Holistic Cost: Implementing and Comparing Approaches to Graph ML for Learned Query Cost Estimation

Saloni Verma
Data and Knowledge Engineering (FIN)
Otto von Guericke University
Magdeburg, Germany
saloni.verma@ovgu.de

Sudheer Kumar Reddy Kandula
Digital Engineering (FIN)
Otto von Guericke University
Magdeburg, Germany
sudheer.kandula@st.ovgu.de

Akshata Balasaheb Bobade
Data and Knowledge Engineering (FIN)
Otto von Guericke University
Magdeburg, Germany
akshata.bobade@st.ovgu.de

Aditya
Masters in Informatik (FIN)
Otto von Guericke University
Magdeburg, Germany
aditya.aditya@st.ovgu.de

Janusz Feigel
Bachelors in Informatik (FIN)
Otto von Guericke University
Magdeburg, Germany
janusz.feigel@st.ovgu.de

Devi Prasad Ilapavuluri
Digital Engineering (FIN)
Otto von Guericke University
Magdeburg, Germany
devi.ilapavuluri@st.ovgu.de

***Abstract*—In the recent times, there is a rapid surge in usage of Graph Neural Networks among several disciplines because of their robust expressivity and explicit representation of data. The database forms the backbone for a lot of applications and websites all over the world. And relationships in the data objects are inherently present in the data we try to model and use. Graphs will represent these best. Using machine learning and deep learning algorithms to find the least expensive query plan has been employed in recent times. In this paper, firstly, we extended on the works of Dwivedi, et all [1] and have tuned certain parameters to attain better results on Zinc dataset for certain models than the ones mentioned in their paper. Secondly, we have proposed an approach which is based on incorporating Graph Neural Network algorithms to solve Cardinality Estimation, Runtime Estimation and Cost Estimation problems for databases taking IMDb dataset into consideration. We have studied and experimented with various configurations for Physical and Logical plans and have obtained results which proved to be promising if not better on comparison with Multi-set Convolutional Neural Network (MSCN) as a baseline. We believe that we have established an early methodology of utilising the potential of Graph Neural Networks for Query Cost Estimation.**

***Index Terms*—Holistic Cost, Cost Estimation, Join Order Optimization, Cardinality Estimation, Graph Neural Networks, Benchmarking Graphs, Graph ML, Regression, Positional Encoding, GCN, Gated GCN, GAT, GraphSage, Message Passing, DGL**

I. Introduction

In recent times, *Graph Neural Networks* (GNN) have gained popularity in domains like including social network, knowledge graph, recommender systems, and life sciences. The power that GNN have for modeling the dependencies between nodes in a graph will enable the breakthrough in the research areas of graph analysis [2].

The past decade has seen the widespread adoption of *Machine Learning* (ML), and specifically neural networks (Deep Learning), in many different applications and systems. The database community also has started to explore how machine learning can be leveraged within data management systems. Recent research therefore investigates ML for classical database problems like parameter tuning , query optimization, and even indexing.

In this paper, we propose an approach borrowing from deep learning that predicts (join-crossing) correlations in the data, which covers up the weak spot of sampling-based techniques. One approach is based on a specialized Deep Learning model called *Multi-set Convolutional Networks* (MSCN) [3] that allows us to express query features using sets. The idea for adopting deep learning for database queries rose from the widespread use of these techniques in tasks of scientific importance and the success that some other research groups have had with similar problems. We learnt and understood their aims, research questions, learnt about their assessments of results and designed our plans and implementation along those lines. It is incredible, the strides that technology has taken and we were able to adapt according to the successes and avoid the shortcomings of the other research teams.

We started off with understanding about Graphs to be able to represent queries in graphical form which would be used as input for the Deep Learning models. The next step was to learn about the various kinds and tuning of models that we could use for our research questions. Then we zeroed in on the research questions that we would focus for the scope of our project. We tried replicating the results of the Bengio paper [1] first to see the output for ourselves and to model our experiments accordingly. We compared the performance of different models like *Graph Convolutional Networks* (GCN),

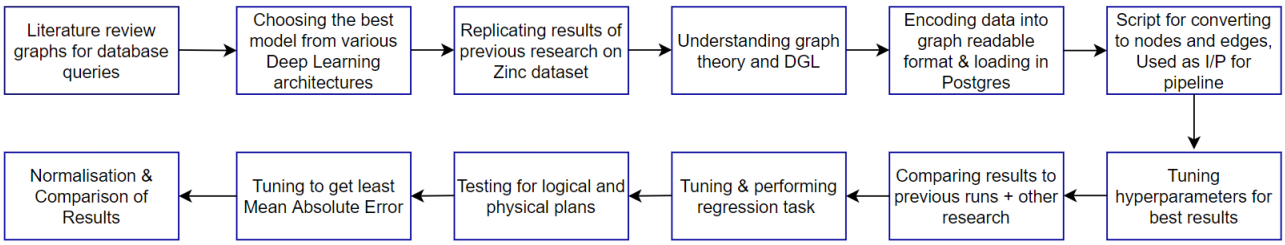

Figure 1. Project Flow and Pipeline

GNN, GraphSage, Gated GCN, and many more. After a comparative analysis and an initial round of proof of concept testing, we settled on Gated GCN with Edge features, Gated GCN with Edge features and Positional Encodings, MSCN, as our focus models. We implemented the models initially on the Zinc dataset [4] and moved on to the IMDB dataset for our final evaluation of Query Cost Estimation. After completing numerous different variations of the tuned models, we started seeing some promising results. We tweaked the embeddings for the logical and physical plans for better results, faced some issues with normalisation but are now presenting our best results in this paper.

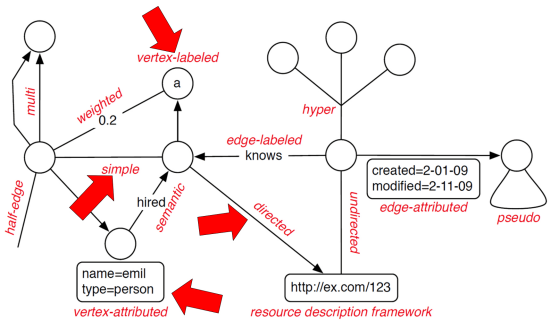

Figure 2. Types of GNNs Architectures for our Research

Our contributions are as follows:

- Converting query plans to graph representable data
- Creating the representations/embeddings for logical and physical plans
- Using *Deep Graph Library* (DGL) to make input readable by the graph models
- Finding the GCN parameters that give the most optimal results for cost and cardinality estimation

The paper is organised in the form of Introduction where we outline the goals and our project timeline. The Background section covers information about Graph Theory, Deep Learning models of our interest, Query optimisation including Cost and Cardinality estimation. We also define the motivation behind this topic, and the related work. Next, we define our Research Questions that define the scope of this project. The Experimental Setup includes the prototype design, datasets i.e Zinc and IMDB. The Evaluation section consists of our hypotheses to check if we were successful in answering the research questions. We also show our Results in graphical form for better comparison and interpretation. Finally, we conclude the paper and share our vision for the possible future work in this domain.

II. Background

GNN are known to capture the dependence in graphs by tapping the message passing between the different nodes of a graph.

Conversely to standard neural networks, GNN retain the state which represents the neighbourhood information with an arbitrary depth. In our research, we focus both on the *Message Passing GNNs* and *Weisfeiler-Lehman (WL) GNNs*.

Graph Neural Network is a type of Neural Network which directly operates on the Graph structure.

This section will illustrate the algorithm described in the paper, the first proposal of GNN and thus often regarded as the original GNN. In the node classification problem setup, each node `v` is characterized by its feature `x-v` and associated with a ground-truth label `t-v`. In a partially labeled graph called G, we have to use these labeled nodes to successfully predict the labels of unlabeled nodes. It learns to represent each node with a d dimensional vector (state) `h-v` which contains the information of its neighborhood. Specifically, However, there are three main limitations with this original proposal of GNN pointed out by this paper: If the assumption of "fixed point" is relaxed, it is possible to leverage *Multi Layer Perceptron* (MLP) to learn a more stable representation, and removing the iterative update process.

This is because, in the original proposal, different iterations use the same parameters of the transition function f, while the different parameters in different layers of MLP allow for hierarchical feature extraction. It cannot process edge information (e.g. different edges = different relationships between nodes)

GraphSage provides a solution by learning the embedding for each node in an inductive way. Each of the nodes is represented by an aggregation of its neighborhood. So even if a new unseen node appears during training time, it can be properly represented by the aggregation of its neighboring nodes.

Pooling aggregator: It performs element-wise pooling function on the given neighboring set. Below shows an example

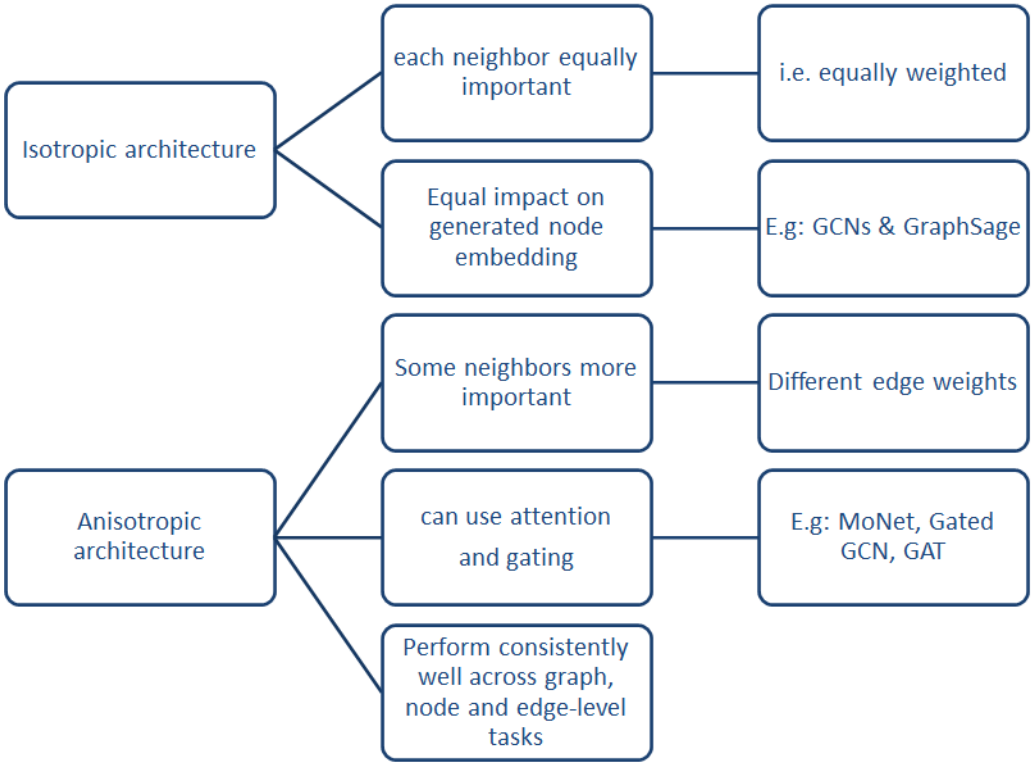

Figure 3. Types of Graphs (Graph Theory): Isotropic and Anisotropic

of max-pooling:

$$\hat{h}_i^{\ell+1} = \mathrm{ReLU}\left(U^{\ell}\ \mathrm{Concat}\ \left(h_i^{\ell}, \mathrm{Max}_{j\in\mathcal{N}_i}\ \mathrm{ReLU}\left(V^{\ell}h_j^{\ell}\right)\right)\right) \tag{1}$$

Isomorphism - Shape is same for the graphs. In our DS, the connections have a semantic meaning. **Isotropic:** Properties of a given graph are identical in all directions. **Anisotropic:** Properties of the material depend on its direction.

We perform experiments on simple edge properties, and homogeneous graphs, un-directed graphs, no weights or labels. Then homogeneous directed, no need for edge attributes, single relationship type. Directed Simple Edge type of graphs **(Anisotropic models)** Extend with vertex properties and vertex labels Ex. Each vertex can be one operator type.

A Logical Plan [5] will just depict what we expect as an output after applying a series of transformations like filter, join clause, where, groupBy, etc clauses on a particular table. It is an abstract of all transformation steps that will be performed and it does not refer to the Driver (Master Node) or Executor (Worker Node). The Physical Plan is responsible for deciding the exact type of join, the sequence of the execution of filter, where, group by clauses, etc. It shows the how-to-compute part of the query definitions. It is executable. We try to Visualise the execution of a query using the Physical plan and a Query Visualiser. This query is from the IMDB dataset (details in the next sectionII-A) and the query plan is also generated in PostgreSQL. This helps us to understand the complex steps that happen in the backend when a complex query is actually executed. The query plan we have is given in the Appendix**??**. It consists of Aggregate, Bitmap Heat Scan, Index Scan, etc which we visualised using a Query Visualiser [6]. The query executed was as follows:

```
EXPLAIN ANALYZE SELECT COUNT(*)
FROM title t,movie_info mi
WHERE t.id=mi.movie_id AND t.kind_id<3
AND t.production_year=2008
AND mi.info_type_id>2;
```

- DGL - It is a Python package useful for Deep Learning on graphs. It is built using existing tensor *Deep Learning* (DL) frameworks (mostly pytorch). It also eases the implementation of new Graph Neural Networks Faster and memory-friendly compared to other GNN frameworks. DGL adapts opt like multi-thread and multi-process acceleration, kernel fusion, and automatic sparse format tuning. When compared to others like PyTorch Geometric, DGL is much faster and memory-friendly.
- Use of deep learning - the implementation ease and expansiveness that deep learning allows and the extension to different areas of learning is immense. The available research and libraries makes the support for such research easy for students coming up with questions on the fly which can be tested by tuning certain hyperparameters.

```
                List of relations
 Schema |       Name        | Type  |
--------+-------------------+-------+
 public | aka_name          | table |
 public | aka_title         | table |
 public | cast_info         | table |
 public | char_name         | table |
 public | comp_cast_type    | table |
 public | company_name      | table |
 public | company_type      | table |
 public | complete_cast     | table |
 public | info_type         | table |
 public | keyword           | table |
 public | kind_type         | table |
 public | link_type         | table |
 public | movie_companies   | table |
 public | movie_info        | table |
 public | movie_info_idx    | table |
 public | movie_keyword     | table |
 public | movie_link        | table |
 public | name              | table |
 public | person_info       | table |
 public | role_type         | table |
 public | title             | table |
(21 rows)
```

Figure 4. IMDB dataset loaded in PostgreSQL

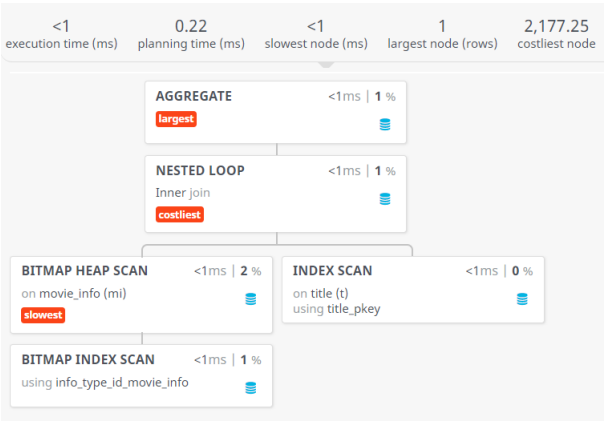

Figure 5. Query Interpretable using Visualiser [6]

- Graphs and their properties - The complex query plans have been made easy to visualise, understand and process through graph form.

A. *The Dataset - IMDB*

We chose the IMDB dataset [7] because it is available online and is easily integrated into the PostgreSQL architecture that we opted for. Certain subsets of the IMDb data are available for personal and non-commercial use. We are allowed to hold local copies of this data, subject to their terms and conditions. It was an ideal dataset for the queries that we were going to generate. The dataset was loaded in Postgres (on an Ubuntu machine), the relations were generated (as shown in the image) and it consists of *title*, *cast information, character names* and more. Once the data was loaded, we could see the relations and data that were working with on the basis of primary keys and foreign keys.


```
[
  {
    "Plan": {
      "Node Type": "Aggregate",
      "Strategy": "Plain",
      "Partial Mode": "Simple",
      "Parallel Aware": false,
      "Startup Cost": 1044.70,
      "Total Cost": 1044.71,
      "Plan Rows": 1,
      "Plan Width": 8,
      "Actual Startup Time": 0.234,
      "Actual Total Time": 0.234,
      "Actual Rows": 1,
      "Actual Loops": 1,
      "Plans": [
        {
          "Node Type": "Index Only Scan",
          "Parent Relationship": "Outer",
          "Parallel Aware": false,
          "Scan Direction": "Forward",
          "Index Name": "person_id_cast_info",
          "Relation Name": "cast_info",
          "Alias": "ci",
          "Startup Cost": 0.56,
          "Total Cost": 1043.42,
          "Plan Rows": 514,
          "Plan Width": 0,
          "Actual Startup Time": 0.042,
          "Actual Total Time": 0.203,
          "Actual Rows": 838,
          "Actual Loops": 1,
          "Index Cond": "(person_id = 172968)",
          "Rows Removed by Index Recheck": 0,
          "Heap Fetches": 838
        }
      ]
    },
    "Planning Time": 0.389,
    "Triggers": [
    ],
    "Execution Time": 0.264
  }
]
```


Figure 6. Physical Query Plan loaded in PostgreSQL

These are the steps for using the open source dataset for our project:

1) Loading IMDB dataset in Posgres
2) Generating physical plans
3) Script for converting data points into nodes and edges
4) Using Pytorch and DGL for creating PKL files that would be used as input for testing and regression task

The IMDB dataset and a physical plan of one query of it can be seen in Figure 4 and in Figure 5. Based on the research Hiprecht etc, al [8], we learnt they proposed a data driven model to overcome previous limitations for cardinality estimation and query answering. This is a workload-driven approach which has two major downsides. One, collection of the training data can be very expensive, since all queries need to be executed on potentially large databases. Two, this training data needs rework i.e. has to be collecte again when workload or data changes. It was assumed that the price would be lower accuracy since workload-driven models make use of more information but this is not the case. The results of their empirical evaluation demonstrate that their data-driven approach not only generalizes better to unseen queries but also provides better accuracy than contemporary learned components.

Model	Layers	Hidden Dimensions	Initial Learning Rate	MAE on Test Set	MAE on Train Set
Gated GCN-E-PE	16	120	0.003	0.1993	0.0509
Gated GCN-E	16	120	0.003	0.2077	0.0447

Table I
RESULTS FOR GATED GCN-E-PE AND GATED GCN-E MODELS ON ZINC DATASET FOR REGRESSION TASK

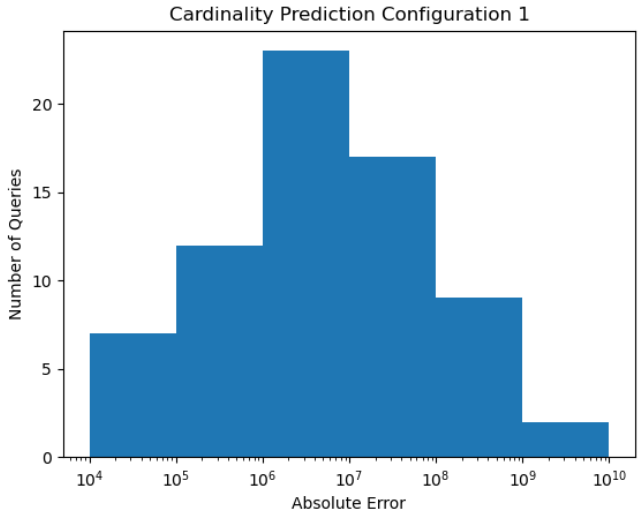

Figure 7. Cardinality Estimation for Configuration 1

We used the CSV files since the relations were apparent once the relations were made in Postgres. Once the CSV files were loaded in to Postgres, the next step was to make them graph readable which we achieved through a script written for converting queries into graphs and edges representation. It was a python script. In this script, we converted the data into *8*ndata (node data) and *edata* (edge data). Once the query plan is generated, we load it into a pickle file. There were specific checks for one datapoint files, and confirmation of nodes and egdes after the graphs were created for a good representation. After that, we use DGL and Pytorch to assign the joblight and synthetic sets into their respective source and destination files, while having the node and edge data clearly define. Synthetic Queries [9] Physical The encoding format (scripts derived from fellow university research) was fed into the DGL framework and Pytorch using Colab. [10]

The JSON files created from the PosgreSQL were the physical query plans recognised from the numbers. It served as an 70 identifiers [11] in terms of the JobLight queries. The final evaluation was done on this subset of our data.

B. Graphs and related Concepts

For our research the relevant types of Graph concept were as follows:

- Simple edges : The connection between 2 vertices(points in a graph). A simple graph is a graph without loops and multiple edges. Two vertices are adjacent if there is an edge that is connecting them. Adjacent edges are edges that share a common vertex. [12]
- Vertex-labeled nodes: Formally, given a graph , a vertex labelling is a function of to a set of labels; a graph with such a function defined is called a vertex-labeled graph. Likewise, an edge labelling is a function of. to a set of labels. [13]
- Directed edges: A directed edge is an edge contained in a directed graph, which means that the edge must also have an orientation, which is represented by using an arrow for the directed edge: the tail and head of this arrow are the nodes representing the beginning and ending points of the edge. [14]

C. Various Graph Neural Network Architectures

The researched GNN Architectures that we referred to were based on the forward and backward snowballing technique from our base Benchmarking paper [1]. The others we researched on the premise of hyperparameter tuning, alternatives to the approach, etc. The research group [15] present graph attention networks (GATs), neural network architectures that work on graph-structured data, using masked self-attentional layers to overcome the shortcomings of prior methods based on graph convolutions or such approximations. By stacking layers where nodes attend over their neighborhoods' features, they enabled (implicitly) by specifying different weights to different nodes in a neighborhood, without costly matrix operations (such as inversion) or dependence on knowing graph structure beforehand.

GraphSAGE [16], an inductive framework that uses node feature-information (example, text attributes) to formulate node embeddings for any previously unseen data. Rather than training individual embeddings for each node, they learnt a function which generates embeddings by sampling and aggregating features from a node's immediate local neighborhood. Another group proposed [17] a unified framework allowing them to generalize CNN architectures to non-Euclidean domains (such as graphs and manifolds) and learn stationary, local, and compositional task-specific features. Moreover, a scholar hypothesised [18] for deep learning is to reach its full potential, it needs to reconsider "hard-coded" approaches by integrating the assumptions about inherent structure of input data directly into our architectures and then learning algorithms, through its structural inductive biases. He validated this hypothesis by developing 3 structurely-infused neural

Configuration	Task	Encoding	Hidden Neurons	Initial Learning Rate	MAE of Normalized Validation Set	Training Duration (min)
1	Cardinalities	Logical	200	0.003	2.0393e+08	8.76
2	Cardinalities	Physical	145	0.003	2.0447e+08	6.81
3	Time	Physical	100	0.003	49.5360	8.45
4	Time	Physical	145	0.003	61.3016	7.41
5	Cost	Physical	120	0.003	42545.8555	8.9
6	Cost	Physical	200	0.004	53855.5820	8.77

Table II
BEST FOUND CONFIGURATIONS FOR THE DIFFERENT TASKS

network architectures (operating on sparse multimodal, graph-structured data), and another algorithm that was structure-informed learning for GNNs, which significantly outperform conventional baseline models and algorithms.

Bresson et al [19]proposed a natural extension of LSTM and ConvNet to graphs with an arbitrary size. They designed a set of analytically controlled experiments on 2 graph problems, i.e. subgraph matching graph clustering for testing various architectures. Their results showed that the proposed graph ConvNets and 1.5-4x faster than graph RNNs and are 3-17% more accurate. They are also 36% more accurate than variational or non-learning techniques. And the most effective graph ConvNet architecture which used gated edges and residuality since residuality plays an essential role in learning multi-layer architectures, they effectively provide a 10% gain of performance.

Another group [20] proposed Position-aware Graph Neural Networks (P-GNNs), a new type of GNNs for computing position-aware node-embeddings. It samples sets of anchor nodes, computes the distance of target node to each of the anchor-set, and moves on to learn a non-linear distance-weighted aggregation scheme over those sets. So, P-GNNs capture positions/locations of nodes wrt the anchor nodes. They have several advantages like being scalable, inductive, and incorporating node feature information.

Xu [21] present a theoretical framework for checking the expressive power of GNNs that capture different graph structures. Their results showed the discriminative power of popular GNN variantslike Graph Convolutional Networks and GraphSAGE - and how they cannot learn to distinguish some very simple graph structures. They then developed a simple architecture that is the most expressive among a class of GNNs and is just as powerful as the Weisfeiler-Lehman graph isomorphism test.

D. Benchmarking GNN

For the benchmarking of GNNs a framework from the benchmarking GNNs paper [1] exists. This framework can be adapted to take new datasets for different tasks like node classification or graph regression. The framework already implements the different GNNs and a learning algorithm. It has also the possibility to change parameters of a choosen model.

E. Join Order Optimization Problem

Learned Cardinalities

From a high-level perspective, applying machine learning to the cardinality estimation problem is straightforward: after training a supervised learning algorithm with query/output cardinality pairs, the model can be used as an estimator for other, unseen queries. There are, however, a number of challenges that determine whether the application of machine learning will be successful: the most important question is how to represent queries (“featurization”) and which supervised learning algorithm should be used. Another issue is how to obtain the initial training dataset (“cold start problem”). In the remainder of this section, we first address these questions before discussing a key idea of our approach, which is to featurize information about materialized sample.

Standard Deep Neural Network architectures such as *Convolutional Neural Networks* (CNN), *Recurrent Neural Networks* (RNN), or simple MLP are not directly applicable to this type of data structure, and would require serialization, i.e., conversion of the data structure to an ordered sequence of elements. This poses a fundamental limitation, as the model would have to spend capacity to learn to discover the symmetries and structure of the original representation. For example, it would have to learn to discover boundaries between different sets in a data structure consisting of multiple sets of different size, and that the order of elements in the serialization of a set is arbitrary.

- The cardinality estimation Problem is to try to predict, how many rows a query will return.
- The time estimation problem is to try to predict, how long the execution time of a query will be, it is depending on the chosen execution plan.
- The cost estimation problem is to try to predict, how much cost a particular execution plan of a query will have.

a) Encoding the Logical Query Plan into Graphs using DGL: For our project we try to predict cardinalities, time and cost for queries and a corresponding execution plan, with GNN. For that the queries have to be encoded as DGL graphs. The nodes have a feature vector with one attribute as 1, depending if the node represents either a predicate, table or column and the others 0. The edges between the nodes have as data a feature vector with a 136 attributes which are either 0 or 1, depending on the corresponding logical query plan.

b) Encoding the Physical Query Plan into Graphs using DGL: The physical plan is an execution plan. The nodes and edges have also feature vectors with 0s or 1s as data. In this case the nodes have 263 attributes that can be 0 or 1 and the

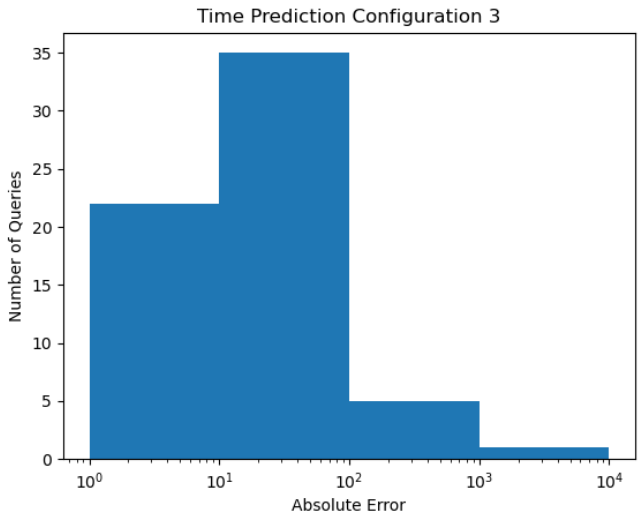

Figure 8. Time Estimation for Configuration 3

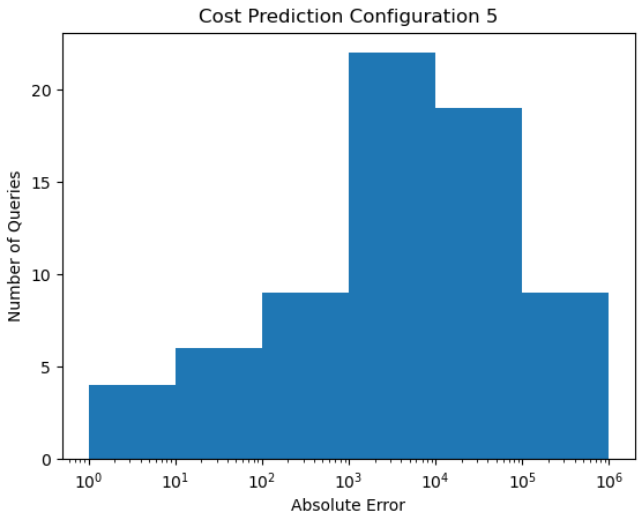

Figure 9. Cost Estimation for Configuration 5

edges have 3 attributes which represent the parent relationship, as either inner, outer or extra.

F. Research Questions

1. How will GNN cope with a general public graph regression task? What is the role of configurations?

2. What is the best architecture achievable for datasets of cost estimations? What is the role of configurations?

3. What can be learned about the generalization and power of models from a comparison against Multi-Set Convolutional Network (MSCN) and baselines?

III. Experimental Setup

A. Benchmarked GNN Architecture

a) GNN Benchmark: We used from the benchmarking framework the Models GCN, Gated GCN and 3WLGNN. The configurations could also be changed, the differences were the usage of positional encodings, the usage of edge features, the number of hidden neurons per layer, the layer number, the batch size and the learning rate schedule patience.

b) ZINC dataset: We trained with 5000 synthetic queries encoded in a logical form for cardinality prediction and in a physical form for cardinality, time and cost prediction. The 5000 as graphs encoded queries were split into 4250 queries for the training dataset and 750 queries for the test dataset. As a validation dataset we used 70 job light queries. The labels were normalized during training and for the validation set the predictions were unnormalized to calculate the real error and to see which real query values were predicted good and which were predicted bad.

c) IMDb dataset: We have studied various GNN models mentioned in [1] and tried to tune various parameters to get different configurations for models in order to find any significant change in the performances and also reduce the

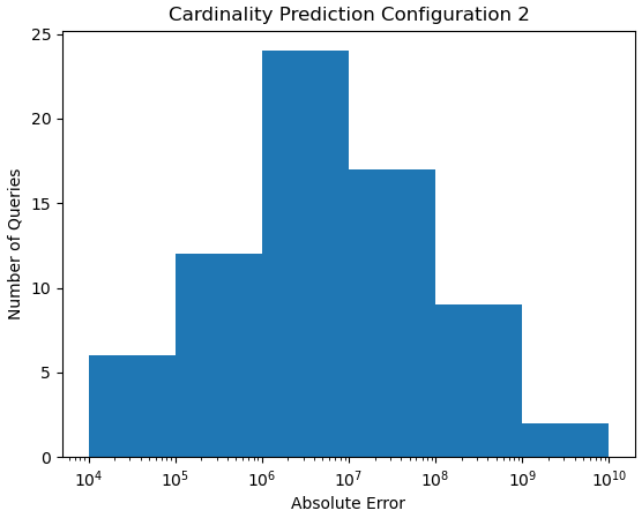

Figure 10. Cardinality Estimation for Configuration 2

error rates in comparison with the results in [1]. We trained our models considering 10k zinc molecular graphs for training set and tested them on 1000 graphs each for testing and validation sets.

d) Evaluating Performance against MSCNs: MSCN also used the from the IMDb Dataset the 5000 synthetic queries for learning and testing and then used the 70 job light queries as the validation dataset. So the MSCN cardinality predictions were gotten and the error then calculated, to have a baseline for comparison.

B. Implementation and Prototype Design

With the Benchmarking Framework and the Datasets, we could solve our research questions. We first did solved the first research question with the Benchmarking Framework and the

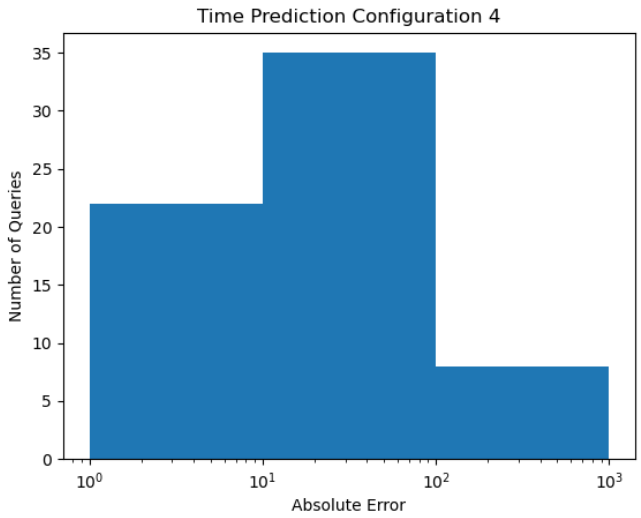

Figure 11. Time Estimation for Configuration 4

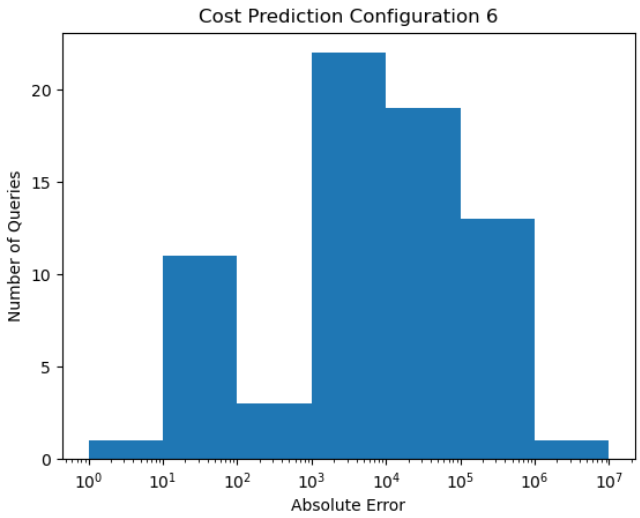

Figure 12. Cost Estimation for Configuration 6

already existing ZINC Framework. For the second Question we generated the physical plans with the cost and let them run to get the time. These were encoded as the DGL graphs of the IMDb datset. Logical and Physical datasets had to be loaded, wherefore a dataloader had to be created, which took the graphs, added to them the corresponding cardinality, time or cost label and added them then into a DGLDataset, which had to have predefined operations. After getting first results for research question 1 and 2, the configurations were changed until configurations were found which had a low error. Afterwards with the results, the third research question could be answered.

IV. Evaluation

A. Hypothesis

Our hypothesis for the 3 research questions were:

1) Graph Neural Networks are capable of a graph regression task and Gated GCN with Positional Encodings and Edge Features will perform good, as it did in the Benchmarking GNNs Paper [1].
2) Gated GCN with Positional Encodings and Edge Features will also perform best on a cost task. Logical and Physical Encoding can both give good results, but it is not clear which performs better.
3) The model will be able to generalize and can get better results than MSCN.

The challenges that we faced while conducting the experiments were the run times of the code i.e the training time for the 5k graphs and then the subsequent test times was really long. Every time the hyperparameters were changed to test for performance, we had to spend some time waiting for a new model to train and then test for further results. The results are not easily interpretable since the deep learning models are complex and work as a black-box.

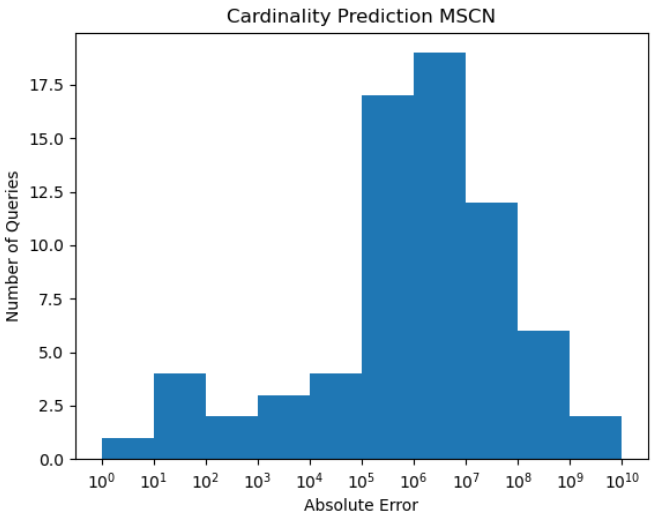

Figure 13. Cardinality Estimation for MSCN

B. Results

We just have to visualise and assume for different query plans which is an extensive task with the sheer amount of queries we use to train our model well. The difference between evaluation measures of different papers (Mean Q Error or Mean Absolute Error) also poses some difficulty in comparing results from different papers across the board. One more issue we ran into was the issue of normalisation for queries in the last stages of testing.

It was identified when the GCN model was suddenly performing poorly, we caught the slight issue and the results were exponentially better than before. These are some things to take care of in the future. This evaluation makes us better learners and makes the model even more robust for unseen

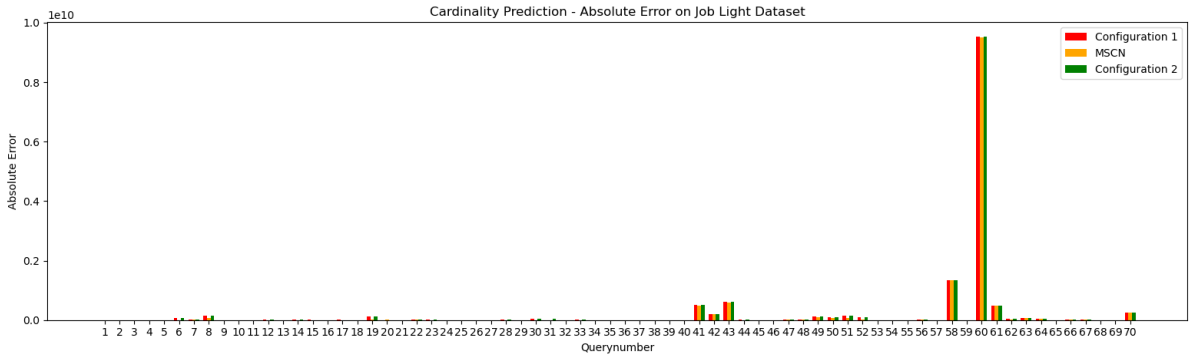

Figure 14. Query-wise Cardinality Prediction on Job light dataset (Linear scale)

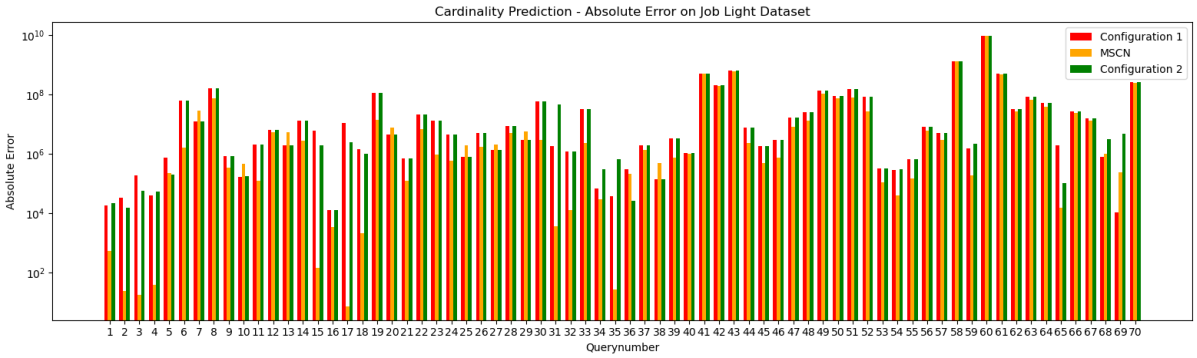

Figure 15. Cardinality Prediction on Job light dataset (Logarithmic Scale)

data. Because our end goal was to generalise to unknown data that comes in the future and to be able to reuse the model without much tuning. The hyper-parameters we concluded with work best for the current dataset and we have tested extensively for the most suitable results.

1) Evaluation results for first research question: As interpretable in Table I the Gated GCN Network with Edge Features performed well on a general regression task. The Positional Encoding can be good, but don't have to be. The learning rate of 0.003 and the 120 neurons in every hidden layer is a configuration that performed good. These insights were useful with the models for the cardinality, time and cost regression. The hypothesis was correct.

2) Evaluation results for second research question: The Gated GCN architecture with Positional Encodings and Edge Features, was the one, which gave on all 3 tasks the best results. The first part of our hypothesis is therefore correct. For the configurations a batch size of 128, 16 Layers and a learning rate schedule patience of 16 gave the best results. The number of hidden layer neurons was best between 100 and 200 and the initial learning rate was the best between 0.003 and 0.004. For cardinality prediction, the logical and physical encoding performed similar. Meanwhile, the Logical Encoding performed a little bit better which affirms and answers the second part of our hypothesis. In Table II are the differing parts of the 6 best models and configurations we could find for the different problems.

The different configurations had a different distribution of the error, which can be seen in this histograms of the 70 validation set job light queries:

3) Evaluation results for third research question: The models did train with normalized labels between 0 and 1 and the MAE was always between 0.01 and 0.1. The error on the train and test set was always similar and on the validation set it was larger by a factor of 1.5 to 2. So the model could generalize, which shows that the first part of our hypothesis was correct. The MSCN model had a MAE of 1.9305e+08 on the job light validation set. The distribution of the error can be seen in the Fig 13 histogram:

Together with the two following bar diagrams, it can be concluded, that there are some very expensive wrong predictions, that our models and MSCN are predicting wrong and that for most of the queries the prediction is relatively correct by our models and MSCN and that MSCN predicts cardinalities for some queries better than our models (mainly queries with small cardinalities). That shows, that the second part of our hypothesis is not correct.

V. Conclusion and Future Work

GNN have been explored to various types of regression and classification problems. However, no research had been found

regarding their use for Cardinality and Cost Estimations. In the field of databases, specifically, Cardinality Estimation has been very significant problem and any development in this path can yield ground breaking results. We have taken one of the early steps towards utilising the power of GNN for solving the Cardinality Estimation and Cost Estimation problem.

Our approach of considering this as a regression problem, have affirmed that the GNN are suitable for cost regression tasks. Even if they could not beat MSCN, they still performed well. We have strong hunch that perhaps with a different encoding of the graph, maybe with undirected edges, or with more queries of the problematic types, the results could be better. There can be a huge scope to research further in this area and some advancements can prove to be of great value. It can be said that our work looks promising to leverage the power of GNN to solve some kinds of database problems in a deeper level through further dedicated research.

REFERENCES

[1] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *arXiv preprint arXiv:2003.00982*, 2020.

[2] A. Jindal, E. Palatinus, V. Pavlov, and J. Dittrich, "A comparison of knives for bread slicing," *Proceedings of the VLDB Endowment*, vol. 6, no. 6, pp. 361–372, 2013.

[3] A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper, "Learned cardinalities: Estimating correlated joins with deep learning," *CoRR*, vol. abs/1809.00677, 2018. [Online]. Available: http://arxiv.org/abs/1809.00677

[4] J. J. Irwin, T. Sterling, M. M. Mysinger, E. S. Bolstad, and R. G. Coleman, "Zinc: a free tool to discover chemistry for biology," *Journal of chemical information and modeling*, vol. 52, no. 7, pp. 1757–1768, 2012.

[5] S. Hussain, "Understanding spark's logical and physical plan in layman's term," Feb 2021. [Online]. Available: https://blog.knoldus.com/understanding-sparks-logical-and-physical-plan-in-laymans-term/

[6] Saloni. [Online]. Available: https://tatiyants.com/pev/#/plans/plan_1614442419418

[7] imdb. [Online]. Available: https://datasets.imdbws.com/

[8] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and C. Binnig, "Deepdb: learn from data, not from queries!" *arXiv preprint arXiv:1909.00607*, 2019.

[9] b. DataManagementLab, "Datamanagementlab/deepdb-public." [Online]. Available: https://github.com/DataManagementLab/deepdb-public/blob/master/benchmarks/job-light/sql/synthetic_queries.sql

[10] "Google colaboratory," Jan 2021. [Online]. Available: https://colab.research.google.com/drive/14jWdQazpJIZn5FXXA1fnBBU4_t835iV9?usp=sharing

[11] b. DataManagementLab, "Datamanagementlab/deepdb-public." [Online]. Available: https://github.com/DataManagementLab/deepdb-public/blob/master/benchmarks/job-light/sql/synthetic_queries.sql

[12] "Virginia commonwealth university." [Online]. Available: http://www.people.vcu.edu/

[13] "Graph labeling," Jan 2021. [Online]. Available: https://en.wikipedia.org/wiki/Graph_labeling

[14] "Glossary." [Online]. Available: http://rosalind.info/glossary/directed-edge/

[15] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[16] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *arXiv preprint arXiv:1706.02216*, 2017.

[17] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 5115–5124.

[18] P. Veličković, "The resurgence of structure in deep neural networks," Ph.D. dissertation, University of Cambridge, 2019.

[19] X. Bresson and T. Laurent, "Residual gated graph convnets," *arXiv preprint arXiv:1711.07553*, 2017.

[20] J. You, R. Ying, and J. Leskovec, "Position-aware graph neural networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 7134–7143.

[21] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

APPENDIX A
SQL QUERY PLAN


```
[{"Plan": {
  "Node Type": "Aggregate",
  "Strategy": "Plain",
  "Partial Mode": "Simple",
  "Parallel Aware": false,
  "Startup Cost": 2205.08,
  "Total Cost": 2205.09,
  "Plan Rows": 1,
  "Plan Width": 8,
  "Actual Startup Time": 0.004,
  "Actual Total Time": 0.005,
  "Actual Rows": 1,
  "Actual Loops": 1,
  "Plans": [
    {
      "Node Type": "Nested Loop",
      "Parent Relationship": "Outer",
      "Parallel Aware": false,
      "Join Type": "Inner",
      "Startup Cost": 6.60,
      "Total Cost": 2205.07,
      "Plan Rows": 4,
      "Plan Width": 0,
      "Actual Startup Time": 0.003,
      "Actual Total Time": 0.004,
      "Actual Rows": 0,
      "Actual Loops": 1,
      "Inner Unique": true,
      "Plans": [
        {
          "Node Type": "Bitmap Heap Scan",
          "Parent Relationship": "Outer",
          "Parallel Aware": false,
          "Relation Name": "movie_info",
          "Alias": "mi",
          "Startup Cost": 6.17,
          "Total Cost": 19.41,
          "Plan Rows": 260,
          "Plan Width": 4,
          "Actual Startup Time": 0.003,
          "Actual Total Time": 0.003,
          "Actual Rows": 0,
          "Actual Loops": 1,
          "Recheck Cond": "(info_type_id > 2)",
          "Rows Removed by Index Recheck": 0,
          "Exact Heap Blocks": 0,
```

```
        "Lossy Heap Blocks": 0,
        "Plans": [
          {
            "Node Type": "Bitmap Index Scan",
            "Parent Relationship": "Outer",
            "Parallel Aware": false,
            "Index Name":
            "info_type_id_movie_info",
            "Startup Cost": 0.00,
            "Total Cost": 6.10,
            "Plan Rows": 260,
            "Plan Width": 0,
            "Actual Startup Time": 0.001,
            "Actual Total Time": 0.001,
            "Actual Rows": 0,
            "Actual Loops": 1,
            "Index Cond": "(info_type_id > 2)"
          }
        ]
      },
      {
        "Node Type": "Index Scan",
        "Parent Relationship": "Inner",
        "Parallel Aware": false,
        "Scan Direction": "Forward",
        "Index Name": "title_pkey",
        "Relation Name": "title",
        "Alias": "t",
        "Startup Cost": 0.43,
        "Total Cost": 8.41,
        "Plan Rows": 1,
        "Plan Width": 4,
        "Actual Startup Time": 0.000,
        "Actual Total Time": 0.000,
        "Actual Rows": 0,
        "Actual Loops": 0,
        "Index Cond": "(id = mi.movie_id)",
        "Rows Removed by Index Recheck": 0,
        "Filter": "((kind_id < 3) AND
        (production_year = 2008))",
        "Rows Removed by Filter": 0}]}]
},
 "Planning Time": 0.224,
 "Triggers": [
 ],"Execution Time": 0.086}]
```