openreview.net/forum?id=shr9PXz7T0>
- [155] J. Tang, Z. Wang, S. Chen, S. Zhou, J. Chen, and J. Bu, “OpenGT: A Comprehensive Benchmark For Graph Transformers,” 2025. [Online]. Available: <https://arxiv.org/abs/2506.04765>

Appendix A

Chapter Appendices

A.1 KGTOSA

A.1.1 The KG-Ben Benchmark: datasets and GNN tasks

Our KG datasets include real KGs, such as MAG, DBLP, YAGO and WikiKG. The datasets are extracted from real KGs of different application domains: general-fact KGs (Yago¹ and ogbl-wikikg2²) and academic KGs (MAG³ and DBLP⁴). These KGs contain up to 42.4 million vertices, 400 million triples, and tens to thousands of node/edge types. Existing HGNN methods for LP tasks on large KGs require excessive computing resources. Hence, we also used YAGO3-10⁵, a smaller version of YAGO, and ogbl-wikikg2 [1], a dataset extracted from Wikidata. It contains 2.5M entities and 535 edge types. Tables A.1 and A.2 summarize the details of the used KGs and our defined NC and LP tasks, respectively. Our benchmark will be available for further study.

Our benchmark includes NC and LP tasks. In KGs, an NC task is categorized into single- or multi-label classifications, as defined in 4.2.1. Our benchmark followed existing benchmarks [1–3] and defined single-label NC tasks. We choose the accuracy metric for these NC tasks to evaluate the performance. The train-valid-test splits are either split using a logical predicate that depends on the task or stratified random split with 80% for training, 10% for validation and 10% for testing.

¹YAGO-4: <https://yago-knowledge.org/downloads/yago-4>

²ogbl-wikikg2: <https://ogb.stanford.edu/docs/linkprop/#ogbl-wikikg2>

³MAG-2020-05-29: <https://makg.org/rdf-dumps/>

⁴DBLP versions March 2022 and Jan, March, and May 2023: <https://dblp.org/rdf/release/>

⁵YAGO3-10: <https://paperswithcode.com/dataset/yago>

Table A.1: Our Benchmark Statistics. The number of nodes and edges (RDF triples) is in millions. The number of node types (n-type) and edge types (e-type) is tens to thousands.

KG-Dataset	#nodes	#edges	#n-type	#e-type
MAG-42M	42.4M	166M	58	62
YAGO-30M	30.7M	400M	104	98
DBLP-15M	15.6M	252M	42	48
ogbl-wikikg2	2.5M	17M	9.3K	535
YAGO3-10	123K	1.1M	23	37

Table A.2: A summary of our GNN tasks: Task Types (TT) are single-label node classification (NC) or missing entity link prediction (LP).

TT	Name	KG	Split	Ratio	Metric
NC	PV	MAG-42M	Time	84/9/7	Accuracy
NC	PD	MAG-42M	Time	87/8/5	Accuracy
NC	PC	YAGO-30M	Random	80/10/10	Accuracy
NC	CG	YAGO-30M	Random	80/10/10	Accuracy
NC	PV	DBLP-15M	Time	79/10/11	Accuracy
NC	AC	DBLP-15M	Time	80/10/10	Accuracy
LP	AA	DBLP-15M	Time	99/0.7/0.3	Hits@10
LP	PO	ogbl-wikikg2	Time	94/2.5/3.5	Hits@10
LP	CA	YAGO3-10	Random	99/0.5/0.5	Hits@10

Table A.2 summarizes the details of the split schema and ratio. Our benchmark includes two NC tasks per KG.

For LP tasks, KGs contain many edge types that vary in importance to a specific task. For example, a user from an academic background will be interested in the predicates (edge type) *discipline* and *affiliation* in Wikidata but not in predicates related to movies and films. Moreover, training HGNNs for an LP task on a large KG of many edge types is computationally expensive. Hence, we define our LP tasks for a specific predicate to enable the HGNN methods to train the models with our available computing resources: a Linux machine with 32 cores and 3TB RAM. We choose the Hits@10 metric to evaluate the task performance following SOTA methods [1, 2, 70]. The train-valid-test splits are either split using three versions of the KG based on time or randomly. The details of the LP tasks are summarized in Table A.2.

A.1.2 Node Classification Datasets:

MAG-42M Paper-Venue (PV): The dataset extracted from The MAG KG [35] to train a GNN model that predicts the publication venue of a paper. The target nodes are $736K$ papers, and the prediction labels are their associated venues. The set of paper is the same set in OGBN-MAG [1] dataset while including all neighbours nodes for a depth of 3 hops. The number of classes (distinct venues) is 349. The training set includes papers published until 2018, the validation set includes papers published in 2019, and the test set includes papers published in 2020. The number of node types is 58, and the number of edge types is 62. The number of edges is 166M, and the number of nodes is 42M, as shown in table A.1.

DBLP-15M Paper-Venue (PV): The dataset extracted from DBLP [34] to train a model that predicts the publication venue of a paper. For the set of labels (distinct venues), we choose the top 50 frequent venues. Any paper that does not belong to the top 50 is excluded. This task includes $1.2M$ papers. The training set includes papers published before 2020, the validation set includes papers published in 2020, and the test set includes papers published in 2021.

MAG-42M Paper-Discipline (PD): The dataset extracted from MAG [35] to train a model that predicts the discipline of a paper based on its authors and citations. The target nodes are $710K$ papers, and the prediction labels are their associated disciplines. The set of paper is the same set in OGBN-MAG [1] dataset while including all neighbours nodes for a depth of 3 hops. For the label set (distinct disciplines), we choose the top 50 frequent disciplines. We exclude any paper that does not belong to this set. The splitting criteria are the same as MAG-42M Paper-Venue (PV) dataset.

DBLP-15M Author Affiliation-Country (AC): The dataset extracted from the DBLP KG [34] to train a model that predicts the affiliation country of an author based on its collaboration history. The target vertices are $92K$ authors with type (*person*). The label set includes the top 50 frequent countries. The author nodes are randomly split into train-valid-test sets.

Yago4-30M KG (PlaceCountry (PC)): We used YAGO4 [27] to train a model that predicts the country of a place based on its neighbour places. There are $734K$ target vertices of type (*place*). The label set includes the top 200 frequent countries. The places nodes are randomly split into train-valid-test sets.

Yago-4 KG (Creative-work Genre (CG)): We also used YAGO4 [27] to train a model that predicts the genre of creative work. There are 128K target vertices of type (*Creative-Work*). The label set includes the top 50 frequent genres. The creative work nodes are randomly split.

A.1.3 Link Prediction Benchmark datasets:

Yago3-10 Connected Airports (CA): We defined an LP task on YAGO3 [98] to predict the missing triples $\langle v_t, p, ? \rangle$, where v_t is of type airport and p is the *isConnected* predicate. The *isConnected* triples in the original validation and testing subsets are used for model validation and testing.

ogbl-Wikikg2 Person Occupations (PO): The ogbl-wikikg2 dataset is used for an LP task that predicts the missing occupation links for a person (edge type *P106* in Wikidata). The dataset is split according to time with three versions of May, August, and November 2015 for training, validation and testing, respectively.

DBLP-15M Author PrimaryAffiliations (AA): We defined another LP task on DBLP [34] to predict the missing triples $\langle v_t, p, ? \rangle$, where v_t is of type author and p is the *PrimaryAffiliation* predicate. Our train, validate, and test sets included the DBLP versions of Jan, March, and May 2023, respectively.

A.2 Node classification Results

Figures A.1, A.2, and A.3 show the results for the six node classification tasks.

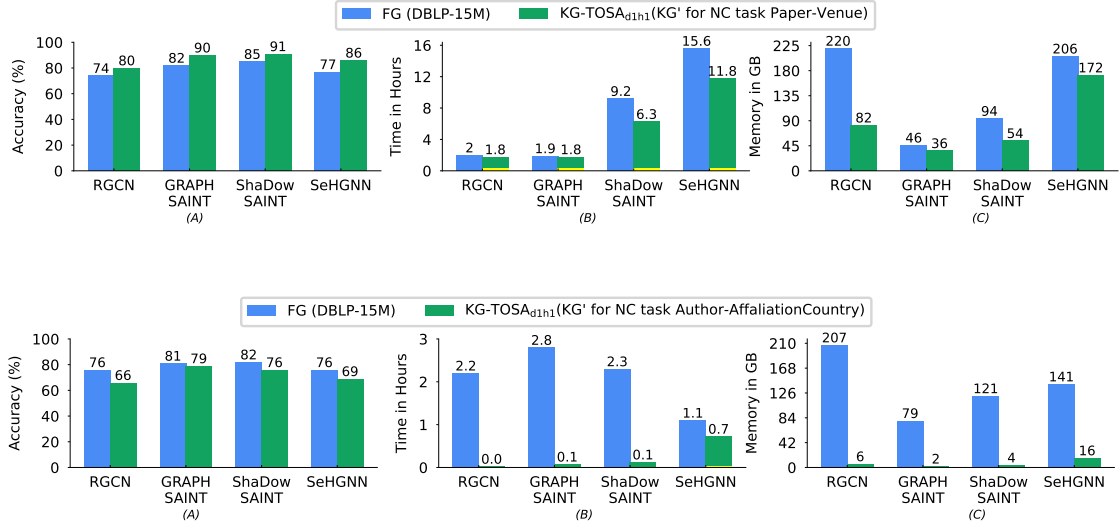


Figure A.1: Performance of RGCN, Graph-SAINT, ShaDow-SAINT, and SeHGNN in the NC tasks. (A) Accuracy (higher is better), (B) Training-Time (lower is better), (C) Training-Memory (lower is better). The figures on top show the results for the paper-venue classification task on DBLP. The figures at the bottom show the results for the Author-Affiliation_Country classification task. KGTOSA enables all methods to reduce memory and time while improving accuracy or keeping comparable scores, even with KG-TOSA's preprocessing time in yellow.

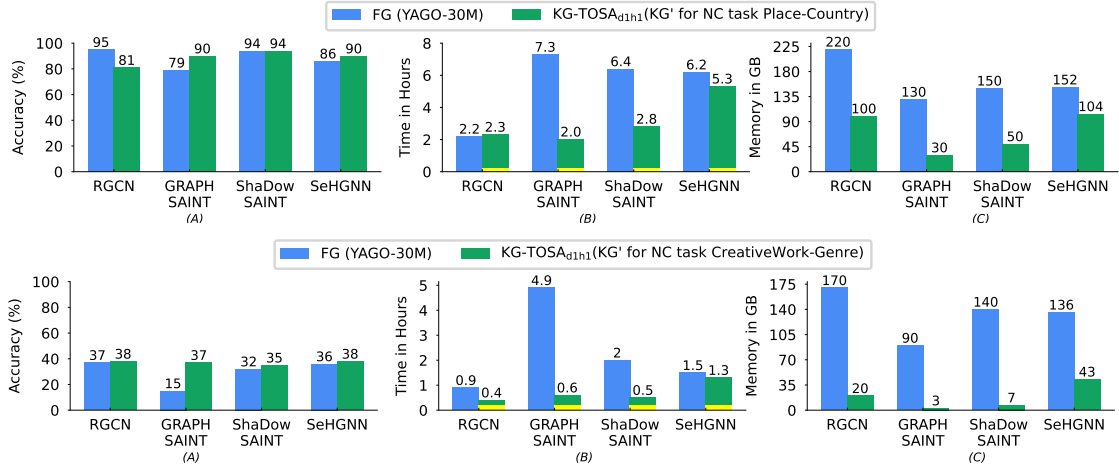


Figure A.2: Performance of RGCN, Graph-SAINT, ShaDow-SAINT, and SeHGNN in the NC tasks. (A) Accuracy (higher is better), (B) Training-Time (lower is better), (C) Training-Memory (lower is better). The figures on top show the results for the place-country classification task on YAGO. The figures at the bottom show the results for the CreativeWork-Genre classification task. KGTOSA enables all methods to reduce memory and time while improving accuracy, even with KG-TOSA's preprocessing time in yellow.

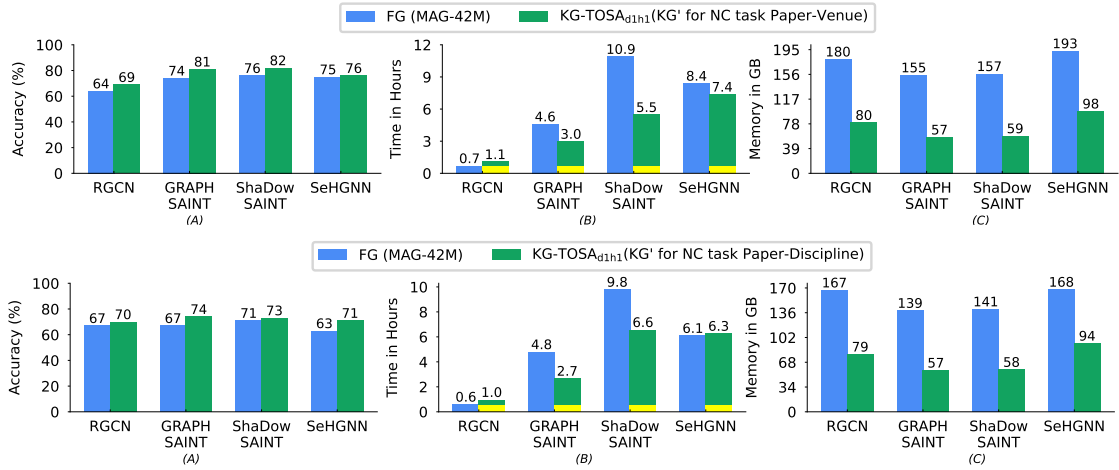


Figure A.3: Performance of RGCN, Graph-SAINT, ShaDow-SAINT, and SeHGNN in the NC tasks. (A) Accuracy (higher is better), (B) Training-Time (lower is better), (C) Training-Memory (lower is better). The figures on top show the results for the paper-venue classification task on MAG. The figures at the bottom show the results for the Paper-Discipline classification task on MAG. KGTOSA enables all methods to reduce memory, time and accuracy, even with KG-TOSA's preprocessing time in yellow.

A.3 Confusion Matrix and Classes Distribution

We performed stratified splitting to ensure a fair, unbiased test set labels distribution.

A.3.1 The MAG-PV labels distribution and normalized confusion matrix

The top-50 frequent labels distribution and the normalized confusion matrix of the MAG- PV task using KGTOSA_{d1h1} are shown in figures A.4 and A.5.

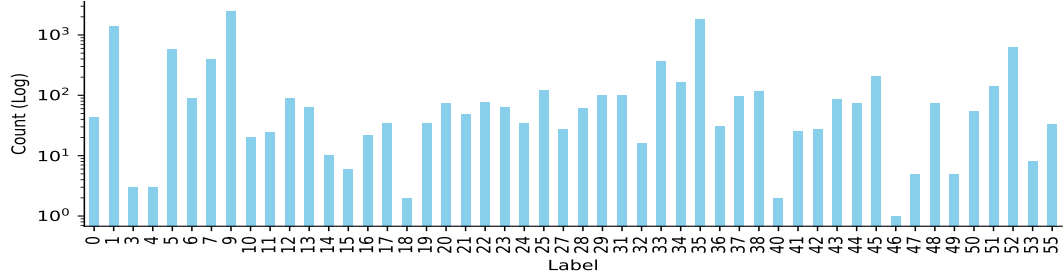


Figure A.4: The top 50 frequent labels distribution of the MAG-PV dataset.

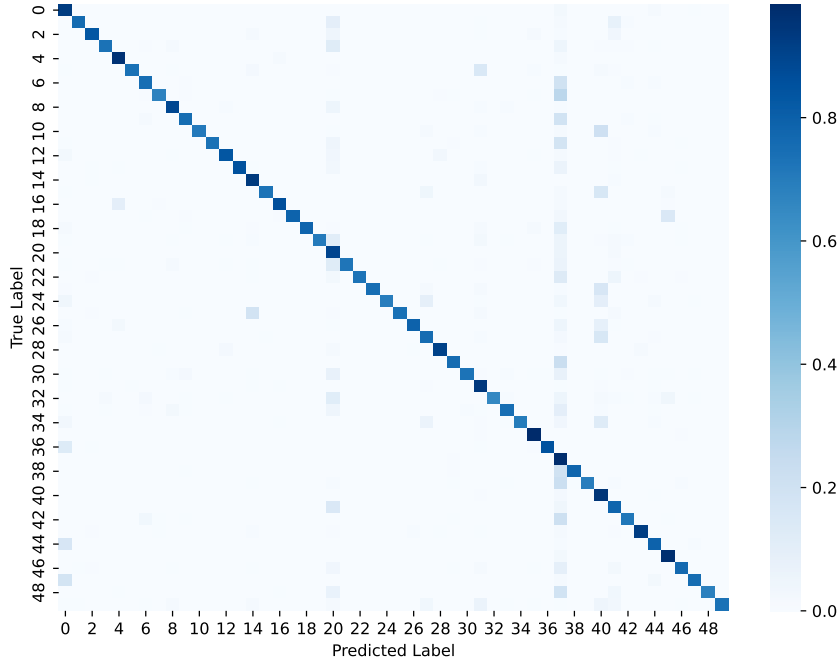


Figure A.5: The normalized confusion matrix of KGTOSA_{d1h1} on top-50 MAG-PV labels using Graph-SAINT. The model accuracy is 81% and the F1(Macro) score is 0.63.

A.3.2 The DBLP-PV labels distribution and normalized confusion matrix

The labels distribution and the normalized confusion matrix of the DBLP-PV task using KGTOSA_{d1h1} are shown in figures A.6 and A.7.

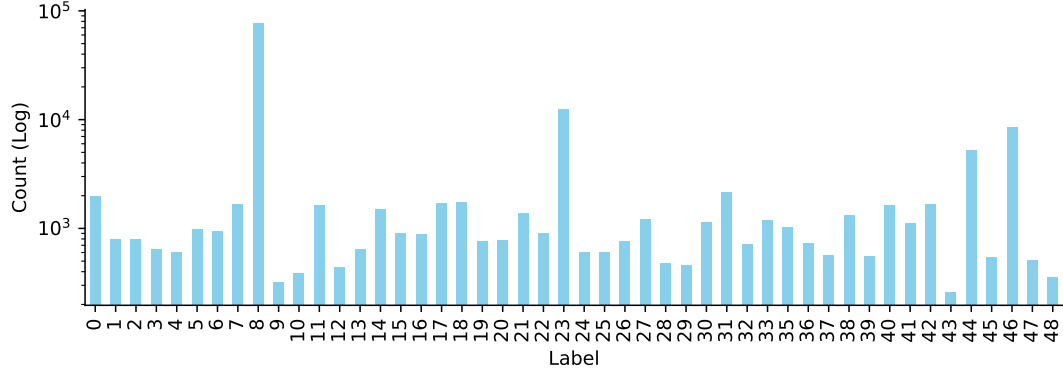


Figure A.6: The label distribution of the DBLP-PV dataset.

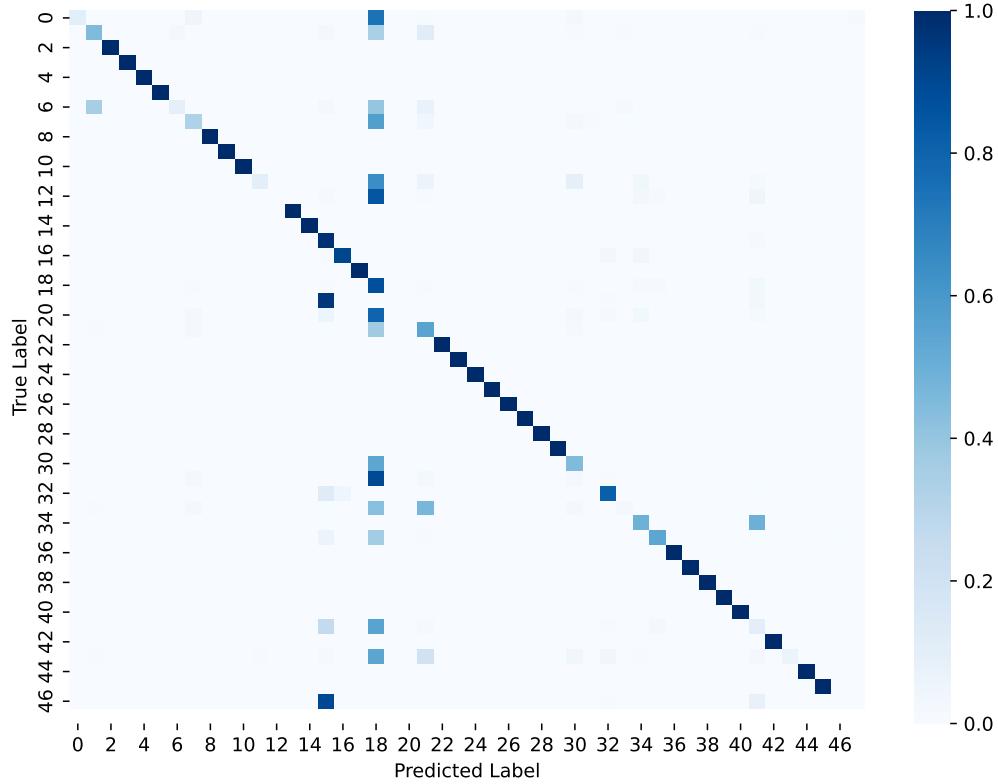


Figure A.7: The normalized confusion matrix of KGTOSA_{d1h1} on DBLP-PV using Graph-SAINT. The model accuracy is 90% and the F1(Macro) score is 0.66.

A.4 Training Convergence Rate Results

Figure A.8 shows the convergence rate of RGCN on the six-node classification tasks.

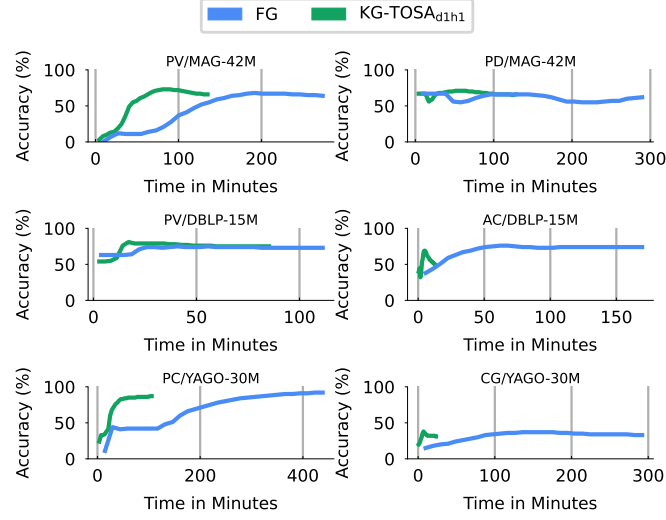


Figure A.8: Convergence rate analysis. RGCN while training the six NC tasks using the full graph (FG) and KG' extracted by KGTOSA. KGTOSA enables the GNN method to generalize faster with comparable accuracy.

A.5 Illustrative example subgraphs sampled using KGTOSA, URW, BRW, and IBS.

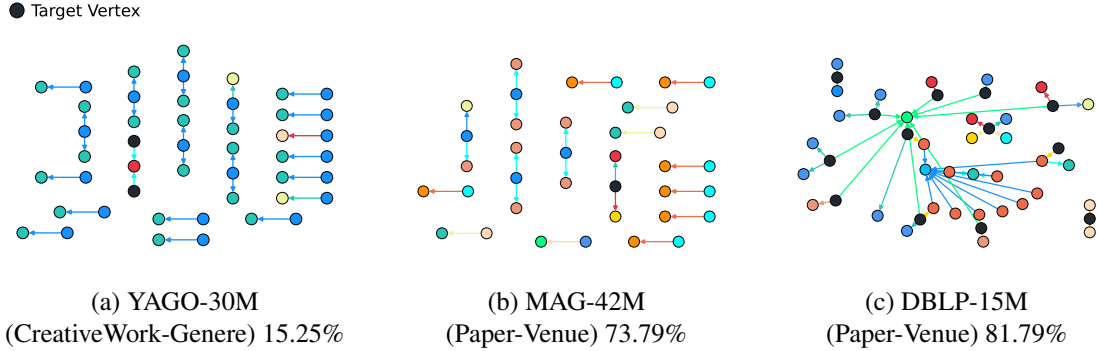
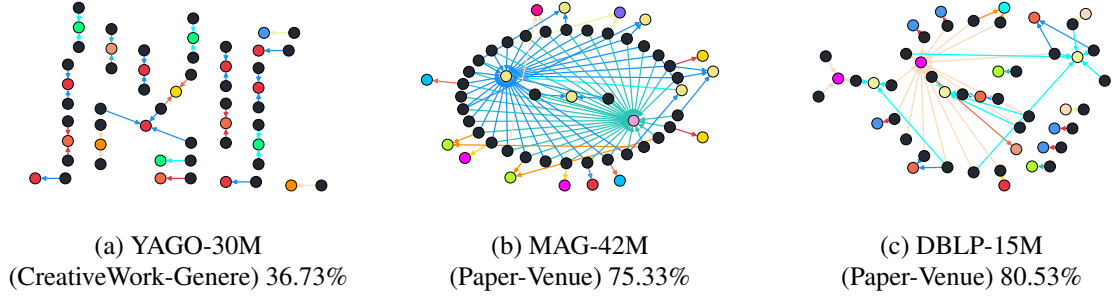
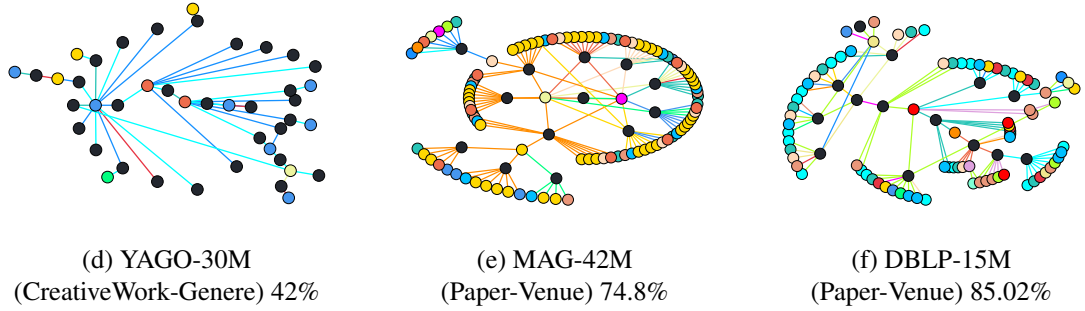


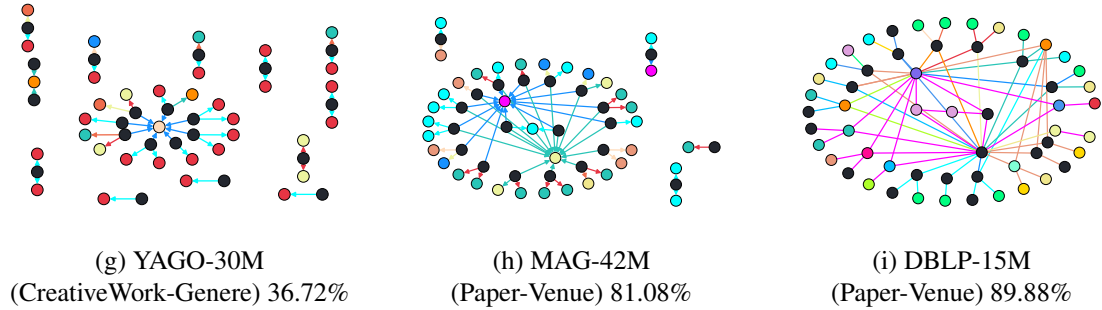
Figure A.9: Generated subgraphs using URW and our TOSG sampling approaches are presented. Rows 1 represent URW Each row includes subfigures sampled from YAGO-30M-CW, MAG-42-PV, and DBLP-15M-PV datasets, where the percentage represents the task accuracy score. Black vertices indicate target vertices $v_t \in \mathcal{V}_T$. Node and edge types are color-coded. The sampling parameters are $bs=20$ and $h=2$. URW sampling doesn't ensure sufficient representation of target vertices ($\mathcal{V}_T$) and may include target disconnected vertices.



Sampler 1: Sampled subgraphs based on the BRW.



Sampler 2: Sampled subgraphs based on the IBS.



Sampler 3: Sampled subgraphs based on the KGTOSA.

Figure A.10: Generated subgraphs using our TOSG sampling approaches are presented. Rows 1 to 3 represent BRW, IBS, and KGTOSA subgraphs, respectively. Each row includes subfigures sampled from YAGO-30M-CW, MAG-42-PV, and DBLP-15M-PV datasets where the percentage represents the task accuracy score. Black vertices indicate target vertices $v_t \in \mathcal{V}_T$. Node and edge types are color-coded. The sampling parameters are $bs=20$ and $h=2$. The baselines BRW and IBS, and KGTOSA, achieve a higher ratio of target vertices connected to non-target vertices, leading to increased diversity in node/edge types. URW sampling, however, doesn't ensure sufficient representation of target vertices ($\mathcal{V}_T$) and may include target disconnected vertices.

A.6 GLOWQA

A.6.1 GLOW Algorithm

Algorithm 4 GETPROMPT: Generate GLOW Prompt from Input OWA Question over KG

Require: Q the user input question, $GLOW_v$ The system variation .

```
1: function GETPROMPT( $Q, GLOW_v$ )
2:    $Q_n, Q_e, KG \leftarrow \text{entityExtraction}(Q)$  ▷ extract the question's node and edge
3:    $KG_{Sc} \leftarrow \text{getKGScheme}(KG)$  ▷ Load the KG schema triples
4:    $v_t, e_t \leftarrow \text{ER-Linking}(Q, KG_{Sc}, Q_n, Q_e)$  ▷ Link the question's node and edge crossponding KG's
   node and edge URIs
5:    $\mathcal{R}_q \leftarrow \text{TextToSPARQL}(KG_{Sc}, v_t)$  ▷ generate  $\mathcal{R}$  equivalent SPARQL query
6:    $\mathcal{R}_{Triples} \leftarrow \text{execSPARQL}(\mathcal{R}_q, KG)$  ▷ Execute the  $\mathcal{R}$  SPARQL Query
7:    $L \leftarrow \text{getPossibleLabels}(v_t, e_t)$  ▷ Extract the set of possible labels
8:    $\mathcal{R} \leftarrow \text{RC-Serialization}(\mathcal{R}_{Triples})$  ▷ Serialize the  $\mathcal{R}$  triples into text.
9:    $GNN_{Ans} \leftarrow \text{GNNPredict}(v_t, e_t)$  ▷ predict the top-K GNN answers for  $v_t$  and  $e_t$ 
10:   $P \leftarrow \text{getPrompt}(Q, v_t, e_t, L, \mathcal{R}, GNN_{Ans}, GLOW_v)$  ▷ generate the GLOW's variation prompt
11:  return  $P$ 
12: end function
```

A.6.2 GLOW Prompt Examples

User Prompt:

Predict the chemical kingdom for the drug Yohimbine from the BioKG knowledge graph.

Answer:

Question Understanding Prompt:

<system>: You are an expert Entity-Extraction NLP system.

<user>: Given the following question, identify 1-The question main entity type, 2- the main entity, 3- the prediction label and

4- the KG name.

Question: { }

Answer:

1-Question main entity: 2-Main Entity:

3-Prediction label: 4-KG name:

Entity/Relation Linking Prompt:

<system>: You are an expert knowledge graph entity-relation Linking NLP system.

<user>: Given the following KG Schema in the basic graph pattern CSV format, one predicate per line:
node type, relation, node type.

KG Schema:

{ }

1- What is the node type in the schema that corresponds to {main entity}? Return only the name.

2- Choose from the schema the BGP (node type, relation, node type) that describes the {main entity type} value {main entity}. Return only the BGP.

3- Choose from the schema the BGP (node type, relation, node type) that describe the {main entity type } {prediction label}.

Return only the BGP.

Answer:

1- 2- 3-

Question To SPARQL Prompt:

<system>: You are an expert text-To-SPARQL translation system.

<user>: Given the following KG Schema in the basic graph pattern CSV format, one predicate per line:
node type, relation, node type.

KG Schema:

{ }

Write a SPARQL query that selects the {main entity type} that satisfy the following BGPs.

1- {question node type} ,[label/name/titile] ,{question node}

2- {Other BGPs}

graph prefix: {KG Prefix}

Answer: SPARQL Query

Do not return any explanation or reasoning details.

SPARQL Query Example:

PREFIX biokg: <http://www.biokg.com/>

SELECT ?drug as ?vt ?kingdom as ?vl

WHERE {

VALUES ?name { "Yohimbine" }

?drug biokg:NAME ?name .

?drug biokg:KINGDOM ?kingdom .

}

Basic Prompt:

<system>: You are an expert open world question answer system.

<user>: Predict the {**Prediction Label**} for the following {**question node type**} from {**KG details**} knowledge graph.

- Do not return any context or analysis.
- Help: The possible list of {**Prediction Label Type**}s are: [{ **Labels List**}]
- Question: {**question node**}

Answer

To generate an instance of this template, replace the question node type, KG, and prediction label with values from one of the queries in table [A.3](#). **An example prompt for OWA-Q #1:**

<system>: You are an expert open world question answer system.

<user>: Predict the **Kingdom** for the following **Drug** from the **BioKG** , a **Biomedical** knowledge graph.

- Do not return any context or analysis.
- Help: The list of **Kingdoms** are: [Organic,Non-Organic]
- Question: **Durg**: Yohimbine

Answer:

GLOW-Bench-G Prompt:

<system>: You are an expert open world question answer system.

<user>: Predict the **Kingdom** for the following **Drug** from the **BioKG** , a **Biomedical** knowledge graph.

- Do not return any context or analysis.
- Help: The possible list of **Kingdoms** are: [**Organic,Non-Organic**]
- Question: **Durg**: and its associated triples. Capsaicin: [("DDI", "DB13677"), ..]

Answer:

GLOW-Bench-N Prompt:

<system>: You are an expert open world question answer system.

<user>: Predict the **Kingdom** for the following **Drug** from the **BioKG** , a **Biomedical** knowledge graph.

- Do not return any context or analysis.

- Help: The possible list of **Kingdoms** are: [**Organic,Non-Organic**]

- Verify the following list of GNN Answers: [Organic,Non-Organic, ..]

- Question: **Durg**: Yohimbine

Answer:

GLOW-Bench-GN Prompt:

<system>: You are an expert open world question answer system.

<user>: Predict the **Kingdom** for the following **Drug** from the **BioKG** , a **Biomedical** knowledge graph.

- Do not return any context or analysis.

- Help: The possible list of **Kingdoms** are : [**Organic,Non-Organic**]

- Verify the following GNN Answer: [Organic]

- Question: **Durg** and its associated triples. Capsaicin: [("DDI", "DB13677"), ..]

Answer:

A.6.3 LLM as-a-Judge Prompt

<system>: You are an expert LLM-as-a-Judge system.

<user>: Given the following list of predicted and true pairs of values.

-Rank the predicted value against the true value using two metrics.

1- Exact Match Rule: you compare the two strings after normalization and remove any special characters.
report 1 if both values are literally and semantically equal and 0 otherwise.

2- Hierarchical/Categorical Match Rule: report 1 if the predicted value is under a subcategory or hierarchically belongs to the true value or is a synonym and 0 otherwise.

- Example:

List of pairs: [[music, art], [painter, artist],[football player, soccer player], [lawyer, judge], [lawyer, player]]

Answer: [[0,1],[0,1],[1,1],[0,1],[0,0]]

- Question:

-List of pairs: {**ListOfPairs**}

-Note: refine each pair and return Answer for exactly {**length(ListOfPairs)**} pairs without explanation.

-Finally: make sure you return only {**length(ListOfPairs)**} pair of answers.

Answer:

A.6.4 Full Details of Our GLOW-Bench

Table A.3: Our GLOW-Bench benchmark characterizes each Open-World Question Template (OW-QT) across four KG-based features: (1) Knowledge Domain (KD): Domain-Specific (DS), Entertainment (E), or Generic (G); (2) Target Entity Type (ET); (3) Reasoning Hops (RH); and (4) Multiple Choice Count (MCC).

OW-QT	KG	KD	Target ET	RH	#Class	MCC	Label Type	Reasoning Path
1	BioKG	DS	Drug	1	2	2–4	Kingdom	Kingdom
2		DS	Drug	1	13	8–16	Superclass	Superclass
3		DS	Drug	2	29	16–32	Class	RelatedPubMed → Class
4		DS	Protein	1	18	16–32	SPECIES	SPECIES
5		DS	Protein	2	15	8–16	R.Keyword	RelatedPubMed → R.Keyword
6		DS	Protein	1	4	2–4	Family	Family
7	CrunchBase	DS	Investor	1	12	8–16	Country	Country
8		DS	Investor	1	14	8–16	InvestRegion	InvestRegion
9		DS	Investor	1	6	4–8	PositionTitle	PositionTitle
10		DS	Investor	1	11	8–16	Company	Company
11	LinkedMDB	E	Film	1	15	8–16	Language	Language
12		E	Film	1	27	16–32	Country	Country
13		E	Film	2	39	32+	Producer	Sequel → Producer
14		E	Film	2	7	4–8	Genre	Sequel → Genre
15		E	Film	1	28	16–32	Subject	Subject
16	YAGO4	G	CreativeW	2	13	8–16	ProdComp	byArtist → ProdComp
17		G	CreativeW	2	14	8–16	Publisher	isBasedOn → Publisher
18		G	CreativeW	1	25	16–32	PublishLang	Author
19		G	CreativeW	2	11	8–16	Genre	ProdComp → Genre
20		G	CreativeW	1	6	4–8	Country	Country
21		G	Person	1	3	2–4	GivenAward	GivenAward
22		G	Person	1	91	32+	Nationality	Nationality
23		G	Person	1	4	4–8	GraduateOfOrg	GraduateOfOrg
24		G	Person	1	102	32+	Occupation	Occupation
25		G	Person	1	8	4–8	SpokenLang	SpokenLang

A.6.5 The GNN Training/Inference Technical Details.

- For each question pattern, we train a node classification GNN using GraphSAINT with RGCN as GNN convolutional layer to support heterogeneous KG subgraphs. The initial node embeddings are iteratively aggregated with embeddings received from neighboring nodes connected to a specific relation until the embeddings of all nodes converge. The final embedding of a

v_t , our main question entity, is obtained through two aggregations: an outer aggregation over each relation type and an inner aggregation over neighbouring nodes $\mathcal{RC}$ of a specific relation and defined by RGCN [57] as follows:

$$h_i^{(l+1)} = \sigma\left(\sum_{r \in \mathcal{R}} \sum_{j \in N_i^r} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)}\right) \quad (8)$$

where l is an RGCN layer, $h_j^{(l+1)}$ is the hidden embedding of node j at layer $l + 1$, σ is element-wise activation function, N_i^r denotes the set of neighbour indices of node i under relation $r \in \mathcal{R}$, $c_{i,r}$ is a normalization constant that can either be learned or chosen in advance (such as $c_{i,r} = |N_i^r|$), $W_r^{(l)}$ is the weight matrix for relation r at layer l , and W_0^l is the initial weight matrix at layer l .

- To construct the training set, we exclude the GLOW-Bench question entity nodes from the training set and extract 1-hop in and out neighbor nodes subgraphs using the KGTOSA sampler [41]. This ensures scalability to large KGs and yields task-relevant and diverse subgraphs for effective training.
- GraphSAINT Training HyperParameters: We set the input dimension ($D = 128$), hidden channels dimension=64, and number of layers ($L = 2$) of our GNN module with dropout rate 0.5 applied to each layer. We train the model with the Adam optimizer, learning rate from 5e-4, 5e-3, 1e-3, 2e-3.
- Subgraph Sampling Hyperparameters: batch-size=20000, walk-length=2, num-steps=10.
- At inference time, the node classification model M is resolved using the KG name, the question’s main entity type, and the target edge. The model M predicts the top-k answer classes based on the Log-Likelihood for each class and feeds them to the LLM prompt. The trained GNN model is hosted via an inference API. At runtime, this API returns the top-K predicted answer nodes, which are then passed to the GLOW pipeline as supportive evidence for the LLM’s reasoning.

A.6.6 The OWA-QA Detailed Results:

Table A.4: The detailed accuracy (%) of GLOW compared to baseline models on existing question answering datasets from [4] and our developed benchmark GLOW-Bench. All results below use Qwen3-8B as the underlying LLM. Best results are marked in **bold**.

LLM	Dataset	RH	AskGNN	GraphSAINT	LLM-Only	GCR	GLOW-L	GLOW-G	GLOW-N	GLOW-GN
Qwen3:8b	Arxiv2023	1	62	41	31	4	59	64	66	81
	ogbn-arxiv	1	70	59	12	14	63	71	60	77
	ogbn-product	1	29	45	34	5	41	44	51	57
	drug-superclass	1	88	84	29	3	50	77	85	92
	drug-kingdom	1	98	96	88	94	91	98	97	96
	protein-SPECIES	1	75	89	8	16	19	71	91	92
	protein-FAMILY	1	67	70	9	13	35	39	69	71
	film-country	1	55	55	40	4	40	60	57	55
	film-subject	1	77	81	15	7	50	30	82	80
	film-language	1	87	83	70	13	73	80	85	86
	person-nationality	1	86	89	27	38	45	89	91	90
	parson-graduateOfOrg	1	51	53	3	25	27	36	54	55
	person-occupation	1	48	46	7	12	15	38	46	53
	person-spokenLang	1	86	83	70	61	83	89	83	85
	person-givenAward	1	93	91	7	50	75	93	93	93
	CWork-PublishedLang	1	75	80	50	60	55	55	85	95
	CWork-country	1	72	51	64	43	62	75	48	79
	person-title	1	79	72	5	41	63	47	77	83
	person-Company	1	97	95	3	93	30	21	96	97
	person-InvestmentRegion	1	98	94	7	46	95	87	96	100
	person-InvestmentCountry	1	98	95	38	14	94	55	97	97
	drug-class	2	46	68	11	15	33	45	70	73
	protein-keyword	2	35	42	5	7	33	53	44	57
	film-genre	2	54	51	50	25	60	62	51	52
	film-producer	2	35	21	7	7	14	24	21	28
	CWork-ProductionCompany	2	20	22	10	29	21	31	25	28
	CWork-Genere	2	20	22	16	7	22	33	19	27
	CWork-publisher	2	31	28	33	17	58	62	27	32

Table A.5: The detailed accuracy (%) of GLOW compared to baseline models on existing question answering datasets from [4] and our developed benchmark GLOW-Bench. All results below use GPT-4o-Mini [5] as the underlying LLM. Best results are marked in **bold**.

LLM	Dataset	RH	AskGNN	GraphSAINT	LLM-Only	GLOW-L	GLOW-G	GLOW-N	GLOW-GN
GPT-4o-Mini	Arxiv2023	1	N/A	41	35	58	59	48	62
	ogbn-arxiv	1	N/A	41	35	58	59	48	67
	ogbn-product	1	N/A	45	N/A	N/A	N/A	N/A	N/A
	drug-superclass	1	N/A	84	8	47	57	73	87
	drug-kingdom	1	N/A	96	31	100	100	98	99
	protein-SPECIES	1	N/A	89	6	10	77	92	93
	protein-FAMILY	1	N/A	70	11	53	43	69	72
	film-country	1	N/A	55	47	47	71	57	58
	film-subject	1	N/A	81	15	73	38	76	75
	film-language	1	N/A	83	80	80	80	86	88
	person-nationality	1	N/A	89	43	56	89	92	87
	parson-graduateOfOrg	1	N/A	53	19	22	33	55	57
	person-occupation	1	N/A	46	7	23	30	46	52
	person-spokenLang	1	N/A	83	70	81	91	83	89
	person-givenAward	1	N/A	91	6	68	96	93	91
	CWork-PublishedLanguage	1	N/A	80	65	60	65	83	87
	CWork-country	1	N/A	51	62	64	77	48	83
	person-title	1	N/A	72	5	38	55	74	77
	person-company	1	N/A	95	3	6	30	96	98
	person-InvestmentRegion	1	N/A	94	9	7	96	100	96
	person-InvestmentCountry	1	N/A	95	41	44	100	97	97
	drug-class	2	N/A	68	8	27	45	69	67
	protein-keyword	2	N/A	42	7	28	44	40	45
	film-genre	2	N/A	51	68	65	70	50	53
	film-producer	2	N/A	21	20	17	20	21	31
	CWork-ProductionCompany	2	N/A	22	10	28	42	25	32
	CWork-Genre	2	N/A	22	27	27	72	19	44
	CWork-publisher	2	N/A	28	62	70	79	25	75

Table A.6: The detailed accuracy (%) of GLOW compared to baseline models on existing question answering datasets from [4] and our developed benchmark GLOW-Bench. All results below use DeepSeek-V3 [6] as the underlying LLM. Best results are marked in **bold**.

LLM	Dataset	RH	AskGNN	GraphSAINT	LLM-Only	GLOW-L	GLOW-G	GLOW-N	GLOW-GN
DeepSeek-V3	Arxiv2023	1	N/A	41	17	50	57	61	73
	ogbn-arxiv	1	N/A	59	58	60	62	63	65
	ogbn-product	1	N/A	45	N/A	N/A	N/A	N/A	N/A
	drug-superclass	1	N/A	84	3	32	48	73	86
	drug-kingdom	1	N/A	96	9	88	100	98	96
	protein-SPECIES	1	N/A	89	2	8	79	92	93
	protein-FAMILY	1	N/A	70	21	76	87	69	81
	film-country	1	N/A	55	57	65	70	57	71
	film-subject	1	N/A	81	23	80	38	80	83
	film-language	1	N/A	83	81	86	86	86	93
	person-nationality	1	N/A	89	72	81	87	91	93
	parson-graduateOfOrg	1	N/A	53	32	48	43	52	54
	person-occupation	1	N/A	46	19	34	34	46	53
	person-spokenLang	1	N/A	83	79	79	82	83	85
	person-givenAward	1	N/A	91	3	56	58	93	94
	CWork-PublishedLanguage	1	N/A	80	47	50	50	84	90
	CWork-country	1	N/A	51	68	75	87	48	79
	person-title	1	N/A	72	41	41	63	74	77
	person-company	1	N/A	95	3	24	33	96	97
	person-InvestmentRegion	1	N/A	94	7	21	100	89	100
	person-InvestmentCountry	1	N/A	95	44	58	100	97	100
	drug-class	2	N/A	68	7	21	39	69	70
	protein-keyword	2	N/A	42	3	40	59	49	52
	film-genre	2	N/A	51	53	60	58	50	62
	film-producer	2	N/A	21	10	30	50	14	34
	CWork-ProductionCompany	2	N/A	22	28	53	57	25	53
	CWork-Genre	2	N/A	22	18	22	27	16	28
	CWork-publisher	2	N/A	28	70	79	87	16	70

Table A.7: The detailed accuracy (%) of GLOW compared to baseline models on existing question answering datasets from [4] and our developed benchmark GLOW-Bench. All results below use DeepSeek-R1-Distill-Qwen:7B [7] as the underlying LLM. Best results are marked in **bold**.

LLM	Dataset	RH	AskGNN	GraphSAINT	LLM-Only	GCR	GLOW-L	GLOW-G	GLOW-N	GLOW-GN
DeepSeek-R1-Distill-Qwen:7B	Arxiv2023	1	48	41	24	7	44	55	59	67
	ogbn-arxiv	1	50	59	13	15	30	30	61	63
	ogbn-product	1	19	45	27	12	36	38	43	51
	drug-superclass	1	53	84	20	5	20	21	69	68
	drug-kingdom	1	98	96	60	90	90	100	98	100
	protein-SPECIES	1	71	89	1	12	4	62	88	90
	protein-FAMILY	1	62	70	7	23	30	35	68	69
	film-country	1	56	55	17	34	44	52	57	64
	film-subject	1	78	81	8	8	11	34	76	82
	film-language	1	86	83	73	32	73	66	86	88
	person-nationality	1	85	89	22	41	30	79	91	89
	parson-graduateOfOrg	1	50	53	5	17	19	36	53	54
	person-occupation	1	47	46	3	12	15	31	46	54
	person-spokenLang	1	82	83	35	56	41	79	81	83
	person-givenAward	1	89	91	8	23	50	70	90	92
	CWork-PublishedLanguage	1	71	80	50	44	50	55	77	81
	CWork-country	1	62	51	37	42	41	70	48	66
	person-title	1	73	72	5	51	44	52	80	82
	person-company	1	93	95	3	90	30	19	96	92
	person-InvestmentRegion	1	92	94	3	33	64	96	100	100
	person-InvestmentCountry	1	93	95	11	56	72	94	97	97
	drug-class	2	41	68	4	13	15	24	69	71
	protein-keyword	2	31	42	3	8	18	22	32	38
	film-genre	2	53	51	35	18	50	56	50	57
	film-producer	2	30	21	7	9	7	7	14	24
	CWork-ProductionCompany	2	18	22	10	23	17	21	25	32
	CWork-Genere	2	17	22	11	17	11	16	16	27
	CWork-publisher	2	25	28	12	23	20	40	16	26

Table A.8: The detailed accuracy (%) of GLOW compared to baseline models on existing question answering datasets from [4] and our developed benchmark GLOW-Bench. All results below use IBM Granite-3.3-8B-instruct [8] as the underlying LLM. Best results are marked in **bold**.

LLM	Dataset	RH	AskGNN	GraphSAINT	LLM-Only	GCR	GLOW-G	GLOW-N	GLOW-GN
Granite-3.3-8B-Instruct	Arxiv2	1	63	41	14	14	32	63	77
	ogbn-arxiv	1	66	59	5	21	31	55	73
	ogbn-product	1	30	45	22	13	45	52	54
	drug-superclass	1	86	84	0	9	22	73	71
	drug-kingdom	1	97	96	8	98	95	98	99
	protein-SPECIES	1	75	89	2	18	45	93	90
	protein-FAMILY	1	67	70	0	33	36	69	72
	film-country	1	56	56	48	60	66	58	55
	film-subject	1	76	81	12	11	27	81	85
	film-language	1	84	83	87	46	80	87	88
	person-nationality	1	86	89	52	88	79	91	92
	parson-graduateOfOrg	1	52	53	6	14	31	56	57
	person-occupation	1	47	46	8	23	23	46	50
	person-spokenLang	1	85	83	48	75	52	83	88
	person-givenAward	1	93	92	3	25	63	91	94
	CWork-PublishedLanguage	1	81	80	45	35	80	81	94
	CWork-country	1	72	51	58	69	75	59	69
	person-title	1	78	72	14	55	64	80	83
	person-company	1	97	96	3	96	42	96	97
	person-InvestmentRegion	1	99	95	4	29	82	98	100
	person-InvestmentCountry	1	99	96	39	69	78	96	97
	drug-class	2	47	68	0	18	27	70	70
	protein-keyword	2	35	42	2	10	36	9	38
	film-genre	2	54	51	35	15	52	54	57
	film-producer	2	35	21	5	5	30	23	29
	CWork-ProductionCompany	2	22	22	7	18	21	25	32
	CWork-Genere	2	21	22	16	22	50	17	33
	CWork-publisher	2	32.7	28	33	46	58	27	36