Holistic Cost: Leveraging Graph Neural Networks 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*—Databases form the backbone for a lot of applications and websites all over the world, and the queries launched over databases, in the graph form of a query plan, are an object of importance for analysis and prediction. Traditional processing models for queries often neglect the graph structure, as models developed specific for graphs were not available. In recent times, there has been a rapid surge in the usage of Graph Neural Networks (GNNs) among several disciplines because of their robust expressivity and explicit representation of data. In this paper we aim to study the applications of GNNs for the supervised learning (regression) tasks of cardinality, runtime and cost estimation. We begin by a scoped study of model tuning on an established regression dataset (Zinc). Next we propose 2 different graph encodings for physical and logical plans, which we evaluate using the IMDB dataset and the Join Order Benchmark. Generally we find that GNNs can be effectively trained for the different regression tasks. We also find that GNNs can be competitive for cardinality estimation, though they are not seen to outperform the state-of-the-art model (MSCN) in all cases. We believe that our study establishes an early methodology of utilising the potential of GNNs for query cost estimation, and we suggest further areas of research.**

***Index Terms*—Cardinality Estimation, Cost Estimation, Graph Neural Networks, GCN, Gated GCN**

I. Introduction

In recent times, *Graph Neural Networks* (GNN) have gained popularity in domains like social networks, knowledge graphs, recommender systems, and life sciences. The power that GNN have for modeling the dependencies between nodes in a graph could enable breakthroughs in research areas of graph analysis [1].

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 context, to date there is limited work applying GNN. This is a specially noteworthy shortcoming because there are many graph-structured problems in data management, such as those related to query optimizations.

In this paper, we use 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 such existing approach is based on a specialized Deep Learning model called *Multi-set Convolutional Networks* (MSCN) [2] that allows us to express query features such as Tables, joins and predicates for specific query 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. For our work we seek to understand to what extent a competitor model based on GNN would be able to meet or outperform the existing approach. For this study it is also relevant to consider other tasks apart from cardinality estimation, such as cost and runtime estimation, and the fine-tuning of GNN models on established datasets.

The contributions of our work, from following our project flow, as shown in Fig. 1, are as follows:

- We fine-tune GNN models, specifically gated GCN with postitional encodings and gate features, to the task of graph regression on a chemical dataset (Zinc), identifying parameters that enhance the performance of the model beyond results previously published.
- We develop novel representations of queries for graph ML, corresponding to physical and logical plans.
- We evaluate the selected models using the aforementioned representations for the tasks of runtime, cost and cardinality estimation.

The rest of the paper is structured as follows: We start by presenting a condensed background for our work (Sec. II), where we introduce basics on cardinality estimation, graph

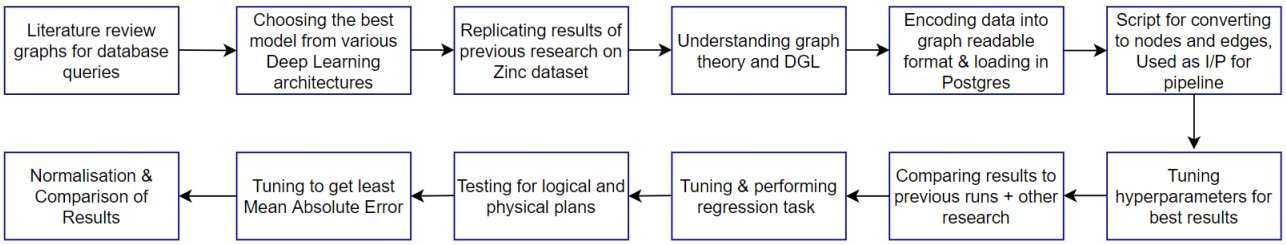

Figure 1. Project Flow and Pipeline

theory and graph neural networks. In Sec. III we present our research questions, and our contribution related to the graph representations developed for the logical and physical plans. In Sec. IV and Sec. V we present the experimental setup, describing the datasets used, and the evaluation results. We conclude the paper in Sec. VI, sharing our views and vision for possible future work in this domain.

II. Background

In this section of the paper, we introduce the two core areas of study in our work: Cardinality Estimation, and Graph Neural Networks.

A. Cardinality Estimation

Before we start defining the task of cardinality estimation, it is essential to define queries and the kinds of plans in which queries can be represented, at different stages of their processing.

Logical Plan

A Logical Plan [3] will just depict what we expect as an output after applying a series of transformations or clauses like filter, join, where, groupBy, etc., on a particular set of tables. Logical plans are an abstraction of all transformation steps that will be performed and they do not refer to implementation details, such as algorithms chosen to implement a specific operator.

Physical Plan

A 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 can visualise the execution of a query using the Physical plan and a Query Visualiser. We provide an example from the IMDB dataset (details in the next section**??**) 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 [4]. 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;
```

```
                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 2. IMDB dataset loaded in PostgreSQL

The IMDB dataset and a physical plan of one query of it can be seen in Fig. 2 and in Fig. 3.

Cardinality Estimation Problem

Cardinality estimation is the task of predicting for a query, the number of rows that are likely to come out as a result of its execution over a database. Cardinality estimation is an essential task for query optimization, since cardinalities are usually the input for cost models that in turn serve to guide optimizers to find the best query change to do.

A related problem to cardinality estimation is the time estimation problem, which is the attempt to try to predict, how long the execution time of a query will be, it is depending on the chosen execution plan. Also related is cost estimation, which seeks to predict, how much cost a particular execution plan of a query will have. Here cost is in general, non-specific units.

Research shows that cardinality estimation remains a difficult task for databases, because of difficulties to form samples able to capture the diverse kinds of correlations that might influence the number of query results. Hence, machine learning is being studied for cardinality estimation.

```
[
  {
    "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 3. Physical Query Plan loaded in PostgreSQL

From a high-level perspective, applying machine learning to a 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 (“i.e cold start problem”).

Standard Deep Neural Network architectures such as *Convolutional Neural Networks* (CNN), *Recurrent Neural Networks* (RNN), or simple *Multi Layer Perceptron* (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. To address this, an architecture called deep sets, or MSCN has been proposed.

MSCN encodes the whole Query for the processing in the Convolutional Neural Network. Whereby it supports processing of multiple Tables and estimates the Cardinality. During the encoding Process the Query gets separated into different Sets (Tables Set, Joins Set and Predicates Set). MSCN aggregates the Joins as one Layer after the next Layer as of its conceptual Methodology. Particularly the Predicate-Encoding done by the MSCN can be described as Normalization of the Predicate Values using the minimum and maximum Values of the respective Column Range. Even though MSCN is State of the Art in many Aspects, the Cardinality Estimation results are still considered to be improvable.

Hilprecht etc, al [5], proposed a data-driven model to overcome previous limitations for cardinality estimation and query answering. Authors call previous approaches as workload-driven, and emphasize two major downsides: One, the 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 collect 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.

B. Graph Neural Networks

a) Graphs and related Concepts: A graph is a mathematical abstraction that consists of nodes and edges. For our research we identified a list of concepts as relevant. Some of these concepts are shown in Fig. 15. We list down the important terms and concepts briefly 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. [6]
- 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. [7]
- 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. [8]
- Isomorphism: Shape is same for two graphs (i.e. there is a one to one mapping between two graphs).
- Isotropic: Properties of a model or function over given graph are identical in all directions.
- Anisotropic: Properties of a model or function over a graph depend on its direction.

For our study we perform experiments on simple edge properties, and homogeneous graphs, undirected graphs with-

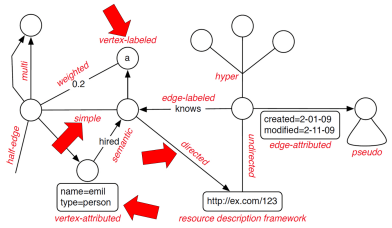

Figure 4. Types of Graphs (Graph Theory): Isotropic and Anisotropic

out weights or labels. Further we investigate homogeneous directed graphs, which does not need edge attributes and are of single relationship type. Directed Simple Edge type of graphs which are Anisotropic models, extends with vertex properties and vertex labels. For example, each vertex can be single operator type.

b) Various Architectures of Graph Neural Networks: A Graph Neural Network is a type of a Neural Network which directly operates on the Graph structure. It takes as input a graph consisting of nodes and edges. It can be directed or undirected. Furthermore it enables for nodes and edges to have data intrinsic to them. Based on their graph input GNNs learn an internal representation of the graph, which can then be used for further machine learning tasks, such as node classification, graph regression, graph classification or link prediction.

Node Embedding and Message-passing:

The graph embedding determines a fixed length vector representation for every entity in our graph. These embeddings are a lower dimensional representation of the graph and preserve the topology of graphs. The graph embedding techniques include whole graph embedding and node embedding. The most commonly used technique is node embedding where the graph structure is preserved by keeping the nodes that are closer to each other in a graph also closer in the embedded space.

In GNNs, the Message-passing is performed between the nodes and it is also known as Node Aggregation as it involves the process of pushing the embeddings from neighborhood nodes around a reference node through directed edges. For a single reference node, the messages or information from all the neighborhood nodes passed via edge neural networks is aggregated which can be done by using several techniques like max-pooling, aggregating etc., A new embedding of this reference node is updated by applying its activation function on its own embedding and the information aggregated from its respective neighboring nodes embedding.

There are many kinds of GNNs nowadays. The primary two classes of GNNs are broadly based on message passing principle and the rest are based on other mechanisms. A second distinction is shown in Fig. 5, which divides GNNs into anisotropic models that weight the edges differently and isotropic models that weight edges equally. GCN and GraphSage architectures are isotropic models based on message passing principle are explained below. On the other hand, GAT, MoNet, Gated GCN architectures are anisotropic models.

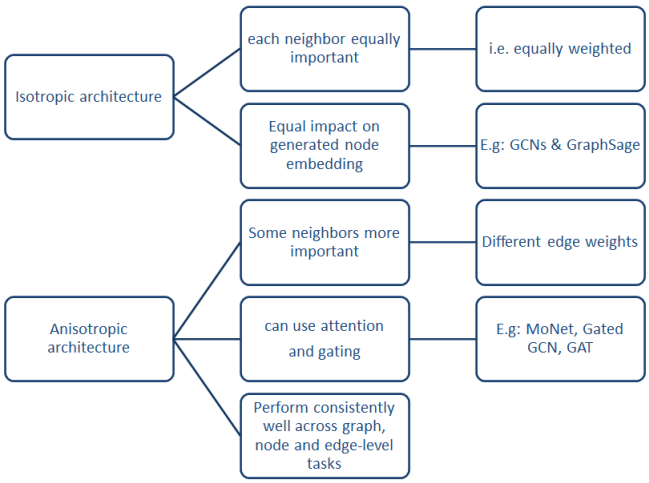

Figure 5. Types of GNNs Architectures for our Research

We present various GNN architectures by firstly introducing an isotropic message passing *Graph Convolutional Networks* (GCN) model that was proposed in the paper Semi-Supervised Classification with Graph Convolutional Networks [9], and thus often regarded as the original GNN.

In GCN, a spatially moving filter convolves over the nodes of the graph which contains feature representation of the relevant data of that node or embeddings. The layer-wise convolution operation on nodes within single node distance or in the immediate layer is performed to preserve the local neighborhood structure of graph. A recursive message passing is performed on each node or edge of a graph and the feature representation is computed for each layer of GCN where each graph node gathers features from its neighbor in order to depict a local graph structure. Furthermore, when various iterations or K layers of message passing is completed, a node learns more about its neighborhood nodes as well as its distant neighbors.

GCN 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, GCN retain the state which represents the neighbourhood information with an arbitrary depth. In our research, we focus both on the *Message Passing GCN*, instead of the variant *Weisfeiler-Lehman (WL) GNNs*.

The research group [10] presents the Graph Attention Net-

works (GATs). This neural network architecture uses masked self-attention layers that work on graph-structured data. This overcomes the shortcomings of all prior methods that are based on graph convolutions that treats all neighbors of a node equally. The key idea of their approach is to stack layers, wherein nodes attend over their neighborhoods' features and arbitrary importance for nodes are achieved implicitly by assigning different weights to different nodes in a neighborhood. The different attention scores or weights that are assigned by attention mechanism to each neighbor helps to identify the more important neighbors. Without the necessity for any expensive matrix operations or relying on knowledge of graph structure in advance, arbitrary importance was achieved.

GraphSAGE [11] is an inductive framework that uses node feature-information (example, text attributes) to formulate node embeddings for any previously unseen data. Instead of training embeddings individualy for each node, they learn a function which generates embeddings by sampling and aggregating features from a node's immediate local neighborhood. As it incorporates explicitly each node's own features from previous layer in node embedding updation, it is an improvement over the simple GCN model. Despite a new unseen node appearing during training time, it can be properly represented by the aggregation of its neighboring nodes. The author suggests three aggregator functions to perform aggregation operation and Pooling aggregator described below is one such example of these functions.

Pooling aggregator: It feeds each neighbor's hidden state through a fully connected layer and a max-pooling operation is applied on the neighboring node's set. Below shows an example of max-pooling:

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

In the paper Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs [12] a framework named mixture model networks (MoNet) was proposed which provides an ability to design CNN architectures for graphs and manifolds and enables to learn stationary, local, and compositional task-specific features. Main idea of their approach is to use functions of graph's pseudo coordinates for convolution like operations and these functions are represented as a mixture of Gaussian Kernels. In the paper [13] GatedGCN architecture is introduced which is a anisotropic variant of GCN. It takes into account the residual connections, batch normalization and edge features along with node features while updating the embeddings. There are two other variants of GatedGCN architecture, that is GatedGCN-E with edge-features and GatedGCN-E-PE [14] with edge-features and positional encodings. They typically provide special importance to graph edges and node positions respectively.

Bresson et al [13]proposed a natural extension of LSTMs and ConvNets to graphs with an arbitrary size. They designed a set of analytically controlled experiments on 2 graph problems, i.e. subgraph matching and graph clustering for testing various architectures. Their results showed that the proposed graph

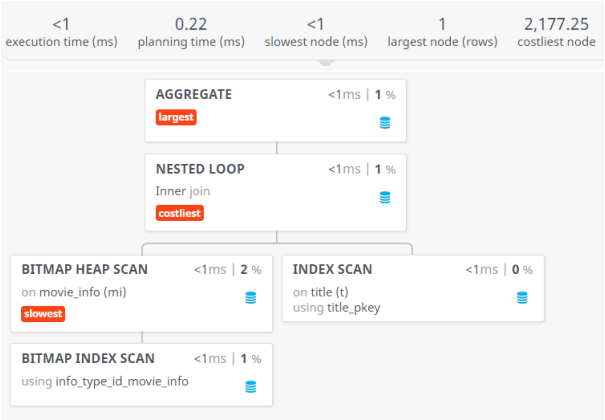

Figure 6. Query Interpretable using Visualiser [4]

ConvNets are 1.5-4x faster than variants with 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 [15] 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 [16] 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 variants like 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.

For the benchmarking of GNNs a framework from the benchmarking GNNs paper [17] 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 chosen model. With this framework authors [17] have been able to produce good results for regression and classification tasks across various datasets. This has inspired us to leverage the power of GNN to solve the Cardinality Estimation and Cost Estimation problems by treating these as regression problems. For evaluation, we have considered as baseline, a robust deep learning model MSCN mentioned in the paper [2] for the Cardinality Estimation problem.

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 I
BEST FOUND CONFIGURATIONS FOR THE DIFFERENT TASKS

III. PROTOTYPE DESIGN

In this section, we show the research questions which we formulated at the beginning of our project. Then we explain about the encodings for both logical and physical query plans. Later, we mention the models of our interest which were used for evaluation and comparison.

A. 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 a baseline, such as the Multi-Set Convolutional Network (MSCN) [2]?

B. Encodings

a) Encoding the Logical Query Plan: Since for our project we intend to predict cardinalities, time and cost for queries and a corresponding execution plan, with GNN, the first task is to adopt/develop a mechanism for encoding queries to be presented to the models. For this, queries have to be encoded as graphs which can then be used in a graph processing library such as *Deep Graph Library* (DGL) that connects to GNN implementations.

For the logical plan we have devised a model where nodes can be either tables, attributes or predicates; and edges can represent join relationships, table-attribute membership, or the connection of predicates to attributes. To fully capture table and attribute information we require nodes to have a one hot encoding for these items. Similarly for operators in predicates we require a one hot encoding. Finally for the values, which are numerical in our case, we use a normalized value. At the end for each node we 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 nodes also have as data a feature vector with a 136 attributes which are either 0 or 1, depending on the corresponding logical query plan. The edges have feature vectors of just size 3 to indicate the kind of relationship. The resulting logical plans can be mostly seen as a non-hierarchical but layered graph, with tables as the parents, attributes in the middle, and different predicates as the last nodes. We developed a recursive method to build this representation given an input SQL query, enhanced with table information (i.e., min-max column values).

b) Encoding the Physical Query Plan: The physical plan is an execution plan, which contains lot of information. In essence this is provided by a DBMS optimizer, for our study we used PostgreSQL. The resulting graph, which is the input to our encoding looks like a hierarchy of operators, each with their physical information. It is a hierarchy with the node that produces the output on top, and scan nodes on the end. When consuming this information from a DBMS, we receive a nested JSON data structure containing nested plans. To encode this we created different node types, corresponding to the operators present in our dataset. These types were: seq scan, hash, nested loop, merge join, gather merge, hash join, materialize, aggregate, sort, gather and an extra filter node type. The last node type mentioned is to cater to the fact that AND or OR predicates appear as a single node, which is not compatible with our representation as used in the logical form. So for each AND predicate we create a tree structure that holds as children all sub-predicates, and we assign as node type the mentioned extra filter node type. As for edge types we have inner or outer relationships, as well as the relationship corresponding to the extra filter nodes. Internally each node has different one-hot-encoded information regarding its properties e.g. whether a sort is done with quicksort or external merge, or whether it uses disk or memory as a sort space. For these properties we considered partial model, sort space type, sort method, hash or merge conditions, join filters, relation name, and others. Filter nodes also have normalized values, as the case of the logical plans. At the end, the nodes have 263 attributes, and the edges have 3 attributes which represent the parent relationship. We developed a recursive method to build this representation given an input SQL query plan in JSON format, enhanced with table information i.e., min-max column values.

The JSON files we used for our study were created from the PosgreSQL physical query plans.

C. Selected Models

We selected the Models GCN, Gated GCN, Gated GCN with Edge Features, Gated GCN with Edge Features and Positional Encodings and 3WLGNN for the regression task on the different datasets. These Models had already performed well on the benchmark in the Benchmarking GNNs paper [17]. On the other end, we have considered the MSCN model, mentioned in [2] paper as a baseline for cardinality estimation based on the logical plans.

IV. Experimental Setup

In this section of the paper, we describe the environment and dataset we used and how we set up and executed our experiments.

A. Benchmarked GNN Architecture

a) GNN Benchmark: We used from the benchmarking framework the Models GCN, Gated GCN and 3WL-GNN. 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) DGL: DGL - For our work in feeding our representations to the GNN models we used DGL. It is a Python package useful for Deep Learning on graphs. It is used as the library for graph encoding in the Benchmarking Architecture. 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 optimizations like multi-thread and multi-process acceleration, kernel fusion, and automatic sparse format tuning. When compared to others like PyTorch Geometric, DGL is reported to be faster and memory-friendly.

c) Zinc dataset: We have studied various GNN models mentioned in [17] 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 error rates in comparison with the results in [17]. We trained our models considering 10k zinc molecular graphs for training set and tested them on 1000 graphs each for testing and validation sets.

B. IMDb Dataset

We chose the IMDb dataset [18] 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.

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

1) Loading IMDb dataset in PosgreSQL
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 on regression task.

We used the CSV files since the relations were apparent once the relations were made in Postgres. Once the CSV files were loaded into 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 *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 [19] and physical queries were taken into account. There were 5000 synthetic graphs for each, the physical and the logical dataset, which were divided into 4250 train graphs and 750 test graphs. There were also 70 job light graphs for each, the physical and logical dataset, which were used as the validation set. The cardinalities, times and cost were normalized and then added as labels to the datasets. The encoding format (scripts derived from fellow university research) was fed into the DGL framework and Pytorch using Colab. [20]

a) Evaluating Performance against MSCNs: MSCN also used from the IMDb Dataset, 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 made and the error then calculated, to have a baseline for comparison.

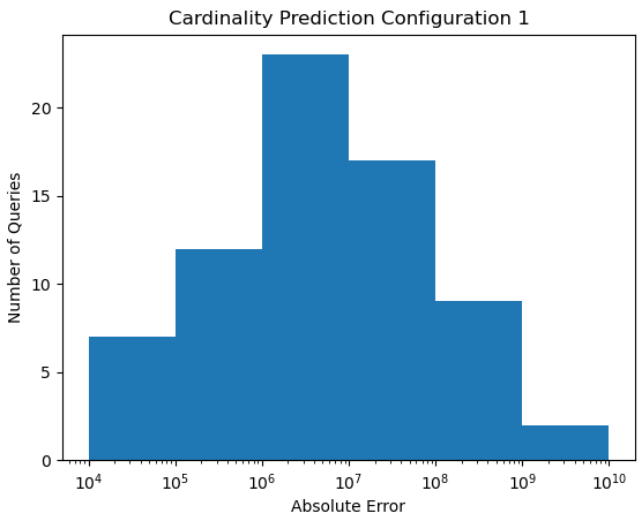

Figure 7. Cardinality Estimation for Configuration 1

C. Implementation

With the Benchmarking Framework and the Datasets, we could tackle our research questions. We first addressed the first research question with the Benchmarking Framework and the already existing ZINC data. For the second research question the pickle files of the Logical and Physical queries could be used to train the network with varying models, configurations and labels i.e., costs, cardinality, runtime. The third research question could then be solved with a function that we wrote

in a csv, the predictions of the net of the 70 job light queries. With that csv, the predictions could be unnormalized and the MAE can be calculated and compared against MSCN.

V. Evaluation

In this section of the paper, we mention about the hypotheses for the research questions, that we had and what the real results are and if they match our hypotheses.

A. Hypotheses

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 [17].
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.

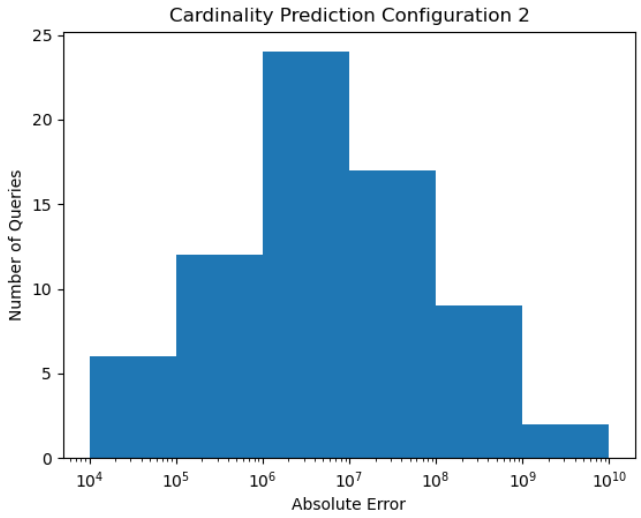

Figure 8. Cardinality Estimation for Configuration 2

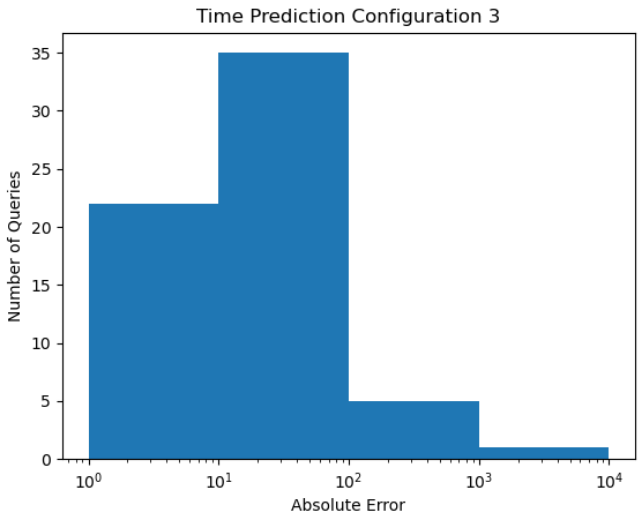

Figure 9. Time Estimation for Configuration 3

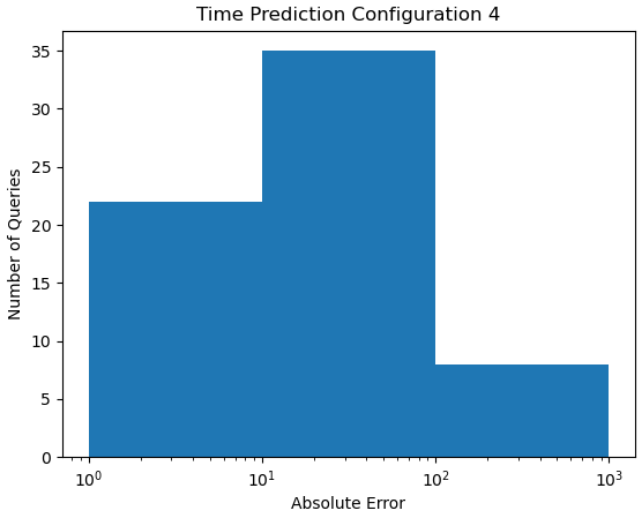

Figure 10. Time Estimation for Configuration 4

B. 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 for the Zinc dataset. The Positional Encoding can be good, but this is not necessarily so in all cases. For instance, this has shown good result on test set by getting least error for some cases but not on training set. The learning rate of 0.003 and the 120 neurons in every hidden layer is a configuration that performed good. On the other hand, tuning of factors like learning rate_reduce factor, weight decay,drop out have not turned out to be useful. Individually batch size did not have a great effect. However,batch size coupled with learning did have some significance to improve performance of models to some extent. Learning rate_schedule_patience factor was also an important one in avoiding the models to overfit. These insights were useful with the models for the cardinality, time and cost regression on IMDb dataset later on. Our hypothesis regarding the capability of Gated GCN with Posititonal Encodings and Edge Features is validated by our experiments.

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

Model	Layers	Hidden Neurons	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 II
RESULTS FOR GATED GCN-E-PE AND GATED GCN-E MODELS ON ZINC DATASET FOR REGRESSION TASK

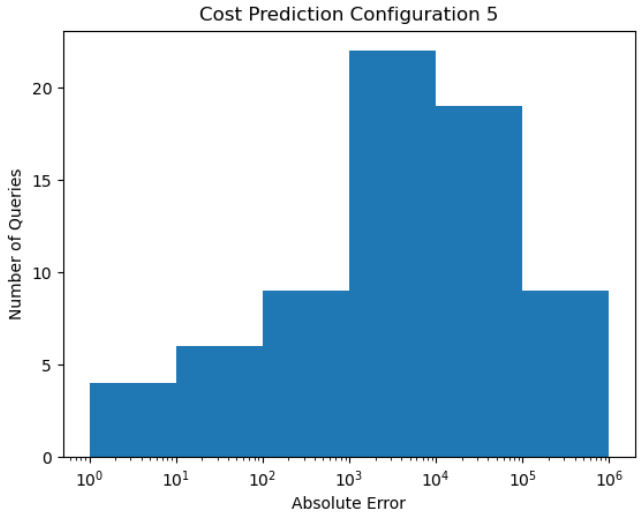

Figure 11. Cost Estimation for Configuration 5

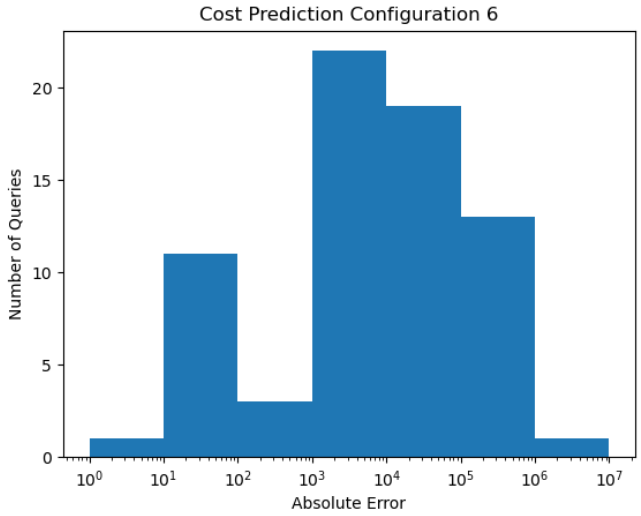

Figure 12. Cost Estimation for Configuration 6

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 the histograms of the 70 validation set job light queries (Figure 7 - 12).

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, suggesting successful learning. 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, suggesting a small tendency towards overfitting. 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.

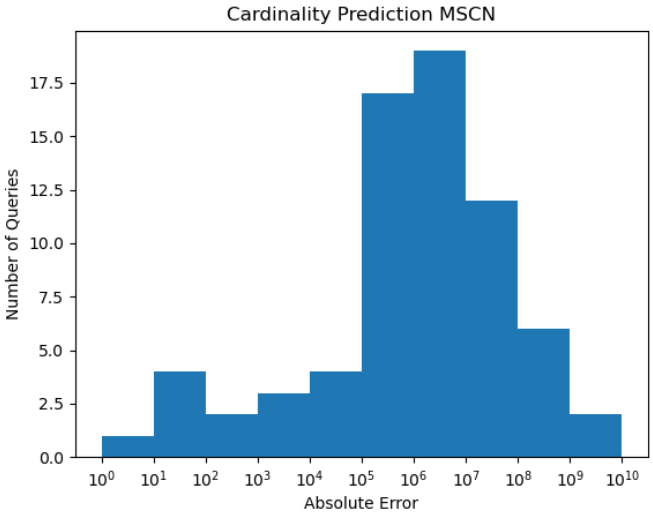

Figure 13. Cardinality Estimation for MSCN

Together with Figure 14 and Figure 15, it can be concluded, that there are some very expensive wrong predictions that our variants of Gated GCN models and MSCN model are predicting wrong and that for most of the queries, the predictions are relatively correct by our models and MSCN and that MSCN predicts cardinalities for some queries better than our models, mainly queries with small cardinalities, while our models have a bit of an edge on cases with large cardinalities. This shows, that the second part of our hypothesis is not correct.

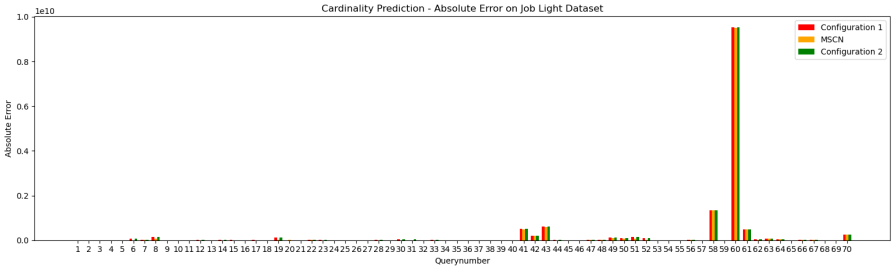

Figure 14. Query-wise Cardinality Prediction on Job light dataset (Linear scale)

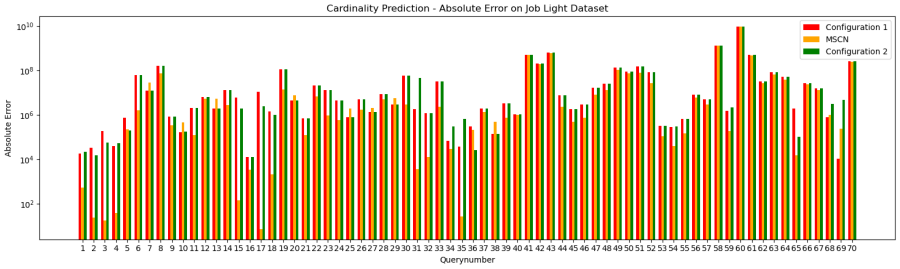

Figure 15. Cardinality Prediction on Job light dataset (Logarithmic Scale)

VI. Conclusion and Future Work

GNN have been explored to various types of regression and classification problems. However, as per the best of our knowledge, no research had been found regarding their use for Cardinality and Cost Estimations. In the field of databases, specifically, Cardinality Estimation has been a very significant problem for years and developments in this area have implications for technological improvement. We have taken in this work 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 on all scenarios, they still performed well. We strongly suggest 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. Techniques like guiding the batch sampling to emphasize more these costly error queries could be important. Also relevant for improvement is the inclusion of bitmaps of samples into our model, as was done for MSCN at training and inference. More model tuning, guided by interpretability tools, could also bring improvements.

In future work would like to incorporate into our results the predictions of modern DBMSs, to put into perspective the improvements that these models bring to the state-of-the-art practice.

We consider that there is a great scope to research further in this area and that it is likely we can identify some advancements that can prove to be of great value. It can be said that our early work validates the promising direction of leveraging the power of GNN to solve some kinds of database problems in a deeper, more holistic level (since graphs can capture more context than other representations), through further dedicated research.

Acknowledgements

We would like to acknowledge and thank a few members who had done some related work in this field. For our work, we had used some previous contributions done by Laxmi Balami, Narasimha Prabhu and a former team including Sankul Rathod working on Cardinality Estimation with other models. We also thank the DBSE project group for providing us an opportunity to work in this amazing project and also for all the feedback. We would also like to specially mention our supervisor Gabriel Campero Durand under whose guidance we could successfully accomplish all the tasks for the project and are grateful for his suggestions and feedback.

References

[1] 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.

[2] 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

[3] 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/

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

[5] 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.

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

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

[8] "Glossary." [Online]. Available: http://rosalind.info/glossary/directed-edge/
[9] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017.
[10] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.
[11] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *arXiv preprint arXiv:1706.02216*, 2017.
[12] 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.
[13] X. Bresson and T. Laurent, "Residual gated graph convnets," *arXiv preprint arXiv:1711.07553*, 2017.
[14] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Computation*, vol. 15, no. 6, pp. 1373–1396, 2003.
[15] J. You, R. Ying, and J. Leskovec, "Position-aware graph neural networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 7134–7143.
[16] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.
[17] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *arXiv preprint arXiv:2003.00982*, 2020.
[18] imdb. [Online]. Available: https://datasets.imdbws.com/
[19] b. DataManagementLab, "Datamanagementlab/deepdb-public." [Online]. Available: https://github.com/DataManagementLab/deepdb-public/blob/master/benchmarks/job-light/sql/synthetic_queries.sql
[20] "Google colaboratory," Jan 2021. [Online]. Available: https://colab.research.google.com/drive/14jWdQazpJIZn5FXXA1fnBBU4_t835iV9?usp=sharing
[21] 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.
[22] b. DataManagementLab, "Datamanagementlab/deepdb-public." [Online]. Available: https://github.com/DataManagementLab/deepdb-public/blob/master/benchmarks/job-light/sql/synthetic_queries.sql
[23] P. Veličković, "The resurgence of structure in deep neural networks," Ph.D. dissertation, University of Cambridge, 2019.

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}]
```