Approximate Query Processing using Deep Generative Models

Saravanan Thirumuruganathan††, Shohedul Hasan‡‡, Nick Koudas‡†, Gautam Das‡
‡†QCRI, HBKU; ‡University of Texas at Arlington; ‡‡University of Toronto
†sthirumuruganathan@hbku.edu.qa, ‡{shohedul.hasan@mavs, gdas@cse}.uta.edu,
‡†koudas@cs.toronto.edu

ABSTRACT
Data is generated at an unprecedented rate surpassing our ability to analyze them. One viable solution that was pioneered by the database community is Approximate Query Processing (AQP). AQP seeks to provide approximate answers to queries in a fraction of time needed for computing exact answers. This is often achieved by running the query on a pre-computed or on-demand derived sample and generating estimates for the entire dataset based on the result. In this work, we explore a novel approach for AQP utilizing deep learning (DL). We use deep generative models, an unsupervised learning based approach, to learn the data distribution faithfully in a compact manner (typically few hundred KBs). Queries could be answered approximately by generating samples from the learned model. This approach eliminates the dependency of AQP to a sample of fixed size and allows us to satisfy arbitrary accuracy requirements by generating as many samples as needed very fast. While we specifically focus on variational autoencoders (VAE), we demonstrate how our approach could also be used for other popular DL models such as generative adversarial networks (GAN) and deep Bayesian networks (DBN). Our other contributions include (a) identifying model bias and minimizing it through a rejection sampling based approach (b) an algorithm to build model ensembles for AQP for improved accuracy and (c) an analysis of VAE latent space to understand its suitability to AQP. Our extensive experiments show that deep learning is a very promising approach for AQP.

1. INTRODUCTION
Approximate Query Processing (AQP). Data driven decision making has become the dominant paradigm for businesses seeking to gain an edge over competitors. However, the unprecedented rate at which data is generated surpasses our ability to analyze them. Approximate Query Processing (AQP) is a promising technique that provides approximate answers to queries at a fraction of the cost needed to answer it exactly. AQP has numerous applications in data analytics, data exploration and data visualization where approximate results are acceptable as long as they can be obtained near-real-time.

Deep Learning for AQP. Deep Learning (DL) has become increasingly popular due to its excellent performance in many complex applications. In this paper, we consider an intriguing question: is it possible to use DL for AQP? Superficially, there exists little connection between these disparate areas. Databases seems intrinsically different from prior areas where DL has shined - such as computer vision and natural language processing. Furthermore, the task of generating approximate estimates for an aggregate query is quite different from common DL tasks such as classification or regression. However, a rigorous investigation shows that AQP using DL models is not only feasible but can be achieved in an effective and efficient manner.

1.1 Outline of Technical Results
AQP and Density Estimation. The database community has pioneered a number of effective techniques for AQP. Sampling is a popular technique that involves a careful selection of a small number of relevant tuples on which the query is executed. The result for the entire dataset is then computed using statistical estimators for the specific queries [39, 10, 17, 4]. The samples could be pre-computed in an offline manner or obtained online during query execution. Pre-computed samples serve as a non-parametric approximation of the underlying data distribution. If the samples approximate the true distribution effectively, the generated AQP estimates are often accurate. One can observe a natural size-accuracy trade-off whereby one could achieve better accuracy by increasing the sample size.

Deep Generative Models for AQP. In this paper, we advocate for an orthogonal approach based on DL that complements existing AQP techniques. Our key insight is to train a DL model to learn the data distribution of the underlying data set effectively. Once such a model is trained, it acts as a concise representation of the dataset and could be used to generate samples from the underlying data distribution. The existing AQP techniques could be transparently applied on these generated samples [29, 38]. Our proposed approach has a number of appealing properties and nicely complements the current AQP ecosystem. The learned DL model is a succinct representation of the data. It is learned in an offline manner and as we will detail it can generate as many samples as required on demand really fast. Offline sampling based approaches are tethered to a pre-computed sample that also bounds the accuracy guarantees. Moreover, if extra samples are required to reach target accuracy goals, this involves an expensive sampling step from the base data, imposing a runtime performance overhead. In contrast, our DL model based approach could be used to generate as many samples as required without the need to touch the underlying data.

The fundamental challenge is to identify a DL based distribution estimation approach that is expressive enough to
learn distributions found in real-world datasets and yet tractable and efficient to train. It must be non-parametric and not make any prior assumption about data characteristics. A large class of DL techniques - dubbed collectively as deep generative models - could be used for this purpose. Intuitively, a deep generative model is an unsupervised approach that learns the probability distribution of the dataset from a set of tuples. Often, learning the exact distribution is challenging, thus generative models learn a model that is very similar to the true distribution of the underlying data. This is often achieved through neural networks that learn a function that maps the approximate distribution to the true distribution. Each of the generative models have their respective advantages and disadvantages. For example, variational autoencoders [12] aim to learn a low dimensional latent representation of the training data that optimizes the log-likelihood of the data through evidence lower bound. Generative Adversarial Networks [21] learn the distribution as a minimax game between two components - generator (that generates data) and discriminator (that identifies if the sample is from the true distribution or not).

**Contributions.** Designing an effective DL model for AQP involves a number of challenges and design trade-offs. In this paper, we make the following major contributions.

1. **DL for AQP.** We investigate the problem of how deep learning could be used for AQP. We focus on the connection between AQP and distribution estimation and propose deep generative models as the most promising DL based approach to represent the data. Our proposed approach has a number of appealing properties and can be easily incorporated into existing AQP systems.

2. **Input Heterogeneity.** DL models are often used in domains such as computer vision and natural language processing where the input is homogeneous and can be represented as high dimensional vectors. Often, this homogeneity simplifies the training of the generative models. In contrast, databases often have a heterogeneous input where the attributes could be of different types such as categorical, integer or real-valued. Holistically treating such as discrete and continuous valued inputs creates a number of challenges that we address.

3. **Minimizing Bias through Rejection Sampling.** We investigate the problem of model bias that could occur when the DL model does not accurately learn the underlying distribution and could only approximate it. Naively generating samples from the DL model produces biased samples and thereby biased estimates. We propose a rejection sampling based approach to address model bias.

4. **DL Ensembles for AQP.** Since a DL model is an approximation of the data distribution, using multiple DL models one can learn finer properties of a distribution more effectively. We formalize this intuition by splitting the dataset into K mutually exclusive partitions and train a generative model for each of them. Specifically, we consider scenarios where the partitions must be made along pre-defined OLAP hierarchies. We formulate optimization problems and propose dynamic programming based algorithms for their solutions.

5. **AQP using Other Generative Models.** Although the bulk of our presentation focuses on variational approaches [12] due to their popularity and simplicity during training, we briefly describe how other popular deep generative models such as GANs and Bayesian networks could be used for AQP.

6. **Experiments.** We conduct extensive experiments over multiple datasets using a large diverse query workload. The results show that using DL models for AQP is both feasible and highly promising.

**Paper Outline.** We introduce background information about the AQP and deep generative models in Sections 2 and 3 respectively. In Section 4 we detail our approach on how to build variational autoencoder models for AQP and address various challenges arising while making variational autoencoders suitable for relational data including data representation and model bias. Section 5 investigates the problem of optimally splitting the dataset into K partitions and building a DL model for each of them, resulting in better approximation of the underlying dataset than the case of utilizing a single DL model. We conduct a comprehensive set of experiments validating our proposed approach in Section 6. Section 7 discusses the relevant prior work and we conclude with some promising directions for future work in 8.

**1. PRELIMINARIES**

Consider a relation $R$ with $n$ tuples and $m$ attributes $A_1, A_2, \ldots, A_m$. Given a tuple $t$ and an attribute $A_i$, we denote the value of $A_i$ in $t$ as $t[A_i]$. Let $Dom(A_i)$ be the domain of attribute $A_i$. Similar to prior work, we restrict our attention to attributes to the fundamental data types such as categorical or numerical and ignore complex attribute types such as text and/or blobs.

**Queries for AQP.** In this paper, we focus on aggregate analytic queries of the general format:

$$\text{SELECT } g, \text{AGG}(A) \text{ FROM } R \text{ WHERE filter GROUP BY G}$$

Of course, both the WHERE and GROUP BY clauses are optional. Each attribute $A_i$ could be used as a filter attribute involved in a predicate or as a measure attribute involved in an aggregate. The filter could be a conjunctive or disjunctive combination of conditions. Each of the conditions could be any relational expression of the format $A \ op \ CONST$ where $A$ is an attribute and $op$ is one of $\{=, \neq, <, >, \leq, \geq\}$. AGG could be one of the standard aggregates $\text{AVG}, \text{SUM}, \text{COUNT}$ that have been extensively studied in prior AQP literature. One could use other aggregates such as $\text{QUANTILES}$ as long as a statistical estimator exists to generate aggregate estimates.

**Performance Measures.** Let $q$ be an aggregate query whose true value is $\theta$. Let $\hat{\theta}$ be the estimate provided by the AQP system. Then, we can measure the estimation accuracy through relative error defined as

$$\text{RelErr}(q) = \frac{|\hat{\theta} - \theta|}{\theta} \quad (1)$$

For a set of queries $Q = \{q_1, \ldots, q_r\}$, the effectiveness of the AQP system could be computed through average relative
error. Let \( \theta_i \) and \( \hat{\theta}_i \) be the true and estimated value of the aggregate for query \( q_j \).

\[
\text{AvgRelErr}(Q) = \frac{1}{r} \sum_{j=1}^{r} \left| \frac{\hat{\theta}_j - \theta_j}{\theta_j} \right|
\]  

We could also use the average relative error to measure the accuracy of the estimate for GROUP BY queries. Suppose we are given a group by query \( q \) with groups \( G = \{g_1, \ldots, g_k\} \). It is possible that the sample does not contain all of these groups and the AQP system generates estimates for groups \( \{g_1, \ldots, g_k\} \) where each \( g_i \in G \). As before, let \( \theta_i \) and \( \hat{\theta}_i \) be the true and estimated value of the aggregate for group \( g_i \). By assigning 100% relative error for missing groups, the average relative error for group by queries is defined as,

\[
\text{AvgRelErr}(q) = \frac{1}{r} \left( (r - r') + \sum_{i=1}^{r'} \left| \frac{\hat{\theta}_i - \theta_i}{\theta_i} \right| \right)
\]  

3. BACKGROUND

In this section, we provide necessary background about generative models and variational autoencoders in particular, that we use for AQP.

**Generative Models.** Suppose we are given a set of data points \( X = \{x_1, \ldots, x_n\} \) that are distributed according to some unknown probability distribution \( P(X) \). The set \( X \) could be a set of images, text or tuples. Generative models seek to learn an approximate probability distribution \( Q(X) \) such that \( Q \) is very similar to \( P \). Most generative models also allow one to generate samples \( X' = \{x'_1, \ldots, x'_m\} \) from the model \( Q \) such that the \( X' \) has similar statistical properties to \( X \). Building effective generative models is a fundamental and challenging problem in Machine learning. Deep generative models often employ powerful function approximators (typically, deep neural networks) for learning to approximate the distribution.

**Variational Autoencoders (VAEs).** VAEs are a class of generative models [12, 6, 5]. VAEs have been shown to model various complicated data distributions and generate samples in domains as diverse as hand written digits [26], celebrity faces [46, 33], text [49, 7], music [47] etc. They are very efficient to train, have an interpretable latent space and could be adapted effectively to different domains such as images, text and music. In this paper, we investigate the suitability of VAE for faithfully generating data utilizing a relation R as a training basis. If successful, the trained model could be used for the purposes of answering queries approximately on the generated data as opposed to using R to answer queries approximately.

**Latent Variables.** In VAE the concept of latent variables is utilized to describe the data. Essentially latent variables are an intermediate data representation that captures data characteristics used for generative modelling. Let \( X \) be the relational data that we wish to model and \( z \) a latent variable. Let \( P(X) \) be the probability distribution from which the underlying relation was derived and \( P(z) \) as the probability distribution of the latent variable. Then \( P(X|z) \) is the distribution of generating data given latent variable.

Since the main objective is to model the underlying data distribution \( P(X) \) we can model it in relation to \( z \) as \( P(X) = \int P(X|z)P(z)dz \) marginalizing \( z \) out of the joint probability \( P(X, z) \). The challenge is that we do not know \( P(z) \) and \( P(X|z) \). The underlying idea in variational modelling is to infer \( P(z) \) using \( P(z|X) \).

**Variational Inference.** We use a method called Variational Inference (VI) to infer \( P(z|X) \) in VAE. The main idea of VI is to approach inference as an optimization problem. We model the true distribution \( P(z|X) \) using a simpler distribution (denoted as \( Q \)) that is easy to evaluate, e.g. Gaussian, and minimize the difference between those two distribution using KL divergence metric, which tells us how different \( P \) is from \( Q \). Assume we wish to infer \( P(z|X) \) using \( Q(z|X) \). The KL divergence then formulated as follows:

\[
D_{KL}[Q(z|X)||P(z|X)] = \sum_{z} Q(z|X) \log \left( \frac{Q(z|X)}{P(z|X)} \right) = E[\log(Q(z|X)) - \log(P(z|X))]
\]

With Bayes rule, we have:

\[
\begin{align*}
D_{KL}[Q(z|X)||P(z|X)] &= E[\log(Q(z|X)) - \log(P(z|X))P(z)] \\
&= E[\log(Q(z|X)) - \log(P(X|z))P(z) - \log(P(X))] \\
&= E[\log(Q(z|X)) - \log(P(X|z)) - \log(P(z))] + E[\log(P(X))] \\
&= E[\log(P(X|z))] - D_{KL}[Q(z|X)||P(z)]
\end{align*}
\]

Equation [7] is known as the variational objective. It essentially establishes a connection between \( Q(z|X) \) which is a projection of the data into the latent space and \( P(X|z) \) which generates data given a latent variable \( z \).

**Encoders and Decoders.** A different way to think of this equation is as \( Q(z|X) \) encoding the data using \( z \) as an intermediate data representation and \( P(X|z) \) generates data given a latent variable \( z \). Typically \( Q(z|X) \) is implemented with a neural network mapping the underlying data space into the latent space (encoder network). Similarly \( P(X|z) \) is implemented with a neural network and is responsible to generate data following the distribution \( P(X) \) given sample latent variables \( z \) from the latent space (decoder network). The variational objective has a very natural interpretation. We wish to model our data \( P(X) \) under some error function \( D_{KL}[Q(z|X)||P(z)] \). In other words, VAE tries to identify the lower bound of \( \log(P(X)) \), in which practice is good enough as trying to determine the exact distribution is often intractable. For this we aim to maximize over some mapping from latent variables to \( \log(P(X|z)) \) and minimize the difference between our simple distribution \( Q(z|X) \) and the true latent distribution \( P(z) \). Since we need to sample from \( P(z) \) in VAE typically one chooses a simple distribution to sample from such as \( N(0, 1) \). Since we wish to minimize the distance between \( Q(z|X) \) and \( P(z) \) in VAE one typically assumes that \( Q(z|X) \) is also normal with mean \( \mu(X) \) and variance \( \Sigma(X) \). The KL divergence...
Advantages of Our Approach. Our proposed approach has a number of appealing properties. First, our approach is complementary to existing AQP based systems and can be easily integrated in them. It can leverage the extensive research on aggregate estimators and approximation error metrics [39]. Second, our approach decouples the AQP system from using, an apriori derived, fixed sample $S$ of size $|S|$ or deriving a new sample on demand to meet certain accuracy constraints. Currently, an AQP system has to choose between two unappealing choices: either (a) continue using the existing sample that results in a bad estimate with large approximation errors, if for a given query the apriori chosen sample is not appropriate or (b) use a very expensive process to retrieve a new relevant sample from the underlying dataset. Our proposed approach offers an alternative. Since the model is a faithful representation of the dataset, it could be used to generate as many sample tuples as required - without the need to access the underlying dataset.

4.2 Using VAE for AQP

In this subsection, we describe how to train a VAE over relational data and use it for AQP.

Input Encoding. Our objective is to train a VAE over the relation $R$. In contrast to homogeneous domains such as images and text, relations often consist of mixed data types that could be discrete or continuous. The first step is to represent each tuple $t$ as a vector of dimension $d$. For ease of exposition, we consider one-hot encoding and describe other effective encodings in Section 4.5. One-hot encoding represents each tuple as a $d = \sum_{i=1}^{m} |\text{Dom}(A_i)|$ dimensional vector where the position corresponding to a given domain value is set to 1. Note that this approach works for both discrete and continuous attributes. For example, if $R$ has two binary attributes $A_1$ and $A_2$, we represent it as a 4-dimensional binary vector. A tuple with $A_1 = 0, A_2 = 1$ is represented as $[1, 0, 0, 1]$ while a tuple with $A_1 = 1, A_2 = 1$ is represented as $[0, 1, 0, 1]$. This approach is efficient for relatively small attribute domains but it could become cumbersome if a relation has millions of distinct values.

Model Building and Sampling from VAE. Once all the tuples are encoded appropriately, we could use VAE to learn the underlying distribution. We denote the size of the input and latent dimension by $d$ and $d'$ respectively. For one hot encoding, $d = \sum_{i=1}^{m} |\text{Dom}(A_i)|$. As $d'$ increases, the result in more accurate learning of the distribution at the cost of a larger model. Once the model is trained, it could be used to generate samples $X'$. The randomly generated tuples often share similar statistical properties to tuples sampled from the underlying relation $R$ and hence are a viable substitute for $R$. One could apply the existing AQP mechanisms on the generated samples and use it to generate aggregate estimates along with confidence intervals.

The sample tuples are generated as follows: we generate samples from the latent space $z$ and then apply the decoder network to convert points in latent space to tuples. Recall from Section 4.5 that the latent space is often a probability distribution that is easy to sample such as Gaussian. It is possible to speed up the sampling from arbitrary Normal distributions using the reparameterization trick. Instead of sampling from a distribution $N(\mu, \sigma)$, we could sample from the standard Normal distribution $N(0,1)$ with zero mean and unit variance. A sample $\epsilon$ from $N(0,1)$ could be con-
verted to a sample $N(\mu, \sigma)$ as $z = \mu + \sigma \odot \epsilon$. Intuitively, this shifts $\epsilon$ by the mean $\mu$ and scales it based on the variance $\sigma$.

4.3 Handling Approximation Errors.

In this subsection, we consider major sources of approximation error when using VAE for AQP. We propose an effective rejection sampling based solution to mitigate errors due to model bias.

**Sampling Error.** Aggregates estimated over the sample could differ from the exact results computed over the entire dataset and their difference is called the sampling error. Both the traditional AQP and our proposed approach suffer from sampling error. The techniques used to mitigate it - such as increasing sample size - can also be applied to the samples from the generative model.

**Errors due to Model Bias.** Another source of error is sampling bias. This could occur when the samples are not representative of the underlying dataset and do not approximate its data distribution appropriately. Aggregates generated over these samples is often biased and needs to be corrected. This problem is present even in traditional AQP [39] and mitigated through techniques such as importance weighting [18] and bootstrapping [15].

Our proposed approach also suffers from sampling bias due to a subtle reason. Generative models learn the data distribution which is a very challenging problem - especially in high dimensions. A DL model learns an approximate distribution that is close enough. Uniform samples generated from the approximate distribution would be biased samples from the original distribution resulting in biased estimates. As we shall show later in the experiments, it is important to remove or reduce the impact of model bias to get accurate estimates. Bootstrapping is not applicable as it often works by resampling the sample data and performing inference on the sampling distribution from them. Due to the biased nature of samples, this approach provides incorrect results [15]. It is challenging to estimate the importance weight of a sample generated by VAE. Popular approaches such as IWAE [8] and AIS [40] do not provide strong bounds for the estimates.

**Rejection Sampling.** We advocate for a rejection sampling based approach [22] that has a number of appealing properties and is well suited for AQP. Intuitively, rejection sampling works as follows. Let $x$ be a sample generated from the VAE model with probabilities $p(x)$ and $q(x)$ from the original and approximate probability distributions respectively. We accept the sample $x$ with probability $\frac{p(x)}{M \cdot q(x)}$ where $M$ is a constant upper bound on the ratio $p(x)/q(x)$ for all $x$. We can see that the closer the ratio is to 1, the higher the likelihood that the sample is accepted. On the other hand, if the two distributions are far enough, then a larger fraction of samples will be rejected. One can generate arbitrary number of samples from the VAE model, apply rejection sampling on them and use the accepted samples to generate unbiased and accurate aggregate estimates.

**Key Challenge.** We have to solve a fundamental problem before one can apply rejection sampling. In order to accept/reject a sample $x$, we need the value of $p(x)$. Estimating this value - such as by going to the underlying dataset - is very expensive and defeats the purpose of using generative models. A better approach is to approximately estimate it purely from the VAE model.

**Rejection Sampling over VAE.** Sample generation from VAE takes place in two steps. First, we generate a sample $z$ in the latent space using the variational posterior $q(z|x)$ and then we use the decoder to convert $z$ into a sample $x$ in the original space. In order to generate samples from the true posterior $p(z|x)$, we need to accept/reject sample $z$ with acceptance probability

$$a(z|x, M) = \frac{p(z|x)}{M \cdot q(z|x)}$$

(8)

where $M$ is an upper bound on the ratio $p(z|x)/q(z|x)$. Estimating the true posterior $p(z|x)$ requires access to the dataset and is very expensive. However, we do know that the value of $p(x, z)$ from the VAE is within a constant normalization factor $p(x) = \frac{p(x, z)}{p(z)}$. Thus, we can redefine Equation (8) as

$$a(z|x, M') = \frac{p(x, z)}{M \cdot p(x) \cdot q(z|x)} = \frac{p(x, z)}{M' \cdot q(z|x)}$$

(9)

We can now conduct rejection sampling if we know the value of $M'$. First, we generate a sample $z$ from the variational posterior $q(z|x)$. Next, we draw a random number $U$ in the interval $[0, 1]$ uniformly at random. If this number is smaller than the acceptance probability $a(z|x, M')$, then we accept the sample and reject it otherwise. That way the number of times that we have to repeat this process until we accept a sample is itself a random variable with geometric distribution $p = P(U \leq a(z|x, M'))$: $P(N = n) = (1 - p)^{n-1}p, n \geq 1$. Thus on average the number of trials required to generate a sample is $E(N) = 1/p$. By a direct calculation it is easy to show [1] that $p = 1/M'$. We approach setting the value of $M'$ as follows:

$$M' = e^{-T}$$

(10)

where $T \in [-\infty, +\infty]$ is an arbitrary threshold function. This definition has a number of appealing properties. First, this function is differentiable and can be easily plugged into the VAE’s objective function thereby allowing us to learn a suitable value of $T$ for the dataset during training [22].

Please refer to Section 4.5 for a heuristic method for setting appropriate values of $T$ during model building and sample generation. Second, the parameter $T$ when set, establishes a trade-off between computational efficiency and accuracy. If $T \rightarrow +\infty$, then every sample is accepted (i.e., no rejection) resulting into fast sample generation at the expense of the quality of the approximation to the true underlying distribution. In contrast when $T \rightarrow -\infty$, we ensure that almost every sample is guaranteed to be from the true posterior distribution, by making the acceptance probability small and as a result increasing sample generation time. Since a should be a probability we change equation Equation (9) to:

$$a(z|x, M') = \min\left[1, \frac{p(x, z)}{M' \cdot q(z|x)}\right]$$

(11)

4.4 Variational Autoencoder AQP Workflow

Algorithm [1] provides the pseudocode for the overall workflow of performing AQP using VAE. In the model building phase, we encode the input relation $R$ using an appropriate mechanism (see Section 4.5). The VAE model is trained on
the encoded input and stored along with appropriate metadata. During the runtime phase, we generate samples from VAE, apply rejection sampling to get samples that are highly likely to be from the original distribution (based on a set threshold $T$). One can then apply existing techniques for generating approximation of aggregates and other approximate query estimates as well as derive confidence intervals.

### 4.5 Making VAE practical for relational AQP

In this subsection, we propose two practical improvements for training VAE for AQP over relational data.

**Effective Input Encoding.** One-hot encoding of tuples is an effective approach for relatively small attribute domains. If the relation has millions of distinct values, then it causes two major issues. First, the encoded vector becomes very sparse resulting in poor performance. Second, it increases the number of parameters learned by the model thereby increasing the model size and the training time. We next consider two effective dense encoding mechanisms for tuples that work well in practice.

**Integer Encoding.** The one-hot encoding represented all attributes - categorical and numerical - as a sparse high dimensional binary vector. The other extreme is integer encoding, where we represent a tuple $t$ with $m$ attributes as an $m$ dimensional numerical vector $t'$. If $t[A_j]$ is numerical, then we set $t'[A_j] = t[A_j]$. For a categorical attribute $A_j$, we impose an arbitrary ordering to its domain values and for each value $v \in Dom(A_j)$, we assign a numeric value $\eta(v)$. By setting $t'[A_j] = \eta(A_j)$, we now treat the categorical attribute as a numeric value. Any mechanism to assign a numeric value to a set of domain values is viable. We begin with a large number $L > |Dom(A_j)|$ and encode the ordered domain values in an equidistant way. For e.g., if $Dom(A_j) = \{Low, Medium, High\}$ and $L = 16$, then $\eta(Low) = 0, \eta(Medium) = 8, \eta(\text{High}) = 16$. During training, we must either use mean squared loss (MSE) or bucketization followed by softmax as the reconstruction loss. Integer encoding results in VAE that are compact and very efficient to train due to the limited number of learnable parameters. However, the MSE reconstruction loss is known to be fragile and requires some careful training.

**Binary Encoding.** A promising approach to improve one-hot encoding is to make the representation denser. Without loss of generality, let the domain $Dom(A_j)$ be its zero-indexed position $[0, 1, \ldots, |Dom(A_j)| - 1]$. We can now consider two effective dense encoding mechanisms for generating approximation of aggregates and other approximate query estimates as well as derive confidence intervals.

### Algorithm 1 AQP using VAE

1: // Model building Phase
2: **Input:** $R = \{x_1, \ldots, x_n\}$ **Output:** VAE model $V$
3: Encode all tuples $x_1, \ldots, x_n$
4: Train VAE model $V$
5: //Online Phase
6: **Input:** VAE model $V$, $T$ **Output:** Aggregate estimate
7: $S_0 = \{}$ // set of samples
8: while samples are still needed do
9: Sample $z \sim q(z|x)$
10: Accept or reject $z$ based on Equation (11)
11: If $z$ is accepted, $S_z = S_z \cup \{z\}$
12: $S = \text{Decoder}(S_z)$ // Convert samples to original space
13: Estimate aggregate and confidence intervals from $S$ using standard techniques

### 4.6 Interpreting VAE for AQP

In this subsection, we visualize the latent space of a trained VAE model and show that it has meaningful symmetry with the original tuple space. While a systematic investigation is beyond the scope of the paper, our illustrations show that tuples with same value for an attribute often get grouped into a dense cluster.

**t-SNE.** t-distributed Stochastic Neighbor Embedding [34] is a non-linear dimensionality reduction technique that could embed high dimensional vectors into low dimensional points. Typically, visualizations using t-SNE are often better than visualization of the output of other dimensionality reduction techniques such as PCA and ICA. This is due to the fact t-SNE strives to ensure the neighbourhood identity of the points. In other words, similar points in the high dimension are modeled by nearby points in the low dimension while dissimilar points are modeled by distance points. Please refer to [34][34] for more details.

**Illustrating Latent Space of VAE.** We now illustrate the latent space of VAE model trained on Census dataset (see Section 6 for more details about the dataset) and show how the data points with different values for some common attributes are distributed in the latent space. The VAE model is trained on the entire dataset. We then select a 1% random sample of the dataset and pass it to the encoder.
to get their corresponding latent representations. We run t-SNE on the latent representations and project them in 2-dimensional space. The perplexity parameter was set to 30 [51]. Figure 2 shows the t-SNE visualization of the latent representations color coded based on the attribute Marital status. We can see that VAE is able to generate meaningful latent representations that have a semantic interpretation. Furthermore, tuples with similar values for marital status are mostly placed near each other.

5. AQP USING MULTIPLE VAES

In our discussion so far, we have assumed a single VAE model is built for the entire dataset learning its underlying distribution. As our experimental results show, even a single model could generate effective samples for AQP. However, it is possible to improve this performance and generate better samples. One way to accomplish this is to split the dataset into say K non-overlapping partitions and learn a VAE model for each of the partitions. Intuitively, we would expect each of the models to learn the finer characteristics of the data from the corresponding partition and thereby generate better samples for that partition. In this section, we investigate the problem of identifying the optimal set of K partitions for building VAE models.

5.1 Problem Setup

Typically, especially in OLAP settings, tuples are grouped according to hierarchies on given attributes. Such hierarchies reflect meaningful groupings which are application specific such as for example location, product semantics, year, etc. Often, these groupings have a semantic interpretation and building models for such groupings makes more sense than doing so on an arbitrary subset of the tuples in the dataset. As an example, the dataset could be partitioned based on the attribute Country such that all tuples belonging to a particular country is an atomic group. We wish to identify a partition $G$ such that all tuples belonging to a particular atomic group $g_i$ is a subset of the data from the atomic groups in $G$. We ensure that each of the groups are at least 5% of the data set size to avoid outlier groups and if necessary merged smaller groups into a miscellaneous group. We train a VAE model on each of the groups for different values of $T$ using variational rejection sampling and computed their R-ELBO. We then construct all pairs, triples, and other larger subsets of the groups and compare the bound obtained by $f(\cdot)$ with the actual R-ELBO value of the VAE trained on the data of these subsets. For each dataset, we evaluated 1000 randomly selected subsets and report the fraction in which the bound

Efficiently solving this problem involves two steps: (a) given a partition, a mechanism to estimate the error of K VAEs trained over the partition without conducting the actual training and (b) an algorithm that uses (a) to identify the best partition over the space of partitions. Both of these challenges are non-trivial.

5.2 Bounding VAE Errors

Quantifying VAE Approximation. The parameters of VAE are learned by optimizing an evidence lower bound (ELBO) given by

$$E[\log P(X|z)] - D_{KL}(Q(z|X)||P(z))$$

(from Equation 7) which is a tight bound on the marginal log likelihood. ELBO provides a meaningful way to measure the distribution approximation by the VAE. Recall from Section 4.3 that we perform rejection sampling on the VAE that results in a related measure we call R-ELBO (resampled ELBO) defined as

$$E[\log P(X|z)] - D_{KL}(R(z|X,T)||P(z))$$

where $R(z|X,T)$ is the resampled distribution for a user-specified threshold of $T$. Given two VAEs trained on the same dataset for a fixed value of $T$, the VAE with lower R-ELBO provides a better approximation.

Bounding R-ELBO for a Partition. Let us assume that we will train a VAE model for each of the atomic groups $g_i \in G$. We train the model using variational rejection sampling [22] for a fixed $T$ and compute its R-ELBO. In order to find the optimal partition, we have to compute the value of R-ELBO for arbitrary subsets $s_i = \{g_{i1}, \ldots, \} \subseteq G$. The naive approach would be to train a VAE on the union of the data from atomic groups in $s_i$ which is time consuming. Instead, we empirically show that it is possible to bound the R-ELBO of VAE trained on $s_i$ if we know the value of R-ELBO of each of $g_{i1}, \ldots$. Let $f(\cdot)$ be such a function. In this paper, we take a conservative approach and bound it by sum $f(r_1, r_2, \ldots, r_k) = \sum_{k=1}^{k} r_i$ where $r_i$ is the R-ELBO for group $g_i$. In other words, $f(\cdot)$ bounds the R-ELBO of VAE trained $\cup_{k=1}^{g} g_i$ by $\sum_{k=1}^{k} r_i$. It is possible to use other functions that provide tighter bounds.

Empirical Validation. We empirically validated the function $f(\cdot)$ on a number of datasets under a variety of settings. Table 1 shows the results for Census and Flights dataset that has been widely used in prior work on AQP such as [32, 17, 19]. Please refer to Section 6 for a description of the two datasets. We obtained similar results for other benchmark datasets. For each of the datasets, we constructed multiple atomic groups for different categorical attributes. For example, one could group the Census dataset using attributes such as gender, income, race etc. We ensured that each of the groups are at least 5% of the data set size to avoid outlier groups and if necessary merged smaller groups into a miscellaneous group. We trained a VAE model on each of the groups for different values of $T$ using variational rejection sampling and computed their R-ELBO. We then construct all pairs, triples, and other larger subsets of the groups and compare the bound obtained by $f(\cdot)$ with the actual R-ELBO value of the VAE trained on the data of these subsets. For each dataset, we evaluated 1000 randomly selected subsets and report the fraction in which the bound
was true. As is evident in Table I the bound almost always holds.

| Dataset | $T = -10$ | $T = 0$ | $T = +10$ |
|---------|-----------|---------|-----------|
| Census  | 0.992     | 0.997   | 0.996     |
| Flights | 0.961     | 0.972   | 0.977     |

5.3 Choosing Optimal Partition

In this section we assume we are provided with the value of R-ELBO for each of the groups $g_i \in G, 1 \leq i \leq l$, a bounding function $f(\cdot)$ and a user specified value $K$. We propose an algorithm that optimally splits a relation $D$ into $K$ non overlapping partitions $S = \{s_1, \ldots, s_K\}$ where $s_i \subseteq G$ and $s_i \cap s_j = \emptyset$ when $i \neq j$. The key objective is to choose the split $S$ in such a way that the $\sum_{i=1}^{K}$ R-ELBO($s_i$) is minimized. Note that there are $K^l$ possible partitions and exhaustively enumerating and choosing the best partition is often infeasible. R-ELBO($g$) corresponds to the actual R-ELBO for atomic groups $g \in G$ while for $s_i \subseteq G$, this is estimated using the bounding function $f(s_i)$. We investigate three scenarios that occur in practice.

Optimal Partition using OLAP Hierarchy. In OLAP settings, tuples are grouped according to hierarchies on given attributes that reflect meaningful semantics. We assume the availability of an OLAP hierarchy in the form of a tree where the leaf node corresponds to the atomic groups (e.g., Nikon Digital Cameras) while the intermediate groups correspond to product semantics (e.g., Digital Camera $\rightarrow$ Camera $\rightarrow$ Electronics and so on). We wish to build VAE on meaningful groups of tuples by constraining $s_i$ to be selected from the leafs or intermediate nodes, be mutually exclusive and have the least aggregate R-ELBO score. We observe that the selected nodes forms a tree cut that partitions the OLAP hierarchy into $K$ disjoint sub-trees.

Let us begin by considering the simple scenario where the OLAP hierarchy is a binary tree. Let $h$ denote an arbitrary node in the hierarchy with $\text{left}(h)$ and $\text{right}(h)$ returning the left and right children of $h$ if they exist. We propose a dynamic programming algorithm to compute the optimal partition. We use the table $\text{Err}[h,k]$ to denote aggregate R-ELBO of splitting the sub-tree rooted at node $h$ using at most $k$ partitions where $k \leq K$. The base case $k = 1$ is simply building the VAE on all the tuples falling under node $h$. When $k > 1$, we evaluate the various ways to split $h$ such that the aggregate R-ELBO is minimized. For example, when $k = 2$, there are two possibilities. We could either not split $h$ or build two VAE models over $\text{left}(h)$ and $\text{right}(h)$. The optimal decision could be decided by choosing the option with least aggregate error. In general, we consider all possible ways of apportioning $K$ between the left and right sub-trees of $h$ and pick the allocation resulting in least error. The recurrence relation is specified by:

$$\text{Err}[h,k] = \begin{cases} \text{R-ELBO}(h) & \text{if } k=1 \\ \min_{1 \leq i \leq k} \{ \text{Err}[\text{left}(h),i] + \text{Err}[\text{right}(h),k-i] \} & \text{otherwise} \end{cases}$$  \hspace{1cm} (12)

The extension to non-binary trees is also straightforward. Let $C = \{c_1, \ldots, c_j\}$ be the children of node $h$. We systematically partition the space of children into various groups of two and identify the best partitioning that gives the least error (eq. 13). This approach works as any non-binary hierarchy could be transformed into a binary one as shown in Figure 3.

Scenario 2: Partitioning with Contiguous Atomic Groups. Given the atomic groups $G = \{g_1, \ldots, g_l\}$, a common scenario is to partition them into $K$ contiguous sub-sets. This could be specified as $K + 1$ integers $b_1 \leq b_2 \ldots \leq b_{K+1} = l$ where the boundary of the $i$-th subset is specified by $[b_i, b_{i+1})$ and consists of a set of atomic groups $\{g_{b_i}, \ldots, g_{b_{i+1}}\}$. This is often desirable when the underlying attribute has a natural ordering such as year. So we would prefer to train VAEs over data from consecutive years such as $\{2016 - 2017, 2018 - 2019\}$ instead of arbitrary groupings such as $\{\{2016, 2018\}, \{2017, 2019\}\}$. This problem could be solved in near linear time ($i.e., O(l)$) by using the approach first proposed in [23]. The key insight is the notion of sparse interval set system that could be used to express any interval using a bounded number of sparse intervals. The authors then use a dynamic programming approach on the set of sparse intervals to identify the best partitioning.

Scenario 3: Partitioning with No Constraints. The final scenario imposes no constraint on which set of atomic groups could be clubbed together. This is often applicable in scenarios where the objective is to obtain the least aggregate R-ELBO error at the cost of grouping arbitrary atomic groups. Unfortunately, the generality of this problem is also its pitfall. This problem is known to be a special instance of the Set-partitioning problem which is known to be NP-Complete. There are $K^l$ possible ways to group $l$ atomic groups into $K$ partitions. However, for the bounding function of addition, it is possible to do better. There is a natural dynamic programming based approach first proposed in [33] that has time complexity $O(l^4)$. It works by building a table $\text{Err}[X,i]$ where the entries correspond to every $X \subseteq G$ and $i \leq K$. The algorithm considers subsets $X$ of increasing size and finds the least error subset by using $\text{Err}[X \setminus S, i-1]$ where $S$ is a subset of elements from the powerset of $X$.
6. EXPERIMENTS

We conduct a comprehensive set of experiments and demonstrate that VAE (and deep generative models) are a promising mechanism for AQP. We reiterate that our proposed approach is an alternate way for generating samples, albeit very fast. Most of the prior work for improving AQP estimates could be transparently used on the samples from VAE. In our evaluation, we investigate the following major questions:

1. Can VAE learn the data distribution from an underlying relation and generate realistic relational samples?
2. How do our proposed alternate schemes for encoding and decoding of tuples impact sample quality?
3. How effective is the proposed rejection sampling methodology as a bias reduction mechanism in our setting?
4. Does the use of multiple VAEs for the same dataset result in improved accuracy for AQP?

6.1 Experimental Setup

Hardware and Platform. All our experiments were performed on a server with 16 cores, 128 GB of RAM and NVidia Tesla K80 GPU. We used PyTorch [44] for training VAE and GAN, buelearn [48] for learning Bayesian Networks and MSPN [37] for mixed sum-product networks (MSPN).

Datasets. We conducted our experiments on two real-world datasets: Census [2] and Flights [1,16]. Both datasets have complex correlated attributes and conditional dependencies that make AQP challenging. The Census dataset has 8 categorical attributes and 6 numerical attributes and contains demographic and employment information. The Flights dataset has 6 categorical and 6 numerical attributes and contains information about on-arrival statistics for the last few years. We used the data generator from [16] to scale the datasets to arbitrary sizes while also ensuring that the relationships between attributes are maintained. By default, our experiments were run on datasets with 1 million tuples.

Deep Generative Models for AQP. In our experiments, we primarily focus on VAE for AQP as it is easy and efficient to train and generates realistic samples [12]. By default, our VAE model consists of a 2 layer encoder and decoder that are parameterized by Normal and Bernoulli distributions respectively. We used binary encoding (Section 4.5) for converting tuples into a representation consumed by the encoder. Each of the samples from the latent space was decoded 10 times stochastically in the tuple space (utilizing our proposed tuple decoding schemes) and aggregated to form a realistic tuple (as detailed in Section 4.5).

In order to generate high quality samples, we use rejection sampling during both VAE training and sample generation albeit at different granularities. During training, the value of threshold $T(x)$ is set for each tuple $x$ so that the acceptance probability of samples generated from $q(z|x)$ is roughly 0.9 for most tuples. We use the procedure from [22] to generate a Monte Carlo estimate for $T(x)$ satisfying acceptance probability constraints. While the trained model already produces realistic samples, we further ensure this by performing rejection sampling with a fixed threshold $T$ (for the entire dataset) during sample generation (as detailed in Section 4.3). There are many ways for choosing the value of $T$. It could be provided by the user or chosen by cross validation such that it provides the best performance on query workload. By default, we compute the value of $T$ from the final epoch of training as follows. For each tuple $x$, we have the Monte-Carlo estimate $T(x)$. We select the 90-th percentile of the distribution $T(x)$. Intuitively, this ensures that samples generated for 90% of the tuples would have acceptance probability of 0.9. Of course, it is possible to specify different values of $T$ for queries with stringent accuracy requirements. We used Wasserstein GAN as the architecture for generative adversarial networks [20].

We used entropy based discretization [13] for continuous attributes when training discrete Bayesian networks. We used the default settings from [37] for training Mixed Sum Product Networks (MSPN).

Query Workload. We used IDEBench [16] to generate aggregate queries involving filter and group-by conditions. We then selected a set of 1000 queries that are diverse in various facets such as number of predicates, selectivity, number of groups, attribute correlation etc.

Performance Measures. As detailed in Section 1.3 AQP using VAE introduces two sources of errors: sampling error and errors due to model bias. The accuracy of an estimate could be evaluated by relative error (see Equation 1). For each query in the workload, we compute the relative error over a fixed size sample (1% by default) obtained from the underlying dataset $R$ and the learned VAE model. For a given query, the relative error difference (RED) computed as the absolute difference between the two relative errors provides a meaningful way to compare them. Intuitively, RED will be close to 0 for a well trained VAE model. We repeat this process over 10 different samples and report the average results. Given that our query workload has 1000 queries, we use box plots to concisely visualize the distribution of the relative error difference. The middle line corresponds to the median value of the difference while the box boundaries correspond to the 25th and 75th percentiles. The top and bottom whiskers are set to show the 95th and 5th percentiles respectively.

6.2 Experimental Results

Evaluating Model Quality. In our first experiment, we demonstrate that VAE could meaningfully learn the data distribution and generate realistic samples. Figure [1] shows the distribution of relative error differences for both datasets over the entire query workload for various sample sizes. We can see that the differences are less than 1% for almost all the cases for the Census dataset. The Flights dataset has many attributes with large domain cardinalities which makes learning the data distribution very challenging. Nevertheless, our proposed approach is still within 3% of the relative error obtained from the samples of $R$.

Impact of Selectivity. In this experiment, we group the queries based on their selectivity and compute the relative error difference for each group. As shown in Figure [2] the difference is vanishingly small for queries with large selectivities and slowly increases for decreasing selectivities. In general, generating estimates for low selectivity queries is challenging for any sampling based AQP. The capacity/model size constraints imposed on the VAE model could result in generating bad estimates for some queries with very low selectivities. However, this issue could be readily ameliorated...
by building multiple VAE models that learn the finer characteristics of data minimizing such errors in these cases.

**Impact of Model Capacity and Depth.** Figures 6 and 7 shows the impact of two important hyper-parameters - the number of latent dimensions and depth of the encoder and decoder network. We vary the latent dimension from 10% to 100% of the input dimension. Large latent dimension results in an expressive model that can learn complex data distributions at the cost of increased model size and training time. Similarly, increasing the depth results in a more accurate model but with larger model size and slower training time. Empirically, we found that setting latent dimension size to 50% (for binary encoding) and encoder/decoder network depth of 2 provides good results.

**Effectiveness of Input Encoding and Output Decoding.** It is our observation that the traditional approach of one-hot encoding coupled with generating a single sample tuple for each sample from the latent space does not provide realistic tuples. It may be suitable for image data but certainly not suitable for relational data. Figure 8 shows how different encodings affect the generated samples. For datasets such as Census where almost all attributes have small domain cardinality, all the three approaches provide similar results. However, for the flights dataset where some attributes have domain cardinality in tens of thousands, naive approaches such as one-hot encoding provides sub-optimal results. This is due to the fact that there are simply too many parameters to be learnt and even a large dataset of 1 Million tuples is insufficient. Similarly, Figure 9 shows that our proposed decoding approach dramatically decreases the relative error difference making the approach suitable for relational data. This is due to the fact that the naive decoding could produce unrealistic tuples that could violate common integrity constraints an effect that is minimized when using our proposed decoding.

**Impact of Rejection Sampling.** Figure 10 shows how varying the value of \( T \) impacts the sample quality. Recall from Section 4.3 that as \( T \to +\infty \), almost all samples from VAE are accepted, while when \( T \to -\infty \), samples are rejected unless they are likely to be from the true posterior distribution. As expected, decreasing the value of \( T \) results in decreased value of relative error difference. However, this results in a larger number of samples being rejected. Our approach allows \( T \) to be varied across queries such that queries with stringent accuracy requirements can use small \( T \) for better estimates. We investigate the impact of rejection sampling on model building and sample generation later in the section.

**One versus Multiple VAEs.** In the next set of experiments, we consider the case where one uses multiple VAEs to learn the underlying data distribution. We partitioned the attributes based on marital-status for Census and origin-state for Flights. We evaluated partitioning data over other attributes and observed similar results. In order to compare the models fairly, we ensured that the cumulative model capacity for both scenarios were the same. For example, if we built \( K \) VAE models with capacity \( C \) each, then we compared it against a single VAE model with capacity \( K \times C \). Figure 11 shows the results. As expected, the sample quality improves with larger number of VAE models enabling them to learn finer data characteristics. Interestingly, we observe that increasing the model capacity for the single VAE case has diminishing returns due to the fixed size of the training data. In other words, increasing the capacity does not improve the performance beyond certain model capacity. Figure 12 compares the performance of partitions selected by the dynamic programming algorithm for the scenario where an OLAP hierarchy is provided. We compare it against a
Other Deep Generative Models. While we primarily focused on VAE, it is possible to leverage other deep generative models for AQP. Figure 13 compares the performance of three common models: VAE, GAN and Bayesian Networks (BN). Generative Adversarial Networks (GANs) [21, 20] are a popular and powerful class of generative models that learn the distribution as a minimax game between two components: generator (that generates data) and discriminator (that identifies if the sample is from the true distribution or not). (Deep) Bayesian networks (BN) are another effective generative model that specifies the joint distribution as a directed graphical model where nodes correspond to random variables and directed edges between nodes signify (direct) dependencies between the corresponding attributes. Please refer to [20] for more details. In order to ensure a fair comparison, we imposed a constraint that the model size for all three approaches are fixed. Furthermore, VAE provides the best results for a fixed model size. GANs provide reasonable performance but was heavily reliant on tuning. Training a GAN requires identifying an equilibrium and tuning of many parameters such as the model architecture and learning rate [21]. This renders the approach hard to use in practice for general datasets. Identifying appropriate mechanisms for training GANs over relational data for AQP is a promising avenue for future research. BNs provide the worst result among the three models. While BNs are easy to train for datasets involving discrete attributes, a hybrid dataset with discrete and continuous attributes, and attributes with large domain cardinalities are challenging. When the budget on model size is strict, BNs often learn a sub-optimal model.

We also evaluated VAE against the recently proposed MSPN that has been utilized for AQP in [32]. Similar to Bayesian Networks, MSPNs are acyclic graphs (albeit rooted graphs) with sum and product nodes as internal nodes. Intuitively, the sum nodes split the dataset into subsets while product nodes split the attributes. The leaf nodes define the probability distributions for an individual variable. MSPN could be used to represent an arbitrary probability distribution [37]. We used the random sampling procedure from [32] for generating samples from a trained MSPN. We observed that MSPN often struggles to model distributions involving large number of attributes and/or tuples and that using a single MSPN for the entire model did not provide good results. As a comparison to train a VAE on 1M tuples of the Census data set on all attributes requires a few minutes versus almost 3.5 hours for MSPN. In addition the accuracy of queries with larger number of attributes for the case of MSPN was very poor and not close to any of the other models. Hence, we decided to provide an advantage to MSPN, building the model over subsets of attributes. That way we let the model focus only on specific queries and improve its accuracy. There were around 120 distinct combination of measure and filter attributes in our query workload. We built MSPN models for each combination of attributes, generate samples from it and evaluate our queries over it. For example, if a query involved an aggregate over \{A_i, A_j, A_k\}, we built an MSPN over the projected dataset containing only \{A_x, A_y, A_z\}. Unlike GAN and BN, we did not control the number of leaf nodes. However, the size of the MSPN models that were trained over attribute subsets were in the same ballpark as the other generative models. Figure 15 presents the performance of VAE and MSPN (build on specialized subsets of attributes) to be superior over GAN and BN. However, in the case of VAE the model was trained over the entire dataset being able to answer arbitrary queries while MSPN was trained over specific attribute subsets utilized by specific queries. Even in this case, providing full advantage to MSPN, the median relative error difference for VAE and
MSPN were 0.060835 and 0.137699 respectively, more than two times better for VAE. This clearly demonstrates that a VAE model can learn a better approximation of the data, being able to answer arbitrary queries while it can be trained an order of magnitude faster than MSPN as detailed next.

Performance Experiments. Our next set of experiments investigate the scalability of VAE for different dataset sizes and values of threshold $T$. Figure 14 depicts the results for training over a single GPU. All results would be substantially better with the use of multiple GPUs. As expected, the training time increases with larger dataset size. However, due to batching and other memory optimizations, the increase is sublinear. Next, incorporating rejection sampling has an impact on the training time with stringent values of $T$ requiring more training time. The increased time is due to the larger number of training epochs needed for the model to learn the distribution. The validation procedure for evaluating the rejection rate uses a Monte Carlo approach \cite{22} that also contributes to the increased training time. However overall it is evident from our results that very large data sets can be trained very efficiently even on a single GPU. This attests to the practical utility of the proposed approach. Figure 15 presents the cost of generating samples of different sizes and for various values of $T$. Not surprisingly, lower values of $T$ require a larger sampling time due to the higher number of rejected samples. As $T$ becomes less stringent, sampling time dramatically decreases. Interestingly, the sampling time does not vary a lot for different sampling sizes. This is due to the efficient vectorized implementation of the sampling procedure in PyTorch and the availability of larger memory that could easily handle samples of large size. It is evident again that the proposed approach can generate large number of samples in fractions of a second making the approach highly suitable for fast query answering with increased accuracy.

7. RELATED WORK

Deep Learning for Databases. Recently, there has been increasing interest in applying deep learning techniques for solving fundamental problems in databases. SageDB \cite{28} proposes a new database architecture that integrates deep learning techniques to model data distribution, workload and hardware and use it for indexing, join processing and query optimization. Deep learning has also been used for learning data distribution to support index structures \cite{29}, join cardinality estimation \cite{27,11}, join order enumeration \cite{31,36}, physical design \cite{45}, entity matching \cite{14}, workload management \cite{35} and performance prediction \cite{40}.

Sampling based Approximate Query Processing. AQP has been extensively studied by the database community. A detailed surveys is available elsewhere \cite{15,39}. Non sampling based approaches involve synopses data structures such as histograms, wavelets and sketches. They are often designed for specific types of queries and could answer them efficiently. In our paper, we restrict ourselves to sampling based approaches \cite{3,4,43,25,9}. Samples could either be pre-computed or obtained during runtime. Pre-computed samples often leverage prior knowledge about workloads to select samples that minimize the estimation error. However, if workload is not available or is inaccurate, the chosen samples could result in worse approximations. In this case, recomputing samples is often quite expensive. Our model based approach could easily avoid this issue by generating samples as much as needed on-demand. Online aggregation based approaches such as \cite{24,52} continuously refine the aggregate estimates during query execution. The execution can be stopped at any time if the user is satisfied with the estimate. Prior approaches often expect the data to be retrieved in a random order which could be challenging. Our model based approach could be easily retrofitted into online aggregation systems as they could generate random samples efficiently. Answering ad-hoc queries and aggregates over rare sub-populations is especially challenging \cite{10}. Our approach offers a promising approach where as many samples as needed could be generated to answer such challenging queries without having to access the dataset. \cite{22} uses mixed sum-product networks (MSPN) to generate aggregate estimates for interactive visualizations. While in the same spirit as our work, their proposed approach suffers from scalability issues that limits its widespread applicability. Even for a small dataset with 1 million tuples, it requires hours for training. This renders such an approach hard to apply for very large data sets. In contrast a VAE model can be trained in a matter of minutes making it ideal for very large data sets. Moreover as our results for MSPN demonstrate even when proving the advantage of tailoring the trained model specifically for attributes included in the query (while training a VAE on all attributes of the relation) the VAE model provides much higher query accuracy, establishing the generality of our approach.

8. CONCLUSION

In this paper, we propose a model based approach for AQP. Our experiments show that the generated samples are realistic and produce accurate aggregate estimates. Naively applying DL models such as VAE for AQP produces unrealistic tuples and incorrect estimates. We proposed improvements for encoding and decoding of relational data that produces realistic samples. We identify the issue of model bias and propose a rejection sampling based approach to mitigate it. We observe that training multiple VAE models produce better samples and proposed dynamic programming based algorithms for identifying optimal partitions. Our proposed approach could integrated easily into AQP systems and can satisfy arbitrary accuracy requirements by generating as many samples as needed without going back to the data. There are a number of interesting questions to consider in the future. Some of them include better mechanisms for generating conditional samples that satisfy certain constraints. Moreover, it would be interesting to study the applicability of generative models in other data management problems such as synthetic data generation for structured and graph databases extending ideas in \cite{42}.

9. REFERENCES

[1] Bureau of transportation statistics. Flights Data Set, 2019.
[2] UCI Machine Learning Repository. Adult Data Set, 2019.
[3] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In ACM Sigmod Record, volume 28, pages 574–576. ACM, 1999.
[38] B. Mozafari. Approximate query engines: Commercial challenges and research opportunities. In Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD ’17, pages 521–524, New York, NY, USA, 2017. ACM.

[39] B. Mozafari and N. Niu. A handbook for building an approximate query engine. IEEE Data Eng. Bull., 38(3):3–29, 2015.

[40] R. M. Neal. Annealed importance sampling. Statistics and computing, 11(2):125–139, 2001.

[41] J. Ortiz, M. Balazinska, J. Gehrke, and S. S. Keerthi. Learning state representations for query optimization with deep reinforcement learning. In Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning, page 4. ACM, 2018.

[42] N. Park, M. Mohammadi, K. Gorde, S. Jajodia, H. Park, and Y. Kim. Data synthesis based on generative adversarial networks. PVLDB, 11(10):1071–1083, 2018.

[43] Y. Park, B. Mozafari, J. Sorenson, and J. Wang. Verdictdb: universalizing approximate query processing. In Proceedings of the 2018 International Conference on Management of Data, pages 1461–1476. ACM, 2018.

[44] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[45] A. Pavlo, G. Angulo, J. Arulraj, H. Lin, J. Lin, L. Ma, P. Menon, T. C. Mowry, M. Perron, I. Quah, et al. Self-driving database management systems. In CIDR, 2017.

[46] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In International Conference on Machine Learning, pages 1278–1286, 2014.

[47] A. Roberts, J. Engel, C. Raffel, C. Hawthorne, and D. Eck. A hierarchical latent vector model for learning long-term structure in music. ICML, 2018.

[48] M. Scutari and J.-B. Denis. Bayesian Networks with Examples in R. Chapman and Hall, Boca Raton, 2014. ISBN 978-1-4822-2558-7, 978-1-4822-2560-0.

[49] S. Semeniuta, A. Severyn, and E. Barth. A hybrid convolutional variational autoencoder for text generation. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 627–637, 2017.

[50] S. Venkataraman, Z. Yang, M. J. Franklin, B. Recht, and I. Stoica. Ernest: Efficient performance prediction for large-scale advanced analytics. In NSDI, pages 363–378, 2016.

[51] M. Wattenberg, F. Vigas, and I. Johnson. How to use t-sne effectively. Distill, 2016.

[52] S. Wu, B. C. Ooi, and K.-L. Tan. Continuous sampling for online aggregation over multiple queries. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pages 651–662. ACM, 2010.

[53] D. Y. Yeh. A dynamic programming approach to the complete set partitioning problem. BIT Numerical Mathematics, 26(4):467–474, 1986.