Towards a GML-Enabled Knowledge Graph Platform

Hussein Abdallah
Concordia University,
hussein.abdallah@concordia.ca

Essam Mansour
Concordia University,
essam.mansour@concordia.ca

***Abstract*—This vision paper proposes KGNet, a graph machine learning (GML) enabled RDF engine. KGNet extends existing RDF engines with GML as a service to automate the training of GML models on KGs. In KGNet, we maintain from the metadata of trained models a transparent RDF graph associated with the target knowledge graph (KG) to enable SPARQL queries to use these models while querying the target KG. The development of KGNet poses research opportunities in various areas spanning GML pipeline automation, GML-Enabled SPARQL query optimization, and KG sampling for task-oriented training. The paper discusses the KGNet potential in supporting GML-enabled queries in real KGs of different application domains. KGNet automatically opts to train models based on a given budget for node classification, link prediction, and semantic entity matching. Using KGNet KG sampling, we achieved up to 62% memory reduction and 35% faster training time while achieving similar or better accuracy w.r.t training pipelines on the full KG. System source-code is available here.**

I. 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 [1]. Graph machine learning (GML) techniques, such as graph representation learning and graph neural networks (GNNs), are widely used in several data science tasks on KGs of diverse application domains, such as categorizing social behaviour in videos [2], drug-drug interaction [3] and fraud detection [4]. In these applications, several problems are formalized as GML models for node classification, link prediction, or semantic entity matching tasks. These GML models are trained using different GML methods. We refer to this type of discovery and exploration as *GML-enabled queries* (Q^{GML}), in which a trained GML model is applied to a specific set of sub-graphs.

RDF engines, which are widely used to store KGs [5], and the SPARQL query language do not support GML-enabled queries. Thus, data scientists are fully responsible for developing ML pipelines to train GML models and apply them to KGs in isolation from RDF engines. For KGs, enabling GML-based queries is a more challenging task due to the dynamic schema of KGs and a wide range of KG embedding (KGE) and GNN methods that could be used in several applications for heterogeneous graphs [6], [7]. It is impractical for average users to have this responsibility.

The KG illustrated in Figure 1 contains information about published papers in DBLP [8]. Traditional SPARQL query language cannot apply GML models on top of a KG, i.e., predict a graph node's class or a missing affiliation link for an author. For example, the venue node in Figure 1 is a virtual node that could be predicted using a node classification model. It will be interesting to query this KG to *get the paper-venue node using a GML node classification model via a SPARQL query.* There is a need for seamless integration of GML Models into RDF engines. Thus, users should be able to easily express their GML-enabled queries by following the SPARQL logic of pattern matching, i.e., avoid the explicit use of user-defined functions (UDF).

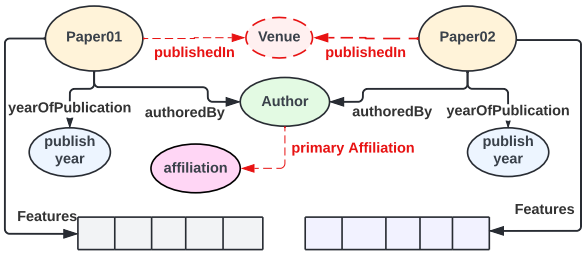

Fig. 1: A KG with nodes/edges in red, which could be predicted by classification and link prediction models on the fly.

```
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.}
```

Fig. 2: Q_V^{GML}: a SPARQL GML-enabled query using a node classification model to predict a paper's venue on the fly.

We envision a system to automatically train GML models and maintain metadata of trained models as an RDF graph (*Meta-GML KG*). The Meta-GML KG is interlinked with the target-node type in the data KG to enable querying both graphs using SPARQL, as illustrated in Q_V^{GML} in Figure 2. This query predicts a venue for each paper using a model of type *kgnet:NodeClassifier*. The GML-enabled triple patterns in lines 8-10 will dereference all *kgnet:NodeClassifier* models that are associated with DBLP nodes of type *dblp:Publication* and predict a class of type *dblp:venue*. We refer to *?NodeClassifier* in the triple pattern ⟨`?paper`, `?NodeClassifier`, `?venue`⟩ as a *user-defined predicate*. Extending RDF engines to support user-defined predicates raises research challenges, such as (*i*) automatic training of GML models for different tasks, (*ii*) optimizing Q^{GML} for GML model selection based on accuracy and inference time, and (*iii*) interacting efficiently with the selected model during the query execution time.

There is a growing adoption of integrating GML with existing graph databases, such as Neo4j [9] or Stardog [10]. However, the above challenges are not addressed by existing graph databases. For example, Neo4j supports some machine learning primitive methods, e.g., PageRank, and shortest-path using the *Cypher* language. Even more, Neo4j Graph Data Science v2.2 [11] supports limited graph embedding methods in a beta version, such as FastRP, Node2Vec, and Graph Sage. However, a user has to train the models separately as an initial step. Addressing these challenges will bring graph machine learning to data stored in RDF engines instead of getting data to machine learning pipelines. This will encourage the development of KG data science libraries powered by the expressiveness of SPARQL for better analysis and insight discovery based on KG structure and semantics. These libraries will empower data scientists with a full breadth of KG machine learning services on top of KGs stored RDF engines.

This vision paper proposes KGNet, a GML-enabled KG engine. KGNet extends existing RDF engines with two main components GML-enabled query manager and GML as a Service (GMLaaS). Our query manager transparently maintains the Meta-GML KG from the metadata of trained models, optimizes the GML model selection, and finally rewrites the Q^{GML} query as a SPARQL query. The rewritten SPARQL query interacts with our GMLaaS via HTTP calls to get the prediction from a particular pre-trained model. GMLaaS supports a repository of GNN and KGE methods and automates the training pipelines to model a specific task based on our task-oriented sampling. The automated pipeline can work to meet a user's budget for training the task within a certain time or memory budget. Our task-based sampling allows KGNet to avoid training using the full graph for higher performance and scalability.

In summary, the contributions of this paper are:

- The first GML-enabled KG platform as an extension of existing RDF stores [1].
- GML as a service to provide automatic training of GML models based on a given memory or time budget.
- GML-enabled query manager to optimize the GML model selection, i.e., opt for the near-optimal model based on constraints on accuracy and inference time.
- A comprehensive evaluation using three GML tasks on real KGs using different GML methods. Using KGNet KG task-based sampling, we achieved 62% memory reduction and 35% faster training while achieving similar or better accuracy w.r.t training using the full KG.

The remainder of this paper is organized as follows. Section II provides a background about existing graph machine learning pipelines. Section III outlines the main research challenges of developing a GMl-enabled KG engine. Section IV presents the KGNet platform. Section V discusses the results of evaluating our automated pipeline for training GML models. Sections VI and VII are related work and conclusion.

[1]The KGNet repository can be accessed at https://gitfront.io/r/CODS/peFMb931cZCo/KGNET/. We will publish the code of KGNet and datasets as an open source upon acceptance.

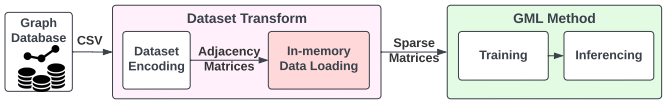

Fig. 3: A traditional GML pipeline [12]. The pipeline starts with extracting the graph data, followed by data transformation into sparse matrices to train models for a GML task. Finally, the inference step is ready to predict results in isolation from the graph databases.

II. Background: ML pipelines for KGs

ML pipelines developed to train models on a KG take a CSV file containing the KG triples as an input. Data scientists are in charge of this extraction process. Moving the KG data is costly, error-prone, and storage-inefficient, training ML models is a tedious and time-consuming task, especially for large KGs. Traditional ML pipelines use KG data as tabular in-memory data frames to perform features engineering and train classical ML classifiers using libraries, such as Scikit-Learn or SparkMLib. Some pipelines avoid the features engineering process by generating KG embeddings for nodes and edges and training models using these embeddings. This kind of pipeline is used to train models for Apple Saga [13], which uses graph ML libraries, such as DGL-KE [14], to generate KGEs. Data scientists are in charge of deciding which ML method to use in training.

With the great success of GNNs, data scientists widely adopted GNN methods to train GML tasks. Open Graph Benchmark (OGB) [12] standardized the GNNs training pipeline. OGB emphasized how AI practitioners tackle GML tasks and how to build a GNN training pipeline. We summarize this pipeline in Figure 3. The pipeline starts by encoding the KG nodes and edges to generate adjacency matrices to be loaded into the memory and form sparse matrices to perform the model training using a specific GNN method.

There are different libraries that provide various implementations for GNN methods, such as DGL [14], and PyG [15]. These libraries provide support to perform data transformation by loading your graph as an in-memory graph data structure and then applying the transformation. These libraries require colossal memory, long processing times for large graphs, and a deep understanding of different graph formats. The OGB pipeline is simple, but it is a semi-automated pipeline that requires human intervention and ML experience to build a successful pipeline in addition to the data transformation cost that is repetitive per task. Finally, data scientists have to save the trained models and use them in isolation from the original KG stored in the graph databases with no integration between traditional queries on these graph databases and these trained models.

III. Challenges of GML-enabled Queries

This section highlights the open research challenges and opportunities raised by extending RDF engines to support GML-enabled queries.

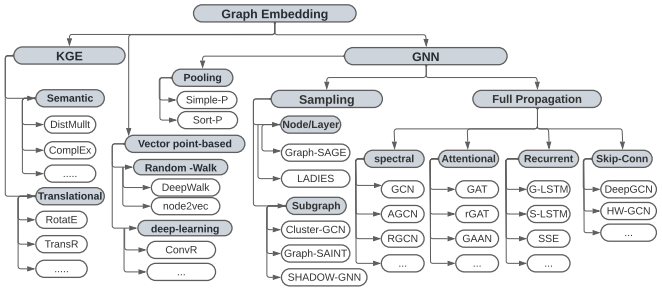

Fig. 4: A taxonomy of methods for training GML models.

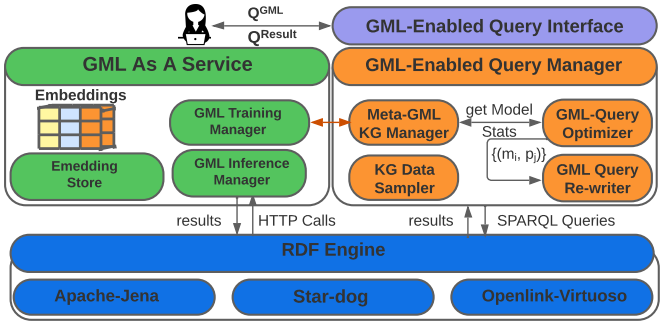

Fig. 5: The KGNet architecture.

A. Automating Training Pipelines for GML On Top of KGs

It is time-consuming and challenging to build AI pipelines that extract semantic information from KGs using GML methods, as it requires a deep understanding of these methods and variations of their implementations. There are numerous methods for training models for GML tasks, as summarized in Figure 4. 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 [7]. Some GNN methods support sampling on full graph, such as Graph-SAINT [16], Shadow-SAINT [17], and MorsE [18]. Examples of GNN full-batch training (without sampling) methods are RGCN [19] and GAT [20]. Our taxonomy has more categories, as shown in Figure 4.

These methods vary in inference accuracy, training time, and memory requirement. Thus, it is challenging to automate a training pipeline for a specific task based on a user's budget for time and memory. Moreover, the GML libraries supporting these methods demand different data transformations to in-memory graph formats, such as hetero-graph in DGL [14] and sparse tensors in PyG data-loaders [21]. The automated pipelines should enable the GML dataset transformation for ad-hoc GML tasks.

Real KGs contain millions to billions of triples, e.g., DBLP [8] and MAG [22]. Training GML models requires colossal computing resources that exceed any machine's capabilities. Moreover, GNN training with these large datasets suffers from over-smoothing that causes accuracy degradation [23], [24]. GNN sampling-based methods cannot overcome these limitations as they perform sampling on the full graph loaded into memory to apply mini-batch training in [25], [26]. Moreover, existing sampling methods use uniform graph sampling strategies that do not lead to high accuracy with a large number of node/edge types in KGs. Task-oriented sampling is an opportunity to optimize training models on a large graph by selecting a representative sub-graph related to the task. It is an open challenge to discover a subset of triples, which could help train a high-accuracy model for a particular task. This helps the GML methods to work with a smaller graph with fewer node/edge types.

B. Seamless Integration Between GML Models and RDF

Enabling ML on top of databases or RDF engines requires interfacing between the trained models and the underlying data management engine. Several systems utilize user-defined functions (UDFs) to implement this interface in [27]–[29]. There will be several UDFs of different trained models. A UDF looks easy to implement, but it comes with its cost for query optimizations in data systems [30]. Having an extensive catalog of UDFs lowers the expressiveness of these ML-based queries. Thus, a user will not be able to express their query easily in a descriptive manner. Moreover, it will be hard to automate the query optimization of ML-enabled queries. Having a seamless integration between GML Models and RDF is an open challenge. We addressed this challenge by maintaining the Meta-GML KG.

C. Optimizing GML-enabled Queries and Benchmarks

Estimating the cost of evaluating a user-defined predicate is more complex than estimating the cost of a traditional predicate in a KG. Cardinality estimation is used for the second one to optimize only the execution time. However, a user-defined predicate in a GML-enabled query can be inferred by multiple models. Each model varies in accuracy and inference time. RDF engines are unaware of this information. This is even more challenging than dealing with user-defined predicates in SQL [31]. We refer to this problem as the best model selection.

The RDF engine has to use the chosen model to perform inference. Optimizing for this step is an open challenge for the rank-ordering of performing inference at once, i.e., one call to a UDF or per instance, which may lead to an extensive number of calls to the UDF. Each model varies in the total number of predictions, e.g., the number of venues the model can predict in our running example. We refer to this as model cardinality. Predicting rank-ordering is an open challenge as RDF engines lack an accurate estimation of the UDF cost.

There are research opportunities for developing benchmarks to evaluate optimization approaches for GML-enabled queries. These benchmarks should be designed to work with large varieties of models for different user-defined predicates. Each GML-enabled query should vary in the number of user-defined predicates and be associated with variables of different cardinalities.

IV. The KGNet Platform

The KGNet architecture consists of three layers, as shown in Figure 5. KGNet *Interface* enables users to submit a GML-enabled query (task) and get results. The middle layer represents the core of KGNet, as it is responsible for linking the

```
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_Classifer',
6   GML-Task:{ TaskType:kgnet:NodeClassifier,
7   TargetNode: dblp:publication,
8   NodeLable: dblp:venue},
9   Task Budget:{ MaxMemory:50GB, MaxTime:1h,
10                Priority:ModelScore} } )};
```

Fig. 6: A GML-enabled insert query that trains a paper-venue classifier on DBLP. The TrainGML function is a UDF that is implemented inside the RDF engine.

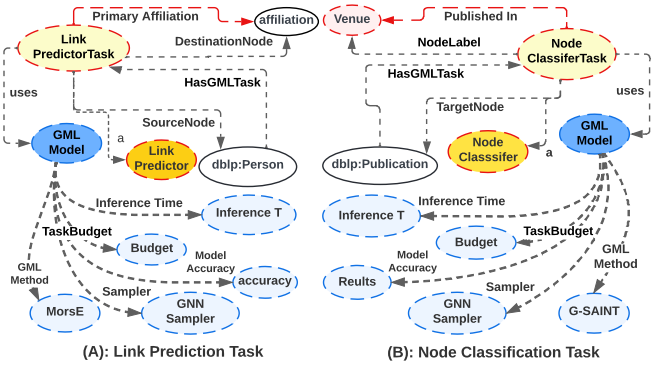

Fig. 7: A Meta-GML KG of two trained models for node classification and link prediction tasks. The white nodes are nodes from the original data KG. The dashed nodes/edges are metadata collected per trained model.

RDF query engine and the GML as a service component. This layer is composed of two main components. *GML-Enabled Query Manager* extends RDF engines with the Meta-GML KG that preserves all the information about the GML-trained models, samples the KG data to select a subgraph relevant to a task, rewrites the GML-enabled query into a traditional SPARQL query, and optimizes query rewriting for accuracy and time constraints.

GML as a Service (GMLaaS) acts as the ML environment for KGNet. GMLaaS mainly automates the training pipelines and provides our query manager with statistics for each trained model to maintain the Meta-GML KG. We use an embedding store to enable fast indexing and searching for similar nodes/edges based on their embedding. GMLaaS manages the pre-trained models and provides a Restful service to query these models for predictions, i.e., inference. An RDF engine interacts with our GMLaaS through remote HTTP calls via an internally implemented user-defined function (UDFs).

A. The GML-Enabled Query Interface

The GML-Enabled query interface is the upper front layer that enables RDF users to submit their GML-Enabled queries Q^{GML} into KGNet and get the results Q^{Result} in form of a set of triples. This layer includes a Q^{GML} parser that parses the query into a tree structure that allows extracting the GML-related triples and KG-data-related triples and identifying the type of the query either select, insert, or delete. Passes this information into the GML-enable query manager and get back the query results.

```
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.}
```

Fig. 8: A SPARQL GML-enabled delete query that deletes a trained model and its meta-data.

```
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.}
```

Fig. 9: A GML-enabled SPARQL query predicting author affiliation link (edge) on DBLP KG.

B. GML-Enabled Query Manager

In KGNet, the GML-Enabled query manager is responsible to maintain a KG of GML-trained models metadata (Meta-GML KG) to be used later for de-referencing GML predicates inside GML queries. Also takes care of the GML training (insert), inferencing (select), and delete query pipelines. a GML-enabled query could be (*i*) a SPARQL *insert* query to train a GML model and maintain its metadata in Meta-GML KG using a SPARQL insert query as shown in Figure 6, (*ii*) a SPARQL *delete* query to delete trained model files and associated embeddings from the GML-aaS component then deletes its metadata from the Meta-GML KG as in Figure 8, or (*i*) a SPARQL *select* query that performs inference for a GML predicate. KGNet query-optimizer picks the best models matching a Q^{GML} query based on the maintained statistics in our Meta-GML KG. Then, our query rewriter translates Q^{GML} into a SPARQL query interacting with our GMLaaS.

Task-oriented sampling: A user can express their interest to train a model for a specific target node(s) in a KG by providing a JSON object that encapsulates all required information to train a GML model, as illustrated in Figure 6. 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 Meta-GML KG manager gets a relevant subgraph (G^R) for the given task and interacts with the GML Training manager to automate the training pipeline for this task. After training is complete, the Meta-GML KG manager will receive the metadata of the trained model, including the model accuracy and inference time to maintain the Meta-GML KG, as shown in Figure 7. The user has to provide at least the

```
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 }
```

Fig. 10: A candidate SPARQL for Q_V^{GML}

```
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 { } }}
```

Fig. 11: A candidate SPARQL for Q_V^{GML}

task type, such as node classification or link prediction, define the task inputs, such as the target nodes and classification labels (Y classes) for a classification task, and set the task budget allowed for training such as memory and time budget. Experienced ML users can provide extra information, such as hyper-parameters, a specific GML method, and others.

Training GML methods with real large KGs is limited due to the GNN scalability issues [12], [14], [26]. GNNs Mini-batch training using graph sampling is a stranded to improve the GNNs scalability [25]. Our *KG data sampler* is a task-oriented sampler that aims at finding a relevant sub-graph to train a GML task. Each GML task targets nodes of a particular type, e.g., dblp:Publication in Q_V^{GML}. The target node is the node to be classified in the node classification task or source/destination nodes directly connected with the target edge in the link prediction task. The full KG may contain triples that are not reachable from a target node v^T or connected via more than three hubs from v^T. These triples do not help a model to generalize or lead to the over-smoothing problem [23], [24]. Our sampler extracts a task-relevant subgraph (g_t^R), which consists of a set of triples with representative triples associated with the target nodes. For GML task t, the size of g_t^R is much smaller than the full KG. This smaller graph will optimize the training time and demand less memory for training.

We developed two methods for sampling the relevant subgraph. The first is node-forward (node-FW) sampling which is designed for the node classification task to select only the outgoing edges/nodes that are directly connected with the target nodes and then generate a complete subgraph. The second method is edge-forward-backward (Edge-FW-BW) sampling. This method is designed for the link prediction task to select the outgoing and ingoing edges/nodes that are directly connected with the target-edge, such as the affiliation edge in figure 9. Our task-oriented sampler extracts the subgraphs matching these criteria and generates a complete graph that will be used for training the task.

From a GML-Enabled Query to SPARQL: The *GML-enabled Query Optimizer* is in charge of optimizing the Q^{GML} query for model selection and rank-ordering for evaluating the user-defined predicate. For Q_V^{GML} in Figure 2, our query optimizer fetches all URIs of the models satisfying the conditions associated with the user-defined predicate *?NodeClassifier*. Our Meta-GML KG is an RDF graph containing optimizer statistics, such as model accuracy, inference time, and model cardinality. Thus, we use a SPARQL query to get the models' URI, their accuracy, inference time, and cardinality. Our query optimizer opts for the near-optimal GML model achieving high accuracy and low inference time. We define this problem as an Integer programming (IP) optimization problem to minimize total execution time or maximize inference accuracy.

The *GML-Enabled Query Re-writer* uses the near-optimal GML model, whose URI is m, to generate a candidate SPARQL query. KGNet currently supports two possible execution plans, whose query templates are shown in Figures 10 and 11. The core idea is to map a user-defined predicate into a user-defined function (UDF), e.g., *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 dominates the query execution cost. For example, Q_V^{GML} predicts the venue of all papers, whose size is |?papers|.

The query template shown in Figure 10 will generate |?papers| HTTP calls. However, the query template shown in Figure 11 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 to the data KG in the GML-enabled query into sets per variable associated with a user-defined predicate. For example, KGNet gets the cardinality of the triple patterns at lines 5 and 6 using a query returning count(?paper) for these two triple patterns. We formulate this problem as another Integer programming (IP) optimization problem [32] that minimizes the total number of HTTP calls or minimizes the constructed dictionary size, which is based on the model cardinality.

C. GML As a Service Component (GMLaaS)

GMLaaS manages GML models in terms of automatic training and interactive inferencing via a Restful service. Moreover, it uses an embedding store to support entity similarity search tasks by measuring the similarity between embedding vectors. The *GML training manager* automates a training pipeline per task as illustrated in figure 12. The automated pipeline gets as input the relevant subgraph, the GML task, the task budget, and the available resources inside the ML environment.

The *Data Transformer* step automates the conversion of the relevant subgraph into GML-friendly sparse-matrices format. This representation is optimized for memory allocations and sparse-matrix operations. This format is compatible with PyGeometric [21] and DGL [14], the most common graph ML data loaders. These graph loaders are optimized for sparse

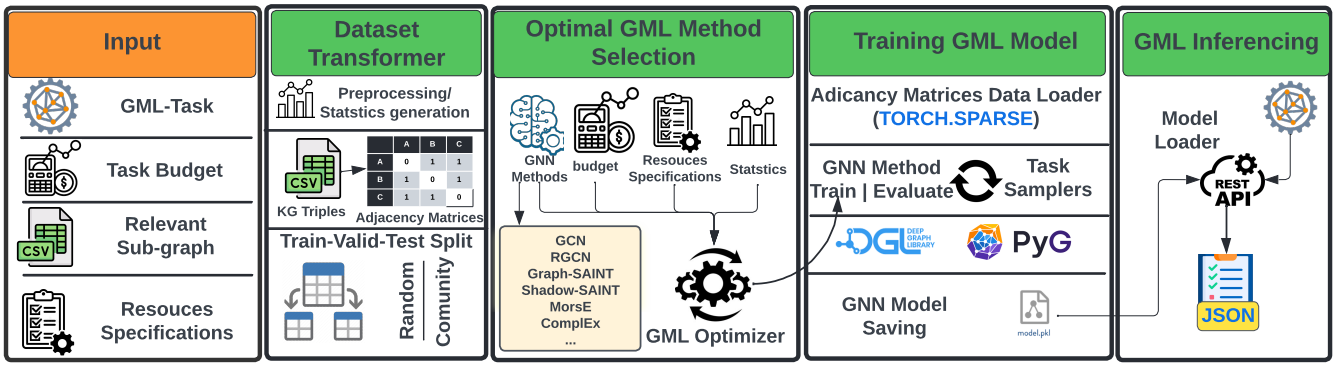

Fig. 12: The automation of training pipeline and inference in our GML-as a service (GMLaaS). GMLaaS interacts with the Meta-GML KG Manager to train a model for a specific task with limited budget. The automated pipeline performs data transformation, optimal GML method selection train a model within the limited budget. GMLaaS supports task inference through RestAPI that is called by a UDF.

knowledge graphs. Our automated pipeline checks the consistency of graph data by validating node/edge types counts, applying custom filters to remove literal data, removing any target class edges if found, and generating statistics about the graph (nodes, edges, and relations). Then, we perform a train-validation-test split according to different strategies, such as Random splitting: using sklearn-learn random train-test split, and Community-based: using one of the edge types that can provide a logical data splitting semantic for the task. KGNet is the first to automate this transformation to enable ad-hoc GML training queries.

The *Optimal GML Method Selection* opts for the best GML method for training a model for a given task. KGNet supports GNN methods, such as GCN [33], RGCN [19], Graph-SAINT [16],Shadow-SAINT [17], Morse [18], and KGE methods, such as ComplEx [7]. Based on the number of sparse-matrices generated, we estimate the total memory size each method may need. Moreover, we estimate the training time based on the dimension of the sparse-matrices and features aggregation approach adopted by each method. For GNN Sampling-based methods, the sampling cost basically depends on the sampling heuristic used [34]. Thus, we are working on a more advanced estimation method based on sampling the sparse-matrices and running a few epochs on them.

KGNet GML-optimizer should decide the resources required for each method and optimize the training settings. This selection enables better scalability in distributed environment settings which is a real challenge while supporting GML on real KGs. Finally, the automated pipeline trains a model and collects statistics related to the evaluation metric and inference time. KGNet creates a URI for the trained model to distinguish between models for inference tasks. The trained model Meta-data is returned back to the GML-Enabled Query Manager to update the Meta-GML KG. The generated meta-data for the link-prediction model is shown in figure 7.a and node-classification model in figure 7.b. For fast similarity search, the *Embedding Store* cub-component is used to store, index, and search these embeddings. The *GML Inferencing* receives HTTP calls for inferencing and serializes the result into a JSON Restful-API response and it back to the RDF engine. The current version uses FAISS [35] embedding store to allow nodes similarity search ad-hoc queries.

V. 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.

A. Evaluation Setup

Compared Methods: We used RGCN [19] as a full-batch training method and GraphSAINT [16], ShadowSAINT [17] as mini-batch sampling-based methods for node classification and MorsE [18] as edge sampling-based method for link prediction. The OGB [12] default configurations are used in both sampling and training.

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, and 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 [8] and Yago-4 [36]). We conducted two node classification tasks and one link prediction. We followed the tasks used in OGB [12]. Statistics about used KG and tasks are provided in table I.

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

B. 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. The training script starts to train the GNN model and then indexes the generated node embeddings

TABLE I: The KGNet's training pipeline is evaluated using two real KGs. Due to our task-oriented sampling, we managed to use four times larger KGs (DBLP and Yago) than the ones reported in OGB [12]. We used GML methods, such as RGCN [19], GraphSAINT [16], and ShadowSAINT [17] for Node Classification (NC) tasks, MorsE [18] for Link Prediction (LP) tasks, and GraphSIANT [16] for Similar Entity (SE) tasks.

Knowledge Graph	#Triples	#Targets	#Edge Types	#Node Types	Tasks
DBLP	252M	50 Venue,51283 Affiliations,1.2M paper	62	22	NC,LP,ES
YAGO4	250M	50 Country	156	73	NC

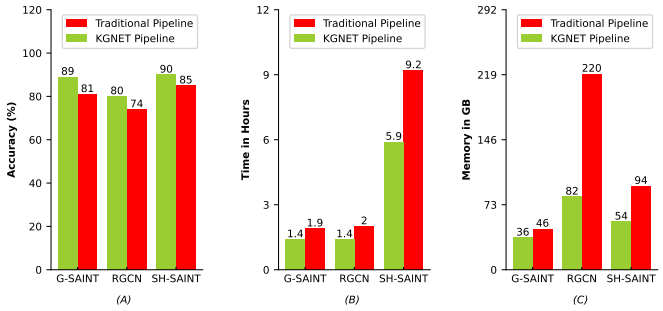

Fig. 13: (a) Accuracy, (B) Training Time, and (C) Training Memory for DBLP KG Paper-Venue node classification task. The KGNet task-oriented sampled subgraph significantly improves accuracy, training time, and memory.

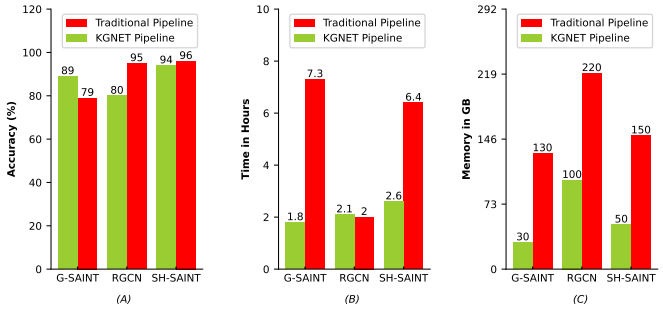

Fig. 14: (a) Accuracy, (B) Training Time, and (C) Training Memory for YAGO-4 KG Place-Country node classification task. The KGNet task-oriented sampled subgraph significantly improves accuracy, training time, and memory.

using FAISS embedding store. The Node-FW task-oriented sampling for paper nodes (target nodes) is used to sample a task-oriented sub-graph to train RGCN, Graph-SAINT, and Shadow-SAINT methods. The task results in Figure 13 show that our KGNet training pipeline outperforms the pipeline on the full graph in all methods with 5% accuracy score. In terms of training memory, the sampled sub-graph uses at least 22% less memory. In terms of training time, the sampled sub-graph uses at least 27% less. These results justify that KGNet task-oriented training pipeline successfully discovers the relevant subgraph for each task.

The **Link prediction task** predicts an affiliation link for an author based on the history of publications and affiliations on DBLP KG. The nodes/edges embeddings are generated using the MorsE [18] edge sampling-based method. MorsE is the state-of-the-art link-prediction sampling-based method. The KGNet task-oriented edge FW-BW is used to discover a

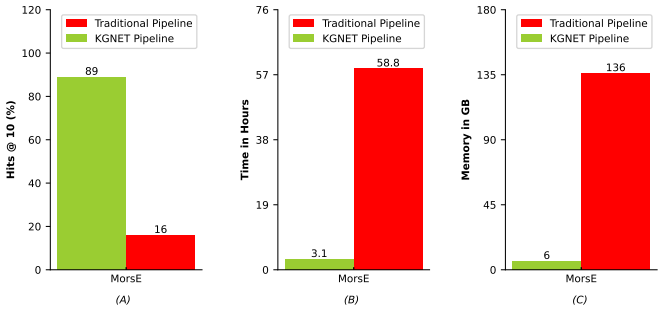

Fig. 15: (a) Accuracy, (B) Training Time, and (C) Training Memory for the DBLP Author Affiliation link prediction task. The KGNet task-oriented edge Sampled subgraph significantly improves the Hits@10 MRR score, training time, and training memory.

relevant sub-graph to train MorsE. Figure 15 shows the results. The KGNet automated pipeline significantly outperforms the pipeline on the full graph in terms of Hits@10 MRR score. In terms of memory and training time, our pipeline uses 94% less memory and time.

VI. Related Work

GraphDB [37] uses Lucene to support full-text search on their graphs. These systems focus only on text embedding and at an early stage of supporting graph embedding methods. Stardog [10] 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. KGNet extends RDF engines with GML-enabled operators using GNN and KGE methods with a graph sampling component for scalability and efficiency. A basic node similarity search implementation of KGNet is demonstrated in [38].

Sematch [39] is an integrated framework to compute the semantic similarity of KGs (concepts, words, and entities) using text similarity techniques and uses similarity metrics that rely on KG structure and statistical information contents from corpora. Sematch is concerned with text analysis applications and uses a python pipeline to match entities with types using SPARQL queries. KGvec2go [40] built a Web API to allow graph embeddings to be accessed and consumed using AI-based applications. KGvec2go is concerned with providing embeddings without evaluating embeddings quality, scalability, or integrability with RDF engines.

Bordawekar et al. [41] 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 RL with DB using text embedding techniques. EmbDI [42] automatically learns local relation embeddings with high quality from relational datasets using a word embedding to support datasets schema matching. Unlike KGNet, these systems did not support either KGE or GNN methods. Moreover, these systems did not automate training models and sampling datasets pipelines.

Yuo Lu et.al. addressed the problem of AI-enabled query optimization for SQL in [27] and introduced the probabilistic predicates (PPs) method that can be trained without any knowledge of the inference models. KGNet automatically trains models and maintains a Meta-GML KG containing statistics that are used for query optimization.

Works RDFFrames [43], DistRDF2ML [44], and Apple Saga [13] 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. Unlike KGNet, these systems do not support GML-enabled queries and follow the two-phase RDF-ML pipeline that moves that RDF data between the RDF engine and ML environment which is costly and non-scalable and even more requires a user with good experience in the ML field, especially for graphs.

VII. Conclusion

This vision paper proposed a novel engine called KGNet for supporting KG search and exploration based on GML models. KGNet addresses several research challenges, such as selecting the optimal GML method to train a model based on a specific budget, optimizing GML-enabled SPARQL queries, and sampling KG data for a given GML task. KGNet bridges the gap between RDF engines and AI pipelines to provide built-in GML-enabled query support. Thus, there is no need to reformat and migrate KG data from RDF engines to data science or AI platforms. KGNet innovate a seamless integration between GML models and RDF engines by maintaining the metadata of trained models and their statistics as an RDF graph, called Meta-GML KG, which is used to optimize the GML-enabled queries. Moreover, a user can easily express their GML-enabled query in a descriptive manner based their GML task, such task type and target nodes. In KGNet, GML as a service supports many different GML methods and automatic training of GML models for link prediction, node classification and semantic entity matching. In KGNet, we easily enable semantic discovery on KGs for users without prior knowledge of embedding techniques and AI pipelines.

References

[1] 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

[2] J. Sun, Q. Jiang, and C. Lu, "Recursive social behavior graph for trajectory prediction," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[3] X. Lin and Z. Q. and, "KGNN: knowledge graph neural network for drug-drug interaction prediction," in *Proceedings IJCAI*, 2020, pp. 2739–2745. [Online]. Available: https://doi.org/10.24963/ijcai.2020/380

[4] Y. Dou, Z. Liu, and et.al., "Enhancing graph neural network-based fraud detectors against camouflaged fraudsters." ACM, 2020. [Online]. Available: https://doi.org/10.1145/3340531.3411903

[5] 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, 2022. [Online]. Available: https://doi.org/10.1007/s00778-021-00711-3

[6] 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

[7] 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.

[8] M. R. Ackermann. (2022) dblp in rdf. [Online]. Available: https://blog.dblp.org/2022/03/02/dblp-in-rdf/

[9] D. Gordon. (2021) Fulltext search in neo4j. [Online]. Available: https://neo4j.com/developer/kb/fulltext-search-in-neo4j/

[10] docs@stardog.com. Stardog documentation-machine learning. [Online]. Available: https://docs.stardog.com/machine-learning#machine-learning

[11] N. Inc. (2022) Neo4j graph data science for machine learning. [Online]. Available: https://neo4j.com/docs/graph-data-science/current/machine-learning/machine-learning/

[12] 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.

[13] 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

[14] 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.

[15] P. Team. (2022) Torch geometric documentation. [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/index.html

[16] H. Zeng, H. Zhou, and et.al., "Graphsaint: Graph sampling based inductive learning method," in *ICLR(8)*, 2020. [Online]. Available: https://openreview.net/forum?id=BJe8pkHFwS

[17] 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

[18] M. Chen, W. Zhang, and et.al., "Meta-knowledge transfer for inductive knowledge graph embedding," in *Proceedings of the 45th International ACM SIGIR*. ACM, 2022, p. 927–937. [Online]. Available: https://doi.org/10.1145/3477495.3531757

[19] M. S. Schlichtkrull, T. N. Kipf, and e. a. Peter Bloem, "Modeling relational data with graph convolutional networks," in *ESWC*.

[20] 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

[21] P. Team. (2022) Torch geometric loader. [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/modules/loader.html

[22] 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

[23] 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 *The Thirty-Fourth AAAI Conference on Artificial Intelligence IAAI 2020*. AAAI Press, 2020, pp. 3438–3445.

[24] J. Liu, K. Kawaguchi, B. Hooi, Y. Wang, and X. Xiao, "Eignn: Efficient infinite-depth graph neural networks," in *Advances in Neural Information Processing Systems*, vol. 34. Curran Associates, Inc., 2021, pp. 18 762–18 773. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/9bd5ee6fe55aaeb673025dbcb8f939c1-Paper.pdf

[25] 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

[26] L. Wu, P. Cui, J. Pei, and L. Zhao, *Chapter6: Graph Neural Networks: Scalability*. Singapore: Springer Singapore, 2022.

[27] Y. Lu, A. Chowdhery, S. Kandula, and S. Chaudhuri, "Accelerating machine learning inference with probabilistic predicates," in *SIGMOD*, 2018. [Online]. Available: https://doi.org/10.1145/3183713.3183751

[28] R. Bordawekar and O. Shmueli, "Enabling cognitive intelligence queries in relational databases using low-dimensional word embeddings," *Computing Research Repository(CoRR)*, vol. abs/1603.07185, 2016. [Online]. Available: http://arxiv.org/abs/1603.07185

[29] 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

[30] 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.

[31] 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

[32] S. Bradley, A. Hax, A. Hax, and T. Magnanti, *Chapter 9: Integer Programming*. Addison-Wesley, 1977.

[33] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *NeurIPS(29)*, 2016, pp. 3837–3845. [Online]. Available: https://arxiv.org/abs/1606.09375

[34] 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

[35] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *IEEE Trans. Big Data*, vol. 7, no. 3, pp. 535–547, 2021. [Online]. Available: https://doi.org/10.1109/TBDATA.2019.2921572

[36] T. P. Tanon, G. Weikum, and F. M. Suchanek, "YAGO 4: A reason-able knowledge base," in *The Semantic Web - 17th International Conference, ESWC*, ser. Lecture Notes in Computer Science, vol. 12123. Springer, 2020, pp. 583–596. [Online]. Available: https://doi.org/10.1007/978-3-030-49461-2_34

[37] graphdb.ontotext.com. (2021) Semantic similarity searches. [Online]. Available: https://bit.ly/3BcILoa

[38] H. Abdallah, D. Nguyen, K. Nguyen, and E. Mansour, "Demonstration of kgnet: a cognitive knowledge graph platform," *ISWC*, 2021. [Online]. Available: http://ceur-ws.org/Vol-2980/paper311.pdf

[39] G. Zhu and C. A. Iglesias, "Sematch: Semantic similarity framework for knowledge graphs," *Knowledge-Based Systems.*, vol. 130, pp. 30–32, 2017. [Online]. Available: https://doi.org/10.1016/j.knosys.2017.05.021

[40] 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

[41] 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

[42] C. Riccardo, P. Paolo, and T. Saravanan, "Creating embeddings of heterogeneous relational datasets for data integration tasks," in *Proceedings of the 2020 ACM SIGMOD*, USA, 2020, p. 1335–1349. [Online]. Available: https://doi.org/10.1145/3318464.3389742

[43] A. Mohamed, G. Abuoda, and et.al., "Rdfframes: Knowledge graph access for machine learning tools," *PVLDB*, vol. 13, no. 12, 2020. [Online]. Available: http://www.vldb.org/pvldb/vol13/p2889-mohamed.pdf

[44] 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