Conditional Generative Model based Predicate-Aware Query Approximation

Nikhil Sheoran[1*], Subrata Mitra[2†], Vibhor Porwal[2], Siddharth Ghetia[3*], Jatin Varshney[3*], Tung Mai[2], Anup Rao[2], Vikas Maddukuri[3*]

[1] University of Illinois at Urbana-Champaign [2] Adobe Research [3] Indian Institute of Technology, Roorkee

Abstract

The goal of Approximate Query Processing (AQP) is to provide very fast but "accurate enough" results for costly aggregate queries thereby improving user experience in interactive exploration of large datasets. Recently proposed Machine-Learning-based AQP techniques can provide very low latency as query execution only involves model inference as compared to traditional query processing on database clusters. However, with increase in the number of filtering predicates (`WHERE` clauses), the approximation error significantly increases for these methods. Analysts often use queries with a large number of predicates for insights discovery. Thus, maintaining low approximation error is important to prevent analysts from drawing misleading conclusions. In this paper, we propose ELECTRA, a predicate-aware AQP system that can answer analytics-style queries with a large number of predicates with much smaller approximation errors. ELECTRA uses a conditional generative model that learns the conditional distribution of the data and at run-time generates a small ($\approx$ 1000 rows) but representative sample, on which the query is executed to compute the approximate result. Our evaluations with four different baselines on three real-world datasets show that ELECTRA provides lower AQP error for large number of predicates compared to baselines.

1 Introduction

Interactive exploration and visualization tools, such as Tableau, Microsoft Power BI, Qlik, Polaris (Stolte, Tang, and Hanrahan 2002), and Vizdom (Crotty et al. 2015) have gained popularity amongst data-analysts. One of the desirable properties of these tools is that the speed of interaction with the data, i.e., the queries and the corresponding visualizations must complete at *"human speed"* (Crotty et al. 2016) or at *"rates resonant with the pace of human thought"* (Liu and Heer 2014).

To reduce the latency of such interactions, Approximate Query Processing (AQP) techniques that can provide fast but "accurate enough" results for queries with aggregates (AVG, SUM, COUNT) on numerical attributes on a large dataset, have recently gained popularity (Hilprecht et al. 2020; Thirumuruganathan et al. 2020; Ma and Triantafillou 2019). Prior AQP systems mainly relied on sampling-based methods. For example, BlinkDB (Agarwal et al. 2013) and recently proposed VerdictDB (Park et al. 2018b), first create uniform/stratified samples and store them. Then at run-time, they execute the queries against these stored samples thus reducing query latency. In order to reduce error for different set of queries — using different columns for filters and groupings — they need to create not one, but multiple sets of stratified samples of the same dataset. This results in a significant storage overhead. Recently, following the promising results (Mondal, Sheoran, and Mitra 2021; Mirhoseini et al. 2017; Kraska et al. 2018) of using Machine Learning (ML) to solve several systems problems, few ML-based AQP techniques have been proposed. These use either special data-structures (DeepDB (Hilprecht et al. 2020)) or simple generative models (VAE-AQP (Thirumuruganathan et al. 2020)) to answer queries with lower latency.

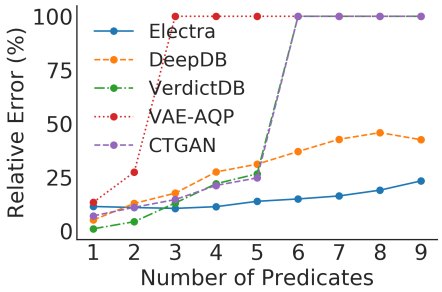


Figure 1: Median relative error vs. no. of predicates. Baselines have high error for important high-predicate queries.

However, a key drawback of these techniques is that as the number of predicates (i.e. `WHERE` conditions) in the query increases, the approximation error significantly increases (Fig. 1 for Flights dataset). ELECTRA is our technique shown in *blue*. The importance of the queries containing such large number of predicates is disproportionally high. This is because, as expert analysts drill down for insights, they progressively use more and more predicates to filter the data to arrive at the desired subset and breakdowns where some interesting patterns might be present. Therefore, lowering the approximation error for such high-predicate queries can prevent wrong conclusions, after laborious dis-

[*] Work done while at Adobe Research

[†] Corresponding Author

section of data in search of insights. As shown in Fig. 1, all baselines fail to achieve this goal and therefore can be detrimental for complex insight discovery tasks.

In this paper, we present ELECTRA, a predicate-aware AQP system that can answer analytics-style queries having a large number of predicates with much lower approximation errors compared to the state-of-the-art techniques. The key novelty of ELECTRA comes from the use of a *predicate-aware generative model* for AQP and associated training methodology to capture accurate conditional distribution of the data even for highly selective queries. ELECTRA then uses the conditional generative model (Sohn, Lee, and Yan 2015) to generate a few representative samples at runtime for providing an accurate enough answer.

The attributes (columns) in a tabular dataset can have categories (groups) that are rare. Moreover, when a large number of predicates are used to filter the data, the number of rows in the original data satisfying such conditions can be very small. If we cannot faithfully generate such rare categories with proper statistical characteristics, AQP results would either have large errors or even miss entire groups for queries using a `GROUP BY`. To overcome this problem, we propose a *stratified masking strategy* while training the CVAE and an associated modification of the CVAE loss-function to help it learn predicate-aware targeted generation of samples. Specifically, using this proposed masking strategy, we help the model to learn rare groups or categories in the data. Our empirical evaluations on real-world datasets show that, our stratified masking strategy helps ELECTRA to learn the *conditional distribution* among different attributes, in a query-agnostic manner and can provide an average 12% improvement in accuracy (Fig. 6) compared to *random* masking strategy proposed by prior work (Ivanov, Figurnov, and Vetrov 2019) (albeit, not for AQP systems but for image generation). Our major contributions in this paper are:

- We propose a novel and low-error approximate query processing technique using predicate-aware generative models. Even for queries with a large number of predicates, our technique can provide effective answers by generating a few representative samples, at the client-side.
- We propose a novel *stratified-masking* technique that enables training the generative model to learn better conditional distribution and support predicate-aware AQP resulting in a significant performance improvement on low selectivity queries.
- We present the end-to-end design and implementation details of our AQP system called ELECTRA. We present detailed evaluation and ablation study with real-world datasets and compare with state-of-the-art baselines. Our technique reduces AQP error by 36.6% on average compared to previously proposed generative-model-based AQP technique on production workloads.

2 Overview of ELECTRA

2.1 Target Query Structure

The commonly used queries for interactive data exploration and visual analytics have the following structure:

SELECT Ψ, **AGG**(N) as α **FROM** T **WHERE** θ [**GROUP BY** C] [**ORDER BY** β]

Aggregate AGG (`AVG`, `SUM` etc.) is applied to a numerical attribute N, on rows satisfying predicate θ. `GROUP BY` is optionally applied on categorical attributes C. One such query example is shown in Fig. 2b. Here an analyst might be exploring what are the average checkout values made by users from a certain demographic (Age "Senior" using "iOS" and "Chrome" Browser) from an e-commerce site across different months. Insights like these helps organizations to understand trends, optimize website performance or to place effective advertisements. ELECTRA can handle a much diverse set of queries including queries with multiple predicates by - breaking down the query into multiple queries; using inclusion-exclusion principle or converting the query's predicates to Disjunctive Normal Form.

Predicates: Predicates are condition expressions (e.g., `WHERE`) evaluating to a boolean value used to filter rows on which an aggregate function like `AVG`, `SUM` and `COUNT` is to be evaluated. A compound predicate consists of multiple (`AND`s) or (`OR`s) of predicates.

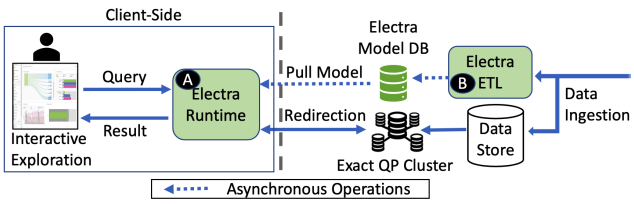
Query Selectivity: The *selectivity* of a query with predicate π is defined as $sel(\pi) := |x \in T : \pi(x) = 1|/|T|$. It indicates what fraction of rows passed the filtering criteria used in the predicates.

2.2 Design of ELECTRA

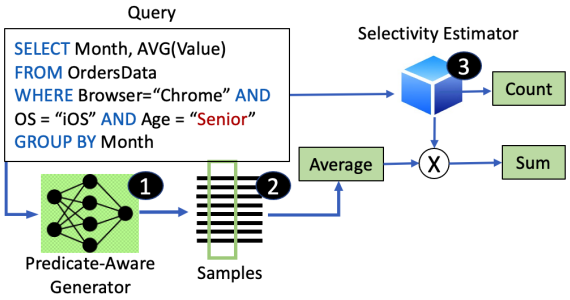
Fig. 2a shows the major components of ELECTRA (marked in green). At a high-level, it has primarily two components: (A) an on-line AQP runtime at the client-side and, (B) an offline server-side ETL (Extraction, Transformation, and Loading) component. The runtime component (A) is responsible for answering the live queries from the user-interactions using the pre-trained ML-models. It pulls these models asynchronously from a repository (Model-DB) and caches them locally. ELECTRA is designed to handle the most frequent-type of queries for exploratory data analytics (Refer § 2.1). Unsupported query-formats are *redirected* to a backend query-processor.

Fig. 2b shows more details of the runtime component. ELECTRA first parses the query to extract individual predicates, `GROUP BY` attributes, and the aggregate function. If the aggregate is either a `SUM` or an `AVG`, ELECTRA feeds the predicates and the `GROUP BY` information to its predicate-aware ML-model (1) and generates a small number of representative samples. For queries with `AVG` as an aggregate, ELECTRA can directly calculate the result by applying the query logic on these generated samples (2). For `SUM` queries, ELECTRA needs to calibrate the result using an appropriate scale-up or denormalizing factor. ELECTRA calculates this factor by feeding the filtering predicates to a *selectivity estimator* (3). For `COUNT` queries, ELECTRA directly uses the selectivity estimator to calculate the query result.

The server-side ETL component ((B) in Fig. 2a) operates asynchronously. For a new dataset, it trains two types of models – a conditional generative model and a selectivity estimator model – and stores them into the Model-DB.



(a) Overall architecture of ELECTRA



(b) ELECTRA Runtime computations

Figure 2: Design of ELECTRA

3 Conditional Sample Generation

In this section, we describe how ELECTRA attempts to make conditional sample generation process accurate, even for queries with large number of predicates, so that the resulting approximation error on these samples becomes very small.

3.1 Problem Description

Let T be a relation with $K = |A|$ attributes where $A = A_N \cup A_C$. A_N denotes the numerical attributes and A_C denotes the categorical attributes. Our aim is to learn a model that can generate a predicate-aware representative sample of T at runtime. This problem can be divided into two sub-problems – first, learning the conditional distributions of the form $P(A_n|A_c)$ where A_n is a subset of A_N and A_c is a subset of A_C and second, generating representative samples using the learned conditional distributions. With respect to the example query in Fig. 2b, the attribute `Value` belongs to A_N. The attributes `Browser`, `OS`, `Age` and `Month` used in the predicates are categorical attributes and belongs to A_C.

ELECTRA targets to support queries with arbitrary number of predicate combinations over the attributes A_C using only a single model. This requires the learning technique to be *query-agnostic* (but yet, sample generation to be *query-predicate specific*).

3.2 ELECTRA's Model Design

We are the first to propose the use of Conditional Variational Auto-encoders (CVAE) (Sohn, Lee, and Yan 2015) along with a query-agnostic training methodology involving a novel masking strategy to reduce the AQP error.

Consider a K-dimensional vector a with numerical and categorical values representing a row of data. Let $m \in \{0,1\}^K$ be a binary mask denoting the set of unobserved ($m_i = 1$) and observed attributes ($m_i = 0$). Applying this mask on the vector a gives us $a_m = \{a_{i:m_i=1}\}$ as the set of unobserved attributes and $a_{1-m} = \{a_{i:m_i=0}\}$ as the set of observed attributes. Our goal is to model the conditional distribution of the unobserved a_m given the mask m and the observed attributes a_{1-m} i.e. approximate the true distribution $p(a_m|a_{1-m}, m)$.

Along with providing full support over the domain of m (arbitrary conditions), we need to ensure that the samples that are generated represent rows from the original dataset. For example: given a set of observed attributes a_{1-m} whose combination does not exist in the data, the model should not generate samples of data satisfying a_{1-m} (since a *false positive*). To avoid this, we learn to generate complete a (both a_m and a_{1-m}) by learning the distribution $p(a|a_{1-m}, m)$ instead of just learning $p(a_m|a_{1-m}, m)$ and hence avoiding generating *false positives*.

Note that the generative models based on VAE(s) (Kingma and Welling 2014) allows us to generate random samples by generating latent vector z and then sampling from posterior $p_\theta(x|z)$. But, we do not have any control on the data generation process. Conditional Variational Auto-encoder models (CVAE) (Sohn, Lee, and Yan 2015) counter this by conditioning the latent variable and data generation on a control variable. In terms of the table attributes a, the control variable a_{1-m} and the mask m, the training objective for our model thus becomes:

$$\mathcal{L}_{\text{ELECTRA}} = -KL(q_\phi(z|a,m)||p_\psi(z|a_{1-m},m)) + \mathbb{E}_{z\sim q_\phi(z|a,m)}[log(p_\theta(a|z))] \quad (1)$$

The loss function is a combination of two terms: (1) KL-divergence of the posterior and the conditional distribution of the latent space i.e. how well is your conditional latent space generation and (2) Reconstruction error for the attributes i.e. how well is your actual sample generation.

The generative process of our model thus becomes: (1) generate latent vector conditioned on the mask and the observed attributes: $p_\psi(z|a_{1-m}, m)$ and (2) generate sample vector a from the latent variable z: $p_\theta(a|z)$, thus inducing the following distribution:

$$p_{\psi,\theta}(a|a_{1-m}, m) = \mathbb{E}_{z\sim p_\psi(z|a_{1-m},m)} p_\theta(a|z) \quad (2)$$

Since a_m and a_{1-m} can be variable length vectors depending on m, inspired from (Ivanov, Figurnov, and Vetrov 2019), we consider $a_m = a \circ m$ i.e. the element-wise product of a and m. Similarly, $a_{1-m} = a \circ (1-m)$.

In terms of a query, the observed attributes are the predicates present in the query. E.g. for the query in Fig. 2b, for attributes A = {`Month, Value, Browser, OS, Age`}, m is $\{1,1,0,0,0\}$ and the observed attributes vector a_{1-m} is {`-, -, "Chrome", "iOS", "Senior"`}.

Model Architecture: ELECTRA's CVAE model consists of three networks - (1) Encoder: to approximate the true posterior distribution, (2) Prior: to approximate the conditional latent space distribution, and (3) Decoder: to generate synthetic samples based on the latent vector z. We use a Gaussian distribution over z with parameters (μ_ϕ, σ_ϕ) and (μ_ψ, σ_ψ) for the encoder and prior network respectively. Fig. 3a shows the model architecture. All the three networks

are used during training, whereas during runtime, only the prior and decoder networks are used. Fig. 3b shows the inputs to these networks during training, corresponding to our example dataset mentioned in Fig. 2b. The input to the Prior network is masked using a novel masking strategy. Fig. 3c illustrates the runtime inputs to ELECTRA's generative model for our example query. It also illustrates the generated samples as obtained from the Decoder network, corresponding to the input query. After generation of such targeted samples that preserve the conditional distribution of the original data in the context of the query received at runtime, ELECTRA simply executes the original query on this very small sample to calculate an answer.

3.3 Stratified Masking

The efficiency of learned conditional distributions depends on the set of masks $M = \{m\}$ used during the training. During each epoch, a mask $m_{b\times k}$ is generated for a batch of data. This mask is applied to the input data to generate the set of observed attributes, for which the samples are generated and the loss function evaluated. In case of $m = 0^K$ i.e. no masking and all the attributes are observed, the prior network would learn to generate latent vectors corresponding to all observed attributes. But at runtime, only a partial set of attributes are observed (as predicates) and hence it would lead to a poor performance. In case of $m = 1^K$ i.e. a masking rate of 100% and all the features are unobserved, the behavior would be similar to that of a VAE. Since the vector $a_{1-m} = \{\phi\}$, the conditioning of the latent vector would have no effect on the learning. Thus, again this would perform poorly on queries with predicates.

In order to counter these and make the training query-agnostic, a random masking strategy (Ivanov, Figurnov, and Vetrov 2019) can be used where a certain fraction of the rows are masked corresponding to each of the input features. However, such random masking does not provide any mechanism to help the model learn better the conditional distribution for rare groups to improve performance for queries with large number of predicates. There are more subtleties. By masking more number of rows for an attribute, the training helps the model to learn better the generation of representative values for that attribute. Whereas, by keeping more unmasked rows, the model can learn better conditional distribution conditioned on those attributes.

Proposed masking strategy: With the above observation, ELECTRA uses a novel masking strategy tailored for AQP. It consists of the following: (a) We completely mask the numerical attributes. (b) For the categorical attributes on which conditions can be used in the predicates, we use a *stratified masking strategy*. In the stratified masking we alter the random masking strategy based on the size of the strata (i.e. groups or categories) to disproportionately reduce the probability of masking for the rare groups. Thus, the ability of the model to learn better conditional distribution improves. We apply this masking strategy in a batch-wise manner during training. In the Evaluation section, we show (Fig. 6) that our masking strategy drastically reduces the approximation error compared to no-masking or random masking strategies.

Algorithm 1 describes the masking strategy where we input a batch of data B of dimension $b \times k$: b denotes the batch size and k denotes the number of attributes. The numerical columns are completely masked. The strata size for each categorical column i (S_i), contains for each value that the column i can take, the fraction of rows in the data where the column takes this value. Using the $weights$ obtained from $S_i[value]$ as the sampling probability and r as the masking factor, we sample $r * b$ rows and mask their column i. Masking factor r controls the overall masking rate of the batch. A higher masking factor simulates queries with less observed attributes i.e. less number of predicates and a lower masking factor simulates queries with more observed attributes i.e. more number of predicates.

Algorithm 1: Stratified Masking Strategy

Input: Batch $\{B\}_{b\times k}$, Masking Factor r, Strata Size S

Output: Binary mask m for the Input Batch B

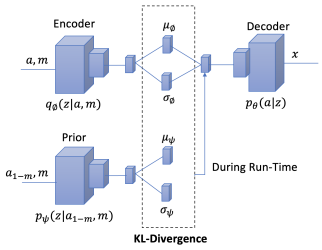
$m \leftarrow 0_{b\times k}$
$num\_cols \leftarrow numerical\_cols(k)$
$cat\_cols \leftarrow categorical\_cols(k)$
for i *in* num_cols **do**
  $\{m_{x,y}\}_{1,i}^{b,i} \leftarrow 1$
end
for $i \in cat\_cols$ **do**
  $values \leftarrow \{B_{x,y}\}_{1,i}^{b,i}$
  $weights \leftarrow \{S_i[values_x]\}_1^b$
  $batch\_indices \leftarrow$ $random.sample(\{x\}_1^b, r * b, weights)$
  for $j \in batch\_indices$ **do**
    $m_{j,i} \leftarrow 1$
  end
end
return m;

4 Implementation Details

Data Transformation. ELECTRA applies a set of data preprocessing steps including label encoding for categorical attributes and mode-specific normalization for numerical attributes. For normalization, weights for the Gaussian mixture are calculated through a variational inference algorithm using sklearn's `BayesianGaussianMixture` method. The parameters specified are the number of modes (identified earlier) and a maximum iteration of 1000. To avoid mixture components with close to zero weights, we limit the number of modes to less than or equal to 3.

Conditional Density Estimator. The conditional density estimator model is implemented in PyTorch. We build upon the code [1] provided by Ivanov et al. The density estimator is trained to minimize the combination of KL-divergence loss and reconstruction loss. To select a suitable set of hyperparameters, we performed a grid search. We varied the depth (d) of the prior and proposal networks in the range [2,4,6,8] and the latent dimension (L) in the range [32,64,128,256].

[1] VAEAC code available at https://github.com/tigvarts/vaeac



(a) Model architecture

Month	Value	Browser	OS	Age
Jan	500	Chrome	iOS	Senior
Jan	100	Chrome	Android	Youth
Jan	250	Safari	iOS	Adult
Feb	350	Chrome	Android	Youth
Feb	200	Safari	iOS	Adult
Feb	600	Chrome	iOS	Senior

Input a to Encoder

Month	Value	Browser	OS	Age
Jan	-	-	iOS	Senior
-	-	-	-	Youth
Jan	-	Safari	-	-
-	-	Chrome	Android	-
Feb	-	-	iOS	Adult
-	-	Chrome	-	Senior

Masked Input a_{1-m} to Prior

(b) Training inputs

Month	Value	Browser	OS	Age
-	-	Chrome	iOS	Senior

Masked Input a_{1-m} to Prior

Month	Value	Browser	OS	Age
Jan	475	Chrome	iOS	Senior
Jan	525	Chrome	iOS	Senior
Feb	625	Chrome	iOS	Senior

Output a from Decoder

(c) Runtime inputs

Figure 3: Model architecture and inputs

For Flights data we use $d = 8$, $L = 64$, for Housing $d = 8$, $L = 64$, and for Beijing PM2.5 we use $d = 6$, $L = 32$. Note that, the depth of the networks and the latent dimension contribute significantly to the model size. Hence, depending on the size constraints (if any), one can choose a simpler model. We used a masking factor (r) of 0.5. The model was trained with an Adam Optimizer with a learning rate of 0.0001 (larger learning rates gave unstable variational lower bound(s)).

Selectivity Estimator. We use NARU's publicly available implementation [2]. The model is trained with the ResMADE architecture with a batch size of 512, an initial warm-up of 10000 rounds, with 5 layers each of hidden dimension 256. In order to account for the impact of *column ordering*, we tried various different random orderings and chose the one with the best performance.

5 Evaluation

5.1 Experimental Environment

All the experiments were performed on a 32 core Intel(R) Xeon(R) CPU E5-2686 with 4 Tesla V100-SXM2 GPU(s).

Datasets: We use three real-world datasets: Flights (of Transportation Statistics 2019), Housing (Qiu 2018) and Beijing PM2.5 (Chen 2017). The Flights dataset consists of arrival and departure data (e.g. carrier, origin, destination, delay) for domestic flights in the USA. The Housing dataset consists of housing price records (e.g. price, area, rooms) obtained from a real-estate company. The Beijing PM2.5 dataset consists of hourly records of PM2.5 concentration. All these datasets have a combination of both categorical and numerical attributes.

Synthetic Query Workload: We create a synthetic workload to evaluate the impact of the number of predicates in the query. For each dataset, we generated queries with *# of predicates* (k) ranging from 1 to $|A_C|$. For a given k, we first randomly sample 100 combinations (with repetitions) from the set of all possible k-attribute selections. Then, for each of these selected k-attributes, we create the `WHERE` condition by assigning values based on a randomly chosen tuple from the dataset. Then, we use `AVG` as the aggregate function for each of the numerical attributes. The synthetic workload thus contains a total of $100 * |A_C| * |A_N|$ queries.

[2] NARU code available at https://github.com/naru-project/naru/

Production Query Workload: We also evaluate ELECTRA using real production queries from five Fortune 500 companies over a period of two months on a large data exploration platform. First, we select queries whose query structure is supported by ELECTRA. We filter out the repeated queries. While we were able to extract the queries, we did not have access to the actual customer data corresponding to those queries. Hence, we use the following methodology to create an equivalent query set for our evaluation. We replace each predicate in the `WHERE` clause by a random categorical attribute and a random value corresponding to that attribute. All attributes that were used in `GROUP BY` were replaced by random categorical attributes from our evaluation datasets. Then, we generate two workloads, one with `AVG` and one with `SUM` as the aggregate function on a random numerical attribute. Thus, we preserve real characteristics of the use of predicates and the grouping logic in our modified workload.

Baselines: We evaluate against four recent baselines.

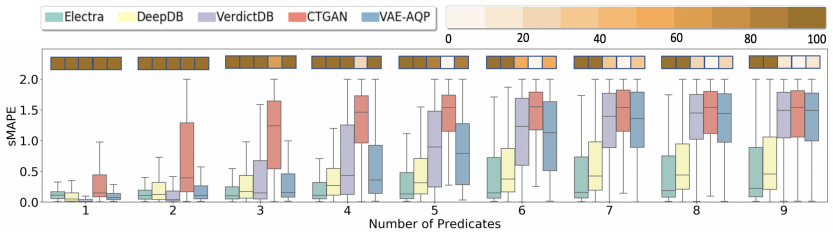
VAE-AQP (Thirumuruganathan et al. 2020) is a generative model-based technique that does not use any predicate information. The code was obtained from the authors. In our evaluations, we generated 100K samples from VAE-AQP for all 3 datasets. Note that this is well over the 1% sampling rate specified by the VAE-AQP authors for these datasets.

CTGAN (Xu et al. 2019) is a conditional generative model. However, the CTGAN technique as described in the paper (Xu et al. 2019) and the associated code (CTGAN code), only support at most one condition for generation. Hence, when using CTGAN as a baseline in our evaluation, we randomly select one of the input predicates as the condition for generating the samples. We keep the generated sample size per query same as ELECTRA.

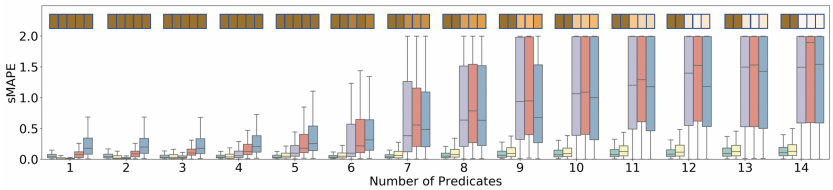
DeepDB (Hilprecht et al. 2020) builds a relational sum-product-network (SPN) based on the input data. For any incoming query, it evaluates the query on the obtained SPN network. We keep the default parameters for DeepDB code obtained from (DeepDB code).

VerdictDB (Park et al. 2018b) is a sampling-based approach that runs at the server-side. It extracts and stores samples (called SCRAMBLE) from the original data and uses it to answer the incoming queries. We keep the default parameters in the code obtained from (VerdictDB code).

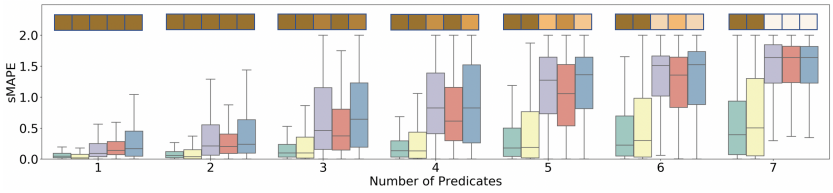
AQP Error Measures: For a query Q, with an exact ground truth result g and an approximate result a, we mea-



(a) Flights

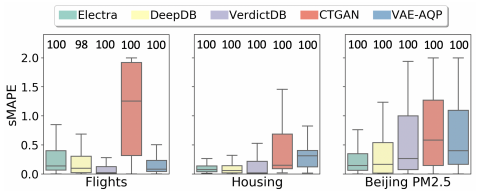


(b) Housing

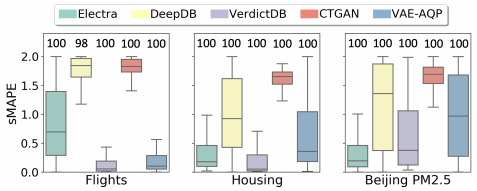


(c) Beijing PM2.5

Figure 4: AQP error vs. # predicates. Heat-map shows % of queries answered.



(a) AVG queries



(b) SUM queries

Figure 5: AQP error vs. # predicates. number on top shows % of queries answered.

Dataset	Sample size			
	500	1000	1500	2000
Flights	15.15	14.93	14.70	**14.60**
Housing	5.96	**5.91**	5.92	5.92
Beijing PM2.5	15.98	15.79	15.77	**15.64**

Table 1: Median R.E. vs. # samples generated

sure the approximation error by computing the Relative Error (R.E.). Since R.E. is an unbounded measure and can become very large, we use a bounded measure called Symmetric Mean Absolute Percentage Error (**sMAPE**) to visualize the approximation errors. sMAPE lies between 0 and 2 and provides a better visualization compared to R.E..

$$sMAPE(Q) = 2 * \frac{|g - a|}{|g| + |a|}$$

5.2 Performance w.r.t. Number of Predicates

Fig. 4 shows the distribution of approximation error (as sMAPE) w.r.t. the number of predicates in the synthetic workload queries. A lower approximation error is better. All the queries could not be successfully evaluated to produce an approximate result because the technique could not match for the used predicates either in the generated data (ELECTRA, CTGAN, VAE-AQP) or in the available samples (VerdictDB) or in the stored metadata (DeepDB). The distribution of the AQP errors in the box-plots were calculated only over the queries that the corresponding technique could support. The heat-map colors over the box-plots show what *percentages of the queries* were supported by different techniques. A technique supporting large fraction of queries is better. It is worthwhile to note, only ELECTRA and DeepDB could answer the queries with $90 - 100\%$ coverage across all the different predicate settings.

Note, the median approximation error is almost always the lowest for ELECTRA (shown in green), as the number of predicates in the queries are increased. The other two generative baselines, CTGAN and VAE-AQP perform poorly in this aspect. DeepDB's performance is great for median approximation comparison, but error at the tail is often quite large for DeepDB, compared to ELECTRA. For one predicate, almost all the techniques can maintain very low AQP error (making any of those practically useful) and ELECTRA often does not provide the lowest error in the relative terms. However, ELECTRA is the overall best solution when we look at the whole spectrum of predicates.

5.3 Performance on Production Workload

Fig. 5 shows the comparisons of the AQP error (as sMAPE) distribution for our production query workload for the three real-world datasets shown separately for `AVG` and `SUM` as the aggregate functions. For AVG as the aggregate function (Fig. 5a), ELECTRA provides the lowest approximation error for both Housing and Beijing-PM-2.5 dataset. For Flights dataset, even though the median error is comparable to DeepDB, VerdictDB and VAE-AQP, the tail error is larger. Please note: since these plots were calculated over the production queries, 70% of the queries had two or fewer predicates. Thus, the benefit of ELECTRA in handling large number of predicates is less visible. The percentages of queries that different techniques were able to successfully answer, are shown over the box-plots.

5.4 Performance on `GROUP BY` Queries

We evaluate ELECTRA's performance on `GROUP BY` queries using slight modification to our synthetic query workload by randomly choosing one of the attributes to be used for `GROUP BY`. We do not use any predicate on that

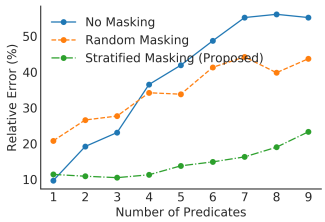


Figure 6: Masking strategy

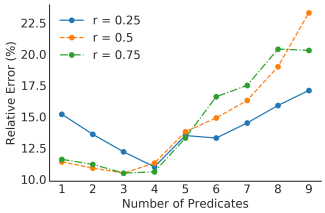


Figure 7: Masking factor

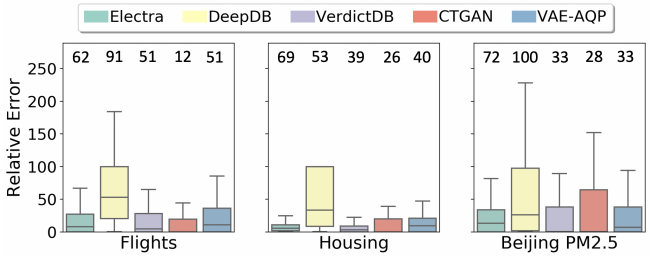


Figure 8: Error & bin-Completeness for GROUP BY queries

attribute. Fig. 8 shows the AQP error over queries with all the different number of predicates. The error for each query was calculated as an average over the individual errors corresponding to each of the *individual groups* that were both present in the ground truth results and the approximate results. The error is computed as the average of the R.E. for each of the different groups corresponding to the GROUP BY column. Approximations might result in missing groups. We use *bin-completeness* measure as follows:

$$\textit{bin-completeness} = \frac{|G_{groups} \bigcap A_{groups}|}{|G_{groups}|}$$

G_{groups} is the set of all the groups that appear in the ground truth and A_{groups} is the set for the approximate result. This *bin-completeness* (§ 5.1) is an important measure for GROUP BY queries. Fig. 8 shows the bin-completeness numbers on top of the box plots. Higher is better. Both ELECTRA and DeepDB can consistently provide much higher bin-completeness, compared to other baselines. However, the error for ELECTRA is much better than DeepDB.

5.5 Ablation Study

Samples Generated per Query. Generating more representative samples by ELECTRA can potentially improve accuracy, but at the cost of larger query latency and memory footprint. Table 1 shows the change in the R.E. for ELECTRA as we increase the number of generated samples. Generating 1000 representative samples, from the learned conditional distribution, is good enough to achieve a low AQP error, and beyond that the improvement is not significant.

Impact of the Masking Strategy. Fig. 6 quantify the benefit of our proposed masking strategy for Flights dataset. We compare the proposed method with a *No Masking* strategy where only *NaN* input values, if any, are masked. We also compare with *Random Masking* strategy that uses a uniform random masking, irrespective of the column type (Ivanov, Figurnov, and Vetrov 2019). We observe that the proposed stratified masking strategy can significantly reduce the AQP error compared to the other two alternative, irrespective of the number of predicates used in the queries.

Masking factor. Fig. 7 shows the sensitivity of ELECTRA corresponding to the masking factor, across the range of predicates for the Flights dataset. A masking factor of 0.5 would be the choice, if we want to optimize for queries with arbitrary number of predicates.

6 Related Work

Approximate Query Processing has a rich history with the use of sampling and sketching based approaches (Chaudhuri, Ding, and Kandula 2017; Cormode 2011; Chaudhuri, Das, and Narasayya 2007). BlinkDB (Agarwal et al. 2013) uses offline sampling, but assumes it has access to historical queries to reduce or optimize for storage overhead. Related works used as baselines were described in § 5.

Representative Data Generation. With the advancement of deep-learning, multiple generative methods have been proposed, primarily using GAN(s) (Brock, Donahue, and Simonyan 2019) and VAE(s) (Ivanov, Figurnov, and Vetrov 2019). These techniques have been successful in producing realistic synthetic samples for images, auto-completing prompts for texts (Radford et al. 2019; Devlin et al. 2018), generating audio wavelets (Oord et al. 2016). Few works used GAN(s) (Xu et al. 2019; Park et al. 2018a) for tabular data generation either for providing data-privacy or to overcome imbalanced data problem.

Selectivity or Cardinality Estimation includes several classical methods with synopsis structures (e.g., histograms, sketches and wavelets) to provide approximate answers (Cormode et al. 2012). Recently, learned cardinality estimation methods have become common including using Deep Autoregressive models to learn density estimates of a joint data distribution (Yang et al. 2019; Hasan et al. 2020; Yang et al. 2020), embedding and representation learning based cost and cardinality estimation (Sun and Li 2019).

7 Conclusion

We presented a deep neural network based approximate query processing (AQP) system that can answer queries using a predicate-aware generative model at client-side, without processing the original data. Our technique learns the conditional distribution of data and generates targeted samples based on the conditions specified in the query. The key contributions of the paper are the use of conditional generative models for AQP and the techniques to reduce approximation error for queries with large number of predicates.

References

Agarwal, S.; Mozafari, B.; Panda, A.; Milner, H.; Madden, S.; and Stoica, I. 2013. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *EuroSys*.

Brock, A.; Donahue, J.; and Simonyan, K. 2019. Large Scale GAN Training for High Fidelity Natural Image Synthesis. *ArXiv*, abs/1809.11096.

Chaudhuri, S.; Das, G.; and Narasayya, V. 2007. Optimized stratified sampling for approximate query processing. *ACM Transactions on Database Systems*.

Chaudhuri, S.; Ding, B.; and Kandula, S. 2017. Approximate Query Processing: No Silver Bullet. In *SIGMOD*.

Chen, S. X. 2017. UCI Machine Learning Repository.

Cormode, G. 2011. Sketch techniques for approximate query processing. *Foundations and Trends in Databases*.

Cormode, G.; Garofalakis, M.; Haas, P. J.; and Jermaine, C. 2012. Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. *Foundation and Trends in Databases*.

Crotty, A.; Galakatos, A.; Zgraggen, E.; Binnig, C.; and Kraska, T. 2015. Vizdom: Interactive Analytics through Pen and Touch. *Proc. VLDB Endow.*

Crotty, A.; Galakatos, A.; Zgraggen, E.; Binnig, C.; and Kraska, T. 2016. The Case for Interactive Data Exploration Accelerators (IDEAs). In *HILDA*.

CTGAN code. 2019. CTGAN Github Repo. https://github.com/sdv-dev/CTGAN.

DeepDB code. 2020. DeepDB Github Repo. https://github.com/DataManagementLab/deepdb-public.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Hasan, S.; Thirumuruganathan, S.; Augustine, J.; Koudas, N.; and Das, G. 2020. Deep Learning Models for Selectivity Estimation of Multi-Attribute Queries. In *SIGMOD*.

Hilprecht, B.; Schmidt, A.; Kulessa, M.; Molina, A.; Kersting, K.; and Binnig, C. 2020. DeepDB: Learn from Data, Not from Queries! *Proc. VLDB Endow.*

Ivanov, O.; Figurnov, M.; and Vetrov, D. 2019. Variational Autoencoder with Arbitrary Conditioning. In *ICLR*.

Kingma, D. P.; and Welling, M. 2014. Auto-Encoding Variational Bayes. In *ICLR*.

Kraska, T.; Beutel, A.; Chi, E. H.; Dean, J.; and Polyzotis, N. 2018. The case for learned index structures. In *SIGMOD*.

Liu, Z.; and Heer, J. 2014. The Effects of Interactive Latency on Exploratory Visual Analysis. *IEEE Transactions on Visualization and Computer Graphics*.

Ma, Q.; and Triantafillou, P. 2019. DBEst: Revisiting Approximate Query Processing Engines with Machine Learning Models. In *SIGMOD*.

Mirhoseini, A.; Pham, H.; Le, Q. V.; Steiner, B.; Larsen, R.; Zhou, Y.; Kumar, N.; Norouzi, M.; Bengio, S.; and Dean, J. 2017. Device placement optimization with reinforcement learning. In *ICML*.

Mondal, S. S.; Sheoran, N.; and Mitra, S. 2021. Scheduling of Time-Varying Workloads Using Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*.

of Transportation Statistics, B. 2019. Carrier On-Time Performance.

Oord, A. v. d.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A.; and Kavukcuoglu, K. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Park, N.; Mohammadi, M.; Gorde, K.; Jajodia, S.; Park, H.; and Kim, Y. 2018a. Data Synthesis Based on Generative Adversarial Networks. *Proc. VLDB*.

Park, Y.; Mozafari, B.; Sorenson, J.; and Wang, J. 2018b. VerdictDB: Universalizing Approximate Query Processing. In *SIGMOD*.

Qiu, Q. 2018. Housing Price in Beijing. https://www.kaggle.com/ruiqurm/lianjia/version/2.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I.; et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8): 9.

Sohn, K.; Lee, H.; and Yan, X. 2015. Learning Structured Output Representation using Deep Conditional Generative Models. In *Advances in Neural Information Processing Systems*.

Stolte, C.; Tang, D.; and Hanrahan, P. 2002. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*.

Sun, J.; and Li, G. 2019. An end-to-end learning-based cost estimator. *arXiv preprint arXiv:1906.02560*.

Thirumuruganathan, S.; Hasan, S.; Koudas, N.; and Das, G. 2020. Approximate Query Processing for Data Exploration using Deep Generative Models. In *36th IEEE ICDE*.

VerdictDB code. 2018. Verdict Project. https://github.com/verdict-project/verdict.

Xu, L.; Skoularidou, M.; Cuesta-Infante, A.; and Veeramachaneni, K. 2019. Modeling Tabular data using Conditional GAN. In *Advances in Neural Information Processing Systems*.

Yang, Z.; Kamsetty, A.; Luan, S.; Liang, E.; Duan, Y.; Chen, X.; and Stoica, I. 2020. NeuroCard: One Cardinality Estimator for All Tables. *Proc. VLDB Endow.*

Yang, Z.; Liang, E.; Kamsetty, A.; Wu, C.; Duan, Y.; Chen, X.; Abbeel, P.; Hellerstein, J. M.; Krishnan, S.; and Stoica, I. 2019. Deep Unsupervised Cardinality Estimation. *Proc. VLDB Endow.*