RLGCNt: Cardinality Estimation based on Rank Gauss Transform Coding and Attention

No Author Given

No Institute Given

Abstract. Cost and cardinality estimation play a pivotal role in determining the selection of query execution plans. However, the cardinality estimation model based on deep learning will suffer from the problem of reduced accuracy when dealing with dynamic workloads. To address the degradation of cardinality estimation under dynamic workloads, we present a novel cardinality estimation model, namely RLGCNt. For query, unevenly distributed data is rank-transformed into a normal distribution using Gaussian transform-based query encoding. This solves the problem of reduced encoding efficiency under dynamic data loads. Furthermore, we incorporate a locality-sensitive hashing module and a Gaussian kernel function module to boost cardinality estimation and capture data similarities effectively. The RLGCNt model also introduces a causal attention mechanism to address the problem of how the query data arrangement affects the cardinality estimation. We conduct a comparison between RLGCNt and deep-learning-based cardinality estimation methods on the public STATS dataset. Experimental results demonstrate that RLGCNt outperforms mainstream cardinality estimation algorithms. Specifically, for dynamic workloads in the STATS dataset, RLGCNt achieves an accuracy 10.7% higher than the baseline method, and for static loads, it attains an accuracy 5.8% higher.

Keywords: cardinality estimation · execution plans · dynamic · attention.

1 Introduction

In the context of query optimization, cost estimation and cardinality estimation are of paramount importance particularly in relational database management systems (RDBMS). Cardinality estimation, in particular, employs algorithms to estimate the size of a query result set. This estimation is instrumental in enabling the query optimizer to choose appropriate query plans, thereby enhancing the overall efficiency of the database system.

Traditional cardinality estimators [6, 9] often ignore the correlation between features and predicates of queries, which reduces estimation accuracy. sampling methods [2, 6] are less effective under dynamic workloads. For queries involving complex joins and multiple tables [13], applying deep learning neural networks [11] for cardinality estimation has become a trend. MSCN [4] based on deep

learning addresses issues faced by traditional estimators, but small sample sizes can reduce effectiveness in large-scale databases and complex workloads. Additionally, the model does not fully capture local data patterns. In the ALECE [7] model, the attention mechanism is significantly influenced by the sequence position of features. Specifically, the one-hot encoding employed within ALECE demonstrates reduced effectiveness under conditions of uneven or highly skewed data distribution. This inefficiency of one-hot encoding can lead to suboptimal performance in the ALECE model, as it fails to adequately represent the data characteristics in such non-uniform data scenarios.

The FACE [16] estimates cardinality through an autoregressive model and assumes that columns are independent of each other, but is affected by the arrangement of data in the query when facing dynamic workloads. Similarly, the PRICE [21] estimates cardinality from a high-dimensional joint probability density function, but is also affected by the arrangement of data in the query, which affects accuracy.

To address the limitations of existing technologies, we propose a novel cardinality estimation method based on attention mechanism and Gaussian kernel function. The core contributions of this paper include the following four parts:

- **Cardinality Estimation Model**: We propose the RLGCNt cardinality estimation model, which improves accuracy for both dynamic and static workloads. By integrating locality-sensitive hashing, Gaussian kernel functions, and an attention mechanism, we cluster data, thus enhancing similarity search and local feature extraction in dynamic workload.
- **Query Encoding Method**: We design a Rank Gaussian Transform-based query encoding method to enhance outlier resilience by transforming with a Gaussian rank function.
- **Causal Attention Mechanism**: We introduce a causal attention mechanism by adding causal masks to improve the handling of the effect of data arrangement on cardinality, addressing the limitations of traditional attention mechanisms.
- **Performance Comparison**: RLGCNt is compared to mainstream models on the STATS dataset, which contains over 180,000 dynamic and static workloads. Experimental results show that RLGCNt is 10.7% more accurate than baseline in dynamic workloads.

2 Related work

2.1 Query-driven Model

The Query-driven model is a deep learning cardinality estimation model that encodes queries. The classic query-driven MSCN model [4] maps feature vectors to a low-dimensional space and then estimates cardinality through a fully connected layer. However, when this model handles dynamic workloads, the accuracy of the cardinality estimation decreases significantly. Deep Sketches [5]

combine convolutional networks and materialized samples for cardinality estimation, but lack data classification, which prevents it from capturing similarities between data. Although this model requires extensive training to adapt to different query modes, it shows significant deviations when faced with unknown queries. Another query-driven model is DBEst [10], which uses a convolutional neural network combined with multiple encoding methods for cardinality estimation, but its performance is average under dynamic workloads because of its poor performance in handling outliers.NNGP [23] is also a query-driven cardinality estimation model, which improves the accuracy of cardinality estimation by using neural networks and capturing spatial similarities between different data in space. However, NNGP has problems in extracting local feature information of queries. In general, query-driven query encoding needs to be improved to address the problems of outliers and local feature extraction.

2.2 Data-driven Model

Another cardinality estimation model is the data-driven model, which performs cardinality estimation based on the modeling of the underlying data. The first is the DeepDB model [3] that uses an RSPN-based data structure. Compared with the previous cardinality estimation methods, the DeepDB model achieves great improvement, but when the query changes, the model often suffers from insufficient generalization capabilities and limited processing of outliers. In addition, Neurocard [19] directly samples the connection results to build an autoregressive model and then trains the autoregressive model in the samples of the complete outer join. However, even if the columns are decomposed and the sub-columns become simpler to a certain extent, there are still several complex columns, and Neurocard fails to capture the similarities between data sufficiently. It is also limited in handling outliers. FACE [16] is a cardinality estimation model based on the rule flow model. It has different processing methods for different data types. If the data are discrete numerical, it is first quantized and processed to turn them into continuous data. Because the FACE model relies too much on continuous data, it affects the accuracy of cardinality estimation.

2.3 Hybrid Model

In addition to the above two models, ALECE [7] combines query-driven and data-driven approaches. Based on the attention mechanism, it can efficiently handle dynamic workloads. It uses the MLP layer of the attention mechanism to merge data and query workload for cardinality estimation. However, the calculation of attention weights is affected both by the arrangement of query data and by its inability to perform fast approximate search, which can lead to errors. PRICE [21] is another hybrid-driven model, but it also ignores the impact of the data arrangement in the query on the cardinality, resulting in reduced efficiency.

The problems with the above three methods can be summarized as follows: the existing cardinality estimation models do not handle outliers in queries well, and ignore the arrangement of query data. Especially in dynamic workloads,

these two problems are particularly prominent, and the local features of data and fast approximate search also need to be improved. Therefore, we address the above four problems.

3 Cardinality estimation

3.1 Problem Definition

Consider a set M with attributes $\{B_1,\ B_2,\ \ldots,\ B_m\}$ and a relation R containing P tuples. Each tuple $u \in R$ is represented as $u = (b_1, b_2, \ldots, b_m)$, where b_j is the value of attribute B_j, and j ranges from 1 to m. The function $o(u)$ indicates the number of tuple u in the relation R. A query Q can be viewed as a function applied to each tuple u. If u meets the query conditions, $Q(u) = 1$; otherwise, $Q(u) = 0$. The cardinality of the query Q is defined as:

$$\text{card}(Q) = |\{u \in R : Q(u) = 1\}|$$

which represents the number of tuples that satisfy the query Q. The selectivity of the query Q is defined as:

$$\text{sel}(Q) = \frac{\text{card}(Q)}{P}$$

Cardinality Estimation (CE) [16, 20] aims to predict the number of tuples card(Q) that satisfy the query conditions of Q without actually running the query.

3.2 Cardinality Estimation of RLGCNt

Deep learning-based cardinality estimation models, which struggle with outliers and uneven data (leading to large estimation errors), are limited in dynamic query scenarios. Additionally, traditional attention mechanisms are easily affected by data arrangement in queries, further degrading estimation accuracy. To address these issues, we propose RLGCNt, which encodes the query using Rank Gaussian Transform coding, combines locality-sensitive hashing and Gaussian kernel functions for joint data-query representation, and finally uses the causal mask in the causal attention mechanism to capture the arrangement of query data. The internal structure of RLGCNt is shown in Fig. 1. In RLGCNt, table join conditions(SJCB, SARC and BRTL)are extracted from the query workload, which include the table join conditions, attribute ranges, and related tables. Then, Rank Gaussian Transform encoding is applied to normalize query distributions, enhancing the model's outlier robustness in large workloads. At the same time, histogram characterizations is used to process the table data. Both representations undergo locality-sensitive hashing (LSH), where high-dimensional data is classified into hash buckets (e.g., Bucket-A, Bucket-B), significantly improving the model's efficiency and robustness. Then, the Gaussian kernel function (GKF) is used to capture data similarities and local features. After LSH

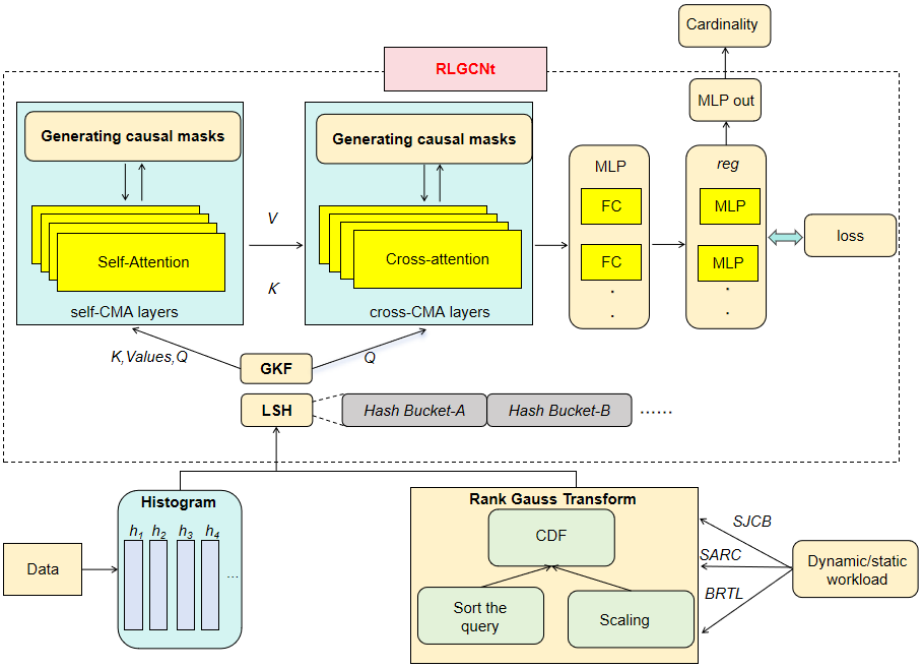

Fig. 1. Cardinality estimation based on RLGCNt.

and GKF processing, the data is submitted to self-attention and cross-attention with multiple heads. The causal mask enables multi-head attention to dynamically prioritize query weights based on data arrangement. The causal mask captures the arrangement of the query data. The output is fed into an MLP layer with stacked fully connected (FC) layers. Similarly, the instance object *reg*, constructed by multiple MLPs, calculates the final cardinality. The regression module (*reg*), constructed via multiple MLPs, is integrated into the loss function for adaptive weight adjustment in self- and cross-CMA layers.

4 RLGCNt Based on Attention Model

4.1 The Rank Gauss Transform Coding

Since one-hot encoding is often ineffective when dealing with queries on massive, non-uniform data and categorical features, we use Rank Gauss Transform to encode each filtering condition, transforming queries from a nonuniform distribution to a normal distribution.

Our approach makes categorical features easier for the attention mechanism to process and solves the problem of outliers and repeated values. The input consists of $SJCB$, $SARC$, and $BRTL$. The first Step involves sorting each element in the query data using the rank data method, specifically by the minimum value.

ranked represents the result after sorting, as shown in Eq. (1). The minimum value sorting method we use assigns repeated values the minimum rank based on their first appearance. xk represents the information vector related to the query, and i represents the index. The next step encodes the input SQL query's join conditions, attribute range conditions, and the list of relevant tables using the Rank Gauss Transform method, as shown in Eq. (2). ng represents the total number of attributes. After that, the length of the SQL conditions is calculated, and the necessary variables are initialized. The process then iterates through all SQL conditions and accumulates the predicted values. The final step involves converting the scaled data into a normal distribution using the Φ function. Φ is the inverse of the Cumulative Distribution Function (CDF) of a probability distribution, and $transformed$ is the new query feature obtained from the output of CDF. Eq. (3) shows that the standard normal distribution value of xk_i is obtained through the inverse cumulative distribution function of the standard normal distribution. Φ represents the CDF of the standard normal distribution.

$$ranked(xk_i) = rankdata(xk_i, method = min) \tag{1}$$

$$scaled\_rank(xk_i) = \frac{ranked(xk_i)}{ng + 1} \tag{2}$$

$$transformed(xk_i) = \Phi^{-1}(scaled\_rank(xk_i)) \tag{3}$$

Once the query is encoded using the Rank Gauss Transform, it is merged with the data features and fed into the multi-head cross-attention layer. Finally, the weight of the query is calculated, and the cardinality is predicted via stacked fully connected (FC) layers.

4.2 Locality Sensitive Hash

The purpose of combining the attention mechanism with locality-sensitive hashing (LSH) is to address the issue that the attention mechanism cannot efficiently approximate query results. This combination reduces the number of query-key-value pair combinations that need to be calculated, thereby greatly speeding up the attention mechanism and improving the accuracy of cardinality estimation. The principle of LSH is to map two similar data points from the original dataset into the same hash bucket. The goal is to ensure that these points remain adjacent in the new bucket, thereby minimizing the probability that non-adjacent data points are mapped to the same bucket.

As shown in Figure 2, tz represents the merged features of the key-value (K, Values) pairs of the input data and the query (Q), which are then divided into different hash buckets such as Hash Bucket-A, Hash Bucket-B, and Hash Bucket-C, etc. The four-line intersecting circle g points to is the similarity matrix generated by the Gaussian kernel function algorithm. The weights are calculated through the causal attention mechanism we proposed (in Section. 5) and then

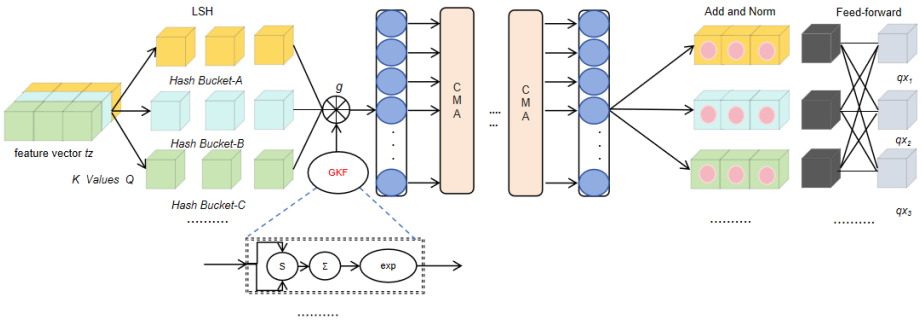

Fig. 2. The LSH of RLGCNt.

transformed through the residual connection and forward feedback layer to obtain the output results qx_1, qx_2, qx_3, ...qx_n. The blue circles represent neurons in the mechanism, which generates feature vectors (rectangles with pink dots), while the black rectangles represent the generated residuals, and finally the gray rectangles represent the feature vectors after processing by the fully connected layer. The remaining content pertains to the Gaussian kernel function and is thoroughly discussed in Section 4.3. Locality-sensitive hashing completes the mapping of data points by creating a hash function and then calculating the hash value to create a hash table. Eq. (4) represents the creation of a hash function, where W_{nl} and b_{nl} represent the weight matrix and bias vector used to create the hash function, and nl represents the number of hash functions. Eq. (5) represents the calculation of the hash value. W_{ib} and b_{ib} represent the weight matrix and bias vector used by the ib-th hash function to compute the hash value, where ReLU is the activation function used, and $hash\_value_{ib}$ represents the calculated hash value corresponding to tz. Eq. (6) represents the connection of the calculated hash values along the last dimension of the feature to obtain the final hash value. Eq. (7) represents the construction of a hash table to combine all hash values h_{nl}, where h represents the hash function. x is the output of locality sensitive hashing.

Storage hash function:

$$(W_1, b_1), (W_2, b_2), \ldots, (W_{nl}, b_{nl}) \tag{4}$$

Calculate hash value:

$$hash\_value_{ib} = ReLU(tzW_{ib} + b_{ib}), i \in [1, nl] \tag{5}$$

Concatenate hash values:

$$\begin{aligned} hashed\_inputs = concat(&hash\_value_1, hash\_value_2, \\ &\ldots, hash\_value_{n_l}, \text{axis} = -1) \end{aligned} \tag{6}$$

Construct a hash table:

$$x = (h_1, h_2, \ldots, h_{nl}) \tag{7}$$

4.3 Gaussian Kernel Function—Cardinality Estimation

We use the combination of the Gaussian kernel function to enhance the ability to capture the local structure of data. The Gaussian kernel function calculates similarity weights to better handle non-linearly related data points and reduce the noise problem caused by attention. First, the feature dimension is expanded, as shown in Eq. (8) and Eq. (9). In these equations, x represents the output of locality-sensitive hashing (LSH), where x_1 and x_2 are parts of x and the expand_dims() function is first used to reshape the input tensor by adding new dimensions, so that x_1 and x_2 can be obtained from the input features for pairwise distance computation. The difference between them is that x_1 inserts a new dimension from the second-to-last dimension to make its shape match the correct form, while x_2 inserts a new dimension from the third-to-last dimension for the purpose of calculating the Euclidean distance. Then, the distance between them according to the Euclidean metric is computed. Eq. (10) shows that after the dimension is expanded, S is obtained by calculating the sum of the squares of the differences between the two feature vectors, and d represents a specific dimension. Therefore, $x_{1,d}$ and $x_{2,d}$ represent the components of x_1 and x_2 in dimension d, respectively. Eq. (11) uses S to calculate the Euclidean distance between them, and D represents the number of feature tuples in the input x. Their similarity is then judged by the Gaussian distribution. Enabling the attention mechanism to better capture the similarity between data. The final Eq. (12) represents dividing the obtained Euclidean distance by σ^2. The role of σ here is to control the Gaussian kernel, which denotes the bandwidth parameter of the Gaussian kernel, and then the range is controlled in (0, 1] through the exponential function. If it is closer to 1, it means that the similarity between the two is higher.

$$x_1 = expand\_dims(x, -2) \tag{8}$$

$$x_2 = expand\_dims(x, -3) \tag{9}$$

$$S = (x_{1,d} - x_{2,d})^2 \tag{10}$$

$$squared\_distance = \sum_0^D S \tag{11}$$

$$\text{Gaussian Kernel} = \exp\left(-\frac{1}{2} \cdot \frac{\text{squared\_distance}}{\sigma^2}\right) \tag{12}$$

4.4 Causal Attention Mechanism

We use the causal attention mechanism to address the issue where the multi-head attention mechanism is affected by the arrangement of query data, thereby reducing the model's over-dependence on future data, limiting the scope of attention, improving the performance of cardinality estimation, and calculating the mask does not increase computational overhead.

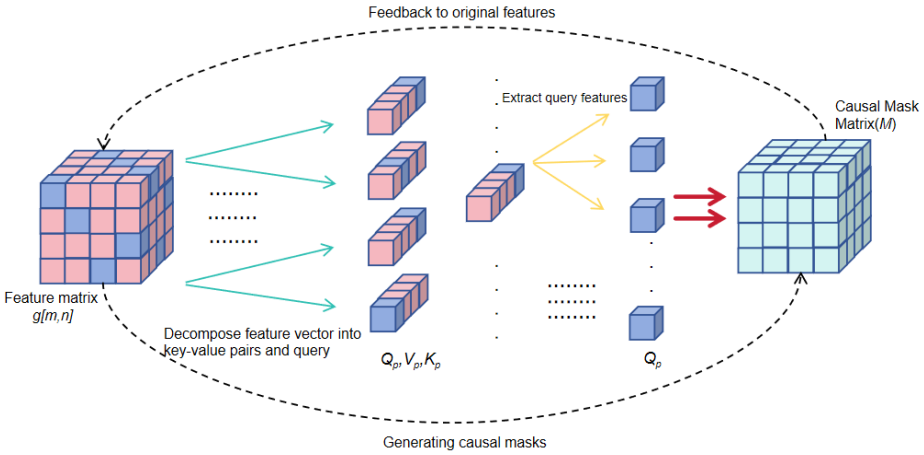

Fig. 3. Mechanism process diagram.

As shown in Fig. 3, the feature matrix on the left is the similarity matrix output by the Gaussian kernel function, represented as the pink matrix, while the blue color represents the query feature vector. First, the required query feature vector is extracted; specifically, the original feature vector is converted into a key-value pair by applying a linear transformation to the weights (as the two green arrows in Fig. 3). Then, the query feature vector is retrieved from the key-value pair (as shown by the yellow arrows). We use the query feature vector to generate a causal mask, as shown by the red arrow. First, a matrix of all ones is generated. Then, elements on and below the main diagonal are kept unchanged, while the rest are set to 0, thus transforming the matrix into an upper triangular form, which serves as the causal mask. Eq. (13) and Eq. (14) represent the equations for generating masks and the equations for the attention mechanism, respectively, where c and j represent the row index and column index of the matrix respectively, and Q_p, K_p and V_p in Eq. (14) represent the values of the key-value pairs after linear transformation, and M represents the causal mask generated in Eq. (13). The final light blue-colored matrix represents the generated causal mask, which is then integrated back into the original feature matrix.

$$\mathbf{M}_{cj} = \begin{cases} 1, & \text{if } c \geq j \\ 0, & \text{if } c < j \end{cases} \tag{13}$$

$$Attention(Q_p, K_p, V_p) = softmax\left(\frac{Q_p K_p^\top}{\sqrt{d_k}} + M\right) V_p \tag{14}$$

5 Loss function

We use MSE [14] as part of the loss function to evaluate the optimization performance. Eq. (15) defines the loss function, which consists of two components. The first component calculates the mean squared error (MSE) between two values, representing the true cardinality and the estimated cardinality, respectively. The second component is the sum of the regularization losses incurred during the forward pass through each fully connected layer.

MSE is given by Eq. (16). The mean squared error between the true cardinality and the estimated cardinality is computed for each sample, where $labels$ represents the true cardinality, $preds$ represents the estimated cardinality, and nv denotes the total number of samples. Finally, we use the Adam optimizer [22] to optimize the model during backpropagation.

$$Loss = MSE(labels, preds) + regularization \tag{15}$$

$$MSE = \frac{1}{nv}\sum_{i=1}^{nv}(labels_i - preds_i)^2 \tag{16}$$

6 Experimental Evaluation

In this section, we introduce the experimental configuration, dataset and dynamic workload used by RLGCNt.

6.1 Experimental Settings and Data Sets

Experimental settings : Our experimental setup included a computer powered by an Intel i7-10700 processor, operating under Ubuntu 20.04.4 and evaluates the optimization method of this paper on more than 180,000 dynamic and static workloads.

STATS[1]: The STATS consists of eight interrelated data tables, forming a complex data ecosystem. These tables include: user information table (storing key data such as user ID, user name, reputation, etc.), post details table (covering post ID, title, content, etc.), post link relationship table, post history change record table, comment interaction table (recording user comments on posts), voting behavior table (tracking voting status of posts and comments), badge recognition table (recording honor badges obtained by users), and label classification table (labeling posts with relevant labels). The dataset is extensive, containing 1,029,842 detailed records.

Dynamic workloadss [7]: The dynamic workloads consist of three parts: insert, update, and delete statements. The initial data set randomly selects two-thirds of these statements in batches and puts them into the relational data table. The remaining one-third is used for update and insert operations.

[1] https://relational.fit.cvut.cz/dataset/Stats

6.2 Evaluation Indicators

Q-error [12]: To show the effect of the model in this paper on cardinality estimation, Q-error is used as an indicator to measure the model in this article and other mainstream methods. The Eq. (17) of Q-error is as follows, where $labels$ represents the true cardinality and $preds$ represents the predicted cardinality.

$$Q - error = max(\frac{labels}{preds}, \frac{preds}{labels}) \tag{17}$$

E2E time [1]: The definition of E2E time is the sum of the time it takes to complete all query executions in the face of different workloads and data sets. It directly reflects the quality of DBMS query performance, and is also an important indicator for measuring the cardinality estimation for query execution speed.

6.3 Comparison Objects

We use the following models to conduct comparative experiments with RLGCNt, including data-driven, query-driven, and hybrid-driven methods.

PostgreSQL [17], mainly obtains the final cardinality through statistical information, which is initially collected manually by the analyze command or automatically collected through the model.

Uni-Samp [8] performs a fast and accurate cardinality estimation without traversing all data sets.

NeuroCard [19] uses Monte Carlo integration to perform progressive sampling in the data set during the probability inference process, which can effectively handle various query situations for the subset problems involved in the query.

FLAT [24] is a data-driven model based on the SPN storage method.

FactorJoin [18] proposes a new binning and upper bounding algorithm to approximate complex factor graph reasoning and combines it with histograms for cardinality estimation.

MLP [15] is not only used for cardinality estimation, but also predicts the final result by fitting the training data for the regression model analysis.

MSCN [4] decomposes the query into tables, join conditions and predicates, and then encodes them one by one.

NNGP [23] estimates the cardinality by the uncertainty of the result.

ALECE [7] processes the underlying data through histogram normalization encoding, and then combines the unique hot encoding and the attention mechanism to perform cardinality estimation.

6.4 Experiment Analysis—Static Workloads

We use ALECE as the baseline for RLGCNt in both static and dynamic workloads. RLGCNt effectively solves the problems of outlier processing and fast approximate querying.

Table 1. Ablation experiment results on STATS dataset–static workloads.

Model	Q-error			
	50%	90%	95%	99%
ALECE	1.687	7.737	16.536	117.331
baseline+LSH+GKF	1.669	7.371	15.650	108.541
baseline+RGC+GKF	1.670	7.909	**15.524**	86.650
baseline+CAM	1.668	7.555	16.042	78.851
RLGCNt	**1.601**	**6.582**	**13.497**	**83.285**

Table 2. Comparative experimental results on the STATS dataset–static workloads.

Model	Q-error				E2E Time (s)
	50%	90%	95%	99%	
PG	1.80	21.84	1.1×10^5	1.8×10^7	12,777
Uni-Samp	**1.33**	6.64	$> 10^{10}$	$> 10^{10}$	15,397
NeuroCard	2.91	192	1,511	1.5×10^5	19,847
MSCN	3.85	39.56	99.81	1,273	15,162
NNGP	8.10	694	3,294	2.3×10^5	20,181
RLGCNt	1.601	**6.582**	**13.497**	**83.285**	**9,565**

As shown in Table 1, LSH denotes the use of locality sensitive hashing before attention, GKF represents the use of the Rank Gaussian Transform instead of the original encoding method, RGC refers to the combination of the Gaussian kernel function with attention, CAM denotes the use of a different attention mechanism instead of the original attention, and RLGCNt represents the model proposed in this paper.

Compared with ALECE, which is based on the attention mechanism, RLGCNt shows a decrease in all four indicators under static workloads.

Table 1 shows that all four indicators have decreased compared with the existing model, with values of 1.669, 7.371, 15.650, and 108.541, respectively. The combination of Rank Gaussian Transform coding and the Gaussian kernel function makes the categorical features smoother and reduces the occurrence of outliers, adding new momentum to the attention-based cardinality estimation.

RLGCNt achieves an overall accuracy improvement of 5.8%.

In Table 2, the four indicators are 1.601, 6.582, 13.497, and 83.285. Compared with traditional cardinality estimation methods such as Uni-Samp, the last three indicators show a significant decrease. Compared with mainstream cardinality estimation methods such as NNGP and MSCN, the four indicators show a significant decrease, with the last two exhibiting the most substantial decline.

The results show that partial locality sensitive hashing, the Gaussian kernel function, and Rank Gaussian Transform coding proposed in this paper offer notable advantages, mitigating the impact of outliers and feature inconsistencies, thereby enhancing the effectiveness of the attention mechanism in cardinality estimation. RLGCNt achieves a substantial reduction in end-to-end processing

time by transforming the query workload into a normal distribution and enabling fast approximate querying.

6.5 Experiment Analysis—Dynamic Workloads

Table 3 shows that RLGCNt has a stronger ability to handle outliers and capture similarities in data features, thereby improving the accuracy of cardinality estimation by 10.7% when facing dynamic workloads.

Table 3. Ablation experiment results on STATS dataset–dynamic Workloads.

Model	**Q-error**			
	50%	90%	95%	99%
ALECE	2.447	11.596	24.899	198.961
baseline+RGC	1.745	7.879	15.954	**68.847**
baseline+LSH	**1.557**	8.454	18.999	143.885
baseline+GKF	1.570	7.307	14.419	148.007
baseline+CAM	2.111	10.001	22.654	166.228
RLGCNt	1.625	**7.200**	**14.090**	85.199

Table 4. Comparative experimental results on the STATS dataset–dynamic workloads.

Model	**Q-error**				**E2E Time (s)**
	50%	90%	95%	99%	
PG	190	1.4×10^5	1.1×10^5	1.8×10^7	7,790
Uni-Samp	**1.35**	12.47	$> 10^{10}$	$> 10^{10}$	6,002
NeuroCard	17.35	1,388	7,402	3.0×10^5	$> 30,000$
FLAT	12.77	1,979	12,897	8.6×10^5	$> 30,000$
FactorJoin	22.62	2,593	31,936	1.6×10^6	$> 30,000$
MLP	2.4×10^6	$> 10^{10}$	$> 10^{10}$	$> 10^{10}$	$> 30,000$
MSCN	20.09	2,870	17,037	2.6×10^5	27,758
NNGP	9.88	827	4,652	2.6×10^5	12,883
ALECE	2.447	11.596	24.899	198.961	2,901
RLGCNt	1.625	**7.200**	**14.090**	**85.199**	**2,623**

As can be seen from Table 4, although the traditional method Uni-Samp performed best in the first indicator, its overall accuracy is low, and the 99th percentile dropped significantly. The structure of RLGCNt proposed in this paper is similar to mainstream methods used for comparison. Compared with traditional cardinality estimation methods, such as PG, all four indicators have decreased significantly. From the last three indicators, RLGCNt reduces the cardinality estimation error to a lower range.

Compared with query-driven methods, such as MSCN and NNGP, RLGCNt has a significant impact in dealing with dynamic workloads, and it produces

fewer extreme estimation values. Compared with data-driven methods, such as NeuroCard, it achieves similar results and demonstrates clear advantages in dynamic workloads.

These results indicate that the accuracy of cardinality estimation has been significantly improved, demonstrating the effectiveness of the four optimization methods proposed in this paper for cardinality estimation.

Following the same principle applied to static loads, RLGCNt shortens the E2E time for dynamic workloads.

7 Conclusion

We propose a cardinality estimation model, RLGCNt, based on the causal attention mechanism. To demonstrate the effectiveness of our model, we conduct extensive experiments on static and dynamic workloads using the STATS dataset. The results show that RLGCNt performs faster and more accurately in cardinality estimation under both static and dynamic workloads. For future research, Fourier neural networks present a promising direction for cardinality estimation.

References

1. Han, Y., Wu, Z., Wu, P., Zhu, R., Yang, J., Tan, L.W., Zeng, K., Cong, G., Qin, Y., Pfadler, A., et al.: Cardinality estimation in dbms: A comprehensive benchmark evaluation. arXiv preprint arXiv:2109.05877 (2021)
2. Harmouch, H., Naumann, F.: Cardinality estimation: An experimental survey. Proceedings of the VLDB Endowment **11**(4), 499–512 (2017)
3. Hilprecht, B., Schmidt, A., Kulessa, M., Molina, A., Kersting, K., Binnig, C.: Deepdb: Learn from data, not from queries! arXiv preprint arXiv:1909.00607 (2019)
4. Kipf, A., Kipf, T., Radke, B., Leis, V., Boncz, P., Kemper, A.: Learned cardinalities: Estimating correlated joins with deep learning. arXiv preprint arXiv:1809.00677 (2018)
5. Kipf, A., Vorona, D., Müller, J., Kipf, T., Radke, B., Leis, V., Boncz, P., Neumann, T., Kemper, A.: Estimating cardinalities with deep sketches. In: Proceedings of the 2019 International Conference on Management of Data. pp. 1937–1940 (2019)
6. Leis, V., Radke, B., Gubichev, A., Kemper, A., Neumann, T.: Cardinality estimation done right: Index-based join sampling. In: Cidr (2017)
7. Li, P., Wei, W., Zhu, R., Ding, B., Zhou, J., Lu, H.: Alece: An attention-based learned cardinality estimator for spj queries on dynamic workloads. Proceedings of the VLDB Endowment **17**(2), 197–210 (2023)
8. Liang, X., Sintos, S., Shang, Z., Krishnan, S.: Combining aggregation and sampling (nearly) optimally for approximate query processing. In: Proceedings of the 2021 International Conference on Management of Data. pp. 1129–1141 (2021)
9. Lin, X., Zeng, X., Pu, X., Sun, Y., et al.: A cardinality estimation approach based on two level histograms. J. Inf. Sci. Eng. **31**(5), 1733–1756 (2015)
10. Ma, Q., Triantafillou, P.: Dbest: Revisiting approximate query processing engines with machine learning models. In: Proceedings of the 2019 International Conference on Management of Data. pp. 1553–1570 (2019)

11. Marcus, R., Papaemmanouil, O.: Plan-structured deep neural network models for query performance prediction. arXiv preprint arXiv:1902.00132 (2019)
12. Moerkotte, G., Neumann, T., Steidl, G.: Preventing bad plans by bounding the impact of cardinality estimation errors. Proceedings of the VLDB Endowment **2**(1), 982–993 (2009)
13. Ning, Z., Zhang, Z., Sun, T., Tian, Y., Zhang, T., Li, T.J.J.: An empirical study of model errors and user error discovery and repair strategies in natural language database queries. In: Proceedings of the 28th International Conference on Intelligent User Interfaces. pp. 633–649 (2023)
14. Prasad, N.N., Rao, J.N.: The estimation of the mean squared error of small-area estimators. Journal of the American statistical association **85**(409), 163–171 (1990)
15. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. pp. 318–362 (1986)
16. Wang, J., Chai, C., Liu, J., Li, G.: Cardinality estimation using normalizing flow. The VLDB Journal **33**(2), 323–348 (2024)
17. Woltmann, L., Olwig, D., Hartmann, C., Habich, D., Lehner, W.: Postcenn: postgresql with machine learning models for cardinality estimation. Proceedings of the VLDB Endowment **14**(12), 2715–2718 (2021)
18. Wu, Z., Negi, P., Alizadeh, M., Kraska, T., Madden, S.: Factorjoin: a new cardinality estimation framework for join queries. Proceedings of the ACM on Management of Data **1**(1), 1–27 (2023)
19. Yang, Z., Kamsetty, A., Luan, S., Liang, E., Duan, Y., Chen, X., Stoica, I.: Neurocard: one cardinality estimator for all tables. arXiv preprint arXiv:2006.08109 (2020)
20. Yang, Z., Liang, E., Kamsetty, A., Wu, C., Duan, Y., Chen, X., Abbeel, P., Hellerstein, J.M., Krishnan, S., Stoica, I.: Deep unsupervised cardinality estimation. arXiv preprint arXiv:1905.04278 (2019)
21. Zeng, T., Lan, J., Ma, J., Wei, W., Zhu, R., Li, P., Ding, B., Lian, D., Wei, Z., Zhou, J.: Price: a pretrained model for cross-database cardinality estimation. arXiv preprint arXiv:2406.01027 (2024)
22. Zhang, Y., Chen, C., Li, Z., Ding, T., Wu, C., Ye, Y., Luo, Z.Q., Sun, R.: Adammini: Use fewer learning rates to gain more. arXiv preprint arXiv:2406.16793 (2024)
23. Zhao, K., Yu, J.X., He, Z., Li, R., Zhang, H.: Lightweight and accurate cardinality estimation by neural network gaussian process. In: Proceedings of the 2022 International Conference on Management of Data. pp. 973–987 (2022)
24. Zhu, R., Wu, Z., Han, Y., Zeng, K., Pfadler, A., Qian, Z., Zhou, J., Cui, B.: Flat: fast, lightweight and accurate method for cardinality estimation. arXiv preprint arXiv:2011.09022 (2020)