ShadowAQP: Efficient Approximate Group-by and Join Query via Attribute-oriented Sample Size Allocation and Data Generation

Rong Gu*
State Key Laboratory for Novel Software Technology
Nanjing University
gurong@nju.edu.cn

Han Li
State Key Laboratory for Novel Software Technology
Nanjing University
han.li@smail.nju.edu.cn

Haipeng Dai*
State Key Laboratory for Novel Software Technology
Nanjing University
haipengdai@nju.edu.cn

Wenjie Huang
State Key Laboratory for Novel Software Technology
Nanjing University
wenjiehuang@smail.nju.edu.cn

Jie Xue
New York University Shanghai
jiexue@nyu.edu

Meng Li*
State Key Laboratory for Novel Software Technology
Nanjing University
meng@nju.edu.cn

Jiaqi Zheng
State Key Laboratory for Novel Software Technology
Nanjing University
jzheng@nju.edu.cn

Haoran Cai
No Affiliation
caihaoran18@163.com

Yihua Huang
Guihai Chen*
State Key Laboratory for Novel Software Technology
Nanjing University
{yhuang,gchen}@nju.edu.cn

ABSTRACT

Approximate query processing (AQP) is one of the key techniques to cope with big data querying problem on account that it obtains approximate answers efficiently. To address non-trivial sample selection and heavy sampling cost issues in AQP, we propose ShadowAQP, an efficient and accurate approach based on attribute-oriented sample size allocation and data generation. We select samples according to group-by and join attributes, and determine the sample size for each group of unique value combinations to improve query accuracy. We design a conditional variational autoencoder model with automatic table data encoding and model update strategies. To further improve accuracy and efficiency, we propose a set of extensions, including parallel multi-round sampling aggregation, data outlier-aware sampling, and dimension reduction optimization. Evaluation results on diversified datasets show that, compared with SOTA approaches, ShadowAQP achieves 5.8× query speed performance improvement on average (up to 12.8×), while reducing query error by 74% on average (up to 95%) at the same time.

PVLDB Reference Format:
Rong Gu, Han Li, Haipeng Dai, Wenjie Huang, Jie Xue, Meng Li, Jiaqi Zheng, Haoran Cai, Yihua Huang, and Guihai Chen. ShadowAQP: Efficient Approximate Group-by and Join Query via Attribute-oriented Sample Size Allocation and Data Generation. PVLDB, 16(13): 4216 - 4229, 2023.
doi:10.14778/3625054.3625059

* represents the corresponding authors.

Proceedings of the VLDB Endowment, Vol. 16, No. 13 ISSN 2150-8097.
doi:10.14778/3625054.3625059

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/nju-lands/ShadowAQP.

1 INTRODUCTION

Approximate query processing (AQP) is proposed to compute approximate answers as exactly as possible in many time-inefficient or resource-consuming query scenarios, such as exploratory analytics [4, 51], interactive visualization [35, 41], and real-time IoT data summarization [30]. Instead of executing queries on the total data, AQP performs sampling over data and generates answers based on the samples to estimate the analytical query results. For join queries, prior works on AQP aggregate join [3, 12, 14, 18, 20, 23, 32] mainly adopt the join-after-sample strategy to avoid joining total data. Naively taking samples from tables cannot guarantee the join matching of sample tables, thereby suffering from high sampling error [12]. As for group-by queries, existing approaches [2, 14, 20, 31, 44] have drawbacks in processing small groups, large group variance, and data outliers issues.

For example, uniform sampling allocates the sample size of each group in proportion to its original size. It would select fewer or even no tuples from small groups, resulting in poor query accuracy [31]. Stratified sampling [2] assigns the sample size equally to each group. Distinct sampler [20] guarantees that at least a certain number of tuples in each sub-group will be selected as samples. However, groups with high variances in the aggregation attribute are more heterogeneous and require more sample tuples than the groups with low variances. This issue also exists in sampling methods like Universe Sampler [20] and Two-Level Sampling [14]. Though works in [44] [31] consider the aggregation attribute variance, they do not think about outlier tuples which have great influence on

the aggregate results. To sum up, a proper sampling size allocation strategy is critical to the accuracy of AQP.

In addition, most existing AQP methods [18, 20, 23] rely on data-scanning based sampling, which usually impose non-negligible I/O overhead. To address the I/O cost issue, many machine learning model-based AQP methods, which aim to learn the underlying table data distribution, have been proposed [27, 28, 43, 55]. They can be mainly classified into sample-based and probability-based approaches. Sample-based approaches [47, 55, 66] generate samples based on learned models and obtain query answers on them. However, prior work [55] does not allocate sample sizes according to group-by or join attributes, and thus can hardly get sufficiently accurate estimation. The probability-based approaches [27, 28, 43, 45, 67] directly approximate query answers based on the probabilities learned by models. These approaches suffer from obvious query errors for join queries because it is difficult to learn the joint probability distribution of tables.

In this paper, we propose ShadowAQP, an efficient and accurate AQP approach based on the attribute-oriented sample size allocation and conditional sample data generation. We divide the AQP process into two steps: sample size allocation and sample tuples acquisition. Firstly, given the sampling ratio, our sample size allocation algorithm determines the reasonable sample quota for each group to improve the estimation accuracy. After that, to efficiently and accurately generate samples, we devise a conditional generative model with various table data encoding techniques and optimization extensions. In this way, our approach can effectively estimate the query results for queries without scanning data.

We mainly confront three obstacles in the design of the proposed approaches. The first challenge is how to effectively select samples from tables under a fixed sampling ratio to give a good approximation. As mentioned above, naively taking samples from tables cannot guarantee the query result accuracy for all groups and the join matching of sample tables, thereby suffering from high sampling error. To address this challenge, we select samples according to the given group-by and join attributes and determine the sample size allocation for each value combination of attributes under the given total sample size. The proposed sample size allocation method uses Markov's inequality to get an upper bound of the error function's expectation, then sets the sample sizes by minimizing the upper bound while ensuring enough samples for small groups.

The second challenge is how to build a generative model over a table with various data types to approximate the underlying data distribution with high fidelity. To address this concern, we propose an approach called Table-CVAE that customizes the general conditional variational autoencoder (CVAE) [49] model with automatically-chosen data encoding strategies. Table-CVAE can capture the conditional distribution of table data with a lightweight neural network model. We also design three model update strategies along with an automatic strategy selection method. During online service, once given a query, Table-CVAE can efficiently generate samples according to the query semantics.

The third challenge is how to reduce query accuracy loss caused by learned model bias and outliers in the original dataset. Besides, how to avoid the dimension explosion for complicated join queries, such as multi-join queries? To address these problems, we propose a set of extensions, including the parallel multi-round sampling aggregation strategy, data outlier-aware sampling method, and dimension reduction optimization.

The main contributions of this paper are summarized as follows.

- **Attribute-oriented sample size allocation**. For the sample size allocation, we propose to select samples according to the group-by and join attributes and determine the sample size quota for each value combination. The proposed sample size allocation algorithm leverages Markov's inequality to obtain an upper bound of the error function's expectation, then determines the sample sizes by minimizing the upper bound.
- **Conditional sample generation with automatic data encoding and model update**. To avoid inefficiently sampling table data by scanning datasets, we propose Table-CVAE, which can pre-encapsulate the table data distribution into a learned conditional generative model with both automatic data encoding and automatic model update.
- **Extensions for query accuracy and efficiency**. To reduce the data bias of model generation, we propose parallel multi-round sampling aggregation strategy to improve query accuracy. To prevent missing data outliers, we propose the data outlier-aware sampling method to identify and separate outlier tuples. To handle complicated join queries, we propose dimension reduction optimization to efficiently generate samples for complicated queries.
- **Extensive experimental performance evaluation**. Evaluation results on synthetic and real-world datasets show that, compared with state-of-the-art approaches, our approach achieves 5.8× speedup on average (up to 12.8×) in query latency, while reducing error by 74% on average (up to 95%) in query accuracy at the same time. Moreover, the model used in our approach is lightweight in size and efficient to train. Finally, the evaluation results show that ShadowAQP is effective in real-world business applications.

2 PRELIMINARIES

AQP aggregate query. In this paper, we mainly focus on AQP approaches for the aggregate query with group-by or join operators. For quick reference, related notations are summarized in Table 1. Consider two table T_1 and T_2 to be joined on the common attribute A_J and then grouped by the categorical attribute A_G, the general format of the aggregate query is given by

SELECT A_G, $AGG(A_{a1})$,..., $AGG(A_{ai})$
FROM T_1 **JOIN** T_2 **ON** $T_1.A_J = T_2.A_J$
WHERE $condition$ **GROUP BY** A_G;

Here, similar to other AQP methods like [19, 55], aggregate functions AGG could be standard aggregate operators, such as AVG, $COUNT$, and SUM. A_{ai} is the aggregation attribute that is involved in aggregate functions. The values of aggregation attributes in our approach are numeric. The query demonstrated above is a two-table join one, and we also support multi-table join query.

Consider an aggregate query on a single table T, AQP first obtains sample table S with a sampling ratio. Then, aggregate functions can be applied on S to obtain the approximate answers. Specifically, the aggregate result of T can be estimated with that of sample table S:

$$AGG(T, A_{ai}) \approx \phi \cdot AGG(S, A_{ai}), \tag{1}$$

Table 1: Notations

Notations	Definition
T_i	Table i
J	Result of joining tables
A_J	Join attribute
A_G	Group-by attribute
A_{ai}	Aggregation attribute
D_J, D_G	Domain of values in join/group-by attribute, respectively
v, u	Value in group-by and join attribute, respectively
ϕ	Scaling factor to scale the aggregate results on the sample
M	Total sample size
m_v, m_u	Number of samples for group value v and u, respectively
n_v, n_u	Group size of group G_v and G_u, respectively
S_i, S_J	Sample of table T_i and join result J, respectively
ρ_i	Sampling ratio for table T_i
k	Sample size threshold for small groups
$SAMPLE(T_i, \rho_i)$	Sampling table T_i with ratio ρ_i
$AGG(T_i, A_{ai})$	Aggregation on attribute A_{ai} of T_i

where ϕ is a scaling factor that scales the aggregate results on the sample according to the sampling ratio.

Consider an aggregate query involved T_1 and T_2, the join-after-sample AQP first separately samples T_1 and T_2 with sampling ratios of ρ_1 and ρ_2, respectively.

$$S_1 = SAMPLE(R_1, \rho_1), S_2 = SAMPLE(R_2, \rho_2). \tag{2}$$

Next, it joins S_1 and S_2 to get the sample of the join result $S_J = S_1 \bowtie S_2$. Its sampling ratio is $\rho_1 \cdot \rho_2$ with respect to J, which is the result of the joining tables. Similarly, the estimated result can be obtained by scaling the aggregate result on the sample table S_J.

Error metric for AQP result. Suppose the tuples are grouped by group-by attribute A_G, and thus each attribute value v in domain D_G corresponds to one group G_v. We denote by μ_v the ground-truth aggregate value of group G_v for the query q, which represents the ground-truth value of $AGG(A_{ai})$ for group G_v, and by $\tilde{\mu}_v$ the estimated value we obtain. Define the error for group G_v as

$$err_v = 1 - e^{-|\tilde{\mu}_v - \mu_v|/\mu_v}, \tag{3}$$

This error is derived from the relative error [28]. We normalize the error, which measures the difference between the estimated value and the ground-truth, in the boundary between 0 and 1 (0%∼ 100%). By taking the exponential of the relative error, we avoid overemphasizing estimates with high errors and provide reasonable treatment of different error values in each group. For group-by queries, the average relative error [55] is adopted to measure query accuracy, which is defined as

$$err(q) = \sum_{v \in D_G} err_v / |D_G|, \tag{4}$$

3 ATTRIBUTE-ORIENTED SAMPLE SIZE ALLOCATION

3.1 Group-by Attribute-oriented Sample Size Allocation

As mentioned above, small groups are usually under-represented in the uniform sample [31]. In addition, groups with high variance are also easy to be under-represented. This is because they need more samples than groups of low variance.

To address the above issues, we propose the group-by attribute-oriented sample size allocation strategy to determine the reasonable sample size for each group. Consider a table T of size N with the group-by attribute of A_G. We denote the domain of A_G by D_G, and the number of groups is $|D_G|$. Recall that for each value $v \in D_G$, we denote G_v as the group of tuples in table T whose A_G-attribute is v, and define group size $n_v = |G_v|$. Each group only needs to store the ratio of the standard deviation to the variance of its aggregation attributes. Thus the storage overhead is not large compared to the size of the original table. Furthermore, we store the statistics data in a hash table manner (in memory or disk), which has very low query complexity. Therefore, the statistics of all these notations can be computed and stored offline with neglectable overhead. Then, they can be used many times online.

Given a total sample size M, the goal of sample size allocation is to determine the number of samples m_v for each group value $v \in D_G$, satisfying $M = \sum_{v \in D_G} m_v$. In practice, M is computed by multiplying the sampling ratio with the table size. The sampling ratio ρ is directly related to the query running time t and query accuracy acc, respectively. It is easy to fit regression model functions $\rho = f(t)$ or $\rho = f(acc)$ with a few (ρ, t) or (ρ, acc) data points. Then, users can determine the sampling ratio ρ according to the query accuracy or running time demand by calculating $f(t)$ or $f(acc)$.

First, to deal with the under-representation issue of small groups, the key idea is to allocate enough sample sizes for small groups to prevent sampling too few samples for them. We define a positive integer $k = \psi \cdot M/|D_G|$ as the lower bound threshold of the sample size for each group. Ensuring $m_v \geq k$ during the sample size allocation can prevent small groups from having too few tuples in the sample. Users can determine k by calculating $\psi \cdot M/|D_G|$, where ψ is the scaling coefficient specified by the user, M is the total sample size, and $|D_G|$ is the number of groups. ψ can be set as 0.3 empirically. The threshold k ensures the base amount of samples for each group, preventing sampling too few samples for them. Additionally, it allocates only a portion of the samples equally to each group, while the remaining samples are allocated using the following sample size allocation method.

Second, as for the issue of large differences in variances among groups, we take the mean and variance into consideration when allocating samples to groups. We bias the sample size allocation by minimizing the expected values of the error function $err(q)$ defined in Equation 4. However, it is extremely difficult as it contains inverse-exponent functions err_v's (Equation 3). Thus, we first establish an upper bound for $err(q)$, and then minimize the upper bound instead of $err(q)$. For $v \in D_G$, since $err_v \in [0, 1]$, we have

$$\begin{aligned}
\mathbb{E}[err_v] &= \int_0^1 \Pr[err_v > \delta]\ d\delta \\
&= \int_0^1 \Pr\left[|\tilde{\mu}_v - \mu_v| > -\mu_v \cdot \ln(1-\delta)\right]\ d\delta \\
&= \int_0^1 \Pr\left[(\tilde{\mu}_v - \mu_v)^2 > \mu_v^2 \cdot \ln^2(1-\delta)\right]\ d\delta.
\end{aligned}$$

We mainly assume that the variables are i.i.d., with bounded and integrable probability density functions or cumulative distribution functions. The mean and variance of the data distribution are μ_v and σ_v^2 for group G_v, respectively. For AVG queries, $\tilde{\mu}_v$ is equal to

$\frac{1}{m_v}$ times the sum of m_v i.i.d. random variables with mean μ_v and variance σ_v^2, and thus the mean of $\tilde{\mu}_v$ is μ_v and the variance of $\tilde{\mu}_v$, *i.e.*, $\mathbb{E}[(\tilde{\mu}_v - \mu_v)^2]$, is σ_v^2/m_v. Thus, by Markov's inequality, we have

$$\Pr\left[(\tilde{\mu}_v - \mu_v)^2 > \mu_v^2 \cdot \ln^2(1-\delta)\right] < \frac{\sigma_v^2}{m_v \mu_v^2 \cdot \ln^2(1-\delta)},$$

for any $\delta \in [0,1]$. It follows that $\mathbb{E}[err_v] \leq \int_0^1 \frac{\sigma_v^2}{m_v \mu_v^2 \ln^2(1-\delta)}\, d\delta$ for all $v \in D_G$, and hence by Equation 4 we have

$$\mathbb{E}[err(q)] \leq \left(\int_0^1 \frac{1}{\ln^2(1-\delta)}\, d\delta\right) \sum_{v \in D_G} \frac{\sigma_v^2}{m_v \mu_v^2},$$

Our goal is to determine the values of m_v's, so that the upper bound of $\mathbb{E}[err(q)]$ in the above inequality is minimized. To avoid small groups being under-represented or missing in the sample table, we need to ensure that the sample size of each group is not less than k. Note that $\int_0^1 \frac{1}{\ln^2(1-\delta)}\, d\delta$ is a constant. So it suffices to minimize $\sum_{v \in D_G} \frac{\sigma_v^2}{m_v \mu_v^2}$, subject to $\sum_{v \in D_G} m_v = M$ and $m_v \geq k$ for all $v \in D_G$. We denote the $\frac{\sigma_v^2}{\mu_v^2}$ as a_v for all $v \in D_G$ and sort a_v's to get an ascending list $a_1, a_2, \cdots, a_{|D_G|}$. Then, the problem is changed to minimize $\sum_{i=1}^{|D_G|} \frac{a_i}{m_i}$, subject to $\sum_{i=1}^{|D_G|} m_i = M$ and $m_i \geq k$ for all $1 \leq i \leq |D_G|$. In an optimal solution of this minimization problem, for all $1 \leq i \leq j \leq |D_G|$, m_i and m_j must satisfy the following two conditions described below.

- If $m_i > k$ and $m_j > k$, then $m_i/m_j = \sqrt{a_i}/\sqrt{a_j}$ (because otherwise we can change the values of m_i and m_j while keeping the values of other variables unchanged to make the value of the objective function smaller).
- If at least one of m_i and m_j is equal to k, then $m_i = k$ (because otherwise we can decrease m_i and increase m_j to make the value of the objective function smaller).

One can easily verify that the sample size allocation satisfying the above conditions is exactly the following:

$$m_1 = m_2 = \cdots = m_p = k, m_i = \frac{\sqrt{a_i} \cdot (M - k \cdot p)}{\sum_{r=p+1}^{|D_G|} \sqrt{a_r}} \text{ for all } i > p,$$

where $p = \min\{j : \sqrt{a_{j+1}} \cdot (M - k \cdot j)/(\sqrt{a_{j+1}} + \cdots + \sqrt{a_{|D_G|}}) > k\}$.

Note that the Markov Inequality based allocation is used to adjust the sample allocation quantity to reduce errors, while k serves as a lower bound for the sample allocation, ensuring that small groups are not overlooked.

3.2 Join Attribute-oriented Sample Size Allocation

For a join query on T and T', the simplest sampling method is to create a uniform sample of each table, say S and S', and use $S \bowtie S'$ to approximate the aggregate statistics of the original join result $J = T \bowtie T'$. However, joining two uniform samples results in fewer output tuples, leading to the poor query accuracy. For example, joining two p-fraction ($0 \leq p \leq 1$) uniform samples will produce less than p^2 of the output tuples of the original join when the distribution of the original tables is non-uniform [12]. The reason is that tuples with some values on the join attribute are under-represented or missing in the uniform samples.

To obtain good samples for join queries, we sample the table according to the join attribute. Similar to group-by attribute-oriented sample size allocation, we determine the sample quota for each set of tuples with the same value on the join attribute to improve the accuracy of approximate results. In other words, we divide the tuples into different groups according to each distinct value on the join attribute. Then assign the sample quota for each group based on the sample size allocation algorithm described in Section 3.1, by considering both the mean and variance of the aggregation attribute across different groups.

In this way, the output cardinality of the join of samples can be guaranteed. Assume that the two tables T and T' are of size n and n', respectively, and they are joined with respect to an attribute A_J. For each value $u \in D_J$, we denote the set of tuples with join value u in T and T' by G_u and G'_u, respectively. Let n_u and n'_u be the number of tuples in G_u and G'_u, respectively. Clearly, the cardinality of the original join output $T \bowtie T'$ can be calculated by

$$|T \bowtie T'| = \sum_{u \in D_J} n_u n'_u. \tag{5}$$

Suppose S and S' are the samples taken from T and T', respectively; m_u and m'_u are the numbers of tuples with the u on the join attribute in both samples. The cardinality of the join sample output $S \bowtie S'$ can be calculated by

$$|S \bowtie S'| = \sum_{u \in D_J} m_u m'_u. \tag{6}$$

Therefore, the cardinality of the join sample output can be determined, and so is its proportional relationship with the original join result. Let ρ_u be the sampling ratio of the group in the original join result for the value u, which can be calculated as

$$\rho(u) = \frac{m_u m'_u}{n_u n'_u}. \tag{7}$$

For *SUM* aggregation, we need to scale the aggregation results with the sampling ratio $\rho(u)$.

We can ensure a given cardinality on the output of the join, in other words, the sampling ratio for join, by conducting a binary search on the sampling ratio for each table as follows. Suppose there is a function $f : [0,1] \to [0,1]$, such that if we use a sampling ratio p for each table, then the sampling ratio for the join is $f(p)$. Apparently, f is monotonically increasing. Also, f can be efficiently evaluated since it suffices to determine how many samples we have for each value on the join attribute in each table according to our sample size allocation algorithm (similar to that in Section 3.1). If q is the desired sample ratio for join, we only need to find a p such that $f(p)$ is roughly equal to q and then use p as the sampling ratio for each table. By the monotonicity of f, p can be computed using binary search, which won't take much time since f can be efficiently evaluated. It works similarly when the sampling ratios of two tables are different.

For multiple aggregation functions, we can sum up their variance (with/without weights) during sample size allocation. The related theoretical analysis is similar in Section 3.1. For a group-by with join

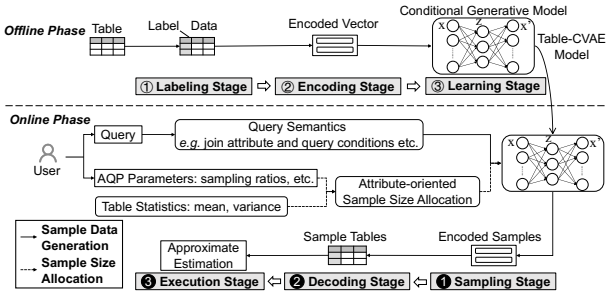


Figure 1: Online and offline workflow of ShadowAQP.

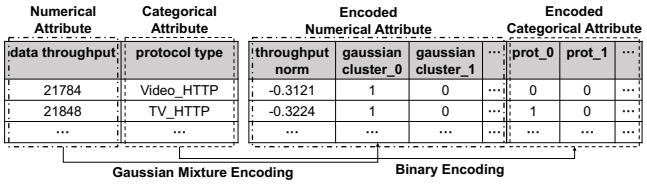


Figure 2: Data encoding strategies of categorical and numerical attributes in table.

aggregation query, we combine the group-by and join attributes, and determine the sample size quota for each value combination.

For joining tables with quite different complexity, we take into account the data distribution of the aggregate attributes in different sample tables. If the data distribution of the aggregate attributes in a particular table is more uneven, allocating more samples to that table can more comprehensively reflect its distribution and reduce query errors. The uniformity is measured by the coefficient of variation (CV), which is the ratio of the standard deviation to the mean of the aggregate attributes in the tables. Reasonable sampling ratios are set based on the relative value of the CVs of the aggregate attributes in the two tables.

4 CONDITIONAL SAMPLE GENERATION BASED ON TABLE-CVAE

Sampling from big data requires frequent random I/O access, resulting in high query latency [15]. To solve this issue, we propose the Table-CVAE approach, which generates samples using the conditional generative model with automatic data encoding strategies.

4.1 Conditional Sampling with Automatic Table Data Encoding

We design a table data generator based on conditional generative models [49] to achieve conditional sample generation. However, existing basic conditional generative models cannot directly train over database tables and generate sample tuples. On the one hand, unlike image datasets with clear labels, the tuples in the table usually do not have explicit labels. On the other hand, the values in tables are often heterogeneous and have various domains.

To address these problems, we propose Table-CVAE (Table Conditional Variational AutoEncoder), a table-oriented generative model to learn the conditional data distribution over database tables for generating targeted sample tuples according to the given query conditions. The overall workflow of ShadowAQP which uses Table-CVAE with the sample size allocation is shown in Figure 1. We incorporate Table-CVAE into the AQP process and divide the whole process into two phases: the model training phase (offline) and the sample generation phase (online).

Model training phase. In this phase, Table-CVAE learns the underlying probability distribution of a table with following stages.

(1) Labeling stage. The goal of this stage is to label tuples in the table, that is to assign a label value y to each tuple x in a table. Table-CVAE also supports multiple label attributes by labeling the data with a combination of the values on the given join and group-by attributes. For example, a tuple that takes the values of 'male' and 'doctor' on the label attributes is labeled with 'male-doctor'.

(2) Encoding stage. Generally, Table-CVAE uses different data encoding methods for different attribute types, as is shown in Figure 2. For categorical attributes, we use *binary encoding* to encode values into integers and turn them into binary representation. For example, 'Video_HTTP' in the protocol attribute is assigned to '0' and is represented as [0,0,0,...,0]. For numeric attributes, we transforms the values into the range $[0, 1]$. The Min-Max scaler is a common way to normalize numeric values with uniform distribution. Suppose the min and max of a numeric attribute are v_{min} and v_{max}. The value v is scaled by $v_{norm} = (v - v_{min})/(v_{max} - v_{min})$.

When the data distribution is non-uniform, the Min-Max scaler does not perform well. We design a Gaussian mixture encoder to solve this issue. It uses a Gaussian mixture model [42] to fit the distribution of each numeric attribute. Since dozens of Gaussian distributions are sufficient in most scenarios, the Gaussian mixture encoder transforms a numeric value into a normalized numeric value and a one-hot encoded No. of Gaussian distribution.

In addition, we design an automatic encoding method selection strategy that automatically chooses a proper encoding method for each numeric attribute. The strategy builds an equi-width histogram for each numeric attribute and computes the standard deviation δ and mean μ of bin sizes. The more uniform the distribution of a numeric attribute is, the lower the δ/μ is. Thus, we use the Min-Max scaler to encode the numeric attributes with low δ/μ while using the Gaussian mixture encoder to encode the rest numeric attributes.

(3) Learning stage. In this stage, the encoded data and labels are fed to the Table-CVAE model for training. Table-CVAE learns a low dimensional latent representation of tables with the encoder network and the decoder network. It usually only needs multiple (usually less than 10) fully connected layers without using any convolutional layers or recurrent layers, resulting in efficient model training and updating.

Sample generation phase. After the training phase, Table-CVAE can generate sample tuples based on the learned conditional probability distribution. The generation phase includes the sampling stage, the decoding stage, and the execution stage.

(1) Sampling stage. In this stage, Table-CVAE generates sample vectors with the latent variables and the given labels. Suppose the dimension of latent variables is n. The data point in the latent space is an n-dimensional vector $\boldsymbol{z}$. The value of each dimension in $\boldsymbol{z}$ comes from a Gaussian distribution $N(\boldsymbol{\mu}, \boldsymbol{\sigma})$. By leveraging the reparameterization trick [21], the decoder network in Table-CVAE will convert $\boldsymbol{z'}$ into $\boldsymbol{z}$ as $\boldsymbol{z} = \boldsymbol{z'} \odot \boldsymbol{\sigma} + \boldsymbol{\mu}$. Therefore, we can generate

sample vectors by sampling from $N(\mathbf{0}, \mathbf{1})$ instead of $N(\boldsymbol{\mu}, \boldsymbol{\sigma})$. Next, we can label $\boldsymbol{z}'$ with the required label values $\boldsymbol{c}$ (in the vector form) so that the decoder network in Table-CVAE can generate sample data with the same label values.

(2) Decoding stage. The decoding stage is responsible for converting the sample data generated from the Table-CVAE model into table tuples. For each encoded numeric value x_e, Table-CVAE first finds out the No. of distribution to which it belongs and obtains the corresponding mean μ and variance σ^2. The original numeric value can be calculated by $x_e * \sigma + \mu$.

(3) Execution stage. In this stage, we executes the queries on the generated samples to obtain the approximate answers. We rewrite the query such that it can be executed on the sample tables instead of the original tables. Similar to other sampling based AQP approaches, its error guarantee can be represented by confidence intervals on the samples with the central limit theorem (CLT) [55][66].

In terms of the number of trained models, it relies on the aggregate, join, and group-by attributes in the queries. For queries that have the same attributes, the same model can be used. For queries that have different attributes, training one model can be achieved as folllows. For different aggregate attributes, we can train only one model that learns all possible columns which might be aggregate attributes in the same table. For different label attributes, such as A and B, a model can be trained using A and B as label attributes. Then the model generates samples based on both attributes, equivalently creating a finer-grained grouping of samples. But these samples can still be considered separately as samples grouped solely based on A or B attributes. For real-world OLAP applications, table attributes used as join or group-by operations do not usually change and are enumerable. Thus, for most new queries, it can utilize pre-trained models. If there is a new query for which no model has been trained for its label attributes, we can still train a model for that data table within reasonable time (as shown in Figure 15(a)).

4.2 Table-CVAE Model Update

Table-CVAE models usually need to be updated because new data may be added to the original table. Since neural network models can be updated with incremental training [7, 39, 48] using stochastic gradient descent [5] parameter optimizations, we propose three Table-CVAE model update strategies for different scenarios.

The first scenario is that the distribution of the new data is similar to the old one. In this case, we update the trained model only with the new data and call this update strategy *Incremental Train_update*. This strategy does not cause large shifts in model weights. It only slightly biases the model towards the distribution of the new data.

The second scenario is when the distribution of the new data is quite different from the old one. We need to involve the old data in training when updating the model to prevent forgetting the learned distribution. We design a *Partial Train_update* strategy that samples from both the old and new data, and train the model on them.

The third scenario is that the model accuracy has a high priority. To ensure accuracy, we update the model with the whole updated data, which is called *Full Retrain_update*. It provides a more accurate model, but takes more time.

Additionally, we design an automatic update strategy selection method based on data distribution. To measure the similarity of the new data distribution and old data distribution, we use Kolmogorov–Smirnov test (K-S test). The two-sample K–S test can check whether the two sets of data come from the same probability distribution or not. We first construct the cumulative distribution function of the aggregation attribute in the old data and new data, denoted by $F_{old}(x)$ and $F_{new}(x)$, respectively. The K-S test statistic measures the largest distance between $F_{old}(x)$ and $F_{new}(x)$ by $D_{KS} = \sup_x |F_{new}(x) - F_{old}(x)|$, where sup_x calculates the supremum of the set of distances.

If D_{KS} is greater than the critical value of the Kolmogorov distribution K_α, the hypothesis about two sets of data coming from the same probability distribution is rejected at level α. The distribution of the new data is similar to that of the old data if $D_{KS} <= K_\alpha$, or is quite different from that of the old data if $D_{KS} > K_\alpha$. Therefore, we use *Incremental Train_update* to update the model if $D_{KS} <= K_\alpha$. Otherwise, *Partial Train_update* is used instead. If a more accurate model is preferred, and longer training time is acceptable, *Full Retrain_update* is a good strategy.

5 EXTENSIONS

5.1 Parallel Multi-round Sampling Aggregation

To mitigate the impact of model bias on query errors, we propose the parallel multi-round sampling aggregation (PMSA) strategy. The key idea of PMSA is to perform multiple rounds of sample generation and aggregation in parallel to obtain multiple sets of query answers for bias reduction, since generating samples by models is efficient. Though solely increasing the sampling ratio can also mitigate bias, it takes more time with a single thread, and is not scalable on large tables or high sampling ratios. We denote by k the total rounds of sampling generation and aggregation. PMSA takes the mean of the answers obtained from the k rounds of aggregation as the final query answer. Though PMSA introduces some execution overhead, it can improve the query accuracy of the results.

To see this, let μ be the true value to be estimated, and $\tilde{\mu}_1, ..., \tilde{\mu}_k$ denote the estimations of μ obtained from the k rounds of aggregation. Then our final answer will be $\bar{\mu} = \frac{1}{k}\sum_{i=1}^{k}\tilde{\mu}_i$. Since the k rounds of sampling are independent, $\tilde{\mu}_1, ..., \tilde{\mu}_k$ can be viewed as i.i.d. random variables. Furthermore, as $\tilde{\mu}_1, ..., \tilde{\mu}_k$ are unbiased estimations of μ, the mean of each $\tilde{\mu}_i$ is equal to μ. Let σ^2 be the variance of each $\tilde{\mu}_i$. The following theorem shows that for the estimation $\bar{\mu}$, the length of the confidence interval $[\mu - \delta, \mu + \delta]$ around μ of a fixed confidence level decreases along with the increase of k.

Theorem 5.1. $\Pr[|\bar{\mu} - \mu| > \delta] \leq \frac{\sigma^2}{k\delta^2}$ *for any* $\delta > 0$.

Proof. Since the random variables $\tilde{\mu}_1, ..., \tilde{\mu}_k$ are independent and the variance of each $\tilde{\mu}_i$ is σ^2, the variance of the summation $\sum_{i=1}^{k}\tilde{\mu}_i$ is $k\sigma^2$. Therefore, the variance of $\bar{\mu}$ is σ^2/k. The mean of $\bar{\mu}$ is clearly μ. By applying Chebyshev's inequality, we directly have $\Pr[|\bar{\mu} - \mu| > \delta] \leq \frac{\sigma^2}{k\delta^2}$. □

We take the Flights[17] dataset as an example and set the sampling ratio to 1‰. We compare the confidence interval width (CI Width) at a 95% confidence level between the theoretical and actual experimental results. As shown in Figure 3, as k increases, the actual CI Width gradually decreases and remains consistently lower than the theoretical value. The trend of the actual CI Width closely aligns

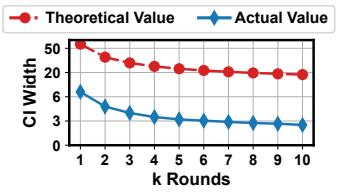


Figure 3: CI width between theoretical and actual results under various rounds of execution.

with the theoretical expectation (CI Width $\propto \frac{1}{\sqrt{k}}$). Consequently, the query error also decreases as k increases.

5.2 Data Outlier-aware Sampling

Some real-world datasets might have outlier tuples on attributes due to the Zipf's law. Outlier tuples contain outlier values on the aggregation attributes that are significantly different from the normal ones. For example, some data records of the network traffic throughput have very large throughput values than normal situations due to network attacks, which become outlier tuples. In essence, outlier tuples are not data noise or bias. Although they are extremely rare, they indeed have a great influence on the aggregate results. It is easy to omit outlier tuples during naive sampling, thus leading to inaccurate query answers in AQP.

The key idea of addressing the outlier tuple issue is to identify and preserve outlier tuples in the samples so as to avoid missing them. Before the labeling stage in Table-CAVE, we separate the outlier tuples from the table based on an outlier boundary, which is a predicate condition to distinguish between normal tuples and outlier tuples. The outlier boundary is calculated with the percentiles of aggregation attributes. Suppose the η-th percentile (*e.g.*, 99%) of the aggregation attribute A_{ai} is $Q_\eta(A_{ai})$. A tuple x whose value on A_{ai} is greater than $\gamma \cdot Q_\eta(A_{ai})$ is treated as an outlier tuple. Then, the outlier boundary for A_{ai} is $\{x|A_{ai}(x) \geq \gamma \cdot Q_\eta(A_{ai})\}$ (*e.g.*, γ is set to 10 by default). A tuple x is treated as an outlier tuple as long as it has an outlier value on any aggregation attributes. Combining the outlier boundary of each aggregation attribute, we have the outlier boundary for the table as follows.

$$\{x|[A_{a1}(x) \geq \gamma \cdot Q_\eta(A_{a1})] \vee \cdots \vee [A_{ai}(x) \geq \gamma \cdot Q_\eta(A_{ai})]\}. \quad (8)$$

We separate the outliers from the data based on table statistics and store them in a specific location. After sample generation, separated outliers are inserted into the samples, and the sampling ratio of outliers is set to 1. Furthermore, every outlier is assigned a weight equal to the sampling ratio ρ, while every non-outlier is sampled with probability ρ and has a weight 1. In this way, outlier tuples will not be missed out. It can also be guaranteed that the aggregate value of the sample is, in expectation, still equal to that of the original set, and hence no bias is brought.

5.3 Dimension Reduction Optimization

In Section 3, we propose an attribute-oriented sample size allocation algorithm, which divides the tuples into multiple groups according to their values on specific attributes and assigns the sample size for each group. However, when there are multiple join attributes or the join attributes have too many values in queries, the number of value combinations might be huge, leading to the combinatorial explosion issue. If there are too many groups, the sample size allocation algorithm becomes ineffective because all groups become small groups, and almost all the tuples require to be selected into the sample table. Consider an extreme example, suppose the number of groups is equal to the number of tuples. There is only one tuple in each group, and all tuples must be selected to avoid missing any group. Moreover, the combinatorial explosion can result in the poor learning performance of generative models because there are too few tuples in each group for training.

Therefore, we propose the dimension reduction optimization to alleviate the combinatorial explosion problem and use it when $M \cdot \psi/|D_G| < 1$, which implies that some groups will have fewer than one sample on average. The key idea of dimension reduction optimization is to reduce the number of value combinations during the labeling stage in Table-CAVE. We partition the values of join attributes into buckets and use the number of the buckets as the new value for labeling. The values of a join attribute are usually discrete and we use ordinal encoding to convert each unique category value into an integer value. Suppose the domain of a join attribute after ordinal encoding becomes $[v_{min}, v_{max}]$. Dimension reduction optimization finds a set of splitting points $\{v_1, v_2, ..., v_r\}$ and divides the domain into a set of buckets $\{[v_{min}, v_1), [v_1, v_2), ..., [v_r, v_max]\}$. For example, if each bucket is of equal width ω then the sequence number (No.) of the bucket that v_i belongs to is $floor((v_i - v_{min})/\omega)$. Then the value on the join attribute is substituted with the No. of the bucket it belongs to. Thus, the number of value combinations is decreased. The join attribute-oriented sample size allocation still works to some extent because tuples in the same bucket have higher probability of joining successfully than the random-sampled tuples.

6 EVALUATION

6.1 Experimental Setup

Hardware and software. Experiments are run on a physical cluster with 10 nodes connected via 1 Gbps Ethernet. Each node has two Intel Xeon E5-2620 v2 CPUs with 6 cores (12 hyper-threaded cores), 32 GB DDR3 memory. We deploy Apache Spark 2.3.2, Hive 3.1.2, and Postgres 14.0 on the cluster. SparkSQL and Postgres are used as distributed and single-node exact query engines for comparison. Machine learning models are trained using PyTorch 1.8.0 on a machine with one NVIDIA RTX3090 GPU.

Datasets and workloads.

(1) TPC-H [57]. We run a join query with 1 group-by and 1 aggregation attributes on TPC-H with a scale factor of 20 by default.

(2) TPC-DS (Query-A and Query-B). We mainly run two queries, TPC-DS(Query-A) and TPC-DS(Query-B), on TPC-DS [56] with a scale factor of 1 and then zoom to the size of 2/3. Both of them are join queries with 1 group-by attribute and 4 aggregation attributes.

(3) Census(Query-C and Query-D). Census [9] is a dataset extracted from the Census Bureau database. We run Census(Query-C) and Census(Query-D) on it. Both conducting self-join on *adult* table with 1 group-by attribute and 3 aggregation attributes. We use IDEBench[16] to scale the dataset up to 150 thousand records.

(4) Flights. Flights [17] is a real-world dataset containing punctuality data for all flights departing from New York in 2013. We query on the self-join of *flight* table with 1 group-by attribute and

3 aggregation attributes. IDEBench is used to scale the dataset up to 300 thousand records.

In addition, all the join result tables contain at least 1 million tuples, and is up to 24 billion.

Model configuration. By default, both the encoder network and the decoder network in ShadowAQP has two fully connected hidden layers with around 100 neurons. The activation function between layers is ReLU. Sigmoid is the output activation function of encoded categorical values, and softmax is the output activation function of encoded Gaussian number values.

Measure metrics.

(1) Query Error. We evaluate approximate query accuracy by computing the difference between the exactly accurate result and the approximate result using Equation 3 and Equation 4 in Section 2.

(2) Query Latency. Query latency is measured from the moment the query is submitted to the moment the result is generated. We repeat workloads three times and record the average results.

Comparison AQP methods. The comparison AQP methods can be divided into three categories:

(1) Traditional sampling based AQP.

a. Uniform sampling based AQP: Every sample is drawn with uniform probability.

b. Stratified sampling based AQP[2]: It divides data groups by attribute values and allocates sample quota evenly [2].

(2) Join-oriented sampling based AQP.

a. Distinct sampler [20]: It ensures that at least a certain number of tuples in each sub-group will be sampled.

b. Universe sampler [20]: It picks tuples whose join key is in a random portion of a projected high dimensional space.

c. Two-Level sampling [14]: For each join key in the chosen space, it can select at least one tuple, and each of the other tuples are sampled independently with another probability.

d. Wander join [23]: It randomly picks a tuple from join tables and finds a set of tuples that can be joined with the picked tuples, and then randomly selects one tuple from the set. Wander join is implemented in XDB [24].

(3) Model-based AQP.

a. DeepGen [55]: It uses deep generative models to learn the data distribution and generates samples from the models.

b. DBEst++ [27]: It performs AQP using integral evaluation on regression models and probability density estimators.

6.2 Effectiveness of Sample Size Allocation and Sample Generation

6.2.1 Effectiveness of sample size allocation method. We evaluate the proposed attribute-oriented sample size allocation method. Figure 4 shows that our approach outperforms the other five cutting-edge comparing methods in query error. Overall, the average query error of our approach is 3.04%, which is only 80% of the uniform sampling's error, 61% of the stratified sampling's error, and 62% of the distinct sampler's error. As shown in Figure 4(a) and Figure 4(c), two-level sampling only performs well in primary key-foreign key joins. Our approach achieves the best performance, since it allocates sample quota more reasonably.

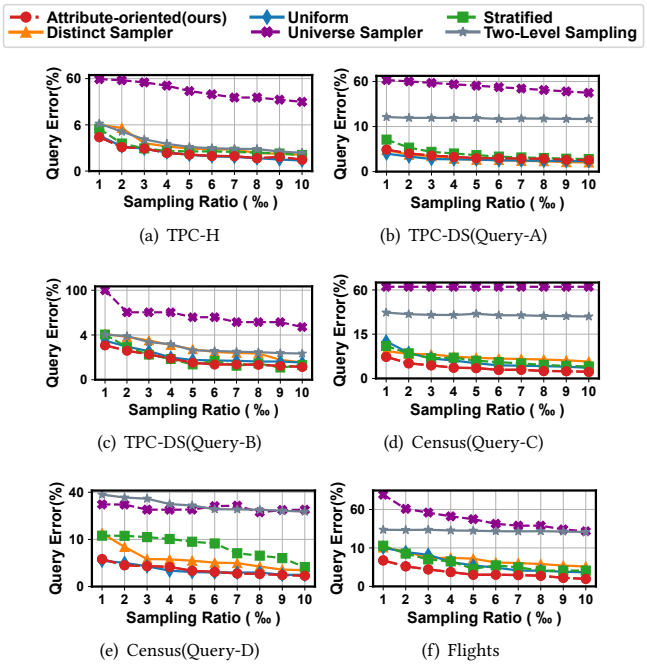


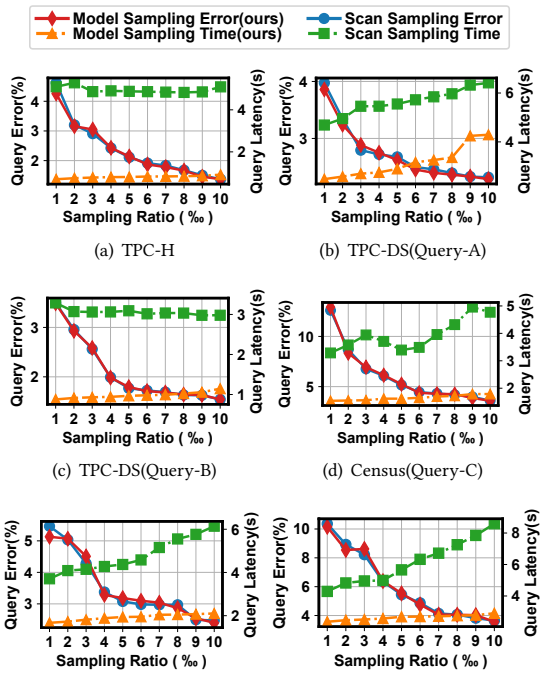
Figure 4: Query errors of attribute-oriented sample size allocation sampling and other sampling methods.

Table 2: Query errors and query latency comparison among ShadowAQP (ours) and other approaches on TPC-DS.

TPC-DS Query ID	#joined tables	#group-by attributes	Query Error			Query Latency(s)		
			Ours	Uniform	Stratified	**Ours**	Uniform	Stratified
44	1	1	**16.52%**	44.20%	41.81%	**9.138**	13.683	25.082
3	2	1	**1.80%**	4.67%	3.29%	**1.173**	9.536	25.034
23	2	1	**22.01%**	40.85%	39.30%	**22.959**	30.835	86.891
83	2	1	**2.61%**	4.58%	3.46%	**0.249**	6.621	10.405
51	2	2	**8.92%**	14.23%	10.85%	**1.037**	8.528	18.074
59	2	2	**5.08%**	9.87%	8.01%	**6.868**	15.109	21.624
32	3	0	**0.44%**	1.90%	0.86%	**1.168**	8.969	23.218
48	3	0	**0.23%**	0.69%	6.30%	**1.526**	8.579	22.734
92	3	0	**1.55%**	2.23%	3.28%	**0.521**	8.256	17.941
58	3	1	**7.48%**	23.14%	22.89%	**7.681**	20.737	33.905
77	3	1	**2.56%**	3.01%	6.40%	**0.766**	10.122	23.652
43	3	2	**1.23%**	1.97%	6.24%	**1.578**	8.321	23.216
55	3	2	**9.13%**	31.56%	24.41%	**9.055**	21.152	34.385
70	3	2	**0.24%**	3.15%	6.59%	**1.537**	8.108	23.088
86	3	2	**5.61%**	7.21%	7.22%	**0.536**	9.587	18.639
31	3	3	**25.43%**	38.93%	39.66%	**4.528**	24.633	100.864
16	4	0	**1.19%**	3.62%	1.60%	**0.815**	10.378	19.534
94	4	0	**4.58%**	5.87%	6.11%	**1.679**	9.064	18.881
27	5	2	**8.07%**	24.48%	23.08%	**14.797**	38.204	58.304
13	6	0	**0.96%**	1.12%	2.03%	**4.896**	42.895	47.827
80	6	1	**2.27%**	6.52%	5.22%	**1.63**	20.565	34.801

6.2.2 Effectiveness of sample generation in ShadowAQP. As shown in Figure 5, we evaluate the proposed model-based sample generation method. Query errors of the model-based sampling method are similar to that of the data scan-based sampling method, which retrieves random tuples from each group rather than generating new tuples using Table-CVAE. However, its query latency is obviously less than the data scan-based sampling method.

6.2.3 Performance of sample size allocation with sample generation on TPC-DS. We evaluate the performance of ShadowAQP, which

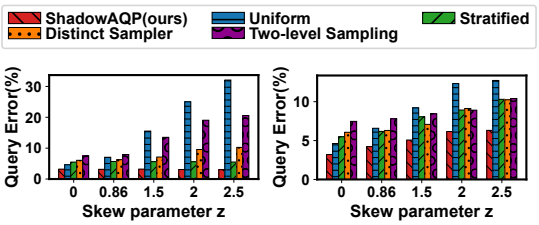


(a) TPC-H (b) TPC-DS(Query-A)

(c) TPC-DS(Query-B) (d) Census(Query-C)

(e) Census(Query-D) (f) Flights

Figure 5: Query errors and query latency of ShadowAQP sample generation and data scan-based sampling approaches.

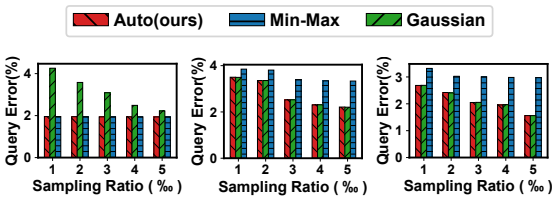


(a) Skew of the group sizes (b) Skew of the aggregation attribute

Figure 6: Impact of skew data on the accuracy.

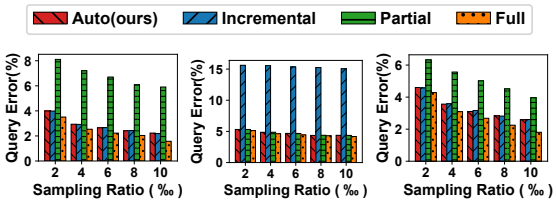
combines the sample size allocation with sample generation techniques, on TPC-DS queries compared with competitive methods. Based on # of joined tables (1∼6) and # of group-by attributes (0∼3), we conducted experiments on TPC-DS with 21 representative workloads. The sampling ratio is 1% by default. However, for complex queries with many groups, we adopt larger sampling ratios. Specifically, for queries 44, 59, 58, 55, 86, and 27 in TPC-DS, we use a 5% sampling ratio, while for queries 23, 51, and 31, we use a 10% sampling ratio. As shown in Table 2, ShadowAQP always achieves both the lowest query error and lowest query latency. This is because that ShadowAQP has a more reasonable sample allocation method and generate samples efficiently with models.

6.2.4 Impact of skew data on the accuracy. We vary the skew of the group sizes and aggregation attributes to evaluate the accuracy



(a) TPC-H (b) Census(Query-C) (c) Census(Query-D)

Figure 7: Effect of auto-encoding method selection strategy.



(a) TPC-H (b) TPC-DS(Query-A) (c) Census(Query-D)

Figure 8: Query errors of different model update strategies.

of our method. As in [2], we use the Zipf distribution to introduce skew with varying values of the z-parameter. We select $z = 0.86$ since it can result in a 90-10 distribution and is used commonly [44].

The dataset is TPC-H with a sampling ratio of 1‰. As shown in Figure 6(a), our method achieves the highest accuracy. Similarly, the error of ShadowAQP is the lowest in Figure 6(b). This is because our method takes the group sizes and aggregation attribute variances into consideration. For cases with a significant skew in group sizes, we can still sample tuples from small groups. As for groups with high variances in the aggregation attribute, we allocate more samples to represent them.

6.2.5 Effectiveness of encoding method selection strategy. We evaluate the performance of different encoding methods and the automatic selection strategy.

As shown in Figure 7, the Min-Max scaler achieves lower query error on TPC-H, while the Gaussian mixture encoder behaves better on Census(Query-C) and Census(Query-D). This is because the distributions of numerical attributes in the TPC-H dataset are more uniform. Using Min-Max scaler allows for a simple and effective encoding of the data, while the Gaussian mixture encoder introduces additional distribution assumptions. However, in the Census dataset, the distributions of numerical attributes are non-uniform. The Min-Max scaler may shrink the differences between different data points, while the Gaussian mixture encoder can better fit the data distribution, thus achieving more accurate results. Nevertheless, the proposed automatic encoding method selection strategy always chooses the better method.

6.2.6 Effectiveness of model update strategy. We use three different settings to evaluate the effectiveness of the proposed three model update strategies, respectively. The sampling ratio of *Partial Train_update* is set to 20%.

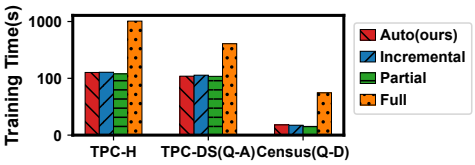
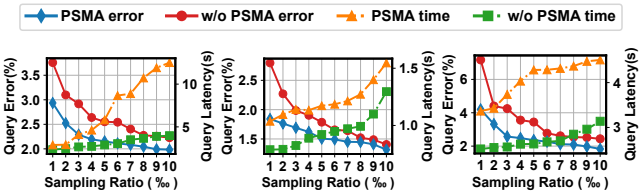


Figure 9: Training time of different model update strategies.



(a) TPC-DS(Query-A) (b) TPC-DS(Query-B) (c) Flights

Figure 10: Effect of parallel multi-round sampling aggregation (PMSA).

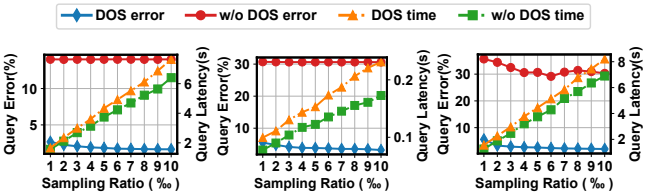
In the first setting, the distribution of the new data is similar to the old data. The result is shown in Figure 8(a). *Incremental Train_update* has lower errors compared to *Partial Train_update* (average error of 2.84% vs. 6.80%). In the second setting, the new and old data are quite different. As depicted in Figure 8(b), *Partial Train_update* achieves lower errors than *Incremental Train_update* (average error of 4.72% vs 15.36%). In the third setting, we assume a higher requirement for model accuracy. As shown in Figure 8(c), *Incremental Train_update* and *Partial Train_update* lead to an increase in query errors by 34.04% and 103.25%, respectively. *Full Train_update* has a similar query error performance to that of the original one. However, the training time of *Full Retrain_update* is about 4 to 5 times that of the other two model update strategies, as the latter updates the model with less training data. Figure 8 verifies that our automatic selection strategy can select the better one between *Incremental Train_update* and *Partial Train_update*.

6.3 Effectiveness of Extensions

6.3.1 Effectiveness of Parallel Multi-round Sampling Aggregation (PMSA). We evaluate performance of our approach with **PMSA** and without PMSA (**w/o PMSA**). We set PMSA multiple sampling round $k = 3$. As shown in Figure 10, PMSA can reduce query errors by narrowing the confidence interval of the approximate answers.

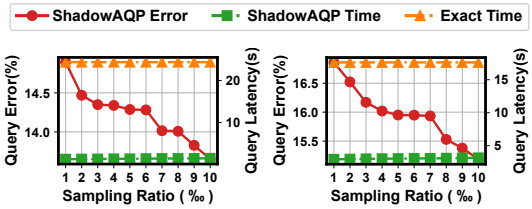
We also compare the query latency of ShadowAQP with **PMSA** and **w/o PMSA**. As shown in Figure 10, the average query latency of **PMSA** is about 1.71 times that of **w/o PMSA**. However, as shown in Figure 14, the query latency of PMSA is still much shorter than that of traditional sampling methods.

6.3.2 Effectiveness of Data Outlier-aware Sampling (DOS). The effectiveness of the **D**ata **O**utlier-aware **S**ampling (**DOS**) optimization is evaluated by comparing our approach with the version without DOS (**w/o DOS**). We inject 0.1‰ outlier values to the aggregation attribute columns in TPD-DS, and 1‰ outlier values to the aggregation attribute columns in Flights.



(a) TPC-DS(Query-A) (b) TPC-DS(Query-B) (c) Flights

Figure 11: Effect of Data Outlier-aware Sampling (DOS).



(a) TPC-H(Complex-1) (b) TPC-H(Complex-2)

Figure 12: Effect of dimension reduction optimization.

As shown in Figure 11, the query error of **w/o DOS** is almost 8.8 times that of **DOS** on average. This is because that DOS can balance the number of samples between groups and preserve data outliers for skewed dataset. In Figure 11, we find that the query latency difference between **DOS** and **w/o DOS** is within 1s. DOS has little overhead because the number of outliers is small.

6.3.3 Effectiveness of Dimension Reduction Optimization. We use two complicated queries, including multiple-joins, which would cause combinatorial explosion issues, to evaluate the effectiveness of the dimension reduction optimization.

We compare the query latency with the exact query method using native SparkSQL. As shown in Figure 12, the average query errors of our approach on the dataset are 14.21% and 15.95%, respectively, which is usually acceptable in complicated query scenarios.

6.4 Comparison with Other AQP Approaches

We also compare our approach with state-of-the-art AQP methods. Since the datasets used in this section don't contain outliers, our method does not employ **DOS** during the experiments here. We set multiple sampling rounds $k = 3$ for PMSA and also compare the version without PMSA. Figure 13 and 14 demonstrate the query errors and query latency of our approach and the other methods, respectively. The two horizontal lines with the 'Exact' label represent the query latency of SparkSQL and Postgres respectively. The query speed performance of our ShadowAQP(with PMSA) achieves 79× and 151× improvement on average compared with exact queries using SparkSQL and Postgres,respectively. The performance comparison among our approach and others is elaborated as follows:

(1) Traditional sampling based AQP. As shown in Figure 13, the average query error of ShadowAQP is only 53.44% of uniform sampling and 40.32% of stratified sampling. Meanwhile, the average query latency of our approach is close to both of them. Thus, our approach outperforms the above traditional sampling methods.

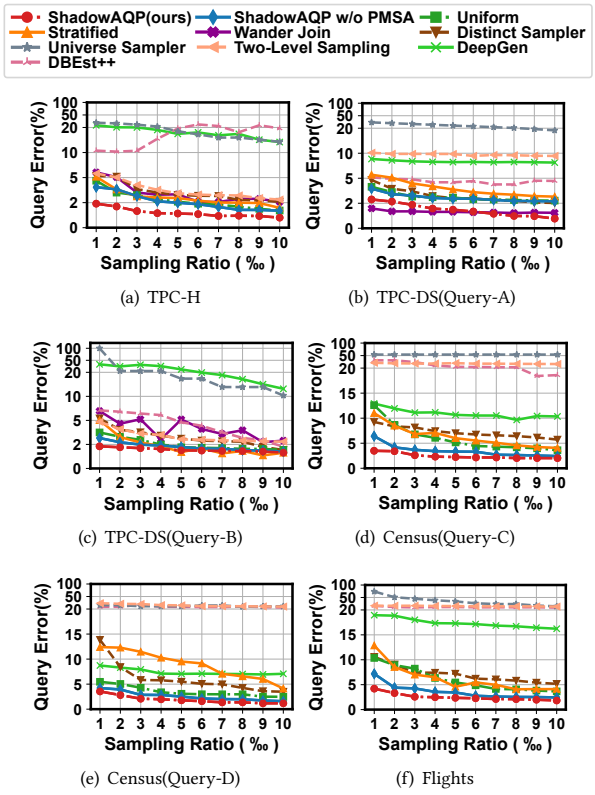


Figure 13: Query errors comparison among ShadowAQP and other AQP approaches under various datasets.

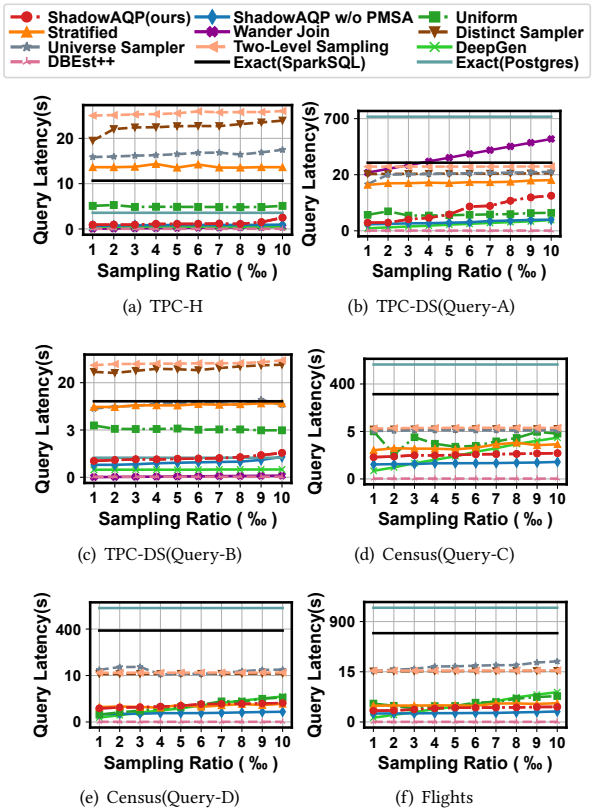


Figure 14: Query latency performance comparison among ShadowAQP and other approaches under various datasets.

(2) Join-oriented sampling based AQP. As shown in Figure 13 and 14, compared to distinct sampler, universe sampler, and two-level sampling, our approach reduces the average query error by 58.94%, 95.77%, and 90.27%, achieving a speedup of 6.81×, 10.03×, and 12.86×, respectively. As for wander join, our approach reduces the average query error by 61.5% in TPC-H and TPC-DS(Query-B) while the query latency is nearly the same. In TPC-DS(Query-A), though the average query error of wander join is lower than ours (1.32% vs 2.23%), its average query latency is 249.65 seconds which is almost 34 times that of ours and even exceeded the time cost of querying exact answers using SparkSQL (166.026 seconds).

(3) Learned Model-based AQP. As shown in Figure 13 and 14, DeepGen has large errors for all datasets because it does not support join-aggregation queries well. Query errors of DBEst++ are small in TPC-DS, but large in other datasets. This is because that DBEst++ cannot handle skewed distribution well. The average query error of ShadowAQP is only 8.88% of DBEst++ and 8.29% of DeepGen. DBEst++ and DeepGen are also learned model-based AQP. Similar to our approach, their query latency is low. However, our approach obviously achieves better accuracy.

We evaluate the training time and model storage overhead of ShadowAQP by comparing it with other model-based approaches on different datasets. Figure 15 shows the training time and model sizes of ShadowAQP, DeepGen, and DBEst++. We can see that the training time of ShadowAQP models is similar to DeepGen or

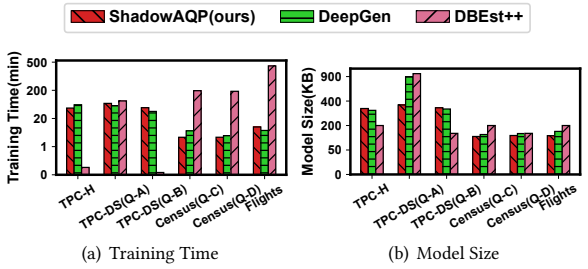
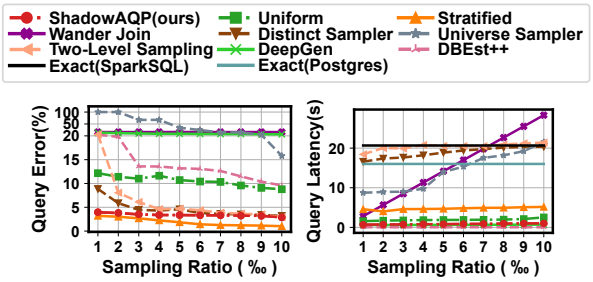


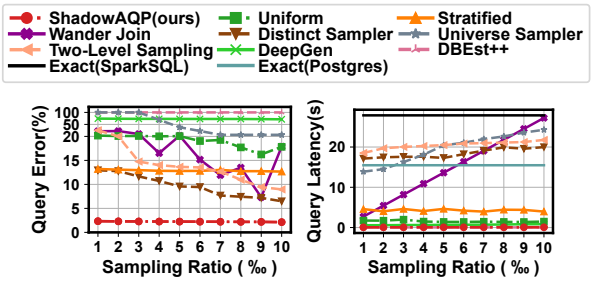
Figure 15: Training time and model size cost of ShadowAQP, DeepGen, and DBEst++ on different datasets.

DBEst++. Besides, like DeepGen and DBEst++, the model sizes of ShadowAQP are around 200 KB. The number of the distinct groups in experiments ranges from several to billions. And, all the model sizes are about hundreds of KB, thus the model size will not be large when there are many distinct values. Thus, all three methods do not bring large-scale models. In summary, ShadowAQP does not introduce excessive model training and storage overhead.



(a) Query Error (b) Query Latency

Figure 16: Performance comparison among ShadowAQP and other AQP approaches under Traffic Analysis-No_Outlier.



(a) Query Error (b) Query Latency

Figure 17: Performance comparison among ShadowAQP and other AQP approaches under Traffic Analysis.

6.5 Performance on Real-world Applications

We evaluate the performance of our approach on a real-world telecommunication network traffic analysis dataset (short as Traffic Analysis) from a large telecommunication company in China. The dataset includes information on the upstream and downstream network traffic and data packets for various communication protocols. We conduct a group-by-join aggregation query on the *traffic* table (about 1.2 million tuples) and the *protocol* table (about 1.5 thousand tuples) with the *protocol_id* join key, and there is 1 group-by attribute and 4 aggregation attributes. The origin Traffic Analysis contains some data outliers. For fair comparison with methods that can not handle outliers, we also prepare a variant dataset called Traffic Analysis-No_Outlier by removing outliers (through formula 8 with $\eta = 0.99$) from the *traffic* table (approximately 1.7K tuples).

We compare our approach with SOTA AQP methods. The round k for PMSA is 3. Figure 16 indicates that our approach is more accurate than others except stratified sampling. However, the average query latency of our approach is only 17.62% of stratified sampling, and is lower than almost other methods. Figure 17 shows that our approach achieves the lowest query error. The query latency of ShadowAQP is within 0.1s, which is lower than most other methods.

7 RELATED WORK

Approximate query processing. AQP is an extensively-studied big data query technique [1, 4, 13, 20, 33, 36, 50, 64] which can be divided into offline synopses and online sampling categories. Offline synopses use prior knowledge to pre-compute the synopses, such as samples [4, 26, 36, 37], histograms [38], sketches [6], and wavelets [10]. However, it relies heavily on prior knowledge and introduces extra storage overhead. Online sampling conducts ad-hoc sampling when queries arrive [1, 8, 20, 32, 33]. It does not require prior knowledge, but has high latency. Our approach can efficiently generate samples according to models with query semantics.

Data-scanning sampling-based AQP join approaches. Sampling over join for AQP is a challenging problem [3, 12, 18, 23, 46, 69]. Ripple join [18] is an adaptive join algorithm for online aggregation. Wander join [23] designs an efficient algorithm that performs random walks to yield a random sample of the join output with pre-built indexes. Our method generates samples from the proposed Table-CVAE model. Quickr [20] handles AQP by injecting a sampling operator that can sample multiple tables. Our method can provide more accurate answers for join queries.

Machine learning Model-based AQP approaches. Machine learning has shown its potential in addressing many challenges in the database area, such as knobs tuning [25, 65], joining order selection [29, 63], query plan cost estimation [52, 61], and cardinality estimation[22, 34, 53, 54, 58–60, 62, 68]. Techniques like [54] for learned cardinality estimation of joins with correlations can be extended for simple AQP. DeepGen [55] employs deep generative models to answer aggregate queries by generating samples from the models. However, it does not consider the query semantics during sample generation. Similar works [47] [40] [45] [66] [67] also adopt machine learning models to estimate the query results.

DBEst [28] transforms aggregation queries into integrals and computes them with density estimators and regression models. Further, DBEst++ [27] extends DBEst and uses mixture density networks well. DeepDB [19] is a deep probability model-based AQP method that supports join queries. But, it requires that the tables must be joined on primary keys and foreign keys.

Sample size allocation for AQP. Sampling has been widely used in AQP [2, 4, 11, 15, 31, 44]. For group-by queries, congressional sampling [2] combines house sampling and senate sampling. STRAT [11] builds sample synopses to minimize the ℓ_2 norm of expected relative error of the whole query workload. However, little work considers the for join query sample size allocation at attribute level.

8 CONCLUSION AND FUTURE WORK

We propose ShadowAQP, an approach based on attribute-oriented sample size allocation and data generation. Experimental results show that it improves query latency and query accuracy performance compared with SOTA ones. In the future, we plan to improve other typical DB workloads by using AIGC techniques.

ACKNOWLEDGMENTS

This work is funded in part by the National Natural Science Foundation of China (No.62072230, 62272223, U22A2031), Jiangsu Province Science and Technology Key Program (No.BE2021729), Fundamental Research Funds for the Central Universities (No.020214380089, 020214380098), and the Collaborative Innovation Center of Novel Software Technology and Industrialization. Haipeng Dai, Rong Gu, Meng Li and Guihai Chen are the corresponding authors.

REFERENCES

[1] Swarup Acharya, Phillip B. Gibbons, and Viswanath Poosala. 1999. Aqua: A Fast Decision Support Systems Using Approximate Query Answers. In *Proceedings of the 25th VLDB International Conference on Very Large Data Bases*. Morgan Kaufmann, 754–757.

[2] Swarup Acharya, Phillip B. Gibbons, and Viswanath Poosala. 2000. Congressional Samples for Approximate Answering of Group-By Queries. In *Proceedings of the 19th ACM International Conference on Management of Data*. ACM, 487–498.

[3] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. 1999. Join Synopses for Approximate Query Answering. In *Proceedings of the 18th ACM International Conference on Management of Data*. ACM, 275–286.

[4] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 29–42.

[5] Shun-ichi Amari. 1993. Backpropagation and stochastic gradient descent method. *Neurocomputing* 5, 3 (1993), 185–196.

[6] Vladimir Braverman and Rafail Ostrovsky. 2013. Generalizing the Layering Method of Indyk and Woodruff: Recursive Sketches for Frequency-Based Vectors on Streams. In *Proceedings of the 16th Approximation, Randomization, and Combinatorial Optimization*. Springer, 58–70.

[7] Lorenzo Bruzzone and Diego Fernández-Prieto. 1999. An incremental-learning neural network for the classification of remote-sensing images. *Pattern Recognition Letters* 20, 11-13 (1999), 1241–1248.

[8] Yang Cao and Wenfei Fan. 2017. Data Driven Approximation with Bounded Resources. *Proceedings of the VLDB Endowment* 10, 9 (2017), 973–984.

[9] Census. 1996. Census Income Data Set. Retrieved February 17, 2023 from https://archive.ics.uci.edu/ml/datasets/Census+Income

[10] Kaushik Chakrabarti, Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. 2000. Approximate Query Processing Using Wavelets. In *Proceedings of the 26th International Conference on Very Large Data Bases*. Morgan Kaufmann, 111–122.

[11] Surajit Chaudhuri, Gautam Das, and Vivek R. Narasayya. 2007. Optimized stratified sampling for approximate query processing. *ACM Transactions on Database Systems* 32, 2 (2007), 9–es.

[12] Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. 1999. On Random Sampling over Joins. In *Proceedings of the 18th ACM International Conference on Management of Data*. ACM, 263–274.

[13] Xingguang Chen and Sibo Wang. 2021. Efficient Approximate Algorithms for Empirical Entropy and Mutual Information. In *Proceedings of the 40th ACM International Conference on Management of Data*. ACM, 274–286.

[14] Yu Chen and Ke Yi. 2017. Two-Level Sampling for Join Size Estimation. In *Proceedings of the 36th ACM International Conference on Management of Data*. ACM, 759–774.

[15] Bolin Ding, Silu Huang, Surajit Chaudhuri, Kaushik Chakrabarti, and Chi Wang. 2016. Sample + Seek: Approximating Aggregates with Distribution Precision Guarantee. In *Proceedings of the 35th ACM International Conference on Management of Data*. ACM, 679–694.

[16] Philipp Eichmann, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. 2020. IDEBench: A Benchmark for Interactive Data Exploration. In *Proceedings of the 39th ACM International Conference on Management of Data*. ACM, 1555–1569.

[17] Flights. 2013. Flights and Airports Data. Retrieved February 18, 2023 from http://www.transtats.bts.gov

[18] Peter J. Haas and Joseph M. Hellerstein. 1999. Ripple Joins for Online Aggregation. In *Proceedings of the 18th ACM International Conference on Management of Data*. ACM, 287–298.

[19] 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 (2020), 992–1005.

[20] Srikanth Kandula, Anil Shanbhag, Aleksandar Vitorovic, Matthaios Olma, Robert Grandl, Surajit Chaudhuri, and Bolin Ding. 2016. Quickr: Lazily Approximating Complex AdHoc Queries in BigData Clusters. In *Proceedings of the 35th ACM International Conference on Management of Data*. ACM, 631–646.

[21] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[22] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. 2019. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In *Proceedings of the 9st Conference on Innovative Data Systems Research*. CIDR Conference, 1–8.

[23] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. 2016. Wander Join: Online Aggregation via Random Walks. In *Proceedings of the 35th ACM International Conference on Management of Data*. ACM, 615–629.

[24] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. 2019. Wander Join and XDB: Online Aggregation via Random Walks. *ACM Transactions on Database Systems* 44, 1 (2019), 2:1–2:41.

[25] Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. 2019. QTune: A Query-Aware Database Tuning System with Deep Reinforcement Learning. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2118–2130.

[26] Kaiyu Li, Yong Zhang, Guoliang Li, Wenbo Tao, and Ying Yan. 2019. Bounded Approximate Query Processing. *IEEE Transactions on Knowledge and Data Engineering* 31, 12 (2019), 2262–2276.

[27] Qingzhi Ma, Ali Mohammadi Shanghooshabad, Mehrdad Almasi, Meghdad Kurmanji, and Peter Triantafillou. 2021. Learned Approximate Query Processing: Make it Light, Accurate and Fast. In *Proceedings of the 11st Conference on Innovative Data Systems Research*. CIDR Conference, 1–11.

[28] Qingzhi Ma and Peter Triantafillou. 2019. DBEst: Revisiting Approximate Query Processing Engines with Machine Learning Models. In *Proceedings of the 38th ACM International Conference on Management of Data*. ACM, 1553–1570.

[29] Ryan Marcus and Olga Papaemmanouil. 2018. Deep Reinforcement Learning for Join Order Enumeration. In *Proceedings of the 1st International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. ACM, 3:1–3:4.

[30] Barzan Mozafari, Jags Ramnarayan, Sudhir Menon, Yogesh Mahajan, Soubhik Chakraborty, Hemant Bhanawat, and Kishor Bachhav. 2017. SnappyData: A Unified Cluster for Streaming, Transactions and Interactice Analytics. In *Proceedings of the 8th Conference on Innovative Data Systems Research*. CIDR Conference, 1–8.

[31] Trong Duc Nguyen, Ming-Hung Shih, Sai Sree Parvathaneni, Bojian Xu, Divesh Srivastava, and Srikanta Tirthapura. 2020. Random Sampling for Group-By Queries. In *Proceedings of the 36th IEEE International Conference on Data Engineering*. IEEE, 541–552.

[32] Frank Olken. 1993. *Random Sampling from Databases*. Ph.D. Dissertation. University of California at Berkeley.

[33] Matthaios Olma, Odysseas Papapetrou, Raja Appuswamy, and Anastasia Ailamaki. 2019. Taster: Self-Tuning, Elastic and Online Approximate Query Processing. In *Proceedings of the 35th IEEE International Conference on Data Engineering*. IEEE, 482–493.

[34] Jennifer Ortiz, Magdalena Balazinska, Johannes Gehrke, and S. Sathiya Keerthi. 2018. Learning State Representations for Query Optimization with Deep Reinforcement Learning. In *Proceedings of the 2nd ACM Workshop on Data Management for End-To-End Machine Learning*. ACM, 1–4.

[35] Yongjoo Park, Michael J. Cafarella, and Barzan Mozafari. 2016. Visualization-aware sampling for very large databases. In *Proceedings of the 32nd IEEE International Conference on Data Engineering*. IEEE, 755–766.

[36] Yongjoo Park, Barzan Mozafari, Joseph Sorenson, and Junhao Wang. 2018. VerdictDB: Universalizing Approximate Query Processing. In *Proceedings of the 37th ACM International Conference on Management of Data*. ACM, 1461–1476.

[37] Jinglin Peng, Dongxiang Zhang, Jiannan Wang, and Jian Pei. 2018. AQP++: Connecting Approximate Query Processing With Aggregate Precomputation for Interactive Analytics. In *Proceedings of the 37th ACM International Conference on Management of Data*. ACM, 1477–1492.

[38] Gregory Piatetsky-Shapiro and Charles Connell. 1984. Accurate estimation of the number of tuples satisfying a condition. *ACM SIGMOD Record* 14, 2 (1984), 256–276.

[39] Robi Polikar, L. Upda, S. S. Upda, and Vasant G. Honavar. 2001. Learn++: an incremental learning algorithm for supervised neural networks. *IEEE Transactions on Systems, Man, and Cybernetics* 31, 4 (2001), 497–508.

[40] Vibhor Porwal, Subrata Mitra, Fan Du, John Anderson, Nikhil Sheoran, Anup B. Rao, Tung Mai, Gautam Kowshik, Sapthotharan Nair, Sameeksha Arora, and Saurabh Mahapatra. 2022. Efficient Insights Discovery through Conditional Generative Model based Query Approximation. In *Proceedings of the 41st ACM International Conference on Management of Data*. ACM, 2397–2400.

[41] Sajjadur Rahman, Maryam Aliakbarpour, Ha Kyung Kong, Eric Blais, Karrie Karahalios, Aditya Parameswaran, and Ronitt Rubinfield. 2017. I've Seen "Enough": Incrementally Improving Visualizations to Support Rapid Decision Making. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1262–1273.

[42] Carl Edward Rasmussen. 1999. The Infinite Gaussian Mixture Model. In *Proceedings of the 12th MIT Advances in Neural Information Processing Systems*. The MIT Press, 554–560.

[43] Nir Regev, Lior Rokach, and Asaf Shabtai. 2021. Approximating Aggregated SQL Queries with LSTM Networks. In *Proceedings of the 25th International Joint Conference on Neural Networks*. IEEE, 1–8.

[44] Philipp Rösch and Wolfgang Lehner. 2009. Sample synopses for approximate answering of group-by queries. In *Proceedings of the 12th International Conference on Extending Database Technology*. ACM, 403–414.

[45] Fotis Savva, Christos Anagnostopoulos, and Peter Triantafillou. 2020. ML-AQP: Query-Driven Approximate Query Processing based on Machine Learning. *arXiv preprint arXiv:2003.06613* (2020).

[46] Michael Shekelyan, Graham Cormode, Peter Triantafillou, Ali Mohammadi Shanghooshabad, and Qingzhi Ma. 2022. Weighted Random Sampling over Joins. *arXiv preprint arXiv:2201.02670* (2022).

[47] Nikhil Sheoran, Subrata Mitra, Vibhor Porwal, Siddharth Ghetia, Jatin Varshney, Tung Mai, Anup B. Rao, and Vikas Maddukuri. 2022. Conditional Generative Model Based Predicate-Aware Query Approximation. In *36th AAAI Conference on Artificial Intelligence*. AAAI Press, 8259–8266.

[48] Shigetoshi Shiotani, Toshio Fukuda, and Takanori Shibata. 1995. A neural network architecture for incremental learning. *Neurocomputing* 9, 2 (1995), 111–130.

[49] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. 2015. Learning Structured Output Representation using Deep Conditional Generative Models. In *Proceedings of the 29th Annual Conference on Neural Information Processing Systems*. ACM, 3483–3491.

[50] Shaoxu Song, Fei Gao, Ruihong Huang, and Yihan Wang. 2021. On Saving Outliers for Better Clustering over Noisy Data. In *Proceedings of the 40th ACM International Conference on Management of Data*. 1692–1704.

[51] Hong Su, Mohamed Zaït, Vladimir Barrière, Joseph Torres, and Andre Cavalheiro Menck. 2016. Approximate Aggregates in Oracle 12C. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management*. ACM, 1603–1612.

[52] Ji Sun and Guoliang Li. 2019. An End-to-End Learning-based Cost Estimator. *Proceedings of the VLDB Endowment* 13, 3 (2019), 307–319.

[53] Ji Sun, Guoliang Li, and Nan Tang. 2021. Learned Cardinality Estimation for Similarity Queries. In *Proceedings of the 40th ACM International Conference on Management of Data*. ACM, 1745–1757.

[54] Ji Sun, Jintao Zhang, Zhaoyan Sun, Guoliang Li, and Nan Tang. 2021. Learned Cardinality Estimation: A Design Space Exploration and A Comparative Evaluation. *Proceedings of the VLDB Endowment* 15, 1 (2021), 85–97.

[55] Saravanan Thirumuruganathan, Shohedul Hasan, Nick Koudas, and Gautam Das. 2020. Approximate Query Processing for Data Exploration using Deep Generative Models. In *Proceedings of the 36th IEEE International Conference on Data Engineering*. IEEE, 1309–1320.

[56] TPC-DS. 2006. TPC-DS Benchmark. Retrieved January 15, 2023 from http://www.tpc.org/tpcds/

[57] TPC-H. 2003. TPC-H Benchmark. Retrieved January 10, 2023 from http://www.tpc.org/tpch/

[58] Xiaoying Wang, Changbo Qu, Weiyuan Wu, Jiannan Wang, and Qingqing Zhou. 2021. Are We Ready For Learned Cardinality Estimation? *Proceedings of the VLDB Endowment* 14, 9 (2021), 1640–1654.

[59] Xiaoyang Wang, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Wei Wang. 2014. Selectivity Estimation on Streaming Spatio-Textual Data Using Local Correlations. *Proceedings of the VLDB Endowment* 8, 2 (2014), 101–112.

[60] Yaoshu Wang, Chuan Xiao, Jianbin Qin, Rui Mao, Makoto Onizuka, Wei Wang, Rui Zhang, and Yoshiharu Ishikawa. 2021. Consistent and Flexible Selectivity Estimation for High-Dimensional Data. In *Proceedings of the 40th ACM International Conference on Management of Data*. ACM, 2319–2327.

[61] Peizhi Wu and Gao Cong. 2021. A Unified Deep Model of Learning from both Data and Queries for Cardinality Estimation. In *Proceedings of the 40th ACM International Conference on Management of Data*. ACM, 2009–2022.

[62] Ziniu Wu, Parimarjan Negi, Mohammad Alizadeh, Tim Kraska, and Samuel Madden. 2022. FactorJoin: A New Cardinality Estimation Framework for Join Queries. *arXiv preprint arXiv:2212.05526* (2022).

[63] Xiang Yu, Guoliang Li, Chengliang Chai, and Nan Tang. 2020. Reinforcement Learning with Tree-LSTM for Join Order Selection. In *Proceedings of the 36th IEEE International Conference on Data Engineering*. IEEE, 1297–1308.

[64] Kai Zeng, Shi Gao, Jiaqi Gu, Barzan Mozafari, and Carlo Zaniolo. 2014. ABS: a system for scalable approximate queries with accuracy guarantees. In *Proceedings of the 33rd ACM International Conference on Management of Data*. ACM, 1067–1070.

[65] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, Minwei Ran, and Zekang Li. 2019. An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. In *Proceedings of the 38th ACM International Conference on Management of Data*. ACM, 415–432.

[66] Meifan Zhang and Hongzhi Wang. 2021. Approximate Query Processing for Group-By Queries based on Conditional Generative Models. *arXiv preprint arXiv:2101.02914* (2021).

[67] Meifan Zhang and Hongzhi Wang. 2021. LAQP: Learning-based approximate query processing. *Information Sciences* 54, 6 (2021), 1113–1134.

[68] Kangfei Zhao, Jeffrey Xu Yu, Zongyan He, and Hao Zhang. 2021. Uncertainty-aware Cardinality Estimation by Neural Network Gaussian Process. *arXiv preprint arXiv:2107.08706* (2021).

[69] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. 2018. Random Sampling over Joins Revisited. In *Proceedings of the 37th ACM International Conference on Management of Data*. ACM, 1525–1539.