LMKG: Learned Models for Cardinality Estimation in Knowledge Graphs

Angjela Davitkova[§]
TU Kaiserslautern (TUK)
Kaiserslautern, Germany
davitkova@cs.uni-kl.de

Damjan Gjurovski[§]
TU Kaiserslautern (TUK)
Kaiserslautern, Germany
gjurovski@cs.uni-kl.de

Sebastian Michel
TU Kaiserslautern (TUK)
Kaiserslautern, Germany
michel@cs.uni-kl.de

ABSTRACT

Accurate cardinality estimates are a key ingredient to achieve optimal query execution plans. For RDF engines, specifically under common knowledge graph processing workloads, the lack of schema, correlated predicates, and various types of queries involving multiple joins, render cardinality estimation a particularly challenging task. In this paper, we develop a framework, termed LMKG, that adopts deep learning approaches for effectively estimating the cardinality of queries over RDF graphs. We employ both supervised and unsupervised approaches that adapt to the subgraph patterns and produce more accurate cardinality estimates. To feed the underlying data to the models, we investigate efficient representations and put forward a novel encoding that represents the queries as subgraph patterns. Through extensive experiments on both real-world and synthetic datasets, we evaluate our models and show that they overall outperform the state-of-the-art knowledge graph approaches and novel learned estimators for RDBMS, NeuroCard and MSCN, in terms of accuracy and execution time while maintaining minimal space overhead.

1 INTRODUCTION

Due to the versatility of the graph model and the ability to create links between different data sources, knowledge graphs are a widely applied concept to model structured knowledge. Since the idea of adapting knowledge graphs for enterprise usage, initially proposed by Google, several major companies such as Facebook and Amazon tailored knowledge graphs to their needs, complemented by substantial academic research efforts in various domains. Specifically in the past years, techniques to mine knowledge graphs have been widely investigated and more recently hugely impacted by deep learning models. Efforts for the improvement of RDF graph representation and embeddings by deep learning models have led to a promising performance in the widely studied tasks of question answering and link prediction [20]. Deep learning has also performed exceptionally well when considering the tasks of graph generation and processing [40, 54]. However, one area remains vaguely explored and that is the usage of deep learning models for cardinality estimation in knowledge graphs.

Intuitively, producing efficient query plans heavily relies on accurate cardinality estimates [24]. Although an RDF database can be seen as a single table composed of three columns (subject, predicate, object), traditional techniques used in relational databases have been shown to perform poorly for SPARQL queries [11, 18, 34, 42]. The **challenges** in cardinality estimation come directly from the nature of RDF data and the lack of a rigid schema. *First*, present correlation between individual predicates renders the use of traditional cardinality estimation techniques, like histograms, inapt [34]. In other words, although the cardinality of two predicates independently may be quite selective, their co-occurrence can be quite common—leading to an inaccurate estimate if independence is assumed. *Moreover*, SPARQL queries typically include many (self-) joins between RDF triples [11]. Hence, to accurately estimate cardinalities, we need to go beyond the join uniformity assumption [18, 42]. *Finally*, it is not uncommon for SPARQL queries to contain more than one type of query pattern, for instance both a star and a chain pattern. Such queries further contribute to the challenges concerning cardinality estimation [18, 42].

In this paper, we introduce *LMKG, a learned model framework for cardinality estimation in knowledge graphs*. LMKG learns to estimate the cardinality of the most used types of queries (i.e., star and chain queries [2]) and additionally, provides algorithms for handling complex queries such as snowflake or tree queries. Motivated by recent research advancements for cardinality estimation in relational databases [27, 52] and the ability of neural networks to detect interconnections between variables, with LMKG, we establish the problem of cardinality estimation of knowledge graph patterns as a deep learning problem, covering both types of learning. LMKG offers the creation of an *unsupervised cardinality estimator* (LMKG-U) by employing autoregressive models with subgraph pattern encodings. By encoding queries as subgraph patterns, LMKG also provides the possibility for creating a *supervised cardinality estimator* (LMKG-S).

LMKG efficiently learns correlations between subgraph patterns and, as a result, provides distinctively accurate cardinality estimates. When designing the models of LMKG, we take into consideration the mentioned cardinality estimation challenges. Thus, to handle the challenge of high correlation between terms and to cope with the large number of (self-) joins, LMKG learns over relevant subgraph patterns and not over independent terms or triples. To deal with the high number of patterns, LMKG provides an efficient sampling approach for generating relevant training data. Furthermore, since knowledge graphs can include data from various sources and consequently have large term domain sizes, we propose the usage of a term compression strategy. The compression strategy enables us to apply the models even when considering knowledge graphs with many distinct terms. Moreover, the compression strategy is a necessity for LMKG-U since the unsupervised model cannot be applied to heterogeneous datasets without it. In addition to most typically used encodings, a novel subgraph encoding coined *SG-Encoding* is introduced. The SG-Encoding can incorporate various subgraph patterns while at the same time maintaining a compact representation.

[§]Equal contribution

Finally, for handling complex queries, we propose approaches tailored to the LMKG models.

We perform a comprehensive overview of the suggested models by examining their suitability based on the datasets, the query types, and the query sizes. Through experimental evaluation on both real-world and synthetic datasets, we perform a thorough analysis of the models of the LMKG framework and we compare our approach with **seven** different cardinality estimators for knowledge graphs or relational databases and we show that LMKG generally outperforms them across several measures.

The main contributions of this paper are:

(1) We formulate the problem of cardinality estimation in knowledge graphs (Section 3) through the lenses of supervised and unsupervised deep learned models.
(2) To tackle the problem of cardinality estimation in knowledge graphs and to handle the respective challenges, we develop the framework LMKG[1] (Section 4) that includes models of different types that can be tailored to a specific dataset or a sample workload (Section 5).
(3) To featurize subgraphs and provide them as input in the models, we explore different representations including our newly introduced *SG-Encoding* (Section 5.1.2).
(4) To reduce the input and output dimensionality when considering heterogeneous knowledge graphs, we propose the usage of a term compression strategy (Section 5.2.1).
(5) We explain how LMKG can handle complex queries (Section 6), analyze the challenges of training data creation (Section 7.1) and give an overview of the most suitable use-cases and limitations of LMKG (Section 7.2).
(6) We report on a comprehensive experimental study, evaluating the LMKG framework against the state-of-the-art approaches, and objectively discuss the challenges of learned knowledge graph cardinality estimation (Section 8).

2 RELATED WORK

Cardinality Estimation in Knowledge Graphs: Early work on cardinality estimation either uses statistics gathered on properties of the ontology [41] or a summary of graph patterns [29]. The Jena ARQ optimizer [44] uses pre-computed statistics and single-attribute synopsis for estimating join selectivities. However, the term independence assumption leads to underestimations. Similarly, RDF-3X [35] does not consider correlation between predicates. Vidal et al. [47] suggest that basic graph patterns can be partitioned into star-shaped groups for which they estimate the cardinality using sampling and a cost model. Characteristic sets [34] is a synopsis for star queries, later extended for join ordering [11]. Using extended characteristic sets (ECS), Meimaris et al. [32] propose an index that allows fast processing of conjunctive queries. However, the focus is solely on improved query processing and join ordering (after an initial filtering phase using the ECS index)—the authors do not provide a method for cardinality estimation. Jachiet et al. [19] use statistics, mainly focused on the predicates. Huang and Liu [18] propose combining Bayesian networks—capturing the joint probability distribution over correlated properties for star patterns—and a histogram for chain patterns. Presto [48] stores statistics of common subgraphs. Stefanoni et al. [42] summarize RDF graphs and use the summaries for estimation. G-Care [38], a benchmarking framework, compares existing approaches for knowledge graphs and adapts estimators used in relational databases for graphs.

Learned Approaches for Cardinality Estimation in Relational DBMS: An early work, Leo [43] adjusts the cardinality estimates later used during query optimization, by monitoring previous queries. Liu et al. [27] provide an effective selectivity estimation technique for relational operators using neural networks. Similarly, MSCN [21], a multi-set convolutional network, represents relational query plans with set semantics, to capture query features. Dutt et al. [8] use neural networks and tree-based ensembles for selectivity estimation of multi-dimensional range predicates. To overcome misestimation, Woltmann et al. [50] suggest a local-oriented approach. Hayek and Shmueli [15] propose the usage of estimated containment rates. Others, explore simple deep learning models [37] and pure data-driven models [16]. Differently, Naru [52] is an unsupervised data-driven synopsis that achieves highly accurate estimates in relational databases with deep autoregressive models and a Monte Carlo integration technique called progressive sampling. The recently proposed NeuroCard [51], extends the idea of autoregressive models for join cardinality estimation in relational databases. As it can also be applied to knowledge graphs, we apply NeuroCard for self-joins, as a main competitor in the experiments. Hasan et al. [14], suggest both autoregressive models and supervised learning models as cardinality estimators. Others [23, 30, 31, 36], shift the focus to optimal plan generation by applying reinforcement learning.

Learning in Graphs: Although not for cardinality estimation over knowledge graphs, deep learning for KGs has been widely researched [20]. Since we need to represent subgraphs efficiently, related work on KG embeddings is relevant, too. For instance, Hamilton et al. [13], Bordes et al. [3], and Wang et al. [49] propose different node embeddings. However, generating node embeddings in presence of unbound terms has not been discussed. Additionally, their main focus is on term or triple and not on a subgraph representation. In more related research, GraphAF [40] and MolecularRNN [39] focus on generating molecular graphs using deep learning. This work does not allow the estimation of the individual term densities. For our encoding, we build on their idea for subgraph representation. Cardinality estimation in graphs can be realized as subgraph isomorphism counting, i.e., determining the number of different subgraph isomorphisms between a given graph and a query pattern graph. Liu et al. [28] model subgraph isomorphism counting as a learning problem by exploring different representations and proposing a dynamic intermedium attention memory network. Chen et. al [6] investigate models for substructure counting, and propose the Local Relational Pooling model. However both previous approaches are trained on synthetic datasets and are computationally costly for larger graphs. Tahmasebi and Jegelka [46] study the theoretical possibility of counting substructures by a graph representation network. Other work also considers subgraph representation improving and learning for different classification and regression tasks [4, 53]. Although related to LMKG-S, all previous approaches focus on smaller graphs or query patterns. Zhao et. al [56] propose a supervised learning model for estimating subgraph counts. They decompose a query into smaller subgraphs, compute their representations using a GNN, and concatenate them as a query-level representation, sent through a MLP for estimation. Although they incorporate all subgraphs from the query for a final prediction, they still create the encodings for smaller query subgraphs, which can introduce misestimates due to the initial query decomposition.

[1]the LMKG code, together with the example datasets, is accessible through the authors' websites

3 PROBLEM STATEMENT

A knowledge graph KG is a finite set of RDF triples. Each triple $t_i \in KG$ is constructed out of three terms (s_i, p_i, o_i), corresponding to a subject $s_i \in S$, a predicate $p_i \in P$, and an object $o_i \in O$. Every term is uniquely identified by a URI where objects can be literals (e.g., strings or integers). More specifically, the subject is a resource or a node in the graph, via which a predicate forms a relationship to another node or a literal value, called an object.

SPARQL[2] is the de-facto standard query language for RDF stores. It is based on matching graph patterns. To execute the matching of graph patterns, the SPARQL query engine has to perform multiple joins to retrieve data from the database, depending on the length of the given query. A SPARQL query can have variables that are not bound to a specific term, referred to as unbound terms. A SPARQL query qp consists of triples forming a graph pattern $\{t_1, ..., t_i, ..., t_k\}$, where $t_i \in KG$ and every triple t_i can have an arbitrary number of variables (e.g., $?x$). Queries are distinguished according to their shapes: chain, star, tree, snowflake, cycle, clique, petal, flower, and graph. Star queries consist of triples $[(s_1, p_1, o_1), \ldots, (s_1, p_k, o_k)]$ such that all triples are centered around the same entity, in this case, subject s_1. Chain queries join triples such that the object of the preceding triple is the subject of the next one. Formally, a chain query consists of k triples $[(s_1, p_1, o_1), (s_2, p_2, o_2), \ldots, (s_k, p_k, o_k)]$, such that $s_i = o_{i-1}$, where $i \in [2, k]$. Although LMKG is mainly proposed for handling the most common queries (star and chain queries [2]), we also provide algorithms for handling complex queries (Section 6). Consider the following SPARQL queries:

```
(1) SELECT ?x WHERE
      { ?x  :hasAuthor  :StephenKing;
            :genre  :Horror. }
(2) SELECT ?x, ?y WHERE
      { ?x  :hasAuthor  ?y.
        ?y  :bornIn  :USA. }
```

The first query is star-shaped and asks for all subjects (i.e., books) that are of genre Horror and have as author StephenKing. In this example, the triples are centered around a single entity, i.e., the same subject. For the second query, we see that the two triples share a common unbound term $?y$, which denotes the object in the first triple and the subject in the second triple, hence, forming a chain query. The query result consists of all authors that have written a piece and are born in the USA.

The cardinality $card(qp)$ represents the number of graph patterns from the knowledge graph KG that match the query graph pattern qp. In this work, we propose *supervised* (LMKG-S) and *unsupervised* (LMKG-U) models that produce accurate estimates $est(qp)$ of the actual cardinality $card(qp)$.

For the **supervised estimator**, we investigate the use of a deep neural network that takes as input the query pattern qp and computes the estimated cardinality $est(qp)$.

For the **unsupervised estimator**, we investigate the application of an autoregressive model. These models decompose the joint density into n conditional probabilities, where n is the number of terms in the input of the model.

4 FRAMEWORK OVERVIEW

LMKG represents a compound of several models. It comprises two phases, depicted in Figure 1. In the first phase, called *creation phase*, the models that need to be created are determined. The next step is to generate adequate training data if there is no sample workload available. Subsequently, the chosen models are tuned and trained. The second phase, called *execution phase*, encompasses the user-system interaction in which the cardinality estimate is computed. Below, the two phases are presented.

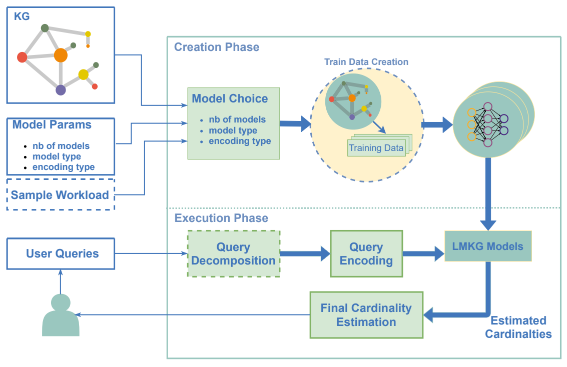


Figure 1: LMKG framework overview

Model Choice: LMKG creates models tailored to a given query workload, where the number and type of models to be created can be specified together with the data encoding. The following models can be created: *Single model*: one model that is trained over queries of different types and sizes. *Models grouped by query type*: multiple models are created where each model is specialized for a different query type, e.g., one model that can answer chain and another model that can answer star queries. *Models grouped by query size*: multiple models are created, specialized for different query sizes, e.g., one for queries with up to 3 joins and another for queries from 4 until 8 joins. We discuss in detail the model grouping strategies and their benefits and drawbacks in Section 7.2. Updates to the KG ultimately lead to increased estimation errors. This can be detected by observing the true cardinalities after query execution, if deployed within an RDF database, or via periodic testing. If the deviation from the original (after training) accuracy is not tolerable, the LMKG models require retraining. Since the grouping strategies create several models, only a few models may be retrained and not all of them.

Training Data Creation: Once the models that need to be created are selected and there is no sample workload available, the next step is to create training data. LMKG creates training data based on the given knowledge graph (Section 7.1). For LMKG-S, the training data contains queries and their assigned cardinalities, where the queries include at least one unbound term. For LMKG-U, the training data includes only patterns with bound terms since the model can estimate their conditional probabilities and use them for queries with unbound terms.

Training: The training step uses data that is either generated or provided as a sample workload. During training, one general model for different queries or several grouped models, each tailored for answering specific types or sizes of queries, are created. The training step involves transforming graph patterns into features and encodings for the chosen deep learning model.

Querying: Given a user-specified query, LMKG provides an estimate of the query result size. As depicted in Figure 1, for a query of a specific type and size, which is already learned by one of the models, LMKG uses the adequate model to directly estimate the cardinality. The query is encoded according to the model. The featurizer then forwards the input through the models and receives a prediction for the cardinality. In this work, we focus only on equality predicates.

[2]https://www.w3.org/TR/sparql11-overview/

To identify which models should be created in LMKG, an expert needs to analyze the characteristics of the dataset at hand. In particular, it is beneficial to know the distribution of the queries in the given workload as well as the dataset distribution, while also being aware of the available memory budget and latency constraints at query time. Having this information, the expert can decide on the number of models that should be created, their type, and the potential grouping. When issuing a query, the user does not need to know the dataset and query workload characteristics nor which models are created. The LMKG framework will guide the query to the appropriate model and deliver the cardinality estimate. For instance, if the expert decided on a supervised model, i.e., LMKG-S, with query type grouping, then two models will be created, one for star and one for chain queries. If the user issues a chain query, the framework will detect the query type and send it to the supervised model trained on chain queries. It is possible that the models receive a query for which they are not specialized, such as a query with both star and chain patterns. Handling of such complex queries including query decomposition and cardinality estimation is explained in detail in Section 6.

5 LMKG MODELS

Neural networks are capable of detecting patterns in high dimensional data, which is an important property considering the high number of co-occurring terms in a knowledge graph. Following existing work on learned cardinality estimation in relational databases, e.g., [21, 37, 52], and to capture and analyze both flavors of learning, LMKG utilizes two models, a supervised and an unsupervised model, coined LMKG-S and LMKG-U, respectively. In the following, we explain the design of these two models in detail and emphasize on the employed pattern encodings. The need for various encodings comes from the ability of the models to train and estimate over different types of inputs. For LMKG-S, we focus on creating a compact encoding that can represent various subgraphs. For LMKG-U, we focus on reducing the input and output dimensionality for handling heterogeneous KGs.

5.1 Supervised Model (LMKG-S)

Supervised deep learning models can efficiently approximate non-linear functions. By increasing the set of learned parameters, different levels of non-linearity and data patterns can be learned. To benefit from their expressiveness, we first address cardinality estimation as a supervised learning problem. We call this model LMKG-S. LMKG-S receives as input a query and predicts its cardinality. Next, we discuss the input encoding, first how triples are encoded and then how the graph structure is encoded.

5.1.1 Term Encoding: To model a triple pattern, we convert the triple terms into numerical values, where an absent term is simply encoded with a value of 0. Each of the terms is separately encoded, and when concatenated they constitute the triple encoding. To capture correlation, we always include all subgraph terms and focus on a more compact representation of them. LMKG currently supports two types of encodings (and optionally an embedding) for terms of triple patterns:

One-hot Encoding in which the bound terms involved in the triple are set to 1. For example, if $|S| = 3$ then the subject with id 2 will be encoded as `[010]`. A single triple encoding constructed out of one-hot encoded terms will occupy $O(|S| + |P| + |O|)$ space.

Binary Encoding in which the bound term is represented with a binary digit. For a KG with 3 unique subjects, the binary encoding of the subject with id 2 is `[10]`. The space of this triple encoding is $O(log_2|S| + log_2|P| + log_2|O|)$. We prefer the binary encoding because we need to compactly represent the knowledge graphs that usually have a large number of unique terms.

5.1.2 Subgraph (SG)-Encoding: Existing work [21], focusing on relational databases, tailors encodings to queries by only representing the presence of a predicate column, not emphasizing enough on the actual values. Although suitable for relational data where columns often have small domains, this is not adequate for heterogeneous knowledge graphs. Further, a suitable encoding should be able to represent different subgraph structures, corresponding to the different query types. Undoubtedly, the most popular graph representations are adjacency lists and adjacency tensors. An adjacency list represents a graph by keeping for each node i a list of nodes to which i points to, while an adjacency tensor A has entries $A_{i,j,l} = 1$ if node i has an edge l to node j, and a zero entry otherwise. Regarding space consumption, adjacency lists are preferred when graphs are sparse, as they do not represent absent edges. Clearly, this advantage vanishes as graphs get denser. Keeping in mind these basic principles, next, we present the SG-Encoding tailored to the needs of LMKG-S.

In LMKG, encodings need to represent every possible subgraph pattern up to a specific size. When working with neural networks the size of the input is fixed. Thus, to represent all possible subgraphs up to a specific size, the size of the encoding needs to be set to the maximal number of features used to represent a dense subgraph of the knowledge graph. That means, for all possible patterns in the KG, a suitable encoding can be in form of an adjacency tensor A of space $O(d * d * b)$, where d is the number of nodes in the knowledge graph (i.e., subjects and objects) and b is the number of predicates. However, for real-life knowledge graphs and query workloads, such tensors would be enormous, thus, impractical. Similarly, adjacency lists are not performing better here, since considering the complete space of possibilities leads to a dense graph. Therefore, we consider subgraph patterns that represent only a subset of the complete graph, such that space consumption can be drastically reduced. Following existing work on molecular graphs [40], we represent a KG subgraph with n nodes as $G = (A, X)$, where $A \in \{0, 1\}^{n*n*b}$ and $X \in \{0, 1\}^{n*d}$. Although suitable for molecular graphs where the number of distinct edges is small, in most KGs this encoding creates a huge sparse tensor A, due to the high number of predicates. To circumvent the large tensors created from the existing representations and adapt them to knowledge subgraphs, we propose a novel subgraph encoding, termed *SG-Encoding*. We define a subgraph pattern to have n nodes and e predicates, where $n < d$ and $e < b$. A subgraph pattern is represented as $SG = (A, X, E)$, where A is the adjacency tensor, X is the node feature matrix and E is the predicate feature matrix. Given an ordering of the nodes and predicates from the subgraph pattern, we define $A \in \{0, 1\}^{n*n*e}$, $X \in \{0, 1\}^{n*d}$ and $E \in \{0, 1\}^{e*b}$, where $A_{ijl} = 1$ if there exists an edge l between the i-th and j-th node. We set X_{im} to 1 if the i-th node is of type m and E_{lk} to 1 if the l-th predicate is of type k. Intuitively, if b is smaller than a threshold value, we can eliminate the matrix E, as proposed in existing work [40].

Initially, each row in X and E is a one-hot encoding of size d and b, respectively, where rows in X represent the node type and rows in E the edge type. For a more compact representation, we provide a modification of the matrices X and E, where instead of a one-hot encoding, we employ a more compact encoding, such as a binary encoding, for each of the nodes and edges in the subgraph. More specifically, the encoding is modified such

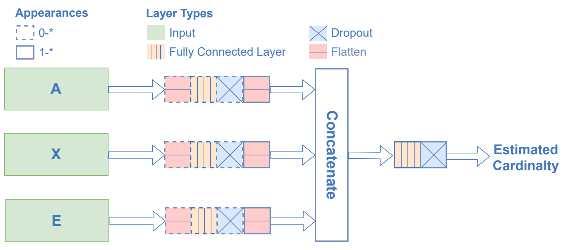


Figure 2: LMKG-S architecture

that $X \in \{0,1\}^{n*\lceil log_2 d \rceil}$ and $E \in \{0,1\}^{b*\lceil log_2 e \rceil}$, thus, directly reducing the space for the subgraph representation.

To create an SG-Encoding, an ordering of nodes and edges is required. They are sorted in ascending order according to their integer ids. As an example, consider the star query and its encoding shown in Figure 3. The encoding process is broken down into three main steps: In Step 1, executed in the creation phase (Section 4), a global mapping of the nodes and predicates to an integer id is created. Next, an ordering of nodes and predicates is created for the given query, where the global ids of the nodes and predicates in the knowledge graph determines their order in the query. Thus, for this example in Figure 3, we will have the ordering shown in Step 2 on the right-hand side. Finally, for a predefined $n = 3$ and $e = 2$, the parts of the encoding are created, resulting in tensor A and matrices X and E (Step 3). For the subpattern $?Book\ \ :hasAuthor\ \ :StephenKing.$, we set $A_{001} = 1$, indicating that $:hasAuthor$ is the first predicate in the edge order, the unbound term is the first node, and $:StephenKing$ the second node in the respective node order. The first two bits in E indicate that $:hasAuthor$ has id 1.

5.1.3 Model Design: Although different approaches exist for handling graph data in supervised deep learning, the usage of lightweight models results in faster execution and lower memory consumption. At the same time, they do work quite effectively when used for cardinality estimation in RDBMS [21, 37]. A standard way of modeling regression tasks is to use multilayer perceptron (MLP). The neural network consists of an input layer, an arbitrary number of hidden layers, chosen according to the input complexity, and an output layer. During training, the neural network optimizes itself based on the provided example queries with precalculated cardinalities. The architecture of LMKG-S is shown in Figure 2. The intuition behind it is combining typical subgraph representations and the capability of simple MLP models for efficient cardinality estimation. Depending on the input size and the need for regularization, some layers are optional. The cardinalities typically follow a skewed distribution where only several queries have large cardinalities. To deal with skewness, cardinalities are log scaled. We next perform min-max scaling, allowing usage of the numerically stable sigmoid function. The minimum and the maximum are the lowest and largest cardinality present in the training data, assuming they are available. The network is defined by $w_{out} = MLP(f(X), f(A), f(E))$ where f is either a flatten layer or another MLP, in which case the model resembles the Deep Sets [55] architecture. $f(X)$, $f(A)$, and $f(E)$ are concatenated and propagated through one or more layers. Every fully connected layer except the output layer uses ReLU, a non-linear activation function defined as $f(x) = max(0, x)$. The output layer uses a sigmoid function $f(x) = 1/(1 + e^{-x})$, having an output between 0 and 1, suitable for the already scaled values. The final output is rescaled in its original domain. As loss function, we use the mean q-error, defined as the relation between the true cardinality and the estimate: $q\_error(y, \hat{y}) = max(\hat{y}/y, y/\hat{y})$.

5.2 Unsupervised Model (LMKG-U)

A deep autoregressive model is an unsupervised deep learning model that efficiently estimates a joint distribution $P(x)$ from a set of samples. The autoregressive property dictates that, for a predefined variable ordering, the output of the model contains the density for each variable conditioned on the values of all preceding variables [10]. Therefore, given as input $x = [x_1, x_2, ..., x_d]$, the autoregressive model produces d conditionals which, when multiplied, will result in the point density $P(x)$. Formally, for $x = [x_1, x_2, ..., x_d]$, the probability is calculated as $P(x) = P(x_1)P(x_2|x_1)...P(x_d|x_1, ..., x_{d-1}) = \prod_{d=1}^{D} P(x_d|x_{<d})$.

Recently, the use of deep autoregressive models has been proposed as a way of estimating the query selectivities in relational databases [14, 51, 52]. We introduce LMKG-U by adapting autoregressive models for cardinality estimation in knowledge graphs. We next discuss the encodings used in LMKG-U, first, term encoding and subsequently a graph pattern encoding.

5.2.1 Term Encoding and Compression: Terms in LMKG-U can be encoded with the previously explained term encodings or an embedding capable of representing heterogeneous KGs. However, unlike the supervised model, the size of the autoregressive models is highly impacted by the input and output dimensionality. More specifically, each logit from the output of the model corresponds to a conditional probability of a term present in the query pattern. Thus, for highly heterogeneous knowledge graphs, the introduced encodings are ineffective since the model size will scale linearly with the term domain values. Even applying embeddings will result in large memory overhead in the case of highly heterogeneous knowledge graphs. To overcome this problem, similar to the column compression of the NeuroCard approach [51], we utilize a novel term compression.

The intuition behind the compression is to encode the input and output in fewer dimensions by representing the term ids through several smaller subterm ids. The number of subterm ids is determined based on the number of term ids such that it can lead to a smaller input dimensionality. For example, if there are 10000 terms, two subterms can already efficiently compress the original term ids, whereas, for 10 million terms, three or more subterms would be needed to enable the training of the model. Our compression is based on the observation that we can reduce the input dimensionality by dividing the term ids with a specific divisor. We start by first recognizing the number of subterms ns that a term should be split into. Then, we set as a divisor st_d to be the ns^{th} root of the number of distinct terms, i.e., $st_d = \lceil \sqrt[ns]{max\_tid} \rceil$. A term with id x is compressed by determining the quotient st_q and reminder st_r when dividing the term x with st_d. When $ns > 2$, the same procedure is repeated for $x = st_q$ and $max\_tid = max\_st_q$, at the end creating ns subterms.

As an example, consider that the number of distinct predicates max_tid is 60000 and we want to split the terms into two subterms $ns = 2$. For this setting, the divisor is $st_d = 245$. We want to compress the predicate x with id 5144. Thus, term x will be compressed in $st_q = 20$ and $st_r = 244$. In this case, all predicates will be split into two subterms having maximal ids st_d and $st_d - 1$. Consequently, with this compression, we reduce the number of dimensions for the input and output from 60000 to 489.

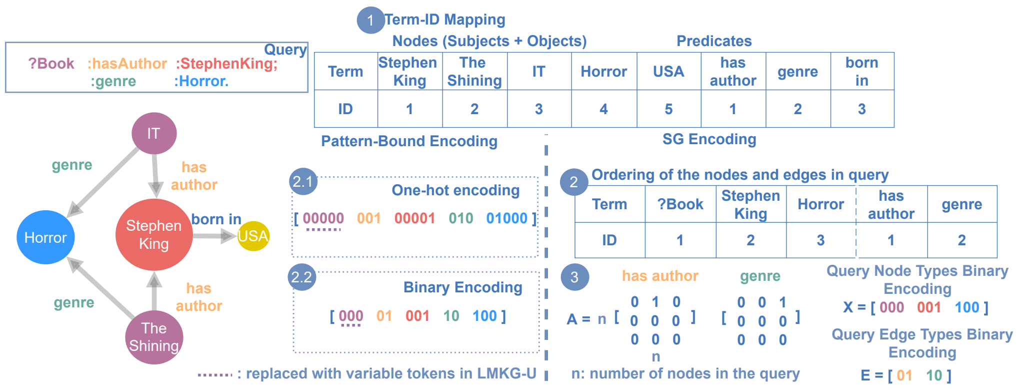

1 Term-ID Mapping
Nodes (Subjects + Objects)
Predicates
Term | Stephen King | The Shining | IT | Horror | USA | has author | genre | born in
ID | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3
Query
?Book :hasAuthor :StephenKing; :genre :Horror.
IT
genre
has author
Horror
Stephen King
born in
USA
genre
has author
The Shining
Pattern-Bound Encoding
2.1 One-hot encoding
[00000 001 00001 010 01000]
2.2 Binary Encoding
[000 01 001 10 100]
: replaced with variable tokens in LMKG-U
SG Encoding
2 Ordering of the nodes and edges in query
Term | ?Book | Stephen King | Horror | has author | genre
ID | 1 | 2 | 3 | 1 | 2
3 has author genre
A = n [0 1 0 / 0 0 0 / 0 0 0] [0 0 1 / 0 0 0 / 0 0 0]
n
Query Node Types Binary Encoding
X = [000 001 100]
Query Edge Types Binary Encoding
E = [01 10]
n: number of nodes in the query

Figure 3: Encoding example

It is observable that the proposed input compression affects not only the model size but also the cardinality estimation accuracy. This is a direct consequence of the setting for parameter *ns*. Although a larger *ns* would lead to fewer dimensions for encoding, the number of input terms will be increased. Thus, the accuracy of the cardinality estimates will be negatively affected since the model would need to learn across multiple input parameters with increased interconnection. Hence, we set the parameter *ns* to achieve an acceptable balance between the model size and the model accuracy. Although introduced for LMKG-U, the term compression can also be used for LMKG-S.

5.2.2 *Pattern-bound Encoding:*

This encoding works only for a model tailored to a certain type of query. For subject-oriented star patterns, the encoding requires ordering of the predicate-object pairs connected to the subject. More specifically, if we have a triple set of size k that is centered around a single unbound subject, the pattern-bound star encoding is a concatenation of all of the k pair encodings and the subject encoding. To demonstrate this encoding, we will make use of the example star-shaped query depicted in Figure 3. The pattern-bound encoding for this query is shown on the left-hand side of Figure 3. In the beginning, a mapping of the nodes and predicates to an integer id is created (Step 1). Then, every term of the query is encoded using either one-hot (Step 2.1) or binary encoding (Step 2.2).

In chain queries, the ordering of the nodes and edges is evident. Intuitively, a predicate connected to a subject is its descendant and its position in the order will follow the one of the subject. The same holds for the object which has as an antecedent the predicate. Given an ordering of the nodes and edges, we can encode the chain query as a concatenation of the term encodings.

Evidently, a serialization of an adjacency list resembles the pattern-bound encoding. In particular, a star pattern represented with a flattened adjacency list will directly correspond to the pattern-bound star encoding. However, for a chain query, an adjacency list will be larger than the pattern-bound encoding. This is because by knowing that an object in a triple will be the subject in the next one, in the pattern-bound encoding, redundant nodes can be eliminated, resulting in reduced encoding size.

When we consider the definition of the SG-Encoding, it is evident that by using only the matrices E and X without the tensor A we achieve the pattern-bound encoding. However, without tensor A the different types of queries or combinations of them cannot be represented in a single model. This is direct evidence of the importance of tensor A. Although we advocate the usage of two different encoding types, for heterogeneous knowledge graphs, because of the high number of unique terms, the SG-Encoding can contribute to increased complexity of the input. This increased input complexity, although minor, will have a negative effect on the estimation accuracy. Hence, the preferred encoding when using LMKG-U is the pattern-bound encoding.

5.2.3 *Model Design:*

In recent years, several autoregressive architectures have been proposed [7, 9]. As one of them, MADE [9] maintains the autoregressive property by adequately masking particular weights in the layers. In our work, we use ResMADE [7], an extension of MADE with residual connections. In autoregressive models, the cardinality for a query is estimated using the conditional probabilities of the terms in the query. For a query with k triples $qp_k = n_1, p_1, ..., p_k, n_{k+1}$, the density is estimated as $P(qp_k) = P(n_1)P(p_1|n_1)...P(n_{k+1}|n_1, p_1, ..., n_k, p_k)$. For a chain query, n_i is an object in the triple t_{i-1} and a subject in the triple t_i where $i \in [2, k]$. For a star query, n_i where $i \in [2, k+1]$, are objects connected to the subject n_1, with p_j for $j \in [1, k]$.

The autoregressive model learns the correlation between the present terms. For answering queries involving variables, we employ wildcard skipping [51, 52]. During training, a randomly chosen subset of columns has their values masked, i.e., replaced with special tokens representing a variable. During prediction, the input will constitute all values of the bound terms and the special tokens for the variables. The cardinality is estimated in a single forward pass of the input through the model. For instance, a forward pass for the triple query ?x :1 :2., will result in three conditionals $p(s = ?x)p(p = :1|s = ?x)$ and $p(o = :2|s = ?x, p = :1)$, which when multiplied, result in the probability for the given pattern. This probability is then multiplied with the size of the dataset (number of triples), resulting in cardinality. For star and chain queries, the point probability is multiplied by the join size.

6 HANDLING COMPLEX QUERIES

Although query optimizers often for simplicity put focus on the most typical queries for query optimization, cardinality estimation of complex queries is a significant problem. A complex query involves an arbitrary number of triples resulting in multiple interconnected star and chain patterns, e.g., snowflake queries or tree queries, which are arbitrary queries without cycles.

6.1 LMKG-S

For representing complex queries, LMKG-S uses the SG-Encoding explained in Section 5.1.2. More specifically, the nodes and the variables are retrieved from the complex query. For each unique node and variable, LMKG-S assigns a different id. The same procedure is performed for the predicates. The mapping between nodes and predicates and their ids is stored in X and E, respectively. The connection between the nodes through the predicates is stored in A, using the newly computed ids. Unlike LMKG-U, which works on the actual data, LMKG-S requires a workload of queries. Thus, for an example workload of complex queries, we show the performance of LMKG-S in the experimental evaluation.

6.2 LMKG-U

LMKG-U uses pattern-bound encoding, which theoretically can also be used to represent any complex pattern since it resembles a flattened adjacency list. However, unlike LMKG-S, a single model of LMKG-U can only represent a single type of query. For instance, for a tree query of a specific size, where the triples are always joined across the same terms, one model can be used. Since there are many variants of complex queries, and the terms for joining can be arbitrary, this will lead to many models all specialized for a specific query type, size, and with prespecified join conditions. Moreover, to find a representative training set, sampling across joins needs to be performed, which is still a challenging task for a large number of arbitrary joins. In addition, large complex queries may lead to many terms and thus, a large model. Therefore, for handling complex queries in LMKG-U, we suggest the re-usage of existing models for star and chain patterns.

The first step when dealing with complex queries is to decompose them into star and chain patterns. Intuitively, the decomposition may lead to different subqueries, and thus, different estimates. For LMKG-U, it is essential to decompose the query to as many possible subqueries for which a model exists. The decomposition starts in a greedy manner with a random star or chain pattern. Next, the triples from the pattern are removed from the query. This process is repeated until no triples are present. As a guideline, one should always choose the pattern whose model has the lowest error, preferably prioritizing larger patterns. In the experiments, we always decompose the queries such that all subqueries can be estimated by the existing models. Once the query is decomposed, we need to combine the individual estimates for a final cardinality estimate of the complex query.

A natural way to handle different query types is to use the existing models and estimate the cardinality through independence assumption. However, as already investigated by previous work [34], this can lead to large misestimates. To solve this problem, we suggest an approach that leverages the power of LMKG-U for purposes beyond the estimation of point densities. An autoregressive model can generate samples and infer values for specific terms conditioned on previously specified terms. Knowing this, we suggest an approach for cardinality estimation of complex queries that uses the capability of LMKG-U to create samples for specific terms. Consider a given query q decomposed into subqueries $sq_1, ..., sq_n$. The algorithm starts with the first subquery sq_1 and produces the cardinality estimate $est(sq_1)$ using the appropriate, existing model. As next, the algorithm samples a prespecified number of values for the term(s) that needs to be joined with the following subquery. We next continue with the subquery sq_2 where its join term with sq_1 is fixed to the generated samples. This iteration continues for the remaining subqueries, in the end, having several different paths and path probabilities. The idea of creating different paths for a query through sampling follows the WanderJoin algorithm [26]. To compute the path probability, as in WanderJoin, we use the unbiased Horvitz-Thompson estimator [17] with inverse probability weighting. The estimate of a path probability incorporates the subquery estimates $est(sq_i)$ computed by the individual models. The final estimate for the complex query is an average over the independent paths.

To avoid generating nonrelevant samples for the query terms when using rejection sampling, we employ likelihood-weighted forward sampling [22, 52]. Similar to Bayesian Networks, terms are sampled in a predefined order and are impacted by the previously created terms, in the end constructing a weighted particle. Weights incorporate the likelihood of already generated terms accumulated throughout the sampling. Thus, the new term value is impacted by the probability of already generated term values.

Consider a complex query formed from two subqueries, sq_{star} and sq_{chain}, joining on the object and subject, respectively. First, the cardinality of sq_{star} is estimated and samples for the object are generated. Each sample will result in a different path. Therefore, for each value in the sample, we fix the subject in the sq_{chain} query and use the respective model for estimating the cardinality.

For star and chain queries, LMKG-U already learns over the joins and it can directly estimate their cardinality. However, WanderJoin, even for the subqueries, samples over the join predicates, and its estimate depends on the sample quality. Therefore, when considering complex queries, LMKG-U produces better subquery cardinality estimates than WanderJoin. However, the samples for the joining terms will be at most as good as WanderJoin. Still, the worse sample quality is not unexpected since the model can produce wrong estimates, whereas WanderJoin always uses correct values drawn from the graph. When the sampling in LMKG-U produces invalid samples, the respective paths have a probability of 0. Therefore, if LMKG-U cannot find representative samples, it falls back to the independence assumption.

7 TRAINING AND MODELS OVERVIEW

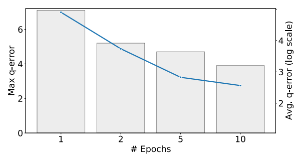
7.1 Training Data Creation

The virtually endless combinations of queries emerging from a KG directly impact the training data creation and the accuracy of the model. For smaller queries and homogeneous KGs, the complete set of patterns for a specific size can be created. However, as the size increases, the creation of all possible patterns of size k creates an intractable problem. Therefore, to generate training data, proper sampling has to be conducted. This sampling has to satisfy the scaled-down property of the knowledge graph. This means that the samples that are generated should exhibit similar properties as the original knowledge graph [25]. As Leskovec and Faloutsos [25] show, this can be achieved by random walk (RW) sampling. We simulate RW sampling by generating different subgraphs matching the query types (avoiding cycles). For star patterns of size k, we randomly pick a node and then simulate a random step k times from the starting node. For chain patterns, we start a random walk from a randomly selected node and stop once the required size is reached. In the end, this results in a large sample representing the incomplete join of the triples. For LMKG-S, we randomly replace bound terms with variables from the created samples. Although we use RW sampling for generating training data, efficient sampling in KGs is a challenging task and, as further shown, the main cause of inaccurate estimates is the quality of the samples, especially visible in KGs with many unique terms.

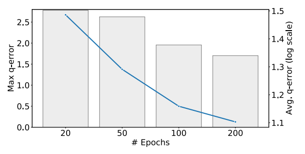
Table 1: Experiment (left) and dataset details (right)

Dataset	SWDF, LUBM20, YAGO
Topology	Chain, Star
Result Size	$[5^0, 5^1], [5^1, 5^2], \ldots$
Query Size	2, 3, 5, 8

Dataset	SWDF	LUBM20	YAGO
Triples	250K	2.7M	15M
Entities	76K	663K	1.7M
Predicates	171	19	91



(a) LMKG-U (sample from LUBM)



(b) LMKG-S (sample from LUBM)

Figure 4: Training time vs. accuracy - bars show max q-error; dots show avg. q-error

7.2 Grouping of Models and Analysis

LMKG can create compounds of models grouped by different criteria. The grouping mainly affects the accuracy and the creation time of the models. LMKG allows the choice between:

A **Single Learned Model** that can be used for the complete knowledge graph and all types of queries, including star-shaped and chain-shaped queries with up to k number of joins. This grouping is suitable for small memory budgets and homogeneous and smaller KGs. It requires less tuning and maintenance during the run-time phase. However, it may produce lower accuracy for heterogeneous KGs due to the high number of patterns that affect the quality of the samples. For this grouping, the user query is directly forwarded to the model.

Query Type Grouping that creates separate models, each specialized for different query types. This grouping can use both subgraph encodings. It enables parallel creation of training data and models, leading to a drastic time reduction. However, having multiple models may lead to increased memory consumption. Additionally, during the execution phase, the query needs to be routed to the appropriate model responsible for the query type.

Query Size Grouping that creates a single model for a range of queries, grouped by their size, e.g., one model can be created for patterns up to size 4 and another for larger ones. This grouping has the same benefits and downsides as the query type grouping.

Besides the different groupings, LMKG offers the creation of different estimators characterized by the specific encoding and the type of learning. We next delineate the benefits and the downsides of the models along these dimensions and how they address the challenges in cardinality estimation in KGs.

Encodings: Both SG-Encoding and pattern-bound encoding can address two of the challenges. They capture term intercorrelations for a specific query by not leaving out any terms in the encoding and both can express many self-joins, assuming that one-hot term encoding is not used.

Pattern-bound Encoding: It is simple to implement and has a small dimensionality since it is tuned to a specific query. However, when the patterns contain reoccurring nodes or predicates, they may be repeated in the encoding, leading to higher dimensionality. The encoding is not generalizable to different query types, thus, it requires the creation of multiple models and needs higher maintenance.

SG-Encoding: Unlike the pattern-bound encoding, it addresses the third challenge since it can simultaneously represent different query types. However, it can have a larger dimensionality, especially noticeable when all the terms in the query are unique.

Supervision: Both models capture term correlations but are highly impacted by the training sample quality. LMKG-S needs to capture enough representative queries which describe the workload. The samples for LMKG-U need to have the same ratio as the original data which can be challenging for larger dimensions.

LMKG-S has a faster training and prediction time, as well as a smaller memory footprint. However, the training data creation is more time consuming since variables need to be included. Additionally, generalizing to queries that are far from the training data is challenging and somewhat impacted by the slight overfit for better results, also leading to worse estimates for outliers.

LMKG-U can create training data faster since the model learns only from bound terms. It also captures the term intercorrelations better than LMKG-S, producing highly accurate cardinality estimates especially suitable for skewed datasets. However, it has a larger memory footprint and higher training and prediction time than LMKG-S. This is especially apparent when handling heterogeneous datasets that have a high number of unique terms.

Guidelines for Combining Models: Although the groupings allow the creation of different models, we briefly mention potential guidelines based on our analysis. Combining models depends on the overall model creation budget as well as the dataset characteristics. When there is a memory constraint, a single LMKG-S model is preferred or two query-type–grouped LMKG-U models in combination with term compression. In cases where a smaller model creation time is needed, LMKG-S is preferred over LMKG-U. If the training data is smaller, LMKG-U can still be considered, even in combination with LMKG-S. If there are no training time constraints, then data characteristics should be examined. For instance, for star queries over datasets having only several nodes with a huge in- or out-degree, i.e., outliers, LMKG-U is preferred. However, when many rare terms appear and we are working with chain queries, training data sampling may be worse for LMKG-U. Thus, in a situation where these two cases appear, a combination of both may be beneficial.

8 EXPERIMENTS

Setup: The experiments for the learned models, implemented in TensorFlow and PyTorch, are carried out on an NVidia GeForce RTX 2080 Ti GPU. The competing approaches are evaluated on a server with an Intel Xeon E7-4830 v3 CPU @ 2.10GHz, 1 TB RAM. For LMKG-U, we adapt and extend the publicly available autoregressive model from Naru [52]. In Table 1 (left), we show the specifications for the experiments. For generating test queries, we group them into buckets depending on the query result size, with boundaries defined by *log* with base 5. For examining different complexities, we chose a query size of 2, 3, 5, and 8 triples, with at least 1 unbound term. For a specific query size and shape, we select 600 test queries where each query is drawn from a bucket for a specific result size. We limit the graph patterns to include only bound predicates due to the competitors' limitations to answer queries with unbound ones. We generate training subgraph patterns of sizes 2, 3, 5, and 8 for all considered datasets according to the explanation in Section 7.1, where the sample size depends on the dataset characteristics.

Datasets: We use one synthetic and two real-world datasets whose characteristics are shown on the right-hand side of Table 1. The SWDF [33] dataset contains fewer triples, but has a high number of interconnections between the terms. We use LUBM [12] , a widely used RDF benchmark, with a scaling factor 20 and YAGO 1 [45], as a larger knowledge base, chosen for

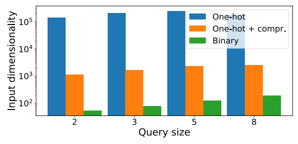


Figure 5: Encodings impact

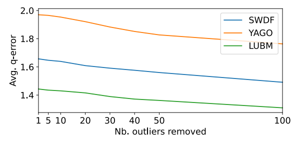
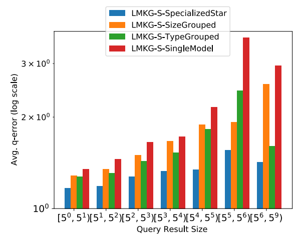
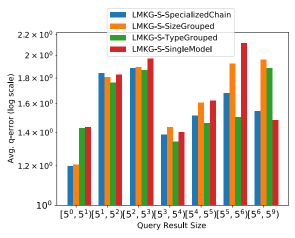


Figure 6: Outliers impact



(a) Star queries



(b) Chain queries

Figure 7: Avg. q-error comparison between specialized and combined models (LUBM)

its large number of distinct term values. The cardinality of the queries follows a skewed distribution and the vast amount of queries have a small cardinality. Moreover, there are extremely large cardinalities, i.e., outliers, which highly impact the models' accuracy.

Competitors: We investigated several approaches that can be applied to our problem, ranging from summary-based or sampling-based methods to recent deep learning approaches. As competitors, we use the approaches in the recently developed benchmarking framework G-CARE [38], the relational learned estimators MSCN [21] and the very recently proposed NeuroCard [51]. *From the family of summary-based approaches:* **Characteristic sets (CSET) [34]:** summarizes entities based on their emitting edges and it is specifically tailored for star queries. **SUMRDF [42]:** estimates the cardinality by relying on the possible world semantics. *From the family of sampling-based approaches:* **Wander-Join [26]:** performs random walks by considering each triple as a vertex and a join as an edge. **Impr [5]:** uses random walks for estimating graphlet counts. **JSUB [57]:** performs random walk sampling over joins, adapted for producing upper bound estimates of the cardinality. *From the family of learning-based approaches:* **MSCN-n [21]:** a multi-set convolutional network using query features as sets and n materialized samples. To train on the same queries as LMKG-S, MSCN learns self-joins over single tables. We use MSCN-0 and MSCN-1k with 0 and 1000 samples, respectively. **NeuroCard [51]:** an autoregressive join cardinality estimator used to perform self-joins over single tables.

For all competitors except for CSET, we use the publicly available implementations [21, 38, 51]. Due to observed drastic overestimates produced by the CSET implementation in G-CARE, following the reference paper, we reimplemented the algorithm. Some approaches in G-CARE are native to knowledge graphs and others are adapted from RDBMS. Like in G-CARE, sampling approaches are executed 30 times. The results are an average over the 30 samples.

8.1 Analysis of LMKG

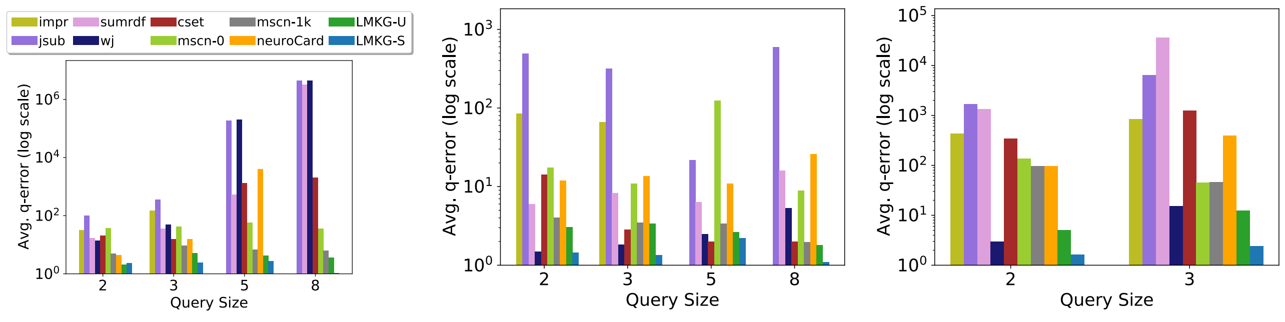
8.1.1 Hyperparameter Tuning: We conducted experiments varying hyperparameters such as epochs, hidden units, and layers. Figure 4 shows how two accuracy metrics change through the training process. For our experiments, we choose either 100 or 200 epochs for LMKS-S and 5 or 10 epochs for LMKG-U. For LMKG-S, for SWDF, LUBM, and YAGO, on average, an epoch takes 13, 16, and 48 seconds, respectively. The training time is directly affected by the sample size. Due to higher complexity in the presence of numerous terms, LMKG-U requires a longer training time. For the sample size considered, for SWDF, LUBM, and YAGO on average, an epoch takes 10, 4, and 20 minutes, respectively. For a larger sample size and many unique terms, an epoch can take up to 36 and 13 minutes, for SWDF and LUBM, respectively. Note that this is a substantial improvement over LMKG-U without compression, where an epoch can take up to 50 minutes for the considered datasets. We varied the number of hidden units (256, 512, 1024) and hidden layers (2–4) depending on the dataset characteristics. We found that **2 or 3 layers of 512 neurons** are a good choice for both models.

8.1.2 Impact of Encoding and Compression: In Figure 5, we show the dimensionality of the encodings for the SWDF dataset when varying the join size of star patterns. For LMKG-U, the output has to be either an embedding or a one-hot encoding since the autoregressive model has to output a single conditional probability per term. However, having heterogeneous datasets with many distinct values per term will create huge embedding matrices and large one-hot encoding vectors (Figure 5), rendering LMKG-U not executable. Therefore, the only option that enables the execution of LMKG-U is term compression. As shown, for $ns = 2$ the compression produces a satisfiable reduction in the dimensionality, and with that, the memory consumption of LMKG-U.

Differently, as an input of LMKG-S, the terms can use any of the previously discussed encodings, even an embedding. Since one of the benefits of a learned estimator is the reduction in memory consumption, we aim at choosing the encoding which produces the smallest model with a marginal accuracy decrease. Thus, we choose the smallest encoding, i.e., binary encoding.

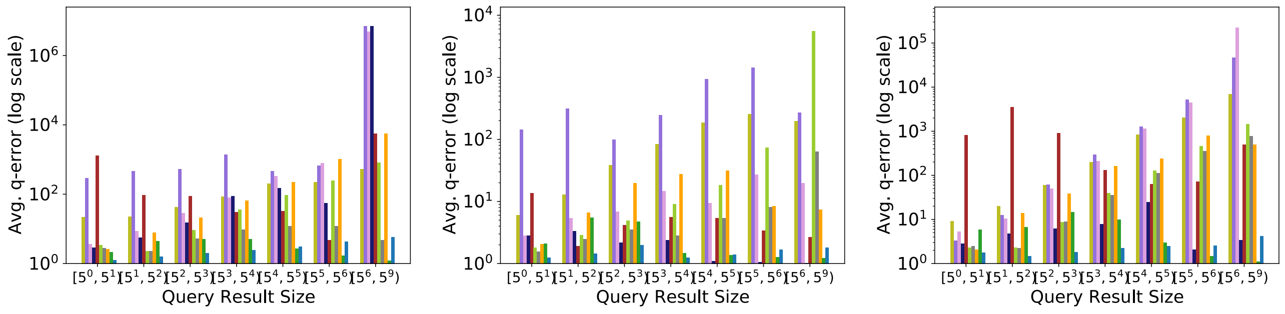
8.1.3 Impact of Outliers: Upon measuring the models' accuracy, it is evident that unlike LMKG-U, whose accuracy is impacted by the domain values of the terms, LMKG-S is extremely affected by outliers. Therefore, in Figure 6 we measure the influence of outliers in star queries, where the impact of outlier removal is most evident. We can see that even if we remove the top-10 outliers from the query data, we achieve a higher accuracy of the model. This trend continues when a larger fraction of the outliers is removed. Although we apply normalization and scaling of the data, the impact of the skewness is still evident. Therefore, given a larger space budget, a possible improvement can be to store the cardinalities of the outliers on the side. For a fair comparison, we proceed without this improvement.

8.1.4 Impact of Grouping: In Figure 7, we show the accuracy of the following LMKG-S models: specialized model for queries of specific type and size, model grouped by size, model grouped by type, and model for every query type and size (SingleModel). We stop after 50 epochs, where every model has two layers and the same configuration. For almost every case, the specialized model overfits the queries and produces the best estimates. The single model, as expected, has the lowest estimation accuracy. Knowing that it trains on a much larger dataset, this accuracy can be acceptable, especially when having a small memory budget. Models grouped by size and type produce good estimates although worse than the specialized model. Based on the accuracy comparison and since the number of specialized models



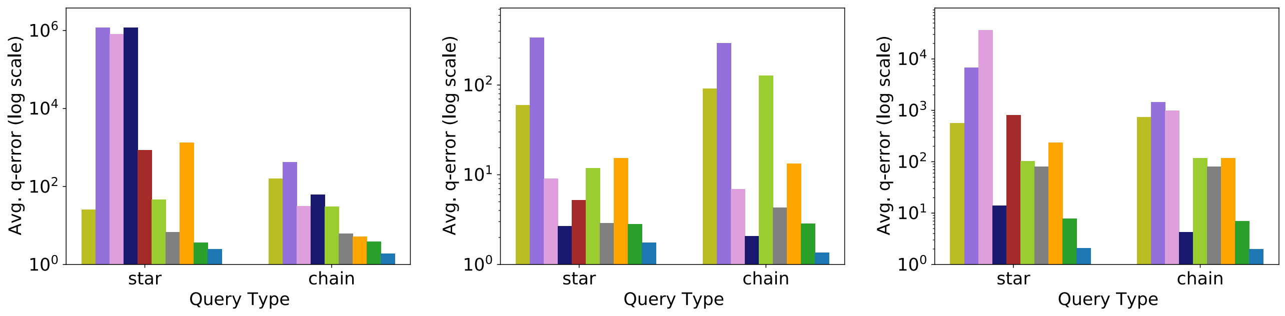
(a) SWDF **(b) LUBM** **(c) YAGO**

Figure 8: Accuracy for query size



(a) SWDF **(b) LUBM** **(c) YAGO**

Figure 9: Accuracy for query result size



(a) SWDF **(b) LUBM** **(c) YAGO**

Figure 10: Accuracy for query type

needed for the various combinations is significantly higher than for the grouped ones, *for the experimental analysis, we chose the query size grouping*.

8.2 Comparison with Competitors

For the comparison with the competitors, for LMKG-S, we used **SG-Encoding** and **query size grouping** (Section 7.2). For LMKG-U, we used **pattern-bound encoding with binary input encoding and term compression** (Section 5.2). We report the accuracy for different query types, query sizes, and query result sizes. For both LMKG models and the learned competitors we tune the architecture and vary the hyperparameters according to the dataset. All models are trained until convergence and the best models are shown.

8.2.1 Accuracy Varying Query Size: Figure 8 depicts the accuracy of the individual approaches when varying the number of joins present in queries. The accuracy is represented through the average q-error. Unlike the others, whose accuracy declines for a larger number of joins, LMKG-S is not impacted by this factor. As depicted, LMKG-U with compression shows a constant performance. However, if the compression was not present, the accuracy of LMKG-U would have been drastically impacted by the large number of terms present in larger queries. The accuracy is also slightly impacted by the quality of the sample for training, which for some datasets is still a challenging task. When compared to NeuroCard (estimates not shown for size 8 due to an out-of-memory error), LMKG-U performs better. This is a result of the sample quality, which is specifically tailored for self joins over knowledge graphs and not RDBMS. When considering the different query sizes, both LMKG-S and LMKG-U perform better than the competitors. As previously explained, when there is no query workload present, we need to generate a large set of representative queries directly from the knowledge graph. Because

Table 2: Memory consumption of different approaches

Dataset	LMKG-U			LMKG-S			SUMRDF	CSET	MSCN	NeuroCard		
	$k = 2$	$k = 3$	$k = 5$	$k = 2$	$k = 3$	$k = 5$	Complete	Complete	0/1K	$k = 2$	$k = 3$	$k = 5$
SWDF	7.5MB	9MB	12MB	4MB	4MB	8MB	1.2MB	816.7 KB	5 / 8 MB	2.8MB	4MB	33MB
LUBM	13 MB	17MB	26MB	4MB	4MB	5MB	8.8MB	8.6 KB	5 / 8 MB	2.8MB	16MB	40MB
YAGO	37MB	50MB	—	4MB	7MB	7MB	342MB	5MB	5 / 8 MB	4MB	10MB	—

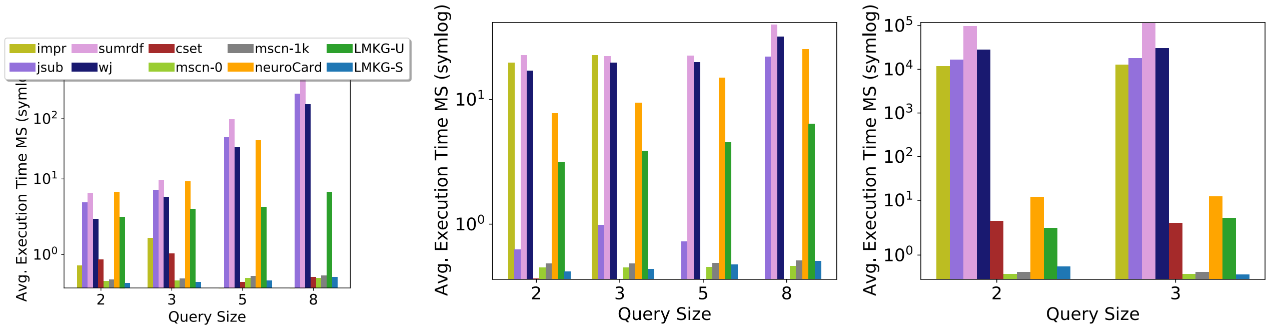


Figure 11: Estimation time when varying query size (SWDF, LUBM and YAGO)

Table 3: Memory consumption of LMKG-U

Dataset	with comp.			without comp.		
	$k = 2$	$k = 3$	$k = 5$	$k = 2$	$k = 3$	$k = 5$
SWDF	7.5MB	9MB	12MB	19MB	43MB	46MB
LUBM	13 MB	17MB	26MB	80 MB	78MB	27MB
YAGO	37MB	50MB	—	—	—	—

of term interconnections in YAGO, it was impossible to create an acceptable amount of representative subgraph patterns from all cardinality ranges of size five and above.

8.2.2 *Accuracy Varying Query Result Size:* Figure 9 shows the accuracy for different query result sizes. Each range contains the same number of queries except for the last ones, where the patterns are sparse. The last buckets are grouped for larger ranges involving the outliers. When varying the query result size, we can most clearly see where the LMKG-S approach fails. LMKG-S is highly prone to outliers, visible for the higher ranges. This is especially visible in YAGO, where the term and dataset size is larger, having larger cardinalities, and thus, a higher possibility for error. Hence, LMKG-S is mainly impacted by the skewness. LMKG-U produces more constant results throughout the ranges. However, for larger datasets, due to sampling reasons, LMKG-U fails to capture the interdependencies between the rarely occurring terms. This is more evident in the smaller ranges, especially visible for YAGO. MSCN represents the predicate values with a single feature which is suitable for relational data. However, this is not adequate for large domain values, especially for larger ranges. MSCN-1K performs better, however, a small sample is still not able to capture the KG's query diversity. On the contrary, our approaches give emphasis on the term values and patterns and provide better estimates. As pointed out, the sampling is a direct reason for the worse accuracy estimates of NeuroCard. When comparing with the existing KG approaches, overall, LMKG-S is always better for smaller ranges, followed by LMKG-U, WJ, and MSCN-1k. CSET and WJ perform better for larger result sizes, however, they are inferior for smaller ranges. Regarding the overall performance, LMKG produces the best accuracy.

8.2.3 *Accuracy Varying Query Topology:* Figure 10 reports the accuracy that is measured for star and chain queries of different sizes. LMKG-S and LMKG-U almost always perform best for both query types. WJ and MSCN-1k perform well and in some cases even outperform LMKG-U. As depicted, both LMKG models are not majorly impacted by the query type, as in the case of some of the other approaches. However, as shown, the LMKG models are impacted by the number of unique terms in the datasets.

8.2.4 *Memory Consumption:* We compare LMKG with the two summary approaches, MSCN and NeuroCard (Table 2). We measure the size of the complete model for LMKG-U and LMKG-S. Although for LMKG-U and LMKG-S a model for size *k* can answer smaller queries we make the following table for completeness. Intuitively, the sampling approaches have an advantage since they directly use the KG and the respective indices. However, this causes a problem in cases where the KG is not available.

LMKG-S has smaller memory than LMKG-U and some of the competitors. MSCN-0 has a smaller footprint due to the much smaller input, at the cost of worse accuracy. CSET is better for SWDF and LUBM, however, for YAGO it has a larger size. The memory of LMKG-U increases with the number of involved terms. Compared to NeuroCard, LMKG-U requires more memory. This is not a result of a superior term compression but due to different encodings used. NeuroCard uses embedding on top of the input, whereas LMKG-U uses only binary encoding.

In Table 3, we show the memory of LMKG-U with and without compression, where LMKG-U without compression has an embedding layer that further reduces the model size. It is evident that the term compression has a great impact on the memory and in some cases, the memory consumption is reduced by more than 3 times. Furthermore, LMKG-U cannot even train on YAGO without compression due to the large number of unique terms.

8.2.5 *Estimation Time:* In Figure 11, we show the estimation time for different types of queries, depending on their size. For the sampling approaches in G-CARE, we measure the time of generating 30 samples required for producing an accurate estimate. A smaller sample size led to much worse accuracy, but

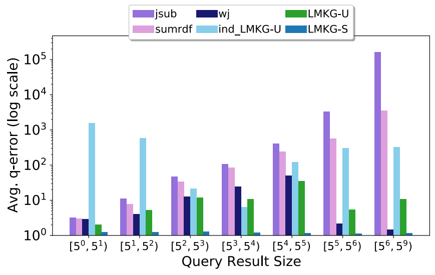


Figure 12: Accuracy for complex queries (SWDF)

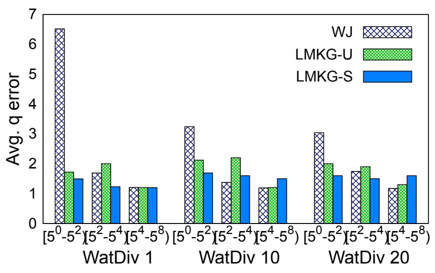


Figure 13: Scalability Experiment (WatDiv)

intuitively, a faster approach. MSCN has a similar prediction time as LMKG-S. LMKG-S performs better than all approaches, except for CSET. LMKG-U outperforms both NeuroCard and the sampling approaches since it requires only a single forward pass and not generation of samples.

8.2.6 *Handling Complex Queries:* As complex queries, we consider all query types that do not contain cycles, i.e., queries created by joining star and chain queries such as snowflake and tree queries. The complex queries were created from the existing chain and star queries for the SWDF dataset by performing joins on either the subjects, subject-object pairs, or objects. By performing a different number of joins and joining on different terms, we created queries of various types. The queries consist of between 3 and 10 triples. In Figure 12, we present the average accuracy for all the considered complex query types by grouping the results for different query result sizes. We do not compare against MSCN and NeuroCard. MSCN cannot be efficiently trained because we cannot create samples from different tables that satisfy multiple join conditions. For NeuroCard, we need to create an endless combination of models depending on the join partners. For LMKG-U, the results are obtained by using the same models as for the other experiments and for LMKG-S we train a model for the new query workload. The cardinalities are estimated according to the explained approaches in Section 6. For LMKG-U, we also evaluate the accuracy when using independence assumption labeled as *ind_LMKG-U* in Figure 12.

The results in Figure 12 show that the LMKG models produce distinctively accurate cardinality estimates even when handling complex queries. LMKG-S outperforms all the competitors and delivers the best cardinality estimates for every range for the considered query types. Additionally, the importance of our proposed estimation approach for LMKG-U is visible when comparing the results to estimates created from following the independence assumption (ind_LMKG-U). Thus, with the suggested approach for complex queries, LMKG-U produces estimates that are comparable to the best competitor WJ throughout the query ranges.

8.2.7 *Scalability:* We evaluated the ability of our approaches to handle scalability in terms of schema diversity and compared the results to the best competitor, i.e., WJ. To perform this experiment, we used the WatDiv Benchmark [1], which is developed for testing RDF data management systems for diverse queries and varied workloads. We varied the scale factor to 1, 10, and 20, where for every dataset, we generated 600 test queries from the same query sizes as the other experiments and grouped them into buckets depending on their result size. In Figure 13, we show the accuracy of the approaches by grouping the results on the query result size for the different scaling factors. The results further underpin the suitability of our cardinality estimation approaches. LMKG-S constantly has the best performance and always produces the most accurate cardinality estimates, except for the outliers present in the higher ranges. Still, even for these ranges, LMKG-S results in comparable accuracy to the best competitor. LMKG-U, as already pointed out, is not affected by the outliers and thus, always has better or comparable results to WJ. It is noticeable that the LMKG models drastically outperform WJ for smaller ranges resulting in a more consistent performance.

8.3 Lessons Learned

The major performance degradation in LMKG-S is not a result of the query complexity but of large outliers. To improve the accuracy for outliers, we suggest the usage of a buffer list. Although LMKG-U has a more constant performance, when having many unique term values, causing numerous correlations, the current sampling has not always proved efficient. Future work involves exploring different sampling for LMKG-U.

Evidently, LMKG requires a higher training time than creating the other approaches. Considering all aspects of query cardinality estimation in KGs, LMKG is well-suited for scenarios where a workload is given or a cardinality for a specified range of queries (e.g., up to k joins) is needed. Additionally, LMKG is practically useful when considering query optimization, where a reordering of different patterns of smaller sizes is required. LMKG is also useful for cardinality estimation of complex queries however the scaling depends on the training data. Thus, a combination of a sampling and a learned approach may be more efficient.

9 CONCLUSION

We addressed the problem of applying deep learning methods for cardinality estimation in knowledge graphs by utilizing both supervised and unsupervised deep learning models. To efficiently feed knowledge subgraphs to our models, we investigated various encodings, utilized a novel term compression, highly beneficial for LMKG-U, and an encoding that is especially useful for LMKG-S. By focusing on the subgraph patterns that constitute the KG, our encodings drastically reduce the input size and enable us to train the models on more than one query type. We analyzed the suitability of our models in different scenarios and explained possible improvements to our framework. Through the experimental evaluation, we showed that LMKG exceeds the state-of-the-art approaches in terms of accuracy while keeping a small memory footprint and requiring less time for generating the estimates.

ACKNOWLEDGMENTS

This work has been partially funded by the German Federal Ministry of Education and Research under grant number 28DE113C18 (DigiVine). The responsibility for the content of this publication lies with the authors.

REFERENCES

[1] Günes Aluç, Olaf Hartig, M. Tamer Özsu, and Khuzaima Daudjee. 2014. Diversified Stress Testing of RDF Data Management Systems. In *ISWC (1) (Lecture Notes in Computer Science)*, Vol. 8796. Springer, 197–212.

[2] Angela Bonifati, Wim Martens, and Thomas Timm. 2017. An Analytical Study of Large SPARQL Query Logs. *Proc. VLDB Endow.* 11, 2 (2017), 149–161.

[3] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *NIPS*. 2787–2795.

[4] Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M. Bronstein. 2020. Improving Graph Neural Network Expressivity via Subgraph Isomorphism Counting. *CoRR* abs/2006.09252 (2020).

[5] Xiaowei Chen and John C. S. Lui. 2016. Mining Graphlet Counts in Online Social Networks. In *ICDM*. IEEE Computer Society, 71–80.

[6] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. 2020. Can Graph Neural Networks Count Substructures?. In *NeurIPS*.

[7] Conor Durkan and Charlie Nash. 2019. Autoregressive Energy Machines. In *ICML (Proceedings of Machine Learning Research)*, Vol. 97. PMLR, 1735–1744.

[8] Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek R. Narasayya, and Surajit Chaudhuri. 2019. Selectivity Estimation for Range Predicates using Lightweight Models. *Proc. VLDB Endow.* 12, 9 (2019), 1044–1057.

[9] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. 2015. MADE: Masked Autoencoder for Distribution Estimation. In *ICML (JMLR Workshop and Conference Proceedings)*, Vol. 37. JMLR.org, 881–889.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

[11] Andrey Gubichev and Thomas Neumann. 2014. Exploiting the query structure for efficient join ordering in SPARQL queries. In *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014*, Sihem Amer-Yahia, Vassilis Christophides, Anastasios Kementsietsidis, Minos N. Garofalakis, Stratos Idreos, and Vincent Leroy (Eds.). OpenProceedings.org, 439–450. https://doi.org/10.5441/002/edbt.2014.40

[12] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. 2005. LUBM: A benchmark for OWL knowledge base systems. *J. Web Semant.* 3, 2-3 (2005), 158–182.

[13] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*. 1024–1034.

[14] Shohedul Hasan, Saravanan Thirumuruganathan, Jees Augustine, Nick Koudas, and Gautam Das. 2020. Deep Learning Models for Selectivity Estimation of Multi-Attribute Queries. In *SIGMOD Conference*. ACM, 1035–1050.

[15] Rojeh Hayek and Oded Shmueli. 2020. Improved Cardinality Estimation by Learning Queries Containment Rates. In *EDBT*. OpenProceedings.org, 157–168.

[16] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: Learn from Data, not from Queries! *Proc. VLDB Endow.* 13, 7 (2020), 992–1005.

[17] Daniel G Horvitz and Donovan J Thompson. 1952. A generalization of sampling without replacement from a finite universe. *Journal of the American statistical Association* 47, 260 (1952), 663–685.

[18] Hai Huang and Chengfei Liu. 2011. Estimating selectivity for joined RDF triple patterns. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011*, Craig Macdonald, Iadh Ounis, and Ian Ruthven (Eds.). ACM, 1435–1444. https://doi.org/10.1145/2063576.2063784

[19] Louis Jachiet, Pierre Genevès, and Nabil Layaïda. 2017. Optimizing SPARQL query evaluation with a worst-case cardinality estimation based on statistics on the data. (May 2017). https://hal.archives-ouvertes.fr/hal-01524387 working paper or preprint.

[20] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. 2020. A Survey on Knowledge Graphs: Representation, Acquisition and Applications. *CoRR* abs/2002.00388 (2020).

[21] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. 2019. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In *CIDR*. www.cidrdb.org.

[22] Daphne Koller and Nir Friedman. 2009. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press.

[23] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph M. Hellerstein, and Ion Stoica. 2018. Learning to Optimize Join Queries With Deep Reinforcement Learning. *CoRR* abs/1808.03196 (2018).

[24] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *Proc. VLDB Endow.* 9, 3 (2015), 204–215. https://doi.org/10.14778/2850583.2850594

[25] Jure Leskovec and Christos Faloutsos. 2006. Sampling from large graphs. In *KDD*. ACM, 631–636.

[26] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. 2016. Wander Join: Online Aggregation via Random Walks. In *SIGMOD Conference*. ACM, 615–629.

[27] Henry Liu, Mingbin Xu, Ziting Yu, Vincent Corvinelli, and Calisto Zuzarte. 2015. Cardinality estimation using neural networks. In *CASCON*. IBM / ACM, 53–59.

[28] Xin Liu, Haojie Pan, Mutian He, Yangqiu Song, Xin Jiang, and Lifeng Shang. 2020. Neural Subgraph Isomorphism Counting. In *KDD*. ACM, 1959–1969.

[29] Angela Maduko, Kemafor Anyanwu, Amit P. Sheth, and Paul Schliekelman. 2007. Estimating the cardinality of RDF graph patterns. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy (Eds.). ACM, 1233–1234. https://doi.org/10.1145/1242572.1242782

[30] Ryan Marcus and Olga Papaemmanouil. 2019. Towards a Hands-Free Query Optimizer through Deep Learning. In *CIDR*. www.cidrdb.org.

[31] Ryan C. Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A Learned Query Optimizer. *Proc. VLDB Endow.* 12, 11 (2019), 1705–1718.

[32] Marios Meimaris, George Papastefanatos, Nikos Mamoulis, and Ioannis Anagnostopoulos. 2017. Extended Characteristic Sets: Graph Indexing for SPARQL Query Optimization. In *ICDE*. IEEE Computer Society, 497–508.

[33] Knud Möller, Tom Heath, Siegfried Handschuh, and John Domingue. 2007. Recipes for Semantic Web Dog Food - The ESWC and ISWC Metadata Projects. In *ISWC/ASWC (Lecture Notes in Computer Science)*, Vol. 4825. Springer, 802–815.

[34] Thomas Neumann and Guido Moerkotte. 2011. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, Serge Abiteboul, Klemens Böhm, Christoph Koch, and Kian-Lee Tan (Eds.). IEEE Computer Society, 984–994. https://doi.org/10.1109/ICDE.2011.5767868

[35] Thomas Neumann and Gerhard Weikum. 2010. The RDF-3X engine for scalable management of RDF data. *VLDB J.* 19, 1 (2010), 91–113. https://doi.org/10.1007/s00778-009-0165-y

[36] Jennifer Ortiz, Magdalena Balazinska, Johannes Gehrke, and S. Sathiya Keerthi. 2018. Learning State Representations for Query Optimization with Deep Reinforcement Learning. In *DEEM@SIGMOD*. ACM, 4:1–4:4.

[37] Jennifer Ortiz, Magdalena Balazinska, Johannes Gehrke, and S. Sathiya Keerthi. 2019. An Empirical Analysis of Deep Learning for Cardinality Estimation. *CoRR* abs/1905.06425 (2019).

[38] Yeonsu Park, Seongyun Ko, Sourav S. Bhowmick, Kyoungmin Kim, Kijae Hong, and Wook-Shin Han. 2020. G-CARE: A Framework for Performance Benchmarking of Cardinality Estimation Techniques for Subgraph Matching. In *SIGMOD Conference*. ACM, 1099–1114.

[39] Mariya Popova, Mykhailo Shvets, Junier Oliva, and Olexandr Isayev. 2019. MolecularRNN: Generating realistic molecular graphs with optimized properties. *CoRR* abs/1905.13372 (2019). arXiv:1905.13372 http://arxiv.org/abs/1905.13372

[40] Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. 2020. GraphAF: a Flow-based Autoregressive Model for Molecular Graph Generation. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=S1esMkHYPr

[41] E. Patrick Shironoshita, Michael T. Ryan, and Mansur R. Kabuka. 2007. Cardinality estimation for the optimization of queries on ontologies. *SIGMOD Rec.* 36, 2 (2007), 13–18. https://doi.org/10.1145/1328854.1328856

[42] Giorgio Stefanoni, Boris Motik, and Egor V. Kostylev. 2018. Estimating the Cardinality of Conjunctive Queries over RDF Data Using Graph Summarisation. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, Pierre-Antoine Champin, Fabien L. Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis (Eds.). ACM, 1043–1052. https://doi.org/10.1145/3178876.3186003

[43] Michael Stillger, Guy M. Lohman, Volker Markl, and Mokhtar Kandil. 2001. LEO - DB2's LEarning Optimizer. In *VLDB*. Morgan Kaufmann, 19–28.

[44] Markus Stocker, Andy Seaborne, Abraham Bernstein, Christoph Kiefer, and Dave Reynolds. 2008. SPARQL basic graph pattern optimization using selectivity estimation. In *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, Jinpeng Huai, Robin Chen, Hsiao-Wuen Hon, Yunhao Liu, Wei-Ying Ma, Andrew Tomkins, and Xiaodong Zhang (Eds.). ACM, 595–604. https://doi.org/10.1145/1367497.1367578

[45] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2008. YAGO: A Large Ontology from Wikipedia and WordNet. *J. Web Semant.* 6, 3 (2008), 203–217.

[46] Behrooz Tahmasebi and Stefanie Jegelka. 2020. Counting Substructures with Higher-Order Graph Neural Networks: Possibility and Impossibility Results. *CoRR* abs/2012.03174 (2020).

[47] Maria-Esther Vidal, Edna Ruckhaus, Tomas Lampo, Amadís Martínez, Javier Sierra, and Axel Polleres. 2010. Efficiently Joining Group Patterns in SPARQL Queries. In *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30 - June 3, 2010, Proceedings, Part I (Lecture Notes in Computer Science)*, Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache (Eds.), Vol. 6088. Springer, 228–242. https://doi.org/10.1007/978-3-642-13486-9_16

[48] Xin Wang, Eugene Siow, Aastha Madaan, and Thanassis Tiropanis. 2018. PRESTO: Probabilistic Cardinality Estimation for RDF Queries Based on Subgraph Overlapping. *CoRR* abs/1801.06408 (2018). arXiv:1801.06408 http://arxiv.org/abs/1801.06408

[49] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge Graph Embedding by Translating on Hyperplanes. In *AAAI*. AAAI Press, 1112–1119.

[50] Lucas Woltmann, Claudio Hartmann, Maik Thiele, Dirk Habich, and Wolfgang Lehner. 2019. Cardinality estimation with local deep learning models. In *aiDM@SIGMOD*. ACM, 5:1–5:8.

[51] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Peter Chen, and Ion Stoica. 2020. NeuroCard: One Cardinality Estimator for All

Tables. *Proc. VLDB Endow.* 14, 1 (2020), 61–73.

[52] Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Peter Chen, Pieter Abbeel, Joseph M. Hellerstein, Sanjay Krishnan, and Ion Stoica. 2019. Deep Unsupervised Cardinality Estimation. *Proc. VLDB Endow.* 13, 3 (2019), 279–292.

[53] Rex Ying, Zhaoyu Lou, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, and Jure Leskovec. 2020. Neural Subgraph Matching. *CoRR* abs/2007.03092 (2020).

[54] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. 2018. GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018 (Proceedings of Machine Learning Research)*, Jennifer G. Dy and Andreas Krause (Eds.), Vol. 80. PMLR, 5694–5703. http://proceedings.mlr.press/v80/you18a.html

[55] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. 2017. Deep Sets. In *NIPS*. 3391–3401.

[56] Kangfei Zhao, Jeffrey Xu Yu, Hao Zhang, Qiyan Li, and Yu Rong. 2021. A Learned Sketch for Subgraph Counting. In *SIGMOD Conference*. ACM, 2142–2155.

[57] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. 2018. Random Sampling over Joins Revisited. In *SIGMOD Conference*. ACM, 1525–1539.