ACE: A Cardinality Estimator for Set-Valued Queries

Yufan Sheng
University of New South Wales
Sydney, Australia
yufan.sheng@unsw.edu.au

Xin Cao*
University of New South Wales
Sydney, Australia
xin.cao@unsw.edu.au

Kaiqi Zhao
The University of Auckland
Auckland, New Zealand
kaiqi.zhao@auckland.ac.nz

Yixiang Fang
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
fangyixiang@cuhk.edu.cn

Jianzhong Qi
The University of Melbourne
Melbourne, Australia
jianzhong.qi@unimelb.edu.au

Wenjie Zhang
University of New South Wales
Sydney, Australia
wenjie.zhang@unsw.edu.au

Christian S. Jensen
Aalborg University
Aalborg, Denmark
csj@cs.aau.dk

ABSTRACT

Cardinality estimation is a fundamental functionality in database systems. Most existing cardinality estimators focus on handling predicates over numeric or categorical data. They have largely omitted an important data type, set-valued data, which frequently occur in contemporary applications such as information retrieval and recommender systems. The few existing estimators for such data either favor high-frequency elements or rely on a *partial independence assumption*, which limits their practical applicability.

We propose ACE, an Attention-based Cardinality Estimator for estimating the cardinality of queries over set-valued data. We first design a distillation-based data encoder to condense the dataset into a compact matrix. We then design an attention-based query analyzer to capture correlations among query elements. To handle variable-sized queries, a pooling module is introduced, followed by a regression model (MLP) to generate final cardinality estimates. We evaluate ACE on three datasets with varying query element distributions, demonstrating that ACE outperforms the state-of-the-art competitors in terms of both accuracy and efficiency.

PVLDB Reference Format:
Yufan Sheng, Xin Cao, Kaiqi Zhao, Yixiang Fang, Jianzhong Qi, Wenjie Zhang, and Christian S. Jensen. ACE: A Cardinality Estimator for Set-Valued Queries. PVLDB, 18(7): 2112 - 2125, 2025.
doi:10.14778/3734839.3734848

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/shengyufan/ACE.

1 INTRODUCTION

Set-valued data where the value of an attribute is a set of elements has emerged as an essential data type in many real-world applications, including information retrieval [8], recommender systems [1], and social networks [43]. For example, in a movie recommender system, each movie's genre is generally associated with a set of categories such as sci-fi, action, and comedy. In X (http://www.twitter.com), each tweet generally has multiple hashtags. Table 1 shows a toy example, where each row corresponds to a tweet and the *Hashtags* column is a set-valued attribute that stores the hashtags of a tweet.

Table 1: Twitter hashtag dataset

Tweet_ID	Hashtags
t_1	{Trump, shot}
t_2	{Spain, Euros, Yamal}
t_3	{Biden, Harris, Trump}
t_4	{Harris, Trump, debate}
t_5	{JD Vance, Trump}
t_6	{Messi, Yamal}
t_7	{Messi, Argentina, Copa America}

Given the prevalence of set-valued data across different domains, set queries play a crucial role in efficiently handling multi-valued attributes and complex relationships. For example, X offers the functionality of searching for tweets by a set of keywords or hashtags using operators such as "OR" and "AND". The SQL standard includes support for storing multi-valued data in a single row [39]. Set-valued data and set queries are supported to varying degrees by modern DBMSs such as Oracle [64], MySQL [63], IBM DB2 [31], SQL Server [73], and PostgreSQL [68]. For example, MySQL supports up to 64 distinct elements in a set-valued attribute [63]. SQL Server enables passing a set as a table-valued parameter. To the best of our knowledge, PostgreSQL offers the best support for set-valued data and set queries. It provides three set query predicates: *superset* (`@>`), *subset* (`<@`), and *overlap* (`&&`). For example, to evaluate public interest in the recent United States presidential election, we can count the number of tweets containing at least one presidential candidate. This can be achieved using the following set query q in PostgreSQL: `SELECT COUNT(*) FROM T WHERE T.Hashtags && ARRAY["Trump", "Harris"].`

To identify efficient query execution plans for complex queries, cardinality estimation of a query step plays a crucial role since it directly influences the efficiency of database query execution. Cardinality estimation has been extensively studied [27, 33, 34, 48, 67, 90], showing its profound impact on the quality of selected query plans [25, 47]. However, most DBMSs provide only limited support for optimizing set query execution. To our knowledge, PostgreSQL is the only DBMS that offers a built-in estimator for set

*The corresponding author.

Proceedings of the VLDB Endowment, Vol. 18, No. 7 ISSN 2150-8097.
doi:10.14778/3734839.3734848

operators but the accuracy is not good enough. In this study, we investigate cardinality estimation for queries over set-valued data, which has not received sufficient attention. While some cardinality estimators for set-valued predicates exist [40, 59, 96], they each have significant shortcomings.

First, most studies **pay more attention to elements with high frequency (P1).** For example, Yang et al. [96] propose two sampling-based cardinality estimators for subset queries that aim to capture the distribution of high-frequency elements. They suffer in accuracy over queries containing low-frequency elements [59].

Second, most existing estimators **do not capture the correlation among elements in a query well (P2)**, which is crucial for accurate estimation. For example, "Harris" and "Trump" appear 2 and 4 times, respectively, in the example. If we assume independence, the estimated cardinality for q is $2 + 4 = 6$, while the actual cardinality is 4 because t_3 and t_4 contain both keywords. Korotkov et al. [40] leverage a probabilistic model [16] to address the element correlation issue. However, their model still relies on random sampling of high-frequency elements, thus missing the correlation for low-frequency elements. Recently, Meng et al. [59] propose to convert a set-valued column into multiple categorical columns. They then utilize existing estimators to capture the correlation between columns. This approach still ignores the correlation among elements within the same subcolumn, leading to unstable estimation accuracy. Besides, this solution relies on that a set query can be converted into categorical sub-queries, which does not support all set queries such as the overlap query.

Third, existing studies mainly focus on capturing data distribution and **overlook the valuable insights in historical query workloads (P3).** The cardinalities of two similar queries can differ on the operators used, even when their elements are identical. For example, the cardinality of the example query ($q = T$ && {"Harris", "Trump"}) is 4 while the cardinality of a similar superset query ($q' = T$ @> {"Harris", "Trump"}) is 2. Learning the data distribution only is insufficient for accurate predictions across various query types. Set Transformer [46] processes input sets using an attention mechanism to capture the correlations between elements, making it a potential candidate for a query-driven estimator. However, our experiments in Section 7 show that its accuracy is unstable and sometimes performs much worse than data-driven estimators, because it is impractical to represent all possible combinations of elements given limited training data. Thus, pure query-driven methods that treat the problem as a supervised learning task also have a severe issue: their accuracy highly depends on the quantity and quality of the training data (i.e., known query workload) [25].

To address the issues above, we propose ACE, an Attention-based Cardinality Estimator for queries over set-valued data. As depicted in Figure 1, ACE leverages information from both the data and the query workload to address **P3**.

To address **P1**, we design a distillation-based data encoder to generate a compact dataset representation. We construct a bipartite graph that models the relationships between the set elements and their corresponding sets. This graph serves as the foundation for the subsequent aggregation step. The aggregator module synthesizes a set embedding by integrating information from all elements of the set, ensuring that even low-frequency elements are not underrepresented. Once the set embeddings are computed, they are

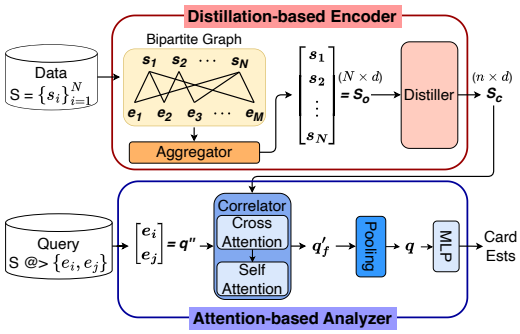


Figure 1: Overview of ACE.

Table 2: Properties of different estimators

Method	Query supported			Data-driven	Query-driven	Low-frequency elements	Element correlation
	Superset	Subset	Overlap				
PostgreSQL	✓	✓	✓	✓	×	×	×
Sampling	✓	✓	✓	✓	×	×	×
OT-S [96]	✓	✓	✓	✓	×	×	×
ST [59]	✓	✓	×	✓	×	✓	⧸
STH [59]	✓	✓	×	✓	×	✓	⧸
ACE (Ours)	✓	✓	✓	✓	✓	✓	✓

concatenated to form the initial representation of the full dataset (i.e., a database table of set-valued data), denoted as S_o. For large datasets, this representation has a high dimensionality, posing significant challenges for downstream learning tasks. To address this issue, a distiller module is designed to produce a more compact representation, S_c, which compresses the original representation with a fixed ratio while preserving as much information as possible.

Next, we design a correlator module to capture element correlations and address **P2**. For each query in a given workload, we apply a cross-attention mechanism to generate the query element embeddings using the data representation obtained from the previous step. The computational complexity of the attention mechanism [5, 11, 75, 81] scales with the size of the input, i.e., the number of rows in the data representation and the number of query elements, which emphasizes the necessity of the distillation step. Then, we utilize the self-attention mechanism to capture the correlation between the learned latent representation of query elements. It is noteworthy that set-valued queries have varying sizes, bringing another challenge for the learning-based estimator. We address the challenge with a pooling module to generate a fixed-sized query embedding $\boldsymbol{q}$ and finally a linear regression model to map the embedding to a cardinality estimation.

Table 2 summarizes the novelty of our estimator ACE compared with existing set-valued query cardinality estimators. Please note that we use the symbol ⧸ because ST and STH can only capture correlations between elements in different columns. Overall, we make the following contributions:

- We propose ACE, a learning-based cardinality estimator for queries over set-valued data, exploiting both the data and query workload distributions.
- We design a distillation-based data encoder to generate a dataset compact representation, reducing the dimensionality while retaining key information.
- We propose an attention-based query analyzer that captures correlations among query elements, followed by a pooling method to address the issue of variable-sized queries.

- We compare ACE and the state-of-the-art estimators on real-world datasets and query workload. The results show that ACE outperforms the SOTA estimators by up to 33.9× in terms of accuracy while offering a stable estimation latency. Additionally, the integration of ACE and PostgreSQL also speeds up the end-to-end execution for complex queries.

2 PRELIMINARIES

We start with our problem statement and technical background for our proposed model. Table 3 lists the frequently used notation.

Table 3: Frequently used notation

Notation	Meaning
$S = \{s_i\}_{i=1}^{N}$	The set-valued dataset
$E = \{e_j\}_{j=1}^{M}$	The element universe of the dataset
d	The dimension of the embedding
B_d/B_q	The batch size of data/query
r	The distillation ratio
$n_{distill}$	The number of layers for the distillation model
n_{cross}/n_{self}	The number of layers for the cross/self attention
$\boldsymbol{s}/\boldsymbol{e}/\boldsymbol{q}$	The set/element/query embedding
$\boldsymbol{S_o}/\boldsymbol{S_c}$	The original/distilled matrix of the dataset
$\boldsymbol{Q}/\boldsymbol{K}/\boldsymbol{V}$	The queries/keys/values of the attention mechanism

2.1 Problem Statement

Definition 1 (**Set-Valued Query**). *A set-valued query q = (operator, literal) is a predicate over the set-valued data, represented by an operator-literal pair. To be consistent with PostgreSQL, operator can be the superset (@>), subset (<@), or overlap (&&) operator, while literal is a subset of E.*

For example, $q = S$ @> $\{e_1, e_3, e_7\}$ is to find the sets over S, each of which is a superset of $\{e_1, e_3, e_7\}$.

Problem. The study aims to propose an estimator that can accurately and efficiently predict the cardinality of a set query.

2.2 Attention Mechanism

The attention mechanism was originally envisioned as an enhancement to the encoder-decoder Recurrent Neural Network (RNN) in sequence-to-sequence applications [5]. In neural networks, attention is a technique that aims to mimic human cognitive attention, and its motivation is that a network should focus on the important parts of the data rather than treating all data equally. It employs an attention function to decide which part of the data should be emphasized. This function maps a query and a collection of key-value pairs, assigns weights by computing the similarity between each pair of the query and a key using some metric, and calculates the weighted sum of values as its output. Therefore, compared to other neural networks, the attention mechanism can achieve better interpretability and have higher representative abilities.

In this study, we use the standard Scaled Dot-Product Attention [81], called *Att*. The keys and values are mapped to matrices $\boldsymbol{K}$ and $\boldsymbol{V}$, of dimensions $n \times d_k$ and $n \times d_v$, where n, d_k, and d_v denote the size of the original input and the dimensions of the matrices $\boldsymbol{K}$ and $\boldsymbol{V}$. A query is first converted to an $m \times d_k$ matrix $\boldsymbol{Q}$, where m indicates the size of the query and is used as input to the dot product. If $\boldsymbol{Q}$ is different from $\boldsymbol{K}$ and $\boldsymbol{V}$, it is called cross-attention. Otherwise, it is self-attention. Since a large dot product result often leads to the vanishing gradient problem, the *Att* function divides the dot product by the factor $\sqrt{d_k}$. The process consists of calculating the dot products of $\boldsymbol{Q}$ with all keys $\boldsymbol{K}$, dividing each by $\sqrt{d_k}$, applying the softmax function to obtain the weights, and obtaining the output by multiplying the weights and the values $\boldsymbol{V}$.

$$Att(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = softmax\left(\frac{\boldsymbol{Q} \cdot \boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V}$$

Next, we adopt the multi-head attention mechanism [81], which linearly projects the queries, keys, and values using h different linear projections and then computes the *Att* function in parallel. The independent attention outputs are then concatenated and linearly transformed into the expected dimension. Compared with single-head attention, this approach processes different projected spaces jointly, thus capturing complex patterns from different perspectives.

$$MultiHead(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = concat(head_1, \cdots, head_h)\, W_o,$$

where $head_i = Att(\boldsymbol{Q}W_i^Q, \boldsymbol{K}W_i^K, \boldsymbol{V}W_i^V)$.

3 OVERVIEW OF ACE

The structure of ACE is shown in Figure 1. The encoder (Section 4) generates a compact data representation, while the analyzer (Section 5) captures correlations between query elements by learning from both the queries and the underlying data.

As in previous studies [30, 50], the first task is representing the dataset properly. Sets in S are combinations of elements, and we can naturally represent a set as the concatenation of the embeddings of its elements. However, this representation is incompatible with neural networks due to the variable sizes of sets. We need to convert variable-sized sets into fixed-sized vectors. Traditional methods, including padding and truncation, have limitations. For example, padding causes storage overheads and increases time complexities. Instead, we propose to learn the representations of sets. Assuming the number of elements that can occur in sets is M, there are $2^M - 1$ possible sets, making it hard to design one model to represent all the sets. We propose to construct a bipartite graph to model the dependencies between elements and sets and to learn an aggregator to obtain a set embedding $\boldsymbol{s}$ by aggregating the information of each element $\boldsymbol{e}$ in the set. Thus, the underlying dataset is represented by a matrix $\boldsymbol{S_o}$ where each row is the embedding of a set.

Next, we aim to learn the representation of the query elements from data. This motivates us to employ a cross-attention mechanism to discover the relation between the underlying dataset and each query element. The original data matrix $\boldsymbol{S_o}$ cannot be used directly in the attention framework for large-scale datasets. For instance, training on our smallest dataset **GN** requires 42GB of GPU memory, even with a batch size of 1. Thus, we design a distiller module to obtain a matrix $\boldsymbol{S_c}$ that preserves the essential knowledge in $\boldsymbol{S_o}$.

As shown in the example query in Section 1, it is crucial to capture correlations between query elements, influenced by the underlying data, to get accurate estimates. Thus, we propose a correlator module to achieve this. We first leverage a data-query cross-attention to measure the relevance between each query element and the underlying data. We then adopt a self-attention mechanism to capture the correlations between the query elements. In addition, the attention mechanism is suitable for dealing with sets as the order of set elements does not affect the output.

To handle the variable-size queries, we utilize a pooling module to derive a fixed-sized vector. This vector extracts pertinent information from the output of the self-attention mechanism and adapts its focus based on the operator type of a set-valued query.

Offline training. The training process is divided into two distinct phases. In the first phase, we employ an unsupervised learning approach to train the data encoder, requiring only a small subset of the dataset. In the second phase, we utilize a supervised learning method to train the query analyzer, using both the query embeddings and the distilled matrix generated by the encoder as input. During this stage, true query cardinalities serve as ground-truth labels. The entire training procedure leverages stochastic gradient descent (SGD) optimization [70].

Online estimation. In the pre-processing phase, a well-trained data encoder can distill the entire dataset into a compact data matrix. When a new query arrives, we can only utilize the learned query analyzer to estimate the cardinality efficiently, taking the query element embeddings and the matrix as input.

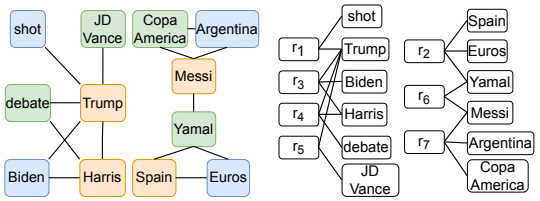
4 DATASET FEATURIZATION

When representing the set-valued dataset, PostgreSQL uses histograms to approximate the distribution of the underlying data. A recent study [59] converts a set into a smaller number of numerical values, models the factorization problem as a graph coloring problem, and proposes a greedy method to address the NP-hard problem. However, this method cannot measure the correlation between the elements in the same partition. Recently, machine learning techniques have opened the opportunity to learn models that outperform many traditional methods [42, 102]. Thus, we aim to learn a model that encodes each set and generates the data representation. We also design a distillation model such that the featurization matrix can be effectively used in the attention mechanism. The details are given in Sections 4.1 and 4.2.

4.1 Set Representation

We follow the setting used in existing studies [40, 45] where the element universe E is finite and fixed, meaning each set consists of known elements. To partition a set into several clusters, the existing work [59] builds an undirected graph based on the underlying data, where edges connect two elements that appear in the same set and uses a greedy algorithm to partition elements into k clusters. Taking the scenario of $k = 3$ as an example, the algorithm proceeds in two phases. In the first stage, it builds a graph and uses the largest first algorithm [41] to obtain initial partitions, ensuring that no elements within a partition are contained in the same set. In the second stage, the algorithm greedily merges partitions to produce the result clusters, as illustrated in Figure 2a, where elements of the same color belong to the same cluster. Although they utilize the existing works [30, 97] to capture the correlation among clusters, the correlation between the elements within the same cluster, such as "Trump" and "Harris", cannot be measured. Unlike the previous method, we represent the dataset as embeddings so that we can leverage the machine learning method to capture the correlation between elements appeared in a query.

However, proposing such a model is non-trivial. As analyzed in Section 3, the number of combinations of M elements equals $2^M - 1$, making it challenging to learn a model that considers all possibilities. Motivated by existing works [10, 91, 105], we build a bipartite graph to model the correlation between elements and sets, as shown in Figure 2b. In this graph, an edge connects an element to a set if the element appears in that set. Now, the problem shifts to representing a set s when its comprised elements are known.



(a) Element graph (b) Element-set graph

Figure 2: Graph construction approaches.

Following this motivation, we propose an aggregator module that takes element embeddings as input. A naïve approach is to adopt pooling methods directly. However, pooling methods discard considerable amounts of information. For example, max-pooling only retains the highest value, ignoring the rest. Additionally, pooling methods are non-trainable operations that cannot adapt to various data, limiting their ability to extract complex representations and potentially leading to suboptimal feature compression. Prior studies [24, 86, 99, 104] show that the Multi-Layer Perceptron (MLP) [28] offers a simple yet effective approach to compute feature representations for each input. Thus, we utilize an MLP to aggregate the information from elements before performing pooling. The process of generating the set embedding $\boldsymbol{s}$ is described as follows (where $\boldsymbol{e_j}$ is the embedding of the element e_j):

$$\boldsymbol{s} = Pool\left(\left\{MLP\left(\boldsymbol{e_j}\right), \forall e_j \in s\right\}\right)$$

Given the lack of inherent order among elements, any symmetric vector function can be used as the pooling operator. Following the prior work [24], we utilize the simple single-layer architecture with the mean-pooling operator.

4.2 Dataset Distillation

The above aggregation model can encode each set as a $1 \times d$ embedding and we can concatenate them together to obtain an $N \times d$ matrix $\boldsymbol{S_o}$ representing the dataset, where N denotes the number of sets in the dataset. Motivated by the existing work [50], we aim to use the attention mechanism to link query elements with the dataset representation. However, when dealing with large-scale datasets, directly using $\boldsymbol{S_o}$ is unrealistic. As introduced in Section 2.2, the correlation in the attention mechanism is captured by the dot product of $\boldsymbol{Q}$ and $\boldsymbol{K^T}$, meaning that the complexity of the mechanism is proportional to the size of the dataset matrix. Since the size of real-world dataset is usually larger than 10^6, we aim to synthesize a small dataset such that models trained on it achieve high performance on the original large dataset.

A naïve method is to draw a small sample from the original data matrix. However, the resultant matrix is lossy, and the performance depends heavily on the sampling quality. Recently, the problem of dataset distillation has been studied in the field of computer vision. Existing works [49, 85, 106] introduce different algorithms that take as input a large real dataset to be distilled and output a small synthetic distilled dataset, which is evaluated via testing models trained on this distilled dataset on a separate real dataset. However, these methods cannot be adopted to solve our problem because the

dataset studied in previous works always has the label information and they distill the data of the same class into a small dataset. For example, the image dataset can be represented as $T = \{(x_g, y_g)\}_{g=1}^{G}$ where G denotes the number of training images, x_g and y_g denote the image and its corresponding label, respectively. Then, they propose various approaches that can compress thousands of training images into just several synthetic distilled images (e.g. one per class) and achieve comparable performance with training on the original dataset. However, our dataset representation lacks the essential label information. Therefore, we aim to propose a distillation model that can compress the large unlabeled dataset.

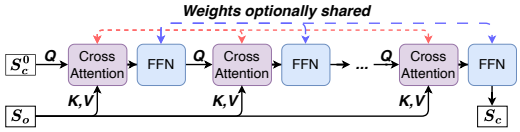


Figure 3: Distillation model.

As shown in the previous work [81], the self-attention mechanism allows the model to aggregate information across tokens. Building on this, we aim to utilize the attention mechanism to compress the dataset $\boldsymbol{S_o}$. Since self-attention produces an output matrix with the same dimensions as its input, we propose an iterative cross-attention model, as illustrated in Figure 3.

Each cross-attention block consists of a single attention layer Att, followed by a feed-forward neural network FFN. Initially, we sample a set of embeddings as the initial value $\boldsymbol{S_c^0}$. Then, we project the distilled matrix to the query $\boldsymbol{Q}$ while mapping the original matrix to the key $\boldsymbol{K}$ and the value $\boldsymbol{V}$. Note that we adopt residual connections [29] and layer normalization [4] in our framework.

$$\tilde{S}_c^i = LayerNorm\left(S_c^{i-1} + Att\left(S_c^{i-1}, S_o, S_o\right)\right),$$
$$S_c^i = LayerNorm\left(\tilde{S}_c^i + FFN\left(\tilde{S}_c^i\right)\right)$$

By iteratively applying the cross-attention mechanism, our model can extract useful information from the original matrix while reducing the size of the matrix simultaneously. This model can also be seen as performing the clustering of the inputs with the latent positions as cluster centers, leveraging highly asymmetric cross-attention layers. Following previous works [35, 93], we share weights between each instance of the cross-attention module (except the first one) for parameter efficiency. Consequently, we utilize the smaller $\boldsymbol{S_c}$ as input of the following query analyzer.

4.3 Encoder Training

The dataset encoder comprises two distinct modules, each with an optimization objective. To address this, we propose a combined loss function that integrates both objectives, allowing training the two modules simultaneously, in line with previous works [26, 56, 77].

The aggregation module aims to generate the set embeddings by integrating the information from elements. To achieve this, we predict whether there is an edge connecting the set and the element based on their embeddings. Following the previous work [91], we use the cross entropy (CE) [6] as the loss function to maximize log probabilities for one-hop structure learning.

$$L_{CE} = \sum_i \left(-\boldsymbol{s_i} \boldsymbol{e_j}^T + log\left(\sum_{e_k \in N(s_i) \cup e_j} \boldsymbol{s_i} \boldsymbol{e_k}^T\right)\right),$$

where $e_j \in s_i$ and $N(s_i) = \{e_l \mid e_l \notin s_i\}$ denote the positive sample and the collection of negative samples, respectively.

Regarding the distillation module, the objective is to compress the dataset while persevering the knowledge as much as possible. Motivated by the previous work [101], we use the maximum mean discrepancy (MMD) [19] as the loss function. The primary purpose of MMD is to determine whether two distributions are similar by comparing their samples. This is achieved by mapping the samples to a high-dimensional feature space using a kernel function and then computing the mean distance between these features. This function is particularly useful in transfer learning [57], which needs to quantify the difference between two sets of data.

$$L_{MMD} = \frac{1}{n^2} \sum_{i,j} k(\boldsymbol{S_{ob}^i}, \boldsymbol{S_{ob}^j}) + \frac{1}{m^2} \sum_{i,j} k(\boldsymbol{S_{cb}^i}, \boldsymbol{S_{cb}^j}) - \frac{2}{nm} \sum_{i,j} k(\boldsymbol{S_{ob}^i}, \boldsymbol{S_{cb}^j}),$$

where k is a kernel function (e.g., the Gaussian kernel) while n and m denote the size of batch data $\boldsymbol{S_{ob}}$ and $\boldsymbol{S_{cb}}$, respectively.

To train these models, we first split the underlying data based on the batch size B_d. Then, the data batches are divided into two parts, the training dataset and the testing dataset. Since the element universe is finite and fixed, we create the fixed representation with dimension $M \times d$, where each row represents one element. For each training batch, we propose a hybrid training method that minimizes an overall loss function L, combining L_{CE} and L_{MMD}. To prevent overfitting, we also use the L2 regularization technique.

$$\min\left(L + \lambda L2_{reg}\right) = \min\left(L_{CE} + L_{MMD} + \lambda \left\|\Theta\right\|^2\right),$$

where λ is the hyper-parameter to adjust the weight.

5 ANALYZER DESIGN

Given a query q and the distilled dataset $\boldsymbol{S_c}$, ACE discovers the relations between query elements and data. Then, we capture the correlation between query elements. The key challenge is the attention mechanism [81] used in ACE. To handle the variable-size input, we also propose an attention-based pooling method. Finally, we employ a linear regression model to predict the cardinality of q, taking the fixed-size embedding as the input. Detailed explanations are provided in Sections 5.1 and 5.2.

5.1 Element Correlation

Before estimating the cardinality of a query q, we need to obtain the query representation. A naïve method is to leverage the trained aggregator to integrate the information from query elements. Because the element embeddings are randomly initialized and fixed in the data encoder, these initial embeddings lack meaningful information. Additionally, the aggregator cannot capture the correlation between query elements $\{e_i \mid e_i \in q\}$. Therefore, we need to propose another method to complete the task.

Considering Figure 2b, each element can also be represented as the collection of sets containing it. Thus, we propose to learn the query representation from the underlying data. A simple method is to flatten $\boldsymbol{S_c}$ into a vector and concatenate the vector with embeddings of query elements. Then, the vector combining data and query information can be fed into an MLP to generate the query embedding. However, we observe that an element has a stronger relation to the sets containing it. Therefore, we leverage the cross-attention mechanism, which can pay more attention to these sets and learn better embeddings of query elements based on the distilled dataset representation.

After obtaining the embeddings, we need to capture the correlations hidden in the embeddings. A simple approach is to use

an MLP to learn the correlations dynamically. However, an MLP applies the same transformation to all inputs regardless of their importance and struggles to capture complex correlations. Motivated by the previous work [50, 81], we utilize the self-attention mechanism, taking these latent embeddings as the input, to capture the correlations between elements.

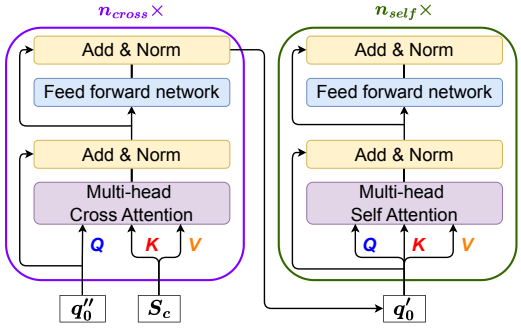


Figure 4: Hybrid attention framework.

In the first stage, we initialize the query embedding by stacking query element embeddings and update the query embedding by considering the information from the dataset. We employ n_{cross} stacked attention layers to capture the correlations between the initial query embedding $\boldsymbol{q}''_0$ and the distilled data matrix $\boldsymbol{S_c}$. Each layer is identical and includes two sub-layers. The first is the multi-head cross-attention sub-layer Att_c where $\boldsymbol{S_c}$ is used as $\boldsymbol{K}$ and $\boldsymbol{V}$ while $\boldsymbol{Q}$ uses $\boldsymbol{q}''_0$ or the output of the last layer. On top of Att_c, the feed-forward sub-layer FFN uses stacked fully connected networks and nonlinear activation functions, e.g., GeGLU [74], to map $\tilde{\boldsymbol{q}}''_i$ into the latent representation $\boldsymbol{q}''_i$. To prevent performance degradation and ease the model training, we also employ a residual connection [29], followed by layer normalization [4].

$$\tilde{\boldsymbol{q}}''_i = LayerNorm\left(\boldsymbol{q}''_{i-1} + Att_c(\boldsymbol{q}''_{i-1}, \boldsymbol{S_c}, \boldsymbol{S_c})\right),$$
$$\boldsymbol{q}''_i = LayerNorm\left(\tilde{\boldsymbol{q}}''_i + FFN(\tilde{\boldsymbol{q}}''_i)\right)$$

The attention sub-layer establishes a bridge between query elements and data. It obtains element representations by aggregating information from the most relevant parts of data embeddings $\boldsymbol{S_c}$, while diminishing others. The effect of particular attention can be realized through learnable parameters of different layers.

In the second stage, we discover and measure the correlation between query elements. We stack n_{self} identical attention layers. Similar to the first stage, each layer consists of a multi-head attention sub-layer and a feed-forward sub-layer. Also, residual connections are employed, followed by layer normalization. Unlike the first stage, this self-attention sub-layer Att_s takes the same inputs of keys, values, and queries. They are either the output of the first module, denoted as $\boldsymbol{q}'_0$, or the previous stacked layers.

$$\tilde{\boldsymbol{q}}'_i = LayerNorm\left(\boldsymbol{q}'_{i-1} + Att_s(\boldsymbol{q}'_{i-1}, \boldsymbol{q}'_{i-1}, \boldsymbol{q}'_{i-1})\right),$$
$$\boldsymbol{q}'_i = LayerNorm\left(\tilde{\boldsymbol{q}}'_i + FFN(\tilde{\boldsymbol{q}}'_i)\right)$$

As introduced above, $\boldsymbol{q}'_0$ can be considered as new embeddings for query elements, which integrate the information from the underlying dataset. The use of the same input for the keys, values, and queries makes each element in the output set of a layer attend to all outputs in the previous layer and thus attend to all elements. More importantly, the self-attention sub-layer quantitatively 'measures' the relevance between a pair of elements, enabling the effective discovery of implicit correlations between elements. Thus, the information from the data and query is encoded into the final output embedding $\boldsymbol{q}'$ that will be processed later.

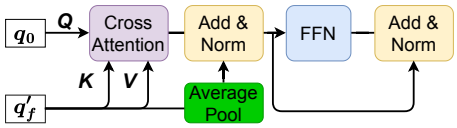


Figure 5: Attention pooling.

5.2 Attention Pooling

Through the hybrid attention framework, the query embedding $\boldsymbol{q}'$ not only links the query with the underlying dataset but also includes the correlation information of query elements, which can be used for the cardinality estimation task. A model for set-input problems should satisfy two fundamental requirements. First, it should be permutation invariant, that is, the output of the model should not change under any permutation of the elements in the input set, which is inherently satisfied by our hybrid attention framework. Second, such a model should be able to process input sets of any size. For example, if the literal of a query is composed of k elements, the dimension of the output query embedding $\boldsymbol{q}'$ will be $k \times d$. Generally, three methods address this problem – pooling, padding, and truncation. Pooling effectively reduces input size by aggregating information from local regions, thereby reducing computational load [17]. In contrast, padding increases input size, while truncation may result in the loss of important information. Additionally, pooling operations introduce a degree of translation invariance, enhancing the model's robustness to changes in the input position [78]. Motivated by prior works [14, 79], we employ an attention-based pooling module to generate the fixed-sized query embedding $\boldsymbol{q}$ for predicting the corresponding cardinality.

As demonstrated in previous work [96], query elements with varying frequencies can have opposite impacts on the cardinality of a query depending on the operator type. For example, consider two queries composed of the same elements. In a superset query, which aims to find sets containing all specified elements, low-frequency elements have a stronger influence on cardinality than high-frequency elements. Conversely, in an intersection query, the resultant sets include at least one of the specified elements, meaning that high-frequency elements have a greater impact on the cardinality. Thus, the frequency information is first appended to $\boldsymbol{q}'$, which generates the $(d+1)$-dim embedding $\boldsymbol{q}'_f$, and then the attention pooling module takes a random initialized embedding $\boldsymbol{q_0}$ and $\boldsymbol{q}'_f$ as inputs. After accessing $\boldsymbol{q}$, the output of the pooling layer, we use a simple linear regression layer LR to predicate the cardinality estimation c. It is noteworthy that we use the logarithm of the frequency as the appending information and modify the conventional residual connection inspired by the prior work [55, 87]. Figure 5 shows the framework of this module.

$$\tilde{\boldsymbol{q}} = LayerNorm\left(AvgPool(\boldsymbol{q}') + Att_c(\boldsymbol{q_0}, \boldsymbol{q}'_f, \boldsymbol{q}'_f)\right),$$
$$\boldsymbol{q} = LayerNorm\left(\tilde{\boldsymbol{q}} + FFN(\tilde{\boldsymbol{q}})\right),$$
$$c = LR(\boldsymbol{q})$$

5.3 Analyzer Training

Fine-tuning the parameters of the query analyzer requires a training dataset of which each record is a 3-tuple $(q_i, \boldsymbol{S_c}, c_i)$, where q_i is the

set-valued query consisting of k elements, and c_i denotes the true cardinality of q_i. In practice, collecting the training dataset, which is split into batches to train our analyzer, is not difficult, and we only need to collect the feedback of executed queries.

Since each module in the analyzer is differentiable, we train the analyzer in an end-to-end manner. Here, we use the weighted mean Q-error function $WMQ(\cdot)$ as the loss function, which takes input of the batch cardinality estimates $\boldsymbol{c'_b}$ and the true cardinalities $\boldsymbol{c_b}$ as well as their weights $\boldsymbol{w_b}$ with batch size B_q.

$$WMQ(\boldsymbol{c'_b}, \boldsymbol{c_b}) = \sum_{i=1}^{B_q} w_i * max\{1, \frac{c'_i}{c_i}, \frac{c_i}{c'_i}\},$$

where w_i is proportional to $\log c_i$, i.e. $w_i = \frac{\log c_i}{\sum_j \log c_j}$. We use the weight in the loss function because it is usually beneficial to emphasize the queries with larger true cardinalities [50].

6 ACE UNDER UPDATES

In this section, we first discuss how to leverage our ACE on dynamic data. Then, we analyze its benefits compared with the state-of-the-art baseline methods. Notably, we use the same setting when working with dynamic data, that is, the element universe is finite and fixed. The update of the element universe is left for future work.

ACE on dynamic data. We focus on dynamic data involving insertions and deletions because one update is equivalent to one deletion followed by one insertion. Based on the structure of ACE, we take a two-stage approach to accommodate dynamic data – (1) dataset representation update and (2) query cardinality estimation.

Given a batch of tuples to be inserted, we first use the aggregator to represent them. Then, we sample the learned tuple matrix and regard sampled embeddings as the initial distilled matrix. Next, we leverage the distiller to update the distilled matrix.

When deleting tuples, considering that our original dataset is split into a collection of dataset slices based on the batch size B_d, we locate the affected slices and only need to update their corresponding distilled matrix by leveraging the trained encoder, as motivated by the previous work [7]. Additionally, we need to update the frequency of elements that are affected by the update.

After obtaining the new distilled matrix, we can feed it along with the embeddings of the queried elements into the trained hybrid attention framework to derive the element embeddings that link the query with the updated dataset and capture the implicit correlation between elements. Subsequently, we incorporate the current frequency information of each element and utilize the attention pooling as well as the linear regression models to get the cardinality estimate for the new dataset.

Comparison and analysis. When working with dynamic data, PostgreSQL reconstructs the affected histograms to approximate the distribution of the updated dataset. However, it still relies on the (partially) independent assumption, which limits its accuracy. The update process of traditional sampling methods involves sampling tuples from the inserted dataset or deleting tuples from the existing samples when encountering insertions or deletions, respectively, leading to their performance heavily depending on the quality of the resultant samples. Additionally, they still pay more attention to elements with high frequency and achieve poor performance on the query with low-frequency elements.

One prior work [96] proposes the improved sampling method based on the pre-constructed trie structure. However, this study does not address how to handle dynamic data updates. Therefore, we propose a straightforward algorithm to support such updates. When deleting data, we adhere to the traditional method by checking if the data is part of the sampling results. If it is, we delete it and re-sample some sets to maintain the sample ratio. When inserting data, updating the trie structure is not feasible because it only retains the most frequent elements. Instead, we first partition data into several clusters based on the elements they contain. Then, we use the sample ratio calculated by the original trie to sample additional sets and update the sampling results. Nonetheless, this method may not well approximate the distribution of elements because of the fixed trie structure. Another recent work [59] proposes two conversion methods that transform the set-valued data into a small number of categorical data and introduces incremental updating methods for dynamic data. However, a significant issue with the proposed method persists. The cluster generation process is based on the dataset before any updates, aiming to alleviate the effect of the correlation between elements within the same cluster. As analyzed in Section 5, the correlation between elements is influenced by the corresponding dataset. Thus, the clusters need to be monitored and reconstructed when necessary because the initial clusters might not work well, which the proposed methods ignore.

Compared to these baselines, the performance of our ACE is superior as the data encoder minimizes the information loss when representing the dataset and the query analyzer effectively captures the useful correlation to obtain accurate estimates.

7 EXPERIMENTS

This section reports the experiments that compare ACE with SOTA baselines. All experiments are evaluated on the Katana server [69] with a 32-core Xeon(R) Gold 6242 CPU @ 2.80GHz, 100GB memory, and an NVIDIA Tesla V100-SXM2 32GB GPU.

7.1 Datasets and Workloads

Datasets. We use three real-world datasets varying in the number of sets N, the size of the element universe M, and the average number of elements within a set $AvgL$ as described in Table 4. The **GN** dataset [59] contains descriptions of natural features, canals, and reservoirs in the United States, each of which might consist of its name, class, and location state. The **WIKI** dataset [82] consists of the first sentence of each English Wikipedia article extracted in September 2017. The **TW** dataset [9] includes tweets posted from April 2012 to December 2012, which are published in UCR STAR [18]. We preprocess the latter two datasets and convert each set to a set of words that do not include stop words.

Table 4: Dataset statistics

Property	GN	WIKI	TW
N	2.2M	5.3M	19.9M
M	89K	858K	559K
$AvgL$	3	12	5

Workloads. We follow the method from the former work [59] to generate our workloads. For each subset query, we uniformly draw 5–10 sets from the set-valued dataset and take the union of the sampled sets as the query. For each superset and overlap query,

we uniformly draw one set from the dataset, and then uniformly draw 2–4 elements from the set as the query. We also consider the frequency of query elements. Following the former work [76], we separate elements into three classes based on their frequency: low ($\leq 0.01\%$), medium, and high ($\geq 0.1\%$). By default, all elements are considered in a regular query. For high-frequency queries, we add a filter to select only high-frequency elements. The generation of low-frequency queries follows a similar approach. The true cardinality of each query is obtained by executing it in PostgreSQL. For each dataset, we generate 1400 queries as the training workload, where the ratio of regular, high-frequency, and low-frequency queries is 3:2:2, while each testing workload consists of 300 queries.

7.2 Experimental Settings

Implementations. Our ACE is implemented with PyTorch [65] and we set the embedding dimension $d = 64$. We train ACE using Adam optimizer [38], with a learning rate of 0.001. We set the data batch size $B_d = 10000$, the distillation ratio $r = 0.001$, and the query batch size $B_q = 100$. The number of layers in the distillation model $n_{distill}$, the cross-attention module n_{cross}, and the self-attention module n_{self} are set to 4, 4, and 8, respectively. When utilizing the multi-head attention mechanism, we follow the existing work [50] by setting the number of heads to 8. For all datasets, we employ negative sampling with 10 samples for each set and perform a grid search for the L2 regularization weight $\lambda \in [0, 0.005]$.

Competitors. We include the following representative methods. (1) **PG** is the 1-D histogram-based cardinality estimator used in PostgreSQL [40]. (2) **Sampling** uniformly samples a collection of sets, where we set the sample ratio as 0.01. (3) **Greek-S** [60] proposes a different method to calculate the caridnality based on the statictis of the sampled dataset. (4) **OT-S** [96] samples a collection of sets based on the constructed trie structure. For a pair comparison, the sampling ratio is the same as the previous approach and we keep the 12 most frequent elements following the setting in the previous work. (5) **Set-Trans** [46] is proposed to capture element correlation and trained in a supervised learning manner. We regard it as a query-driven estimator. (6) **ST** and **STH** [59] convert the set-valued set into a small number of numerical data and employ the existing estimators. Based on the observation of the former work, we use DeepDB [30] and NeuroCard [97] as the employed estimators for **ST** and **STH**, respectively.

Evaluation metrics. We use four metrics to evaluate all methods. (1) **Q-error** [61] measures the distance between the estimated cardinality c_p and the true cardinality c of a query. In particular, Q-error = $max\{1, \frac{c_p}{c}, \frac{c}{c_p}\}$. (2) **Building time** denotes the construction time of traditional methods or the training time of Set-Trans and ACE. For ST and STH, the building time consists of conversion time as well as offline training time. (3) **Storage overhead** is the memory size used by a method. (4) **Estimation latency** is the average estimation time per query.

7.3 Overall Performance

We first conduct extensive experiments to evaluate the overall performance. We do not compare ST and STH on the overlap query since they are incompatible with this query type.

Estimation Accuracy. Tables 5 - 7 show the estimation error for various queries. We observe ACE has the best performance compared to other baselines in most cases. The mean Q-error of ACE in all cases is smaller than 10, and none of the other methods can reach this level of performance. Moreover, at the 95% quantile, PG, Sampling, OT-S, ST and STH averagely result in up to 16.7×, 29.6×, 27.7×, 13.5×, 10.2× larger Q-error than that of ACE, respectively.

Table 5: Estimation error for subset queries

Dataset	Method	Regular				High-frequency				Low-frequency			
		Mean	50%	95%	99%	Mean	50%	95%	99%	Mean	50%	95%	99%
GN	PG	8.53	6.54	16.9	33.2	4.12	3.87	6.03	10.6	**2.75**	**2.25**	**6**	**9**
	Sampling	1.36	1.21	2.54	3.91	1.11	1.09	1.31	1.45	62.3	13	166	203
	Greek-S	1.39	1.14	1.75	8.48	1.32	1.16	1.87	5.48	18.2	14.1	44.4	53.3
	OT-S	**1.08**	**1.06**	**1.21**	**1.32**	**1.09**	**1.07**	**1.23**	**1.32**	63.7	13	170	203
	Set-Trans	1.51	1.33	2.46	3.56	1.77	1.42	4.14	6.57	134	111	325	513
	ST	3.81	2.73	9.53	18.9	6.06	4.77	14.2	25.1	21.2	18	48.1	66.1
	STH	2.75	2.46	5.69	8.67	1.12	1.09	1.35	1.46	17.1	20.5	32.1	54.5
	ACE	1.26	1.13	2.34	4.17	1.69	1.44	3.26	5.22	3.11	2.81	8.56	12.4
WIKI	PG	9.49	5.68	19.7	41.3	4.74	3.51	11.9	14.6	4.14	3.83	7.67	9.51
	Sampling	28.1	1.37	188	299	24.5	1.41	151	207	28.7	22.6	54.5	89
	Greek-S	2.24	1.85	5.13	8.38	2.73	1.82	7.24	10.7	25.6	18.1	45.5	53.4
	OT-S	25.3	1.45	109	215	21.7	1.61	131	189	30.7	23.3	52.1	84
	Set-Trans	2.84	1.62	5.97	11.2	2.83	2.14	6.19	12.4	6.97	5.25	17.1	37.8
	ST	7.39	1.78	7.05	14.7	5.39	2.33	28.9	35.3	13.8	7.11	54	135
	STH	7.32	5.32	19.7	30.8	10.1	8.23	22.8	33.9	12.4	10.2	48.6	89.2
	ACE	**2.04**	**1.36**	**4.93**	**8.71**	**2.37**	**1.77**	**5.35**	**7.27**	**2.43**	**1.75**	**5.96**	**13.5**
TW	PG	5.85	4.39	13.8	23.2	4.01	3.19	7.74	12.2	3.69	2.88	9.13	12.7
	Sampling	5.04	4.03	12.1	36.2	3.57	3.24	13.5	29.1	19.1	15.4	52	76
	Greek-S	1.81	1.42	2.49	3.92	1.93	1.81	3.54	11.9	14.4	11.5	38.1	53.5
	OT-S	4.99	3.83	9.67	28.4	4.82	3.55	11.6	31.2	21.9	16.2	60	97
	Set-Trans	2.05	1.89	3.72	4.47	2.74	2.54	5.81	9.58	375	260	922	1464
	ST	4.38	3.74	8.82	12.4	4.74	3.31	7.32	15.4	30.9	23.1	74	127
	STH	5.07	3.24	12.3	15.2	3.11	2.76	5.94	11.1	19.5	4.57	72.7	130
	ACE	**1.46**	**1.34**	**2.17**	**2.81**	**1.66**	**1.53**	**2.94**	**4.24**	**1.88**	**1.61**	**3.81**	**4.92**

Table 6: Estimation error for superset queries

Dataset	Method	Regular				High-frequency				Low-frequency			
		Mean	50%	95%	99%	Mean	50%	95%	99%	Mean	50%	95%	99%
GN	PG	67.5	4.26	198	1785	96.6	3.5	192	2728	6.73	6	12	17
	Sampling	16.7	5	70	187	21.1	2.45	94	229	37.1	35.2	76.7	90.2
	Greek-S	12.3	5.93	44.4	53.3	8.05	3.46	33.3	53.3	25.8	17.4	50.6	53.3
	OT-S	20.2	6	81	195	17.9	2.21	78	192	36.7	34.5	77.1	91.3
	Set-Trans	12.5	5.77	46.9	117	12.4	4.32	46.5	133	7.53	5.22	18.6	33.5
	ST	40.2	2.21	52.7	272	18.8	2.12	22.2	100	9.71	7	28.5	37.4
	STH	16.3	**1.52**	34.6	161	10.3	**1.45**	22.2	150	5.08	5	11	15
	ACE	**5.19**	2.91	**17.2**	**36.1**	**6.54**	2.19	**20.9**	**49.6**	**2.15**	**2.11**	**3.18**	**3.49**
WIKI	PG	175	8	271	2330	439	8.61	703	5479	9.44	3.75	33	89
	Sampling	13.3	**2.31**	65	145	15.1	**1.39**	71	179	32.9	28.1	83	189
	Greek-S	10.9	3.98	38.1	53.4	8.91	2.35	33.4	93.5	15.4	12.4	30.4	53.5
	OT-S	14.8	2.57	72	159	11.6	1.51	61	159	33.7	29.3	77	186
	Set-Trans	19.2	7.86	80.2	158	15.1	5.43	60.2	108	22.6	10.7	81.8	96.1
	ST	130	4.92	245	1570	130	2.77	101	1576	30.6	13.3	118	241
	STH	16.1	3.98	69.6	169	9.31	2.58	35.4	143	29.3	11	118	239
	ACE	**6.44**	3.34	**17.9**	**37.4**	**8.33**	3.16	**30.8**	**75.2**	**2.88**	**2.67**	**8.23**	**11.6**
TW	PG	179	2.39	99	1014	128	2.09	48	2323	14.3	4.75	53.5	152
	Sampling	17.5	2.31	83	223	10.7	**1.32**	60	173	173	149	251	380
	Greek-S	9.83	3.21	44.5	74.3	9.22	2.43	26.7	80.3	22.7	19.1	47.6	53.3
	OT-S	15.6	**1.96**	87	210	8.91	1.36	44	135	161	137	251	380
	Set-Trans	20.2	6.67	82.7	128	13.3	4.67	65.6	118	13.9	10.5	37.2	60.7
	ST	49.9	2.37	81.1	578	43.7	2.05	76.6	553	37.9	10	160	347
	STH	12.2	2.08	48.6	306	11.7	2.83	44.1	471	37.7	10.1	160	374
	ACE	**6.79**	2.32	**22.5**	**60.7**	**8.41**	2.04	**26.8**	**67.1**	**3.93**	**2.61**	**8.76**	**11.7**

Table 7: Estimation error for overlap queries

Dataset	Method	Regular				High-frequency				Low-frequency			
		Mean	50%	95%	99%	Mean	50%	95%	99%	Mean	50%	95%	99%
GN	PG	**3.28**	**1.26**	9.82	34.1	2.99	**1.31**	9.47	27.4	12.7	4.15	68.1	108
	Sampling	4.18	1.28	9.95	42.1	2.93	1.32	9.19	29.2	239	2.89	481	5954
	Greek-S	4.99	1.33	11.1	75.2	2.99	1.35	8.66	28.1	13.6	2.71	33.3	277
	OT-S	3.41	1.28	9.96	30.6	3.01	1.33	8.81	29.8	235	3.08	502	6409
	Set-Trans	10.7	4.73	46.2	110	10.9	5.51	22.1	94.7	51.2	30.3	221	346
	ACE	3.34	1.46	**9.29**	**18.1**	**2.66**	1.56	**6.39**	**17.6**	**9.62**	**2.67**	**22.7**	**80.6**
WIKI	PG	8.28	1.81	13.1	94.2	4.28	2.32	15.1	40.7	27.1	14.9	165	368
	Sampling	4.29	1.88	13.2	43	4.28	2.34	14.2	37.4	43.5	2.49	214	496
	Greek-S	4.52	1.93	16.6	38.8	4.31	2.32	13.5	38.5	8.44	3.29	22.9	54.8
	OT-S	5.52	1.78	13.1	59.3	4.16	2.26	12.9	37.1	35.5	2.54	196	414
	Set-Trans	23.9	11.9	98.7	131	37.7	19.5	108	260	26.6	14.3	104	239
	ACE	**3.98**	**1.75**	**8.99**	**15.8**	**2.72**	**2.14**	**7.52**	**10.7**	**8.01**	**2.46**	**22.6**	**38.5**
TW	PG	6.33	2.26	19.5	85.9	4.64	2.38	11.3	37.4	18.3	14.3	134	377
	Sampling	6.21	2.26	19.6	82.1	4.67	2.42	10.8	37.6	28.1	2.83	175	473
	Greek-S	6.37	2.33	22.4	76.4	4.61	2.41	11.1	36.2	11.6	3.59	20.6	320
	OT-S	6.25	2.21	19.3	84.1	4.66	2.35	10.6	37.6	28.6	2.81	167	287
	Set-Trans	51.1	16.5	162	446	69.2	28.9	191	580	16.2	8.68	57.7	166
	ACE	**5.61**	**2.19**	**16.4**	**51.5**	**2.96**	**2.21**	**7.92**	**13.7**	**6.72**	**2.79**	**17.9**	**73.3**

PG estimates the cardinality of a query based on the independence assumption, which often leads to poor performance on regular and high-frequency queries due to ignoring the correlation

between elements. However, its estimates are more accurate for low-frequency queries, as the correlation between low-frequency elements can sometimes be disregarded. Moreover, we observe that its performance on low-frequency overlap queries is bad because of insufficient statistics targets.

Sampling, Greek-S and OT-S show the opposite trend compared to PG. Their performance on regular and high-frequency queries is better than that on low-frequency queries because they focus more on high-frequency elements. Greek-S firstly determines the geometric mean of the upper (ω) and lower (α) bounds for the number of qualifying tuples based on probabilistic estimates, which is then used to return a more accurate cardinality estimate. OT-S improves upon the traditional sampling method by leveraging the trie structure, leading to better performance in most cases. However, the performance of these methods is not stable across three datasets, as it heavily depends on the quality of sampling results.

Set-Trans only utilizes the information of the workload, regarding the problem as a supervised learning task. However, its performance is unstable across datasets and queries since it is impossible to enumerate all combinations given limited training data. ST and STH are SOTA methods that can utilize any data-driven estimator to predict cardinality based on the constructed clusters and the corresponding conversion algorithm. In general, our ACE outperforms these methods, verifying that the partial independence assumption in these methods is not reasonable for some scenarios. Additionally, we observe that the results on low-frequency queries differ significantly from those reported in the former work [59], as it first filters out low cardinality elements before selecting the query element, thereby ignoring the actual low-frequency elements.

Construction Efficiency. Referring to Table 8, the training time of ACE is acceptable and shorter than STH and ST in most cases. It requires less than 2, 7, and 10 minutes to fine-tune its parameters for the GN, WIKI, and TW datasets, respectively. Although Sampling and OT-S require less construction time, their Q-error performance is worse, especially on low-frequency queries. Notably, Greek-S is excluded from our analysis, as it does not influence the sampling process. Besides, we observe that the time of Set-Trans and ACE on different types varies because of the variable-size query, where a subset query usually has more elements than other types of queries.

Table 8: Building time (minutes) of different methods

Dataset	Type	Sampling	OT-S	Set-Trans	ST	STH	ACE
GN	Subset	0.03	0.11	2.76	0.43	4.02	1.88
	Superset	0.03	0.11	2.06	0.43	3.94	1.56
	Overlap	0.03	0.11	2.03	-	-	1.58
WIKI	Subset	0.06	0.61	6.03	10.9	13.2	6.77
	Superset	0.06	0.61	1.97	10.9	12.9	2.85
	Overlap	0.06	0.61	1.68	-	-	2.67
TW	Subset	0.21	1.31	6.16	9.73	11.7	9.35
	Superset	0.21	1.31	5.07	9.73	11.4	5.13
	Overlap	0.21	1.31	5.14	-	-	5.16

Storage Overhead. Table 9 shows the experimental results. We denote the size of samples as the storage overhead of sampling-based methods, which increases with the size of datasets. As Greek-S does not change the sampling process, its results are excluded from the table. ST and STH need to maintain the parameters of DeepDB and NeuroCard, respectively, with STH typically incurring higher space costs due to its more complex structure. In contrast, Set-Trans maintains a consistent model size across datasets, as its storage requirements are determined by the embedding dimensions. However, its overall size is slightly larger than ours due to the additional inclusion of inducing points. Our ACE demonstrates a stable storage, exhibiting only a marginal increase as the dataset size expands, primarily due to the benefits of its distillation process.

Table 9: Storage overhead (MB) of different methods

Dataset	Sampling	OT-S	Set-Trans	ST	STH	ACE
GN	0.28	0.29	10.6	3.31	16.1	8.11
WIKI	3.21	3.33	10.6	29.6	79.7	8.26
TW	4.77	4.79	10.6	11.2	58.1	8.36

Estimation Latency. As illustrated in Table 10, PG needs the least time to estimate the cardinality but its performance is not acceptable, while the latency of the sampling-based methods increases with the size of the dataset because they need to traverse all samples. Greek-S, in particular, exhibits significantly higher latency than the other methods, as it involves more computational steps, resulting in increased time costs. As shown in previous work [59], the time complexity of ST depends on the number of clusters, which leads to lower estimation time, while STH reduces the number of nodes kept on each trie to speed up the prediction process. However, both methods need to convert the query before estimating the cardinality, which cannot be executed in GPUs. Set-Trans uses the least time to predicate the cardinality because it only takes the query as the input and regards the problem as a supervised learning task. ACE is a fully learning-based estimator with the best performance and the most stable latency across three datasets.

Table 10: Estimation latency (ms) of different methods

Dataset	PG	Sampling	Greek-S	OT-S	Set-Trans	ST	STH	ACE
GN	1.05	124.9	207.2	128.6	2.91	3.86	12.39	4.54
WIKI	3.64	381.1	876.1	392.4	3.28	25.67	83.12	5.17
TW	2.79	1163	7808	1175	3.07	19.57	41.07	6.17

Real-world Cases. In Section 1, we introduce a real-world application for set-valued data, i.e., tag search. Privacy policies render user data confidential in most applications, such as Twitter and Wiki. We utilize the data released by a recipe website (http://www.food.com) and its real user search queries [52, 58] to show the necessity of supporting set queries. In this dataset, the total number of keywords is 632 and there are 500K recipes, each tagged with several keywords. For the query workload, we extract 10K distinctive queries for superset and overlap queries supported by this website. For each query type, we randomly select 1000 and 200 queries for the training and validation, respectively, while the remaining are used for testing. As ST and STH cannot support the overlap queries, the two methods have no corresponding results. As shown in Table 11, ACE consistently outperforms other baseline methods using the real-world set-valued data and query workload.

Table 11: Real-world tag search

Method	Superset				Overlap			
	Mean	50%	95%	99%	Mean	50%	95%	99%
PG	15.1	4.25	51	186	1.17	1.13	1.74	2.16
Sampling	14.4	4.62	59	122	1.26	1.31	1.88	3.71
Greek-S	4.42	2.31	17.9	17.9	1.21	1.13	1.57	2.11
OT-S	3.68	2.27	12.5	12.8	1.21	1.16	1.44	1.98
Set-Trans	3.41	2.19	8.47	20.9	1.61	1.45	2.24	4.79
ST	4.05	2.29	11.8	23.4	-	-	-	-
STH	10.2	7.07	29.1	50.6	-	-	-	-
ACE	**3.18**	**2.01**	**7.53**	**15.2**	**1.02**	**1.01**	**1.05**	**1.11**

7.4 Performance on Dynamic Data

We follow previous studies [50, 59] to conduct experiments on dynamic data. We use about 70% of the sets as the initial dataset to train our data encoder and the remaining as the collection of insertion data. The size of each insertion is equal to the data batch size B_d. 90% of the insertion is used to train the query analyzer while the remaining is used to evaluate the performance. Additionally, we might randomly delete some sets from the current dataset before any insertions. To simulate the real-world scenarios, the number of deleted sets is only a small fraction of the entire dataset, meaning that the number of affected slices is much lower than the others.

Regarding the workload, we only conduct the experiments on the superset and subset query since ST and STH are incompatible with the overlap query. To train the query analyzer, we utilize the generated workload as the base workload. Then, we randomly select 20 and 10 queries from the base workload as the training and validation sets, respectively, once an insertion completes. When evaluating the performance, we generate 100 queries after any insertion as the evaluation queries of the current dataset and finally report the average value. Note that the true cardinality of a query might change due to the dynamic data. Thus, we need to use PostgreSQL to obtain the true cardinality values and filter the queries without any results in the training or evaluation process.

Table 12: The performance on dynamic data

Dataset	Method	Superset					Subset				
		Q-error				Update	Q-error				Update
		Mean	50%	95%	99%	Time (s)	Mean	50%	95%	99%	Time (s)
GN	PG	69.6	5.01	204	2031	-	8.39	7.23	15.1	35.7	-
	Sampling	18.7	7	84	173	0.01	**1.37**	**1.16**	3.18	5.28	0.01
	Greek-S	15.4	9.21	53.3	64.7	0.01	1.61	1.52	2.89	6.06	0.01
	OT-S	24.2	7.2	97.2	258	0.06	1.62	1.59	2.82	6.98	0.06
	Set-Trans	28.8	11.3	78.2	211	0.26	5.77	4.83	7.18	9.49	0.34
	ST	45.7	4.35	78.1	395	0.16	4.51	3.17	12.7	20.2	0.16
	STH	20.5	4.02	49.9	190	0.17	3.88	2.56	8.76	11.5	0.17
	ACE	**5.35**	**3.09**	**15.3**	**42.7**	0.34	1.59	1.51	**2.77**	**5.01**	0.41
WIKI	PG	136	6.68	341	3249	-	10.1	4.93	18.2	39.5	-
	Sampling	14.8	4.01	71	140	0.01	29.8	1.95	197	371	0.01
	Greek-S	11.5	4.17	44.5	53.4	0.01	4.31	1.75	9.42	13.6	0.01
	OT-S	18.5	3.22	90	199	0.06	37.9	2.18	163	323	0.06
	Set-Trans	28.1	13.7	114	243	0.44	4.79	3.11	8.75	20.4	1.29
	ST	145	6.77	358	1994	0.41	9.63	3.55	17.4	31.2	0.41
	STH	22.4	5.68	101	274	0.81	8.76	5.88	25.4	32.1	0.81
	ACE	**8.57**	**3.17**	**16.7**	**29.1**	0.68	**3.55**	**1.69**	**7.54**	**9.27**	1.56
TW	PG	187	3.43	101	1341	-	6.85	4.17	12.6	25.7	-
	Sampling	17.3	**2.35**	81	224	0.01	6.03	5.32	13.1	37.7	0.01
	Greek-S	12.4	4.47	48.4	97.1	0.01	3.22	2.83	6.06	10.5	0.01
	OT-S	19.5	3.45	109	263	0.06	5.88	4.02	13.5	31.2	0.06
	Set-Trans	39.5	9.77	108	190	0.81	4.68	3.06	7.66	10.2	1.54
	ST	55.7	3.37	92	768	0.25	7.28	4.96	14.8	22.3	0.25
	STH	16.7	2.94	64	320	0.47	8.21	4.04	13.9	30.1	0.47
	ACE	**10.1**	3.22	**37.7**	**84.1**	0.97	**2.14**	**1.67**	**5.28**	**8.22**	1.72

Referring to Table 12, ACE always has the best performance. Compared to the static data, PG, Sampling, and Greek-S achieve similar estimation accuracy while the Q-error of other methods increases when working on dynamic data. Compared to Sampling, the update progress of OT-S depends on the trie structure built based on the initial dataset, and this structure does not capture the distribution of elements well when encountering new data. ST and STH incrementally update their corresponding trie structure on dynamic data but fix the generated clusters. However, the latest set always brings a change in the element correlation. Therefore, the elements within the same cluster might be heavily correlated when updating the dataset, leading to performance degradation. In terms of our ACE, we also observe performance degradation because the data encoder is trained based on the initial dataset and might not output the best representation of the updated data. However, its performance is more stable than others since we utilize the query information in the query analyzer, which can mitigate this effect.

We also report the average time of each update. Since our ACE is both data- and query-driven, it requires training the query analyzer for better performance when the data matrix updates. Compared to ACE, all baseline methods do not require any training progress, and thus they need less time to update. However, their representation abilities are not as powerful as that of ACE.

7.5 End-to-End Query Runtime

We evaluate the performance of ACE in terms of end-to-end query runtime in PostgreSQL. We utilize the latest IMDb dataset [32] that includes set-valued attributes and extract another database, Food, from the published datasets crawled from http://www.food.com [2, 51, 88]. For the query workload, because current benchmarks, such as JOB [47], do not contain queries with set-valued predicates, we follow the existing work [59] to generate queries. Specifically, we firstly use SQLsmith [72] and the AI SQL generator [89] to generate a query template. Then, we follow the template to generate queries with various granularities. To guarantee the validity of synthetic queries, we follow the same process to generate the set-valued predicates. Note that the set-valued predicate on Food utilize the real user queries, as described in Section 7.3. For the IMDB dataset, we generate 30 queries, each containing 4–8 predicates over 3–5 tables, while for the Food dataset, we generate 20 queries, each containing 3–6 predicates over 3 tables.

To inject cardinalities, our ACE(P) configuration extends the patch from the previous work [3] to accept external estimates for set-valued predicates and the existing support for predicates over categorical and numerical attributes. In addition to the baseline given by PostgreSQL (PG), we compare with four other baselines. Because ST and STH naturally support queries containing predicates over set-valued attributes, ST and STH configurations inject the estimated cardinalities of ST and STH into the query optimizer for all types of predicates. Note that we use PostgreSQL as the estimator for ST and STH to guarantee a fair comparison. Additionally, configurations GS(P) and Set(P) utilize the same approach as ACE(P) but with estimates from Greek-S and Set-Trans, respectively.

Table 13 shows the end-to-end (E2E) running time and Q-error. Our method, ACE, achieves the best overall performance. Notably, queries with set-valued predicates show a notable improvement due to more accurate estimations. We observe that the E2E time of GS(P) is even longer than that of the PG because the estimation latency of Greek-S is significantly larger.

Table 13: End-to-end (E2E) time and Q-error

Method	IMDB					Food				
	Mean	50%	95%	99%	E2E Time (s)	Mean	50%	95%	99%	E2E Time (ms)
PG	30.3	5.11	113	440	106.9	10.3	3.43	20.5	79	7327.8
GS(P)	17.4	4.66	62.1	139	81.1	9.41	2.78	19.2	75.8	8007.7
Set(P)	19.4	4.86	80.2	158	82.3	8.17	2.44	16.1	59.1	6285.4
ST	12.1	3.39	44.5	93.1	68.2	6.53	2.05	11.5	30.4	4667.4
STH	11.8	3.47	45.1	87.2	66.1	6.77	2.01	13.3	30.7	4528.6
ACE(P)	10.1	2.77	33.7	80.1	40.5	5.53	1.78	9.01	25.2	3004.9

7.6 Ablation Study

As shown in Table 14, we verify the effectiveness of the main components in ACE. Here, we conduct extensive experiments on the WIKI dataset. The results on other datasets are similar and omitted.

Table 14: Ablation study

Ablation settings					Subset				Superset				Overlap			
AG	DS	CA	SA	AP	Mean	50%	95%	99%	Mean	50%	95%	99%	Mean	50%	95%	99%
×	✓	✓	✓	✓	4.47	2.69	17.2	31.8	12.6	5.34	44.4	147	6.69	2.94	13.4	20.1
✓	×	✓	✓	✓	4.11	2.72	15.6	32.1	13.7	8.41	49.9	162	7.73	3.23	10.5	18.7
✓	✓	×	✓	✓	2.87	1.58	6.26	13.7	8.19	4.14	19.8	45.8	4.31	2.45	9.74	18.3
✓	✓	✓	×	✓	3.31	1.65	5.62	11.4	8.97	3.79	25.8	55.7	5.93	3.11	11.6	22.3
✓	✓	✓	✓	×	2.24	1.63	5.12	10.9	6.94	3.39	19.5	52.3	4.62	2.35	9.68	16.9
✓	✓	✓	✓	✓	2.04	1.36	4.93	8.71	6.44	3.34	17.9	37.4	3.98	1.75	8.99	15.8

Aggregator (AG). To replace our aggregator, we can use traditional methods, such as padding and pooling, to generate the fixed-size set embedding. Since padding often leads to higher storage overhead, we leverage the mean-pooling method, which has a similar cost to our original design. When comparing results, we observe at least a 60% increase in estimation error at the 50% quantile. This is because the mean-pooling method typically treats all elements equally, which is not powerful enough to obtain high-quality embeddings.

Distillation (DS). To replace the distillation module, we propose a random sampling method, setting the sample ratio to 0.001 for a fair comparison. When comparing the results, we observe a significant increase in estimation error ranging from 94.1% to 112%. This is because the sampling method captures only a fraction of the dataset's information, whereas the distillation model is designed to compress the matrix while preserving as much information as possible. Additionally, since the sampling method give more attention to high-frequency elements while the low-frequency elements predominantly influences the accuracy of superset queries, we observe the most pronounced fluctuations in these queries.

Cross-attention (CA). The stacked cross-attention layers serve to link data and queries, mapping the query elements into a latent space to capture their correlation effectively. As the dimensions of the data matrix S_c and query element embeddings q_i are fixed, we can adopt a straightforward method without the attention mechanism to processing them. For example, we flatten S_c into a vector and concatenate the vector with q_i to generate another vector. Subsequently, the generated vector is fed into a multi-layer perceptron (MLP) with the same number of layers as in our original model. However, experiments reveal that this simplified approach yields worse estimation performance compared to ACE, with a decrease exceeding 16.7%. This performance drop occurs because a straightforward neural network is not powerful enough to discover the implicit relations between elements and data. This finding underscores the necessity and effectiveness of incorporating cross-attention layers.

Self-attention (SA). The stacked self-attention layers are designed to capture correlations between query elements effectively. To evaluate their contribution, we replace this module with a multi-layer perceptron (MLP). When compared to ACE, the modified framework results in 1.49×, 1.31×, and 1.41× larger Q-error than that of ACE at the 99% quantile for superset, subset, and overlap queries, respectively. These results validate the superiority of self-attention layers in accurately modeling inter-element dependencies, which ultimately leads to more precise cardinality estimation.

Attention Pooling (AP). The attention pooling module is designed to address the issue of variable-size input while ensuring permutation invaiance. Since any symmetric function can be used to solve this problem, we compare the performance of the attention pooling method with a mean-pooling method. Our analysis verifies the effectiveness of the attention pooling method. Compared to ACE, the mean-pooling method results in a slight increase in estimation error, with 1.39×, 1.25×, and 1.07× larger mean Q-error for superset, subset, and overlap queries, respectively. This performance degradation occurs because mean-pooling weights each embedding equally regardless of its importance [100].

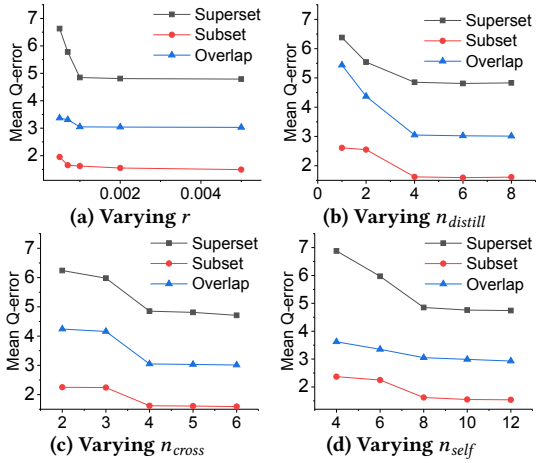


Figure 6: Estimation performance.

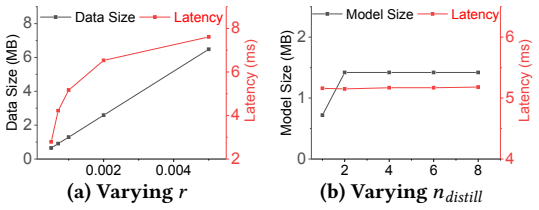


Figure 7: Size and latency.

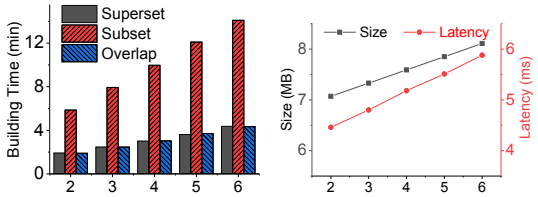
7.7 Hyper-parameter Study

To study the effects of important hyper-parameters, we build different ACE versions and observe their performance. Similarly, we only show the comparison results on the WIKI dataset.

Effects of r and $n_{distill}$. We first study hyper-parameters in our data encoder. Figure 6a and 7a show the performance varying distillation ratios r. We observe that the size of the distilled matrix is clearly influenced by r. When the value of r gets larger, ACE produces more accurate estimates, but with higher estimation latency. Besides, the performance improvement becomes marginal when r exceeds 0.001.

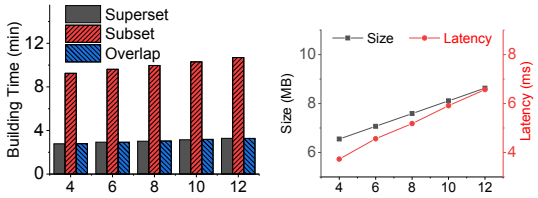
Another important hyper-parameter is the number of layers in our distillation model, denoted as $n_{distill}$. Figure 6b illustrates the estimation performance with varying $n_{distill}$. We observe that the Q-error decreases when $n_{distill}$ is less than 4, after which it stabilizes. As shown in Figure 7b, the model size remains constant for $n_{distill} \geq 2$ as we share weights between each layer except the first one. Moreover, the estimation latency is similar across these values because the distilled matrices have the same size.

Effects of n_{cross} and n_{self}. We also study the effects of hyper-parameters in our query analyzer. Figures 6c and 6d shows the mean Q-error with varying the values of these two hyper-parameters. We observe that increasing n_{cross} and n_{self} both lead to better estimates. This improvement is due to the enhanced ability of more stacked cross-attention layers to discover the relationship between queries and the underlying data, while additional self-attention layers help better capture the correlation between query elements.



(a) Building time (b) Size and latency

Figure 8: Other metrics with varying n_{cross}.



(a) Building time (b) Size and latency

Figure 9: Other metrics with varying n_{self}.

The values of n_{cross} and n_{self} also affect the building time, the model size, and the estimation latency. Figures 8 and 9 illustrate the experiment results of these metrics. We have two observations. First, larger n_{cross} or n_{self} always leads to a more complex structure, resulting in longer building time, larger model size, and higher estimation latency. Second, the effect of n_{cross} is more significant than that of n_{self} on these metrics. For example, the building time for subset queries increases from about 6 minutes ($n_{cross} = 2$) to 14 minutes ($n_{cross} = 6$), compared to an increase from 9 minutes ($n_{self} = 4$) to 11 minutes ($n_{self} = 12$). This is because the distilled matrix with a larger size is used as one input of the cross-attention sub-module. Therefore, taking into account all aspects, we set the values of n_{cross} and n_{self} to 4 and 8, respectively.

8 RELATED WORK

Cardinality estimators for numerical and categorical data. Data-driven methods aim to tightly approximate the data distribution by using statistical or machine learning models. Sampling-based methods [21, 53] estimate cardinality from the sampled data. The simple yet efficient 1-D Histogram [71] is used in DBMSs such as PostgreSQL. It maintains a histogram for each attribute. M-D histogram-based methods [12, 20, 62, 67, 84] build multi-dimensional histograms to capture the dependency among attributes. However, the decomposition of the joint attributes is still lossy such that they need to make partial independence assumptions.

Probability models [23, 80] utilize the Bayesian network (BN) to model the dependence among attributes, assuming that each attribute is conditionally independent given its parents' distributions. BayesCard [92] revitalizes BN using probabilistic programming to improve its inference and model construction speed. Deep autoregressive models [27, 97, 98] decompose the joint distribution to a product of conditional distributions, which have high accuracy but low efficiency and require large storage space. DeepDB [30] and FLAT [109] build upon a Sum-Product Network (SPN) [66] that approximates the joint distribution using multiple SPNs.

Query-driven methods focus on modeling the relationships between queries and their true cardinalities. LW-XGB and LW-NN [13] formulate the cardinality estimation as a regression problem and apply gradient-boosted trees and neural networks to solve the problem, respectively. The KDE-based join estimator [37] combines kernel density estimation (KDE) with a query-driven tuning mechanism. Fauce [54] and NNGP [107] assume that the workload follows a Gaussian distribution and adopt deep ensembles [44] and neural Gaussian process [36] to estimate the mean and variance of the distribution. A few works [50, 90] consider both data and workload. These approaches are limited to querying numerical and categorical data, which are difficult to deploy for set-valued data.

Cardinality estimator for set-valued data. PostgreSQL treats each element as a binary attribute and employs either independence assumptions or the probabilistic model [16] to estimate the cardinality of set-valued queries [40]. Yang et al. [96] improve the sampling method and propose two estimators: OT-sampling uses a trie structure to focus on highly frequent elements, which struggles with low-frequency elements; DC-sampling leverages the workload type information and employs a divide-and-conquer strategy, which is only applicable for the specified types. Hadjieleftheriou et al. [22] propose a hash sampling algorithm for set similarity queries, which differs from the problem studied in this paper. Meng et al. [59] propose two algorithms to convert set-valued data into multi-column categorical data and use data-driven methods to estimate the query cardinality. The conversion process can be regarded as approximately solving the NP-hard graph coloring problem, making it difficult to well capture the correlation among elements. All these existing methods only utilize the information of the underlying data. To the best of our knowledge, there is no learning-based estimator that leverages the underlying data and workload simultaneously.

Attention applications. The attention mechanism has been applied to various problems [83, 94, 95, 103, 108]. Recently, it has been adapted for database optimization [15]. The most closely related work [50] estimates the cardinality for SPJ (Select-Project-Join) queries. However, its featurization method is not suitable for our problem as the number of columns (less than 100) is significantly smaller than the number of sets (exceeding 10^6). Thus, our ACE needs new designs for the data encoder and the query analyzer.

9 CONCLUSION

We presented ACE, a versatile learned cardinality estimation model that makes high-quality estimates for set-valued queries. We first propose a distillation-based data encoder to represent the entire dataset using a compact matrix. To capture correlations between query elements, we then propose an attention-based query analyzer. Since query lengths can vary, we employ a pooling module to derive the fixed-size vector. Extensive experimental results demonstrate superior performance of ACE compared to the state of the art.

ACKNOWLEDGMENTS

This work is partially supported by Australian Research Council (ARC) DP230101445, DP230101534 and DP240101006. Jianzhong Qi is supported by ARC Future Fellowships FT240100170. Wenjie Zhang is supported by ARC FT210100303. Christian S. Jensen is supported in part by the Innovation Fund DK centre, DIREC. This work is also supported in part by Guangdong Talent Program under Grant 2021QN02X826, and Shenzhen Research Institute of Big Data under grant SIF20240002.

REFERENCES

[1] Jaan Altosaar, Rajesh Ranganath, and Wesley Tansey. 2021. RankFromSets: Scalable set recommendation with optimal recall. *Stat* 10, 1 (2021), e363.

[2] Alvin. 2021. Recipes and Reviews. https://www.kaggle.com/datasets/irkaal/foodcom-recipes-and-reviews

[3] Mehmet Aytimur, Silvan Reiner, Leonard Wörteler, Theodoros Chondrogiannis, and Michael Grossniklaus. 2024. LPLM: A Neural Language Model for Cardinality Estimation of LIKE-Queries. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 1–25.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *arXiv preprint arXiv:1607.06450* (2016).

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *ICLR*.

[6] Christopher M. Bishop. 2006. *Pattern recognition and machine learning*. Springer New York.

[7] Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. 2021. Machine unlearning. In *S&P*. 141–159.

[8] S. Castro, P. Meena Kumari, S. Muthumari, and J. Suganthi. 2023. Information Retrieval using Set-based Model Methods, Tools, and Applications in Medical Data Analysis. *Machine Learning for Healthcare Systems: Foundations and Applications* (2023), 187.

[9] Lisi Chen, Gao Cong, Xin Cao, and Kian-Lee Tan. 2015. Temporal spatial-keyword top-k publish/subscribe. In *ICDE*. 255–266.

[10] Yankai Chen, Yixiang Fang, Yifei Zhang, and Irwin King. 2023. Bipartite graph convolutional hashing for effective and efficient top-n search in hamming space. In *WWW*. 3164–3172.

[11] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. 2021. Rethinking attention with performers. In *ICLR*.

[12] Amol Deshpande, Minos Garofalakis, and Rajeev Rastogi. 2001. Independence is good: Dependency-based histogram synopses for high-dimensional data. *ACM SIGMOD Record* 30, 2 (2001), 199–210.

[13] Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek Narasayya, and Surajit Chaudhuri. 2019. Selectivity estimation for range predicates using lightweight models. *Proceedings of the VLDB Endowment* 12, 9 (2019), 1044–1057.

[14] Meng Joo Er, Yong Zhang, Ning Wang, and Mahardhika Pratama. 2016. Attention pooling-based convolutional neural network for sentence modelling. *Information Sciences* 373 (2016), 388–403.

[15] Jia-Ke Ge, Yan-Feng Chai, and Yun-Peng Chai. 2021. WATuning: a workload-aware tuning system with attention-based deep reinforcement learning. *Journal of Computer Science and Technology* 36, 4 (2021), 741–761.

[16] Lise Getoor, Benjamin Taskar, and Daphne Koller. 2001. Selectivity estimation using probabilistic models. In *SIGMOD*. 461–472.

[17] Hossein Gholamalinezhad and Hossein Khosravi. 2020. Pooling methods in deep neural networks, a review. *arXiv preprint arXiv:2009.07485* (2020).

[18] Saheli Ghosh, Tin Vu, Mehrad Amin Eskandari, and Ahmed Eldawy. 2019. UCR-STAR: The UCR Spatio-Temporal Active Repository. *SIGSPATIAL Special* 11, 2 (2019), 34–40.

[19] Arthur Gretton, Karsten Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex Smola. 2006. A kernel method for the two-sample-problem. In *NIPS*. 513–520.

[20] Dimitrios Gunopulos, George Kollios, Vassilis J. Tsotras, and Carlotta Domeniconi. 2000. Approximating multi-dimensional aggregate range queries over real attributes. *ACM SIGMOD Record* 29, 2 (2000), 463–474.

[21] Peter J. Haas, Jeffrey F. Naughton, and Arun N. Swami. 1994. On the relative cost of sampling for join selectivity estimation. In *PODS*. 14–24.

[22] Marios Hadjieleftheriou, Xiaohui Yu, Nick Koudas, and Divesh Srivastava. 2008. Hashed samples: selectivity estimators for set similarity selection queries. *Proceedings of the VLDB Endowment* 1, 1 (2008), 201–212.

[23] Max Halford, Philippe Saint-Pierre, and Franck Morvan. 2019. An approach based on bayesian networks for query selectivity estimation. In *DASFAA*. 3–19.

[24] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*. 1025–1035.

[25] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Tan Wei Liang, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, Zhengping Qian, Jingren Zhou, Jiangneng Li, and Bin Cui. 2022. Cardinality Estimation in DBMS: A Comprehensive Benchmark Evaluation. *Proceedings of the VLDB Endowment* 15, 4 (2022), 752–765.

[26] Yu Hao, Xin Cao, Yufan Sheng, Yixiang Fang, and Wei Wang. 2021. KS-GNN: Keywords Search Over Incomplete Graphs via Graphs Neural Network. In *NeurIPS*. 1700–1712.

[27] Shohedul Hasan, Saravanan Thirumuruganathan, Jees Augustine, Nick Koudas, and Gautam Das. 2020. Deep learning models for selectivity estimation of multi-attribute queries. In *SIGMOD*. 1035–1050.

[28] Trevor Hastie, Robert Tibshirani, Jerome H. Friedman, and Jerome H. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Vol. 2. Springer.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*. 770–778.

[30] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: Learn from Data, not from Queries! *Proceedings of the VLDB Endowment* 13, 7, 992–1005.

[31] IBM. 2024. Array Types and Values. https://www.ibm.com/docs/en/db2-for-zos/13?topic=types-array-values

[32] IMDb. 2024. Non-Commercial Datasets. https://developer.imdb.com/non-commercial-datasets/

[33] Yannis Ioannidis. 2003. The history of histograms (abridged). In *VLDB*. 19–30.

[34] Yannis E Ioannidis and Stavros Christodoulakis. 1991. On the propagation of errors in the size of join results. In *SIGMOD*. 268–277.

[35] Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. 2021. Perceiver: General perception with iterative attention. In *ICML*. 4651–4664.

[36] Lee Jaehoon, Bahri Yasaman, Novak Roman, Schoenholz Sam, Pennington Jeffrey, and Sohl-dickstein Jascha. 2018. Deep Neural Networks as Gaussian Processes. *ICLR* (2018).

[37] Martin Kiefer, Max Heimel, Sebastian Breß, and Volker Markl. 2017. Estimating join selectivities using bandwidth-optimized kernel density models. *Proceedings of the VLDB Endowment* 10, 13 (2017), 2085–2096.

[38] Diederik P Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

[39] Martin Kleppmann. 2017. *Designing Data-intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media, Inc.

[40] Alexander Korotkov and Konstantin Kudryavtsev. 2016. Selectivity Estimation for Search Predicates over Set Valued Attributes. *International Journal of Database Theory and Application* 9, 10 (2016), 285–294.

[41] Adrian Kosowski and Krzysztof Manuszewski. 2004. Classical coloring of graphs. *Contemp. Math.* 352 (2004), 1–20.

[42] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. In *SIGMOD*. 489–504.

[43] Kenneth A Lachlan, Patric R Spence, Xialing Lin, Kristy Najarian, and Maria Del Greco. 2016. Social media and crisis management: CERC, search strategies, and Twitter content. *Computers in Human Behavior* 54 (2016), 647–652.

[44] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and scalable predictive uncertainty estimation using deep ensembles. In *NIPS*. 6405–6416.

[45] Geon Lee, Chanyoung Park, and Kijung Shin. 2022. Set2Box: Similarity Preserving Representation Learning for Sets. In *ICDM*. 1023–1028.

[46] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. 2019. Set transformer: A framework for attention-based permutation-invariant neural networks. In *ICML*. 3744–3753.

[47] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really? *Proceedings of the VLDB Endowment* 9, 3 (2015), 204–215.

[48] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. 2016. Wander join: Online aggregation for joins. In *SIGMOD*. 2121–2124.

[49] Guang Li, Ren Togo, Takahiro Ogawa, and Miki Haseyama. 2020. Soft-label anonymous gastric x-ray image distillation. In *ICIP*. 305–309.

[50] Pengfei Li, Wenqing Wei, Rong Zhu, Bolin Ding, Jingren Zhou, and Hua Lu. 2023. ALECE: An Attention-based Learned Cardinality Estimator for SPJ Queries on Dynamic Workloads. *Proceedings of the VLDB Endowment* 17, 2 (2023), 197–210.

[51] Shuyang Li. 2020. Recipes and Interactions. https://www.kaggle.com/datasets/shuyangli94/food-com-recipes-and-user-interactions

[52] Shuyang Li, Yufei Li, Jianmo Ni, and Julian McAuley. 2022. SHARE: A System for Hierarchical Assistive Recipe Editing. In *EMNLP*. 11077–11090.

[53] Richard J. Lipton, Jeffrey F. Naughton, and Donovan A. Schneider. 1990. Practical selectivity estimation through adaptive sampling. In *SIGMOD*. 1–11.

[54] Jie Liu, Wenqian Dong, Qingqing Zhou, and Dong Li. 2021. Fauce: Fast and accurate deep ensembles with uncertainty for cardinality estimation. *Proceedings of the VLDB Endowment* 14, 11 (2021), 1950–1963.

[55] Tianyi Liu, Minshuo Chen, Mo Zhou, Simon S. Du, Enlu Zhou, and Tuo Zhao. 2019. Towards understanding the importance of shortcut connections in residual networks. In *NeurIPS*. 7892–7902.

[56] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Multi-Task Deep Neural Networks for Natural Language Understanding. In *ACL*. 4487–4496.

[57] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I. Jordan. 2017. Deep transfer learning with joint adaptation networks. In *ICML*. 2208–2217.

[58] Bodhisattwa Prasad Majumder, Shuyang Li, Jianmo Ni, and Julian McAuley. 2019. Generating Personalized Recipes from Historical User Preferences. In *EMNLP-IJCNLP*. 5976–5982.

[59] Zizhong Meng, Xin Cao, and Gao Cong. 2023. Selectivity Estimation for Queries Containing Predicates over Set-Valued Attributes. *Proceedings of the ACM on Management of Data* 1, 4 (2023), 1–26.

[60] Guido Moerkotte and Axel Hertzschuch. 2020. alpha to omega: the G(r)eek Alphabet of Sampling. In *CIDR*.

[61] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. 2009. Preventing bad plans by bounding the impact of cardinality estimation errors. *Proceedings of*

the VLDB Endowment 2, 1 (2009), 982–993.
[62] M. Muralikrishna and David J. DeWitt. 1988. Equi-depth multidimensional histograms. In *SIGMOD*. 28–36.
[63] MySQL. 2017. SET Data Type. http://download.nust.na/pub6/mysql/tech-resources/articles/mysql-set-datatype.html
[64] Oracle. 2023. PL/SQL Collections and Records. https://docs.oracle.com/en/database/oracle/oracle-database/23/lnpls/plsql-collections-and-records.html
[65] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*. 8026–8037.
[66] Hoifung Poon and Pedro Domingos. 2011. Sum-product networks: A new deep architecture. In *ICCV Workshops*. 689–690.
[67] Viswanath Poosala and Yannis E Ioannidis. 1997. Selectivity estimation without the attribute value independence assumption. In *VLDB*. 486–495.
[68] PostgreSQL. 2024. Array Functions and Operators. https://www.postgresql.org/docs/17/functions-array.html
[69] UNSW Sydney PVC (Research Infrastructure). 2010. Katana. (2010).
[70] David Saad. 1998. Online algorithms and stochastic approximations. Vol. 5. 6.
[71] P Griffiths Selinger, Morton M Astrahan, Donald D Chamberlin, Raymond A Lorie, and Thomas G Price. 1979. Access path selection in a relational database management system. In *SIGMOD*. 23–34.
[72] Andreas Seltenreich, Bo Tang, and Sjoerd Mullender. 2022. Bug Squashing with SQLsmith. https://github.com/anse1/sqlsmith
[73] SQL server. 2023. Table-valued Parameters in SQL Server. https://learn.microsoft.com/en-us/sql/relational-databases/tables/use-table-valued-parameters-database-engine?view=sql-server-ver16
[74] Noam Shazeer. 2020. GLU Variants Improve Transformer. *arXiv preprint arXiv:2002.05202* (2020).
[75] Zhuoran Shen, Mingyuan Zhang, Haiyu Zhao, Shuai Yi, and Hongsheng Li. 2021. Efficient attention: Attention with linear complexities. In *WACV*. 3531–3539.
[76] Yufan Sheng, Xin Cao, Yixiang Fang, Kaiqi Zhao, Jianzhong Qi, Gao Cong, and Wenjie Zhang. 2023. WISK: A workload-aware learned index for spatial keyword queries. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–27.
[77] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature* 529, 7587 (2016), 484–489.
[78] Zhou Tao, Chang XiaoYu, Lu HuiLing, Ye XinYu, Liu YunCan, and Zheng XiaoMin. 2022. Pooling operations in deep learning: from "invariable"' to "variable"'. *BioMed Research International* 2022, 1 (2022), 4067581.
[79] Hugo Touvron, Matthieu Cord, Alaaeldin El-Nouby, Piotr Bojanowski, Armand Joulin, Gabriel Synnaeve, and Hervé Jégou. 2021. Augmenting Convolutional networks with attention-based aggregation. *arXiv preprint arXiv:2112.13692* (2021).
[80] Kostas Tzoumas, Amol Deshpande, and Christian S. Jensen. 2011. Lightweight graphical models for selectivity estimation without independence assumptions. *Proceedings of the VLDB Endowment* 4, 11 (2011), 852–863.
[81] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*. 6000–6010.
[82] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85.
[83] Fei Wang, Mengqing Jiang, Chen Qian, Shuo Yang, Cheng Li, Honggang Zhang, Xiaogang Wang, and Xiaoou Tang. 2017. Residual attention network for image classification. In *CVPR*. 3156–3164.
[84] Hai Wang and Kenneth C. Sevcik. 2003. A multi-dimensional histogram for selectivity estimation and fast approximate query answering. In *Conference of the Centre for Advanced Studies on Collaborative research*. 328–342.
[85] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. 2018. Dataset Distillation. *arXiv preprint arXiv:1811.10959* (2018).
[86] Zhen Wang, Liqiang Zhang, Liang Zhang, Roujing Li, Yibo Zheng, and Zidong Zhu. 2018. A Deep Neural Network With Spatial Pooling (DNNSP) for 3-D Point Cloud Classification. *IEEE Transactions on Geoscience and Remote Sensing* 56, 8 (2018), 4594–4604.
[87] Benjamin Warner. 2022. Tinkering With Attention Pooling. https://benjaminwarner.dev/2022/07/14/tinkering-with-attention-pooling.
[88] Alexander Wei. 2024. Recipes with Ingredients and Tags. https://www.kaggle.com/datasets/realalexanderwei/food-com-recipes-with-ingredients-and-tags
[89] Widenex. 2024. Widenex GPTs. https://gpts.widenex.com/
[90] Peizhi Wu and Gao Cong. 2021. A Unified Deep Model of Learning from both Data and Queries for Cardinality Estimation. In *SIGMOD*. 2009–2022.
[91] Xueyi Wu, Yuanyuan Xu, Wenjie Zhang, and Ying Zhang. 2023. Billion-Scale Bipartite Graph Embedding: A Global-Local Induced Approach. *Proceedings of the VLDB Endowment* 17, 2 (2023), 175–183.
[92] Ziniu Wu, Amir Shaikhha, Rong Zhu, Kai Zeng, Yuxing Han, and Jingren Zhou. 2020. BayesCard: Revitilizing Bayesian Frameworks for Cardinality Estimation. *arXiv preprint arXiv:2012.14743* (2020).
[93] Tong Xiao, Yinqiao Li, Jingbo Zhu, Zhengtao Yu, and Tongran Liu. 2019. Sharing Attention Weights for Fast Transformer. In *IJCAI*. 5292–5298.
[94] Yuanyuan Xu, Wenjie Zhang, Ying Zhang, Maria Orlowska, and Xuemin Lin. 2024. TimeSGN: Scalable and Effective Temporal Graph Neural Network. In *ICDE*. 3297–3310.
[95] Yuanyuan Xu, Wenjie Zhang, Ying Zhang, Xiwei Xu, and Xuemin Lin. 2025. Fast and Accurate Temporal Hypergraph Representation for Hyperedge Prediction. In *Proceedings of the SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2025)*. ACM.
[96] Yang Yang, Wenjie Zhang, Ying Zhang, Xuemin Lin, and Liping Wang. 2019. Selectivity estimation on set containment search. *Data Science and Engineering* 4, 3 (2019), 254–268.
[97] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2021. NeuroCard: One Cardinality Estimator for All Tables. *Proceedings of the VLDB Endowment* 14, 1, 61–73.
[98] Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Xi Chen, Pieter Abbeel, Joseph M Hellerstein, Sanjay Krishnan, and Ion Stoica. 2019. Deep Unsupervised Cardinality Estimation. *Proceedings of the VLDB Endowment* 13, 3, 279–292.
[99] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD*. 974–983.
[100] Afia Zafar, Muhammad Aamir, Nazri Mohd Nawi, Ali Arshad, Saman Riaz, Abdulrahman Alruban, Ashit Kumar Dutta, and Sultan Almotairi. 2022. A comparison of pooling methods for convolutional neural networks. *Applied Sciences* 12, 17 (2022), 8643.
[101] Hansong Zhang, Shikun Li, Pengju Wang, Dan Zeng, and Shiming Ge. 2024. M3D: Dataset Condensation by Minimizing Maximum Mean Discrepancy. In *AAAI*. 9314–9322.
[102] Jintao Zhang, Chao Zhang, Guoliang Li, and Chengliang Chai. 2024. PACE: Poisoning Attacks on Learned Cardinality Estimation. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 1–27.
[103] Linhan Zhang, Qian Chen, Wen Wang, Chong Deng, ShiLiang Zhang, Bing Li, Wei Wang, and Xin Cao. 2022. MDERank: A Masked Document Embedding Rank Approach for Unsupervised Keyphrase Extraction. In *Findings of the Association for Computational Linguistics: ACL 2022*. Association for Computational Linguistics, 396–409.
[104] Yan Zhang, Jonathon Hare, and Adam Prugel-Bennett. 2019. Deep Set Prediction Networks. In *NeurIPS*. 3212–3222.
[105] Zeyu Zhang, Jiamou Liu, Kaiqi Zhao, Song Yang, Xianda Zheng, and Yifei Wang. 2023. Contrastive learning for signed bipartite graphs. In *SIGIR*. 1629–1638.
[106] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. 2021. Dataset Condensation with Gradient Matching. In *ICLR*.
[107] Kangfei Zhao, Jeffrey Xu Yu, Zongyan He, Rui Li, and Hao Zhang. 2022. Lightweight and accurate cardinality estimation by neural network gaussian process. In *SIGMOD*. 973–987.
[108] Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. 2020. GMAN: A graph multi-attention network for traffic prediction. In *AAAI*. 1234–1241.
[109] Rong Zhu, Ziniu Wu, Yuxing Han, Kai Zeng, Andreas Pfadler, Zhengping Qian, Jingren Zhou, and Bin Cui. 2021. FLAT: Fast, Lightweight and Accurate Method for Cardinality Estimation. *Proceedings of the VLDB Endowment* 14, 9 (2021), 1489–1502.