Unsupervised Selectivity Estimation by Integrating Gaussian Mixture Models and an Autoregressive Model

Zizhong Meng
Nanyang Technological University
zizhong001@e.ntu.edu.sg

Peizhi Wu
University of Pennsylvania
pagewu@cis.upenn.edu

Gao Cong
Nanyang Technological University
gaocong@ntu.edu.sg

Rong Zhu
Harbin Institute of Technology
rzhu@hit.edu.cn

Shuai Ma
Beihang University
mashuai@buaa.edu.cn

ABSTRACT

Selectivity estimation is a fundamental database task, which has been studied for decades. A recent trend is to use deep learning methods for selectivity estimation. Deep autoregressive models have been reported to achieve excellent accuracy. However, if the relation has continuous attributes with large domain sizes, the search space of query inference on deep autoregressive models can be very large, resulting in inaccurate estimation and inefficient inference. To address this challenge, we propose a new model that integrates multiple Gaussian mixture models and a deep autoregressive model. On the one hand, Gaussian mixture models can fit the distribution of continuous attributes and reduce their domain sizes. On the other hand, deep autoregressive model can learn the joint data distribution with reduced domain attributes. In experiments, we compare with multiple baselines on 4 real-world datasets containing continuous attributes, and the experimental results demonstrate that our model can achieve up to 20 times higher accuracy than the second best estimators, while using less space and inference time.

1 INTRODUCTION

Selectivity estimation aims to estimate the fraction of a query predicate accurately and fast in small memory usage without actual execution [11]. It has applications in approximate query processing and query optimization [45]. Although selectivity estimation has been studied for decades, it is still a notoriously difficult problem and the accuracy may drop significantly as the query complexity increases [30, 55].

The difficulty of selectivity estimation is to learn the joint data distribution of a relation and estimate selectivity accurately and efficiently. DBMSes [33, 36, 42] estimate selectivity based on histograms or sketches. These estimators can conduct query inference very fast, but may have poor accuracy due to the independence assumption that they make. Traditional learning based estimators, such as probabilistic graphical model [47] and kernel density estimation [22, 26], relax the assumption slightly, but still have their shortcomings. For example, the kernel density estimation [22, 26] is struggled with high dimensional data, and the probabilistic graphical model [47] is inefficient in estimation.

Recently, deep learning models have been employed for selectivity estimation. Deep autoregressive modes (AR models) are employed in the estimators [49, 54, 55], which learn the joint data distribution without independence assumptions. As reported in the experimental results [49, 54, 55], estimators based on AR models often achieve state-of-the-art accuracy. After AR models are trained, estimators invoke a sampling method, such as progressive sampling [54, 55], to estimate the selectivities of queries with range predicates since enumerating all points in a query range is prohibitively expensive. However, when there are attributes with large domain sizes in the dataset, such as spatial attributes or temporal attributes, the estimation based on a sampling method like progressive sampling can be inaccurate since the sample space is very large. For example, to manage the efficiency of progressive sampling, estimators may use up to 10^4 samples in estimation. However, the sample space of a real-life dataset can be much larger, e.g., the sample space of `HIGGS` used in previous work [11] is up to 10^{41}. The large difference may render progressive sampling inaccurate on estimation.

To address the aforementioned challenge, we aim to design a model that is able to reduce the domain sizes of some attributes, and thus the sample space, such that the AR model can learn more accurate joint distribution of dataset with large domain sizes. In this work, we propose a new model for selectivity estimation that integrates a deep autoregressive model with Gaussian mixture models, which is called IAM. On the one hand, the AR model can be used to capture the strong correlations of data and learn the joint data distribution. On the other hand, Gaussian mixture models (GMMs) can be used to learn the data distribution of continuous attributes and reduce domain sizes of these attributes. Furthermore, we integrate GMMs and an AR model into one single model, such that training can be performed end-to-end, which is a desirable feature with practical benefits for machine learning algorithms. IAM has two advantages compared with previous AR model based estimators. For training, since domain sizes of some attributes are reduced by GMMs and the data distribution is simplified, we can learn a better AR model with the same model size. For inference, the search space is smaller, and the progressive sampling is more accurate and efficient.

Another challenge of integrating an AR model and GMMs in IAM is how to handle range queries. It is prohibitively expensive to enumerating all distinct values in the query range and summing their estimated probabilities. To address the challenge, previous work [54, 55] of using AR models propose to use the progressive sampling algorithm to estimate selectivities for range queries. Unfortunately, this sampling method is biased when applied to our model IAM since we need to consider the probabilities on mixture models. To address the challenge, we propose a modification to the progressive sampling algorithm based on the estimated probabilities on mixture models, making the sampling algorithm to be unbiased on IAM. This enables IAM to perform accurate and efficient estimation.

Contributions. Our work has the following contributions:

- We propose a novel model IAM that integrates GMMs and an AR model for selectivity estimation, which effectively addresses the problem of the existing AR model based estimators [49, 54, 55] on data containing continuous attributes with large domain sizes. IAM is able to learn the data distribution on data with continuous attributes efficiently and effectively.
- We propose a modification to the progressive sampling algorithm by considering the probabilities on mixture models, so that it can do unbiased sampling for query inference in IAM. This algorithm enables IAM to perform efficient and accurate estimation for range queries.
- We conduct comprehensive experiments to compare IAM with other baselines on four real-life datasets with continuous attributes with large domain sizes. Our results show that IAM achieves up to 20 times higher accuracy than the second best estimators at tail, and efficient estimation. IAM has the best query performance while being integrated into the query optimizer of Postgres.

2 PROBLEM FORMULATION

2.1 Problem Definition

Consider a relation T with n attributes $\{A_1, A_2 \cdots A_n\}$. We denote the domain of A_i by $Dom(A_i)$ and the domain size of A_i by $|A_i|$. We introduce the following definitions:

Definition 2.1 (Query Predicates). A query q is a conjunction of single attribute's predicate: $R = R_1 \wedge R_2 \wedge \cdots \wedge R_n$, where $R_i \subseteq Dom(A_i)$ can be a range or a point predicate. Each predicate contains an attribute, an operator and a value (e.g., $A_1 = 4$ or $A_2 > 3$).

Definition 2.2 (Selectivity). Selectivity is defined as the fraction of tuples in T that satisfy q: $actsel(q) = |\{\theta(t) = 1 | t \in T\}|/|T|$, where $\theta(t)$ indicates whether t is contained in the answer of q and $|T|$ is the number of tuples in T.

Definition 2.3 (Selectivity Estimation). Selectivity estimation aims to estimate the selectivity of query q without actually answering q. We use $estsel(q)$ to denote the estimated selectivity of query q.

Supported Queries. Following previous work [4, 22, 24, 37, 41, 55], we support selectivity estimation on queries with conjunctions and disjunctions of predicates. We focus on conjunctions in the rest of the paper. Note that disjunctions can be easily supported by the inclusion-exclusion principle, e.g., $R_i \vee R_j = R_i + R_j - R_i \wedge R_j$.

For operators, we support range queries ($\neq, <, \leq, >, \geq$) and point queries (=). Both types of queries can be represented by the aforementioned representation of q. For example, given a query $(30 \leq A_1 \leq 100) \wedge (A_2 = \texttt{man})$, it will be represented as $\langle A_1 \leq 100, A_1 \geq 30, A_2 = \texttt{man}\rangle$.

Problem. Given a relation T, we aim to construct a data-driven model by learning from T. This model can be used to perform selectivity estimation *accurately* and *efficiently* for each incoming query q.

We aim to support relations containing continuous attributes with large domain sizes (e.g., spatial attributes or temporal attributes). Moreover, we focus on range predicates on continuous attributes. This is because 1) predicates on continuous attributes are usually range predicates. For example, given a points of interest (POIs) dataset, we may want to find POIs in a spatial range, rather than at a specific latitude and longitude; 2) The selectivities of point predicates on such attributes are usually 0 or $\frac{1}{|T|}$, and thus point predicates on such attributes are easy to estimate.

Extension to support Joins. Our model supports multi-way, multi-key equi-joins, similar to previous work [24, 54].

2.2 Formulation as Joint Distribution Estimation

We consider the problem of selectivity estimation as joint distribution estimation. A joint probability distribution of relation T is a probability distribution in which each of $A_1, \cdots, A_n$ falls in a value specified for that attribute. We use $P(a_1, \cdots, a_n)$ to represent joint probability distribution. Given a query q, actual selectivity can be calculated by using the joint probability distribution: $actsel(q) = \sum_{a=(a_1,\cdots,a_n)\in Dom(A_1)\times\cdots\times Dom(A_n)} P(a) \cdot \theta(a)$ where $\theta : A_1 \times A_2 \times \cdots \times A_n \to \{0, 1\}$ is a function indicating whether $(a_1, \cdots, a_n)$ is contained in q. Thus, the essential problem of selectivity estimation is accurately estimating the joint distribution.

Formulation on Multiple Tables. The joint distribution can be extended to multiple tables as follows. Given a dataset, the join schema is generated from full outer join on all tables, $T = T_1 ⟗ T_2 ⟗ \cdots ⟗ T_N$. The joint probability distribution is defined on all attributes of all tables in the dataset:

$$P(T) = P(T_1.A_1, T_1.A_2, \cdots, T_N.A_k) \tag{1}$$

3 BACKGROUND ON DEEP AUTOREGRESSIVE MODELS

Decomposition of Joint Distribution We formulate selectivity estimation as a joint distribution problem. If all the entries of the joint distribution are stored, we can return exact selectivity for a query. However, this is prohibitively expensive due to the huge number of entries. To factorize the joint distribution into lower dimensions to reduce space consumption, some classical methods such as histograms and Bayesian Network are based on independence assumption or conditional independence assumption. They are struggled on data with strong correlations. In this paper, we factorize the joint probability using autoregressive decomposition without any independence assumption as follows:

$$\hat{P}(A_1, \cdots, A_n) = \prod_{i=1}^{n} \hat{P}(A_i | A_1, \cdots, A_{i-1}) \tag{2}$$

Deep Autoregressive Models. AR models are feed-forward deep learning models that predict values from past values, which can be used to approximate joint distributions. Some recent research which propose to adopt AR models for selectivity estimation achieve accurate results empirically [17, 49, 54, 55], We use AR models as the distribution estimator. We follow the previous work [54, 55] to use ResMADE [9] as the AR model since ResMADE has been demonstrated to have a good trade-off on accuracy and efficiency [55]. Our proposed technique also supports other AR models but exploring them is not our focus.

Model Training. The input of the AR model is all tuples in a dataset, and the output is the estimated probability distribution. During the training, the weights (or parameters) of the AR model are learned from data by minimizing the cross-entropy loss [20] between the true distribution P and the estimated distribution $\hat{P}$, given in Equation 3.

$$loss_{AR} = CrossEntropy(P, \hat{P}) = -\frac{1}{|T|} \sum_{\boldsymbol{t} \in T} P(\boldsymbol{t}) \log \hat{P}(\boldsymbol{t}) \quad (3)$$

The model parameters are optimized by minimizing the loss using back propagation [3] with stochastic gradient-based methods [27].

To train an AR model on T, we need to encode tuples in T as inputs of the model and decode model outputs to get probability distribution. We adopt the encoding and decoding strategies of previous work [54, 55].

Encoding Strategy. For each input attribute A_i for the AR model, we map its domain values into a range of integer values: $[0, |A_i|)$. The mapping will keep the original order of domain values. For example, if $Dom(A_i) = \{\texttt{dog}, \texttt{cat}, \texttt{monkey}\}$, the encoding will be $\texttt{dog} \rightarrow 1$, $\texttt{cat} \rightarrow 0$, $\texttt{monkey} \rightarrow 2$, which follows the lexicographical order.

Decoding Strategy. For each encoded attribute A_i, the output layer will allocate $|A_i|$ dimensional vector for the distribution $\hat{P}(a_i|\boldsymbol{a}_{<i})$. For example, for $Dom(A_i) = \{\texttt{dog}, \texttt{cat}, \texttt{monkey}\}$, an example of the output distribution is $\langle 0.1, 0.3, 0.6 \rangle$.

Query Estimation. The AR model is a point estimator for each tuple $\boldsymbol{t}$. It will be much harder for an AR model to answer range queries. Given a range query q that $R = R_1 \wedge R_2 \wedge \cdots \wedge R_n$, a naive method for selectivity estimation on q is exhaustive numeration: $estsel(q) = \sum_{\boldsymbol{t} \in R} \hat{P}(\boldsymbol{t})$, where $\boldsymbol{t} \in R$ represents all tuples in R. Unfortunately, this is prohibitively expensive since R can contain $O(\prod_i |A_i|)$ in the worst case. To address this, previous work [54, 55] use progressive sampling. Progressive sampling samples each tuple according to the attribute order by focusing on the high probability values. In a nutshell, to get a sample $\boldsymbol{s}$ from R, we sequentially sample s_i from $\hat{P}(A_i|A_i \in R_i, \boldsymbol{s}_{<i})$, where $\hat{P}(A_i|A_i \in R_i, \boldsymbol{s}_{<i})$ is the probability distribution of A_i in R_i given $\boldsymbol{s}_{<i}$. It could be obtained by inputting $\boldsymbol{s}_{<i}$ into the AR model. With progressive sampling, a tuple with higher probability in R is more likely to be sampled. Selectivity estimation of one sample is $\prod_i \hat{P}(A_i \in R_i|\boldsymbol{s}_{<i})$. The final estimation from multiple samples can be obtained by averaging the selectivity estimation of all samples. Progressive sampling is unbiased for selectivity estimation with AR models [55].

Join Queries. AR models can also be used to estimate selectivities of join queries [54]. We follow the methods in previous work [24, 54] to learn joint distribution on join results. Specifically, IAM captures the correlations across joins in a join schema of relations in a single AR model, and employs Exact Weight algorithm [57] to generate unbiased samples of the full outer join, which are fed into the AR model to learn the joint distribution. Interested readers may refer to Neurocard [54] for details.

4 PROPOSED MODEL

Section 4.1 gives an overview of our proposed model. We present the technique for fitting data in GMMs to reduce domain sizes in Section 4.2. Then we present our model that integrates an AR model and GMMs in Section 4.3.

4.1 Overview

Challenges. As discussed in Section 3, we use progressive sampling for estimation on the deep autoregressive models and the sampling space can be up to $\prod_i |A_i|$. For continuous attribute A_i with a large domain size, e.g., spatial coordinates, the domain size $|A_i|$ can be as large as $|T|$ when there are no duplicated values in

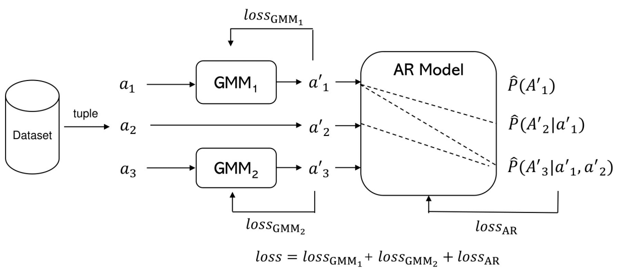


Figure 1: Overview of IAM.

A_i. With the huge sample space, the AR models will suffer in two aspects. 1) The AR models will become inaccurate in selectivity estimation since the number of samples cannot be too large, e.g., 10K, in practice. 2) It is time-consuming to perform query inference for selectivity estimation. Neurocard [54] proposes to use column factorization to reduce model size dramatically, but the sample space size is not reduced. Hence Neurocard suffers from the two issues as well.

Overview of high-level idea. To address the challenges, we propose to use a machine learning method to fit (or preprocess) a continuous attribute, and produce a new set of attributes with much smaller domain sizes. Then we learn an AR model with the new attributes. To the best of our knowledge, the idea has not been explored for modeling data distribution in either machine learning or database literature.

Specifically, we propose a new model that integrates GMMs into an AR model, which is called IAM. Figure 1 shows the framework of IAM. IAM uses GMMs as a preprocessing to get a new set of attribute values (left part in Figure 1), i.e., we leverage GMMs to "cluster" the values of an attribute and use their corresponding component indexes of GMMs to replace the original values. The idea has several advantages: 1) Domain size of a continuous attribute fitted by a GMM can be dramatically reduced to the number of components (e.g., from millions to dozens in our experiments). For GMMs, we only need to store a few parameters, i.e., weights, means and covariances. Hence, the model size becomes very small and the training of the AR model converges much better. 2) As the domain size is decreased, the sample space in query inference of the AR model becomes much smaller. Therefore, selectivity estimation will be more efficient and more accurate. 3) The component indexes of GMMs are used as input attribute values to the AR model, and this enables us to integrate GMMs and an AR model. During model construction as shown in Figure 1, for each tuple in the dataset, IAM has two steps: 1) We use GMMs to fit continuous attributes to reduce their domain sizes; 2) Tuples composed by the new attribute values are fed into the AR model to learn the data distribution.

Alternatives. We proceed to discuss several alternative methods to model continuous variables. First, equi-depth histograms [34] learn the data distribution by splitting data into buckets where each bucket contains same number of data points, and thus we can reduce the domain size to the number of buckets. Second, spline based histograms [35] construct a spline with a given number of data points to approximate the CDF with minimal maximum error. Third, uniform mixture models (UMMs) learn multiple overlapping buckets with width adjusting, so it is more flexible compared with histograms. QuickSel [37] learns a UMM with training queries. We integrate equi-depth histograms, splines and UMMs into AR models. However, the performance of these methods are not comparable with IAM. We present the experimental results in Section 6.6. Finally, we also consider KDE-based methods [22, 26] that use the kernel functions to define the shape of local probability distributions. However, KDE-based methods usually use at least thousands of samples to construct the Gaussian kernels, and for each tuple in a given relation we need to consider all the kernels. Hence, the estimation is very inefficient, and we do not use it.

Remarks. DeepSpace [49] and QuickSel [37] also adopt mixture models in a different way from IAM. We next clarify the differences. (1) IAM **vs. DeepSpace [49].** DeepSpace uses GMMs in a very different way from IAM. On the one hand, DeepSpace feeds the original data values into a AR model and outputs the distribution of the original data, which is the same as Neurocard in this sense. However, IAM takes GMMs as a preprocessing method for the continuous attributes to reduce their domain sizes. Then IAM feeds values of the reduced domain into a AR model and outputs the data distribution with reduced domain. The difference between DeepSpace and Neurocard is that the AR model of DeepSpace outputs GMM parameters while Neurocard outputs a vector to represent the learned probability for continuous attributes. Hence, the domain sizes of the continuous attributes and the search space size in the AR model of DeepSpace is the same with those of Neurocard, but much larger than those of IAM. (2) IAM **vs. QuickSel [37].** QuickSel learns uniform mixture models to fit the data distribution from training queries. In contrast, our model uses mixture models to fit some attributes and cluster them to reduce their domain sizes.

4.2 Fitting Continuous Attributes using Gaussian Mixture Models

We consider alternative design choices for using GMMs to fit continuous attributes, and we next present our key design choices.

One Gaussian Mixture Model for One Attribute. We use one GMM to fit one attribute as shown in Figure 1 for two reasons.

- A GMM can be used to fit either one attribute or multiple attributes. However, fitting multiple attributes with one mixture model will be less memory efficient. Consider fitting n attributes with GMMs. If we use one GMM, the covariance matrix will take $O(n^2)$ memory; If we use n GMMs, each for an attribute, all the n covariance matrices will only take $O(n)$ memory.
- AR models are also capable of learning the correlations of attributes. Therefore, it would not be necessary to explore the ability of GMMs to capture the correlations among attributes, although it can capture correlations when being used to fit multiple attributes. Our preliminary experiments on comparing with the option of using GMMs to fit with multiple attributes show that it did not have better estimation accuracy.

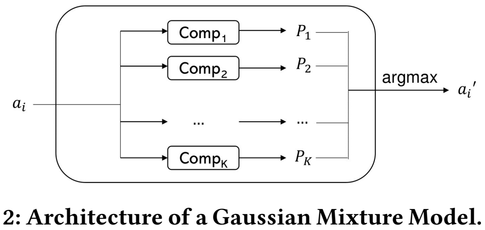


Figure 2: Architecture of a Gaussian Mixture Model. Input an attribute value of A_i and output a value of the new attribute.

Using Gaussian Mixture Models to Reduce Domain Sizes. Now we consider an attribute A_i that is fitted by a GMM with K components in Figure 2. For an attribute value a_i of A_i from a tuple $\boldsymbol{t}$, the GMM computes its new attribute value as follows: 1) a_i is fed into each component of GMM to compute $P_1, \cdots, P_K$, where P_k is the probability that a_i is in the k-th component of GMM. 2) we choose the index of the component with the maximum probability as the new attribute value for the input value, i.e., $a_i' = \text{argmax}\{P_1, \cdots, P_K\}$. Then a_i' will be used as a value of input tuple for the AR model. Compared to directly feeding original attribute value a_i into the AR model, the input layer size and output layer size for the new value a_i' is reduced from $|A_i|$ to K and the size of sample space for this attribute in query inference is reduced as well.

An alternative method would be sampling the component id based on the probabilities $P_1, \cdots, P_K$ as the new attribute value. In contrast, we choose the component index with the maximum probability for the following two considerations.

- **Training efficiency.** Consider feeding one tuple of a dataset to GMMs. If we choose the component index with maximum probability, we get one tuple after applying GMMs on the tuple, and take it as a input to the AR model. However, if we sample component ids based on the probabilities of a GMM, and generate q samples for each GMM, we get q output values from each GMM. Then we need to choose a output value from each GMM and concatenate them into new tuples as inputs to the AR model. Hence, we will have q^p new tuples if we have p GMMs. In this case, one tuple from the dataset will generate q^p tuples as the input to the AR model. The total number of tuples for the AR model will be very large, which is inefficient for training.
- **Accuracy.** The output probabilities from the GMMs are different after each epoch of training if we use the aforementioned sampling strategy. Since we use the output of the GMMs as the input data for the AR model, the input data are different in each epoch of training and thus our model will become more difficult to converge. However, when we only use the component index with maximum probability, the GMMs could converge fast, making the AR model converge fast too.

In our preliminary experiments, we find the two methods achieve similar accuracy in query inference while the first method is more efficient. Therefore, we opt the first method in IAM.

When to Use Gaussian Mixture Models. Another problem is which attributes do we use GMMs. We have the following design choices for IAM.

- For a continuous attribute with a large domain size, we use GMM to fit it. Empirically, we use GMM when the domain size is larger than 1000. Our empirical study in Section 6 shows that our method is robust to various skewness of data.
- For categorical attributes, we do not use GMMs to fit when measuring distance between two distinct values in a categorical attribute is not easy. In this case, we support column factorization [54] on large domain categorical attributes to reduce model size by following Neurocard. Suppose domain size of attribute A is 10^{2n}, we could factor it into two attributes $subA_1$ and $subA_2$ of domain size 10^n. Then the model size is reduced from 10^{2n} to 2×10^n. This factorization does not loss any information since it is also based on chain rule: $P(A) = P(subA_1, subA_2) = P(subA_1)P(subA_2|subA_1)$.

To determine the number of components K for mixture models, we can use Variational Bayesian Gaussian Mixture (VBGM) [51] to get the number of components.

Model Training. Now we discuss the training procedure for one GMM. We first present why the classic EM algorithm is not suitable for IAM, and then present our method.

Expectation–maximization (EM) algorithm is a typical method to estimate parameters of GMM by iteratively finding maximum likelihood [44]. Given a set of values from attribute A_i to fit a GMM with K components, we first initialize weights $\phi_{k=1\cdots K}$, means $\mu_{k=1\cdots K}$ and covariances $\Sigma_{k=1\cdots K}$. Each iteration includes two steps: 1) Expectation step (E step). Calculate the probability $\hat{P}_k(a_i)$ for each value of A_i in each component k based on current parameters. 2) Maximization step (M step). Update three parameters of GMM based on their maximum likelihood estimation. We iteratively execute E step and M step until convergence. In the M step, the parameters can be only updated based on all tuples. However, this is not suitable for IAM since we want to integrate mixture models with an AR model and use batches of tuples to update the model parameters, which is a common method in deep learning models. Hence we use a different algorithm.

Our algorithm is inspired by KeOps [6]. Given a set of values from attribute A_i, suppose we want to fit these values into a GMM with K components. GMM takes weights $\phi_{k=1\cdots K}$, means $\mu_{k=1\cdots K}$ and covariances $\Sigma_{k=1\cdots K}$ of the components as parameters. GMM initializes $\phi_{k=1\cdots K}$, $\mu_{k=1\cdots K}$ and $\Sigma_{k=1\cdots K}$ by VBGM. We only use uniform samples from dataset. Hence, the initialization is efficient. In each training epoch, GMM takes each value as input and fits the data by maximum likelihood estimation. GMM uses negative log likelihood loss as the objective function:

$$loss_{\text{GMM}}(A_i) = \frac{1}{|T|} \sum_{a_i \in A_i} -\log\left(\sum_{k=1}^{K} \phi_k \mathcal{N}(a_i|\mu_k, \Sigma_k)\right), \quad (4)$$

where $\mathcal{N}(a_i|\mu_k, \Sigma_k)$ is the probability of a_i in k-th Gaussian component.

The loss function can be optimized by stochastic gradient descent (SGD) [27] with back propagation [3] on batches of tuples. Since SGD with batches of tuples is also used to optimize the AR model, we can seamlessly integrate GMMs and the AR model in our training algorithm. After training, we replace the input attribute value a_i with its new value a'_i, which is the component id with the maximum probability:

$$a'_i = \underset{k\in\{1\cdots K\}}{\text{argmax}} \,(P_k(a_i)) = \underset{k\in\{1\cdots K\}}{\text{argmax}} \,(\phi_k \mathcal{N}(a_i|\mu_k, \Sigma_k)) \quad (5)$$

We use A'_i to represent the new attribute generated from A_i. For an attribute A_i fitted by a GMM, the domain size of A'_i is K, which is much smaller than the domain size of A_i ($|A'_1| \ll |A_1|$ and $|A'_3| \ll |A_3|$ in Figure 1). For an attribute A_i that is not fitted by any GMM, the new attribute A'_i has the same domain as A_i ($Dom(A'_2) = Dom(A_2)$ in Figure 1). Our model may employ multiple GMMs, and we train them in parallel.

4.3 Integration of Gaussian Mixture Models and a Deep Autoregressive Model

We proceed to present the proposed method IAM that integrates GMMs into an AR model.

Model Architectures. Figure 1 illustrates IAM. IAM consists of two parts. 1) First, we use GMMs to fit continuous attributes in a relation T in parallel by following the approach in Section 4.2. 2) Second, We concatenate the output of all GMMs ($a'_1$and a'_3 in Figure 1) and attribute values not in any GMM (a'_2 in Figure 1) as a new tuple ($\langle a'_1, a'_2, a'_3\rangle$ in Figure 1). Then we take the new tuple as a input into an AR model as we discussed in Section 3. For example, as shown in Figure 1, suppose $\{|A_1| = 100, |A_2| = 3, |A_3| = 50\}$ and $K_{\text{GMM}_1} = 10, K_{\text{GMM}_2} = 5$. After encoding, the new attributes domain will be $\{Dom(A'_1) = \{0, 1, \cdots, 9\}, Dom(A'_2) = \{0, 1, 2\}, Dom(A'_3) = \{0, 1, \cdots, 4\}\}$. Then we feed the encoded tuples into an AR model.

Separate Training. A straightforward idea of training would be first training multiple GMMs for continuous attributes to reduce their domain sizes, and then training a AR model with the new tuples. Unfortunately, training these models separately is inefficient and could be less optimal. For deep learning algorithms, it is ideal that all parameters of the model can be simultaneously trained for one loss function since this is normally much beneficial than training model separately. To achieve this, we propose a joint end-to-end training method that combines multiple GMMs and an AR model into one model.

Joint Training Algorithm. We illustrate our end-to-end joint training procedure with the example in Figure 1. Suppose that we have a set of tuples T with attributes A_1, A_2, A_3, where A_1 is a continuous attribute to be fitted into GMM$_1$, A_2 is a categorical attribute, and A_3 is a continuous attribute to be fitted into GMM$_2$. The forward propagation of our model training consists of two steps. (1) First, for each tuple $\boldsymbol{t} = \langle a_1, a_2, a_3\rangle$ in relation T, the attribute value a_1 is fed into GMM$_1$ and we get a new value a'_1 based on Equation 5; the attribute value a_3 is fed into GMM$_2$ and we get a new value a'_3; a_2 is not fed into any GMM, and $a'_2 = a_2$. Then tuple $\boldsymbol{t}$ is reduced to a new tuple $\boldsymbol{t'} = \langle a'_1, a'_2, a'_3\rangle$. Notice that the calculation in GMM$_1$ and GMM$_2$ can be done in parallel. (2) Then $\boldsymbol{t'}$ is fed into an AR model and we get the estimation $\hat{P}(A'_1)$, $\hat{P}(A'_2|a'_1)$ and $\hat{P}(A'_3|a'_1, a'_2)$. Note that $\hat{P}(A'_i|\boldsymbol{a'}_{<i})$ is different from $\hat{P}(A_i|\boldsymbol{a}_{<i})$, since different a_1 and a_3 might generate same a'_1 and a'_3 after fed into GMM$_1$ and GMM$_2$. Total loss of the joint model IAM is the sum of all the GMMs loss and the AR model loss:

$$loss = \sum loss_{\text{GMM}} + loss_{\text{AR}}, \quad (6)$$

where $loss_{\text{GMM}}$ and $loss_{\text{AR}}$ can be computed by Equation 4 and Equation 3, respectively. We use SGD [27] with back propagation [3] on batches of tuples to minimize the total loss.

Column Order. Previous work [55] shows the left-to-right order is effective for the AR models. We evaluate the impact of column order on IAM in our experiments and get similar result as [55].

Support of Join Queries. We use the method discussed in Section 3 to train for full outer join, which follows Neurocard. Since the continuous attributes fitted by GMMs are usually not join keys, we train the AR model on join tuples that contain new attributes values from GMMs by following Neurocard.

5 QUERY INFERENCE

We proceed to present the query inference method of IAM for selectivity estimation. We focus on queries that contain range predicates on continuous attributes, as well as both point and range predicates on categorical attributes. It is trivial for estimating selectivities of point predicates on continuous attributes, as discussed in Section 2. We first discuss the query construction in Section 5.1. Next we present an unbiased progressive sampling algorithm for IAM in Section 5.2. Finally we present the query inference procedure in Section 5.3.

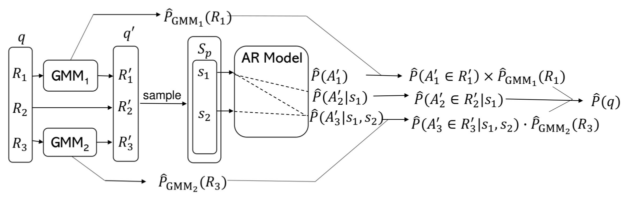

$\hat{P}_{\text{GMM}_1}(R_1)$
q
q'
S_p
AR Model
R_1
GMM$_1$
R'_1
s_1
$\hat{P}(A'_1)$
$\hat{P}(A'_1 \in R'_1) \times \hat{P}_{\text{GMM}_1}(R_1)$
R_2
sample
R'_2
$\hat{P}(A'_2|s_1)$
$\hat{P}(A'_2 \in R'_2|s_1)$
$\hat{P}(q)$
s_2
$\hat{P}(A'_3|s_1, s_2)$
$\hat{P}(A'_3 \in R'_3|s_1, s_2) \cdot \hat{P}_{\text{GMM}_2}(R_3)$
R_3
GMM$_2$
R'_3
$\hat{P}_{\text{GMM}_2}(R_3)$

Figure 3: Overview of query inference. GMMs and AR model follow Figure 1.

5.1 Query Construction

We use the notations from Figure 1 and Section 4. Figure 3 gives the overview of query inference. Consider a query q with query range $R = R_1 \wedge R_2 \wedge \cdots \wedge R_n$, where $R_i \subseteq Dom(A_i)$. Since IAM uses the AR model to learn the joint distribution, a natural idea is following previous work [54, 55] to use progressive sampling for estimating q. However, we learn the joint distribution $\hat{P}(A'_1, \cdots, A'_n)$ but not $\hat{P}(A_1, \cdots, A_n)$ as discussed in Section 4.3. Hence, we need to transform query q with query range $R = (R_1 \subseteq Dom(A_1)) \wedge \cdots \wedge (R_n \subseteq Dom(A_n))$ to a query q' with query range $R' = (R'_1 \subseteq Dom(A'_1)) \wedge \cdots \wedge (R'_n \subseteq Dom(A'_n))$.

Recall that for the input attributes $\{A'_1, \cdots, A'_n\}$ of the AR model, some of them are reduced by GMMs and others are same as original attributes. Hence, we construct a query q' with $R' = R'_1 \wedge R'_2 \cdots \wedge R'_n$ for IAM by the following two rules.

- For attribute A'_i that is the same as A_i, we set $R'_i = R_i$.
- For attribute A'_i reduced from A_i by a GMM, each distinct value in $Dom(A'_i)$ represents a component index in the GMM and R_i might intersect with any components in the GMM. Hence, we set $R'_i = Dom(A'_i)$.

For example, the constructed query q' in Figure 3 is $R' = R'_1 \wedge R'_2 \wedge R'_3$, where $R'_1 = Dom(A'_1)$, $R'_2 = R_2$, and $R'_3 = Dom(A'_3)$.

5.2 Unbiased Progressive Sampling Algorithm

Challenge to Vanilla Progressive Sampling Algorithm. With the constructed query q', a natural idea is to use the progressive sampling algorithm discussed in Section 3 to estimate $\hat{P}(q') = \hat{P}(R'_1 \wedge R'_2 \cdots \wedge R'_n)$. However, our goal is estimating $\hat{P}(q) = \hat{P}(R_1 \wedge R_2 \wedge \cdots \wedge R_n)$. For the attribute A'_i reduced from A_i by a GMM, we define $R'_i = Dom(A'_i)$, but $R_i \neq Dom(A'_i)$. Hence, sampling s_i from $\hat{P}(A'_i|A'_i \in R'_i, \mathbf{s}_{<i})$ is biased.

To address the challenge, we propose to use the probability of R_i on each component of the GMM to correct the bias. We denote this probability as a K-dimensional vector $\hat{P}_{\text{GMM}}(R_i) = \langle \hat{P}^1_{\text{GMM}}(R_i), \cdots, \hat{P}^K_{\text{GMM}}(R_i) \rangle$, where K is the number of components in the GMM and $\hat{P}^k_{\text{GMM}}(R_i)$ is cumulative probability of R_i on k-th Gaussian component. We use a sampling method to calculate $\hat{P}^k_{\text{GMM}}(R_i)$. First, we randomly generate S samples from each Gaussian distribution in the GMM, where S is a hyper-parameter. Second, we calculate the number of samples S_k that locate in the query region R_i for each component $k = 1 \cdots K$. Then we get $\hat{P}^k_{\text{GMM}}(R_i) = \frac{S_k}{S}$. Notice that the first step is a one-time preprocessing that can be done before any query is processed.

High-level Idea for Unbiased Progressive Sampling. Now we present the unbiased progressive sampling algorithm for IAM. When we sequentially generate a sample $\mathbf{s} = \langle s_1, s_2, \cdots, s_n \rangle$, there are two situations for each s_i.

- If attribute A'_i is directly from A_i, we sample $s_i \sim \hat{P}_{AR}(A'_i|A'_i \in R'_i, \mathbf{s}_{<i})$ and store the probability $\hat{P}_{AR}(A'_i \in R'_i|\mathbf{s}_{<i})$.
- If attribute A'_i is reduced from A_i by a GMM, we first correct the bias: $\hat{P}(A'_i|A_i \in R_i, \mathbf{s}_{<i}) = \hat{P}_{AR}(A'_i|A'_i \in R'_i, \mathbf{s}_{<i}) \times \hat{P}_{\text{GMM}}(R_i)$. Then we sample $s_i \sim \hat{P}(A'_i|A_i \in R_i, \mathbf{s}_{<i})$ and store the probability $\hat{P}(A_i \in R_i|\mathbf{s}_{<i})$.

where $\hat{P}_{AR}(A'_i|A'_i \in R'_i, \mathbf{s}_{<i})$ can be obtained from the output of the AR model. The selectivity estimation of q based on $\mathbf{s}$ is $\prod_i \hat{P}(A_i \in R_i|\mathbf{s}_{<i})$. The final selectivity estimation of q can be obtained by averaging selectivity estimation of multiple samples.

We use the example in Figure 3 to illustrate the procedure of drawing a sample $\mathbf{s}$ as following:

- Forward $\mathbf{0}$ into the AR model and get $\hat{P}_{AR}(A'_1)$. Feed R_1 into GMM$_1$ and get $\hat{P}_{\text{GMM}}(R_1)$. Calculate $\hat{P}(A'_1|A_1 \in R_1) = \hat{P}_{AR}(A'_1|A'_1 \in R'_1) \times \hat{P}_{\text{GMM}}(R_1)$ to correct the bias. Sample $s_1 \sim \hat{P}(A'_1|A_1 \in R_1)$. Compute and store $\hat{P}(A_1 \in R_1)$.
- Forward $\langle s_1 \rangle$ into the AR model and get $\hat{P}_{AR}(A'_2|\langle s_1 \rangle)$. $\hat{P}(A'_2|A_2 \in R_2, \langle s_1 \rangle) = \hat{P}(A'_2|A'_2 \in R'_2, \langle s_1 \rangle)$. Sample $s_2 \sim \hat{P}(A'_2|A_2 \in R_2, \langle s_1 \rangle)$. Compute and store $\hat{P}(A_2 \in R_2|\langle s_1 \rangle)$.
- Forward $\langle s_1, s_2 \rangle$ into the AR model and get $\hat{P}_{AR}(A'_3|\langle s_1, s_2 \rangle)$. Feed R_3 into GMM$_2$ and get $\hat{P}_{\text{GMM}}(R_3)$. Calculate $\hat{P}(A'_3|A_3 \in R_3, \langle s_1, s_2 \rangle) = \hat{P}_{AR}(A'_3|A'_3 \in R'_3, \langle s_1, s_2 \rangle) \times \hat{P}_{\text{GMM}}(R_3)$ to correct the bias. Compute and Store $\hat{P}(A_3 \in R_3|\langle s_1, s_2 \rangle)$.

The probability of $\mathbf{s}$ is $\hat{P}(A_1 \in R_1) \times \hat{P}(A_2 \in R_2|\langle s_1 \rangle) \times \hat{P}(A_3 \in R_3|\langle s_1, s_2 \rangle)$.

Theorem 5.1. *The aforementioned progressive sampling on IAM is unbiased.*

For a sample $\mathbf{s}$, consider sampling s_i for attribute A'_i. There are two situations. 1) Attribute A'_i is not generated from any GMM. Then A'_i is same as A_i and $R'_i = R_i$. Hence, $\hat{P}(A'_i|A_i \in R_i, \mathbf{s}_{<i}) = \hat{P}_{AR}(A'_i|A'_i \in R'_i, \mathbf{s}_{<i})$. Drawing s_i from $\hat{P}(A'_i|A_i \in R_i, \mathbf{s}_{<i})$ has been proved unbiased in [55]. 2) Attribute A'_i is generated from a GMM. We consider the range $R_i = \bigcup_k R^k_i$, where R^k_i is the subset of R_i contained in the k-th component. Hence

$$\begin{aligned}\hat{P}_{\text{GMM}}(A'_i) &= \langle \hat{P}_{\text{GMM}}(R^1_i), \cdots, \hat{P}_{\text{GMM}}(R^K_i) \rangle \\ &= \langle \frac{s(R^1_i)}{s(A'_i = 1)}, \cdots, \frac{s(R^K_i)}{s(A'_i = K)} \rangle\end{aligned},$$

where $s(R^k_i)$ is the number of tuples in R^k_i and $s(A'_i = k)$ is the number of tuples in the k-th component, which also represents the number of tuples with $A'_i = k$. Since we define $R'_i = Dom(A'_i)$, we have

$$\begin{aligned}\hat{P}_{\text{AR}}(A'_i|A'_i \in R'_i, \mathbf{s}_{<i}) &= \hat{P}_{\text{AR}}(A'_i|A'_i \in Dom(A'_i), \mathbf{s}_{<i}) \\ &= \langle \frac{s(A'_i = 1, \mathbf{s}_{<i})}{s(\mathbf{s}_{<i})}, \cdots, \frac{s(A'_i = K, \mathbf{s}_{<i})}{s(\mathbf{s}_{<i})} \rangle\end{aligned}$$

Since $s(\mathbf{s}_{<i})$ is the number of tuples given $\mathbf{s}_{<i}$, we consider for any k, $\frac{s(A'_i=k, \mathbf{s}_{<i})}{s(A'_i=k)} = C$, where C is a constant that only depends on $\mathbf{s}_{<i}$. Hence

$$\begin{aligned}&\hat{P}_{\text{AR}}(A'_i|A'_i \in R'_i, \mathbf{s}_{<i}) \times \hat{P}_{\text{GMM}}(A'_i) \\ =&\langle \frac{s(R^1_i)}{s(A'_i = 1)}, \cdots, \frac{s(R^K_i)}{s(A'_i = K)} \rangle \times \langle \frac{s(A'_i = 1)}{s(\mathbf{s}_{<i})}, \cdots, \frac{s(A'_i = K)}{s(\mathbf{s}_{<i})} \rangle \\ =&C\langle \frac{s(R^1_i)}{s(\mathbf{s}_{<i})}, \cdots, \frac{s(R^K_i)}{s(\mathbf{s}_{<i})} \rangle = C\hat{P}(A'_i|A_i \in R_i, \mathbf{s}_{<i})\end{aligned}$$

Therefore, drawing s_i from $C\hat{P}(A'_i|A_i \in R_i, \mathbf{s}_{<i})$ is unbiased.

5.3 Query Inference with Unbiased Progressive Sampling

Overview of Query Inference. Figure 3 demonstrates an overview of query inference based on the model in Figure 1. There are two steps to estimate a query. The first step is initialization, i.e., constructing q' for IAM. The second step is the proposed progressive sampling. For each sample $\mathbf{s}$, we sequentially sample $s_1, s_2, \cdots, s_n$ and get the selectivity estimation based on the method in Section 5.2.

Query Inference Procedure. Now we outline the query inference procedure for a query q in Algorithm 1. We first initialize the final result as 0 (line 1). In step 1, we construct a query q' (line 2) based on the method discussed in Section 5.1 (e.g., $R'_1 = Dom(A'_1)$, $R'_2 = R_2$ and $R'_3 = Dom(A'_3)$ in Figure 3). In step 2, we follow the unbiased progressive sampling algorithm in Section 5.2 to generate multiple samples. For each sample $\mathbf{s}$, we first initialize it as an all 0 vector (line 4) and set its probability $\hat{p}$ as 1 (line 5). Then we sequentially sample $s_1, s_2, \cdots, s_n$ for each attribute. For the i-th attribute, we feed $\mathbf{s}$ into the AR model (line 7) and get the probability distribution $\hat{P}_{\text{AR}}(A'_i|A'_i \in R'_i, \mathbf{s}_{<i})$ (lines 8-9). If A'_i is reduced from a GMM, we first calculate S_k for each component in the corresponding GMM based on method in Section 5.2 and get $\hat{P}_{\text{GMM}}(R_i) = \langle \frac{S_1}{S}, \cdots, \frac{S_K}{S} \rangle$ (line 11). Then we use $\hat{P}_{\text{GMM}}(R_i) \times \hat{P}_{\text{AR}}(A'_i|A'_i \in R'_i, \mathbf{s}_{<i})$ to correct the bias (line 12). If A'_i is not reduced by any GMM, we directly use $\hat{P}_{\text{AR}}(A'_i|A'_i \in R'_i, \mathbf{s}_{<i})$ (line 14). Then we multiply the range probability $\hat{P}(A_i \in R_i|\mathbf{s}_{<i})$ with $\hat{p}$ to update $\hat{p}$ (line 15) and draw s_i from $\hat{P}_{\text{AR}}(A'_i|A'_i \in R_i, \mathbf{s}_{<i})$ (lines 16-17). After multiple samples are generated in parallel, the final result can be obtained by averaging the selectivity estimation of all samples (lines 18–19).

Handling Unqueried Columns. We use *wildcard-skipping* [55] method which uniformly masks a subset of columns and replaces their original values in the tuple with a special token as the training data for IAM. This method can reduce query inference time by skipping progressive sampling for the unqueried columns.

Batch Query Inference. IAM supports batch query inference. Given a batch query size b and the number of samples S_p in progressive sampling, we initialize S_p vectors as samples for each query. In each iteration, we consider the $S_p \times b$ samples as a batch and do unbiased progressive sampling for IAM.

6 EXPERIMENTS

We conduct experiments to evaluate the following aspects:

- **Accuracy.** We evaluate the accuracy of IAM and compare with various baselines including state-of-the-art baselines.
- **Inference Time.** We compare the inference time of IAM with baselines.
- **End-to-End Time.** We evaluate how IAM impact on a query optimizer.
- **Training Time.** We show how maximum error changes with the increase of the number of epochs for model training.
- **Alternative Domain Reducing Methods.** We compare GMM with 3 alternative domain reducing methods.
- **Number of Mixture Components.** We evaluate the effect of the number of components in the GMMs on accuracy.
- **Impact of GMM Sample Number.** We evaluate the impact of the number of GMM sample on accuracy and estimation time.

We also evaluate the impact of unbiased sampling algorithm, the impact of hyper-parameters and the impact of data and query

Algorithm 1: Query Inference on IAM

```
Input: Query $q = \langle R_1, \cdots, R_n \rangle$, an IAM model, $S$ samples
       drawn from each Gaussian component in each
       GMM, number of samples $S_p$ in progressive
       sampling;
Output: Selectivity estimation of $q$;
1  $\hat{P} = 0$;
2  $q' : R'_q = \langle R'_1, \cdots, R'_n \rangle$ ;                         // Step 1
3  for $j = 1$ to $S_p$ do                                 // Step 2
4      $\mathbf{s} = \mathbf{0}_n$;
5      $\hat{p} = 1$;
6      for $i = 1$ to $n$ do
7          Feed s into AR model;
8          $\hat{P}_{\text{AR}}(A'_i|\mathbf{s}_{<i})$ = the $i$-th output of AR model;
9          Obtain $\hat{P}_{\text{AR}}(A'_i|A'_i \in R'_i, \mathbf{s}_{<i})$ from $\hat{P}_{\text{AR}}(A'_i|\mathbf{s}_{<i})$ by
             renormalizing;
10         if $A'_i$ is from GMM then
11             Calculate $\hat{P}_{\text{GMM}}(R_i)$;
12             $\hat{P}(A'_i|A_i \in R_i, \mathbf{s}_{<i}) = \hat{P}_{\text{AR}}(A'_i|A'_i \in$
                 $R'_i, \mathbf{s}_{<i}) \times \hat{P}_{\text{GMM}}(R_i)$;
13         else
14             $\hat{P}(A'_i|A_i \in R_i, \mathbf{s}_{<i}) = \hat{P}_{\text{AR}}(A'_i|A'_i \in R'_i, \mathbf{s}_{<i})$
15         $\hat{p} = \hat{p} \times \hat{P}(A_i \in R_i|\mathbf{s}_{<i})$;
16         Draw sample $s_i \sim \hat{P}(A'_i|A_i \in R_i, \mathbf{s}_{<i})$;
17         $\mathbf{s}[i] = s_i$
18     $\hat{P} = \hat{P} + \hat{p}$
19 return $\frac{\hat{P}}{S_p}$
```

Table 1: Datasets in Evaluation.

Dataset	Rows	Cols.Cat	Cols.Con	Joint
WISDM	4.8×10^6	2	3	10^{21}
TWI	1.9×10^7	0	2	10^{13}
HIGGS	1.1×10^7	0	7	10^{41}
IMDB	2.0×10^{12}	13	5	10^{19}

distribution. Due to the page limitation, we report them in the technical report [25].

6.1 Experimental Setup

6.1.1 Datasets. We use three single table real-world datasets with different attribute types, column numbers and domain sizes. Table 1 describes them. "Cols.Cat" and "Cols.Con" refer to the number of categorical columns and the number of continuous columns, respectively. "Joint" is the product of all domain sizes. Note that we use datasets containing continuous attributes as IAM aims to address the challenges arising from continuous attributes.

WISDM [52]. A real-world dataset contains accelerometer and gyroscope time-series sensor data collected from phones and smartwatches of 51 test subjects. We use 5 columns with both categorical attributes and continuous attributes: `subject_id` (categorical, 51), `activity_code` (categorical, 18), `x` (continuous, 1×10^6), `y` (continuous, 1×10^6), and `z` (continuous, 1×10^6).

TWI. We use a collection of geo-tagged tweets posted in the U.S. in 2016. We use 2 continuous attributes with large domain sizes: `latitude` (continuous, $3{\times}10^6$) and `longitude` (continuous, 3×10^6). This can be considered as a typical spatial dataset.

HIGGS [2]. This dataset has 21 features which are kinematic properties measured by the particle detectors in the accelerator. Following the previous work [11] on selectivity estimation, we use 7 high-level features as attributes: `m_jj` (continuous, 1×10^6), `m_jjj` (continuous, 5×10^5), `m_lv` (continuous, 3×10^5), `m_jlv` (continuous, 5×10^5), `m_bb` (continuous, 1×10^6), `m_wbb` (continuous, 7×10^5), and `m_wwbb` (continuous, 8×10^6) in our experiments.

We use two statistical methods to measure the *correlation* and *skewness* of the datasets. Nonlinear Correlation Information Entropy (NCIE) [50] is used to measure correlations. Smaller NCIE indicates stronger correlation. The NCIEs are 0.33, 0.37. 0.67 for WISDM, TWI and HIGGS, respectively. Hence, WISDM and TWI have stronger correlations while HIGGS has a relatively weak correlation. We use Fisher's definition [60] to measure skewness. Closer to 0 indicates similar to Gaussian distribution. The skewness are 2.3, -1, 81 for WISDM, TWI and HIGGS, respectively. Hence, HIGGS has stronger skewness while WISDM and TWI have weaker skewness.

To test the accuracy of join queries and end-to-end execution time on a query optimizer, we also use the real-world IMDB dataset with extra attributes.

IMDB. Previous work [24, 29] reported that IMDB is a good dataset for selectivity estimation evaluation as it has strong attributes correlations [30]. Since IMDB dataset does not have continuous attribute with large domain size, we concatenate `x`, `y`, `z` attributes from WISDM to table `movie_info`, and `latitude`, `longitude` attributes from TWI to table `title`. The concatenated attribute values are uniformly sampled from WISDM and TWI.

6.1.2 *Estimators.* We compare IAM [1] with 11 other estimators, including several classic methods (Sampling, MHIST, BayesNet, Postgres and KDE), three query-driven supervised learning methods (UAE-Q [53],MSCN [29] and QuickSel [37]), two state-of-the-art data-driven unsupervised learning methods (DeepDB and Neurocard), and a hybrid learning method (UAE [53]). FLAT [58] ran out of memory on our datasets.

Sampling. Sampling keeps a portion of tuples sampled uniformly from the original dataset. For a fair comparison, the portion is based on the space consumption of IAM, which are 0.63%, 0.02% and 0.23% for WISDM, TWI and HIGGS respectively.

Postgres [42] (version 9.6.6). It keeps 1D histograms of data.

MHIST [41]. Multi-dimensional histogram is a classic statistical method without independence assumptions. This method is used as the state-of-the-art multi-dimensional histogram method.

BayesNet [7]. BayesNet shows high accuracy in previous work [55]. Following the previous work [55], we use Chow-Liu tree [7] as the Bayesian Network since empirically it has the best result given the allowed space. We follow the parameter setting for BayesNet used in previous work [55] for a fair comparison.

KDE [22, 26]. Similar to IAM, KDE also learns continuous data distribution but leverages Gaussian kernels. Its bandwidth is calculated based on the Scott's rule [48]. KDE also uses queries as feedback to optimize bandwidth. We use the open source code published by authors [21].

MSCN [29]. This method is based on multi-set convolutional neural networks [29]. We use the open source code published by the authors [28]. We generate 1K samples for bitmaps on each dataset and use two layers (256 hidden units) of multilayer perceptrons on the query encoding, which is its default setting [29]. We also consider the other two query-driven deep learning methods Sup [18] and Model_QE [10]. They have similar results with that of MSCN, and we will not report them.

QuickSel [37]. This method learns uniform mixture models from training queries. We use the code published by the authors [56].

DeepDB [24]. DeepDB is a recent unsupervised data-driven estimator. This method learns joint data distribution based on the Sum-Product Network (SPN) [38]. We use its open source code [23]. We set the number of samples per SPN to 1M, which follows previous work [24, 54] for a fair comparison.

Neurocard [54]. Based on Naru [55], Neurocard further supports join operator and optimizes Naru with column factorization to better handle categorical attributes with large domain sizes. IAM further optimize Neurocard to address the challenges arising from continuous attributes. We use its source code [59]. We follow the settings of its source code [59], such as hidden layer setting and the size of progressive sampling for query inference (set as 8K samples), which strikes a good balance between accuracy and efficiency.

UAE and UAE-Q [53]. UAE learns the joint data distribution from both data and queries with a deep autoregressive model. UAE-Q learns the joint data distributions from queries only.

For a fair comparison, IAM, UAE follow Neurocard in the parameter settings for neural networks, which use 4 hidden layers (256, 128, 128, 256 units). To reduce training cost, we use join sampler technique [54] to sample 10^6 tuples from the datasets as training data for IAM, UAE and Neurocard. For the columns fitted by GMMs in IAM, Neurocard and UAE handle them with column factorization. We set the maximum subcolumn size of column factorization as 2^{11}, and columns with large domain sizes are factorized into 2 subcolumns in our datasets. IAM, UAE and Neurocard do not use column factorization for other columns as the domain sizes are small. We train IAM for each dataset. We use a GMM for each continuous attribute of each dataset. The number of GMM components is set at 30 on all the datasets, and can be decided by VBGM automatically. When conducting query inference, we generate 10K samples for each component in GMM.

We tune hyper-parameters of baselines by following previous work to get good accuracy and efficiency in estimation. We run estimators, including GPU-based estimators (KDE, MSCN, Neurocard, UAE-Q, UAE and IAM) and other CPU based estimators, on a Tesla V100 Xeon E5-2698.

6.1.3 *Query Workloads.* We generate 2K testing queries for each dataset. The query generation method for single table datasets follows [55]. We first draw some attributes from the dataset. For each selected attribute A_i, if A_i is a categorical attribute, we uniformly draw a value from its domain and randomly draw an operator from $\{=, \leq, \geq\}$. If A_i is a continuous attribute, we randomly draw a value between the maximum and the minimum of its domain and uniformly draw an operator from $\{\leq, \geq\}$. Then we generate a query by the conjunction of each predicate. For example, a query in TWI could be $(\texttt{latitude} \leq 43.5) \wedge (\texttt{longitude} \geq -32.7)$. For IMDB dataset, we only generate join queries. We follow the method in Neurocard [54]. The queries are uniformly distributed to each join graph of the benchmark JOB-light [54] (18 in total). For each join graph, we draw a tuple from the inner join result. We randomly place 5–11 operators based on whether the column supports range filter or point filter. The true selectivity of a query is obtained by executing it on Postgres. We generate another 10K queries as training data in the same way for MSCN, KDE,

[1]https://github.com/mzz235711/IAM

Table 2: Estimation errors on WISDM.

Estimator	Mean	Median	95th	99th	Max
Sampling	4.15	1.06	2.57	123	821
Postgres	17.9	4.17	54.5	199	1908
MHIST	46.7	1.55	12.5	1154	8837
BayesNet	3.5	1.33	6.91	31.9	437
KDE	17.0	4.30	50.0	207	1952
DeepDB	2.72	1.14	4.75	34.0	333
MSCN	5.37	1.83	20.9	59.0	197
QuickSel	8499	51	50036	91191	193798
Neurocard	1.30	1.06	2.07	4.73	36.6
UAE	1.30	1.06	2.07	4.73	36.6
UAE-Q	3.32	1.46	6.56	34.17	384
IAM	**1.25**	**1.05**	**1.89**	**4.52**	**14.3**

Table 3: Estimation errors on TWI.

Estimator	Mean	Median	95th	99th	Max
Sampling	93.6	1.02	2.63	2119	15962
Postgres	84.1	1.12	21.6	1709	26671
MHIST	58.2	1.14	9.09	146	29431
BayesNet	185	1.04	4.50	24.9	189983
KDE	7.58	**1.00**	1.36	68	2170
DeepDB	243	1.04	68	8110	38340
MSCN	2.09	1.11	2.91	9.89	586
QuickSel	171	**1.00**	7.26	5635	30398
Neurocard	5.01	1.01	3.68	47.3	1889
UAE	**1.04**	1.01	**1.15**	**3.00**	**11.6**
UAE-Q	1.44	1.05	1.99	9.00	71.5
IAM	1.26	1.01	2.21	9.32	26.0

Table 4: Estimation errors on HIGGS.

Estimator	Mean	Median	95th	99th	Max
Sampling	24.0	1.10	128	564	2368
Postgres	11.6	2.52	49.0	146	939
MHIST	104	5.33	253	1580	20528
BayesNet	2.44	1.33	5.23	12.2	815
KDE	6.72	1.56	23.4	93	882
DeepDB	2.31	1.23	4.46	14.9	504
MSCN	2.75	1.79	7.61	15.2	49.3
QuickSel	3025	7.58	3250	54299	1092969
Neurocard	1.28	1.09	2.24	4.41	16.0
UAE	1.27	**1.07**	2.08	4.32	16.0
UAE-Q	2.72	1.83	4.70	8.70	516
IAM	**1.25**	**1.07**	**1.98**	**3.86**	**9.00**

QuickSel, UAE-Q and UAE so that the training queries and the testing queries have the same distribution.

Evaluation Metrics. We report the Q-Error [30, 55] as the accuracy metric: $error(q) = \max(\frac{actsel(q)}{estsel(q)}, \frac{estsel(q)}{actsel(q)})$. We assume $actsel(q) \geq \frac{1}{|T|}$ and $estsel(q) \geq \frac{1}{|T|}$ to avoid divided by zero. We report the error in different quantiles. We especially concern the tail cases (95%, 99% and maximum), as most existing estimators have good accuracy in median case, but they have very different performance at tail cases. In addition, we also evaluate end-to-end querying time.

Table 5: Estimation errors on IMDB.

Estimator	Mean	Median	95th	99th	Max
Postgres	164	5.00	386	3436	29800
DeepDB	260	2.14	122	1520	158226
MSCN	366	4.97	144	2506	144795
Neurocard	13.9	1.59	44.5	300	1136
UAE	13.9	1.59	44.5	300	1136
UAE-Q	137	3.59	301	4395	9933
IAM	**9.35**	**1.54**	**36**	**156**	**581**

6.2 Accuracy Evaluation

Tables 2–5 report the errors of all the estimators on the four datasets and the lowest error is in bold. We observe that IAM has significantly better accuracy than all the other estimators in the tail cases (i.e., 95th, 99th, and Max), and matches with the best estimator in terms of mean and median. Overall, IAM outperforms Sampling by 1.12−614 times, Postgres by 25.6−1026 times, BayesNet by 2.03 − 7307 times, KDE by 10.3 − 1068 times, DeepDB by 2.15−1475 times, MSCN by 1.32−22.5 times, QuickSel by 1.00 − 1.2×10^5 times and Neurocard by 1.02 − 26 times. We next discuss the main findings.

(1) Sampling has good performance in terms of median in each dataset. However, the accuracy of Sampling becomes worse in the tail cases. This is because Sampling cannot handle low-selectivity queries well, which is also reported in previous work [55].

(2) Postgres has large errors. That is because Postgres is based on the independence assumption, which results in poor accuracy when attributes have strong correlations (WISDM and TWI).

(3) MHIST outperforms Postgres on most cases because it uses multi-dimensional histograms. However, it cannot perform well on maximum error case because of its uniform spread assumption. The side effect of this assumption on MHIST is more obvious when the dataset has strong skewness (HIGGS).

(4) KDE performs poorly, particularly on datasets with large domain sizes and categorical attributes (WISDM). This may be because 1) the bandwidth tuning required by KDE is difficult with large domain sizes and 2) it is difficult to learn the distribution of discrete data with Gaussian kernels.

(5) MSCN can handle well on large domain attributes and strong correlations. However, MSCN still has up to 5.8×10^2 error at tail cases. That may be because 1) MSCN heavily relies on distribution of sampled training queries. Its performance becomes poor at tail cases because the probability that training queries hit low-selectivity data is very small; and 2) When a dataset has strong skewness, it is difficult for the training queries to represent the data distribution accurately. Hence, MSCN has relatively bad performance on WISDM.

(6) QuickSel has large errors in most cases. Similar to histogram, QuickSel makes uniform distribution assumption within a bucket, which may not hold. QuickSel has particularly bad performance on high dimensional datasets (HIGGS) since multi-dimensional buckets are hard to capture correlations.

(7) UAE-Q outperforms the other two query-driven supervised methods, even if it does not use samples as does MSCN. Similar to other query-driven methods, it does not perform well on datasets with strong skewness (WISDM and IMDB) as the training queries cannot represent the data distribution well. UAE performs better than UAE-Q by learning from both data and query workload.

(8) BayesNet has relatively good performance in most cases. However, it has large errors on maximum error cases since discretization [12] used by BayesNet will lead to information loss

for continuous attributes. Additionally, BayesNet is based on conditional independence assumption, its performance will drop quickly when datasets have strong correlations (e.g., TWI).

(9) DeepDB has good performance in terms of median. However, the accuracy degrades significantly in tail error cases when attributes have strong correlations. This is because DeepDB makes independence assumption on some nodes in SPN. In addition, DeepDB uses linear function for continuous domain in leaf nodes, which may result in extremely large errors when attributes in datesets, such as TWI, have non-linear distributions.

(10) IAM achieves the best performance on tail errors and mean errors on most datasets. Neurocard, UAE and IAM are all based on the AR model. However, IAM outperforms Neurocard by up to 26 times at tail cases, and outperforms UAE by up to 2.6 times at tail cases. The improvement could be attributed to two reasons: First, since domain sizes of continuous attributes are very large, it is difficult for Neurocard and UAE to learn distribution accurately with a reasonable model size even if Neurocard and UAE use column factorization to reduce the input and output layer sizes. In contrast, IAM can learn the distribution more accurately since the domain sizes are reduced by mixture models (e.g., which is reduced from 10^6 to 10^2 in TWI). Second, for query inference, sample space will be extremely large for Neurocard and UAE. However, sample space for IAM is much smaller since we use GMMs to reduce the domain sizes, and thus IAM has higher accuracy using the same number of samples.

We present the accuracy of join queries on IMDB for Postgres, MSCN, DeepDB, Neurocard, UAE-Q, UAE and IAM, which support join queries. As expected, Postgres has the worst accuracy on the median. IMDB has more attributes and join queries are more complicated. This indicates that it is hard for 1D-histogram to approximate complex distribution. MSCN performs better than Postgres but worse than data-driven methods. This is because we use more columns and tables for join queries, and it is more difficult for the training queries to represent the data distribution. DeepDB has better accuracy than Postgres but worse than Neurocard and IAM. This is because 1) DeepDB assumes conditional independence on table subsets, and 2) DeepDB uses linear functions to learn continuous attributes with complex distribution. IAM, Neurocard and UAE are both based on AR models. IAM outperforms Neurocard and UAE by 1 time on the 99th and the maximum. Compared with Neurocard and UAE, IAM uses GMMs for continuous attributes to reduce sample space, and thus it achieves better estimation than Neurocard and UAE.

6.3 Inference Time

6.3.1 Single Query Inference. Figure 4 shows the estimation time of all estimators with single query inference. Note that Neurocard, UAE-Q and UAE have the same procedure in query inference, and we use Neurocard in Figure 4 to represent both of them. KDE, MSCN, Neurocard, UAE-Q, UAE and IAM are GPU-based. The other estimators are CPU-based. Since the estimation time is affected by many factors (such as CPU or GPU, coding language, deep learning framework, etc.), we would need to be careful in comparing the running time.

IAM can produce estimations in reasonable time, i.e., around 10ms on single table datasets and around 40ms on IMDB. IAM is much more efficient than Neurocard in query inference for two reasons. First, the input and output layer sizes of Neurocard are larger than those of IAM since IAM uses mixture models to reduce domain sizes. The model size of Neurocard is 1–2 times larger than that of IAM as shown in Table 6. Therefore, forward computation and progressive sampling are more expensive for Neurocard. Second, for continuous attributes fitted by the GMMs in IAM, Neurocard uses column factorization to reduce the model size. However, column factorization increases the number of column, and Neurocard needs more forward computation in query inference. DeepDB is comparable with IAM on inference time. MSCN is very fast in estimation. MSCN needs scanning samples, and thus its inference time is impacted by the materialized sample ratio. QuickSel constructs buckets for all overlapped training query regions. Hence, its inference time is impacted by the size of training query workloads.

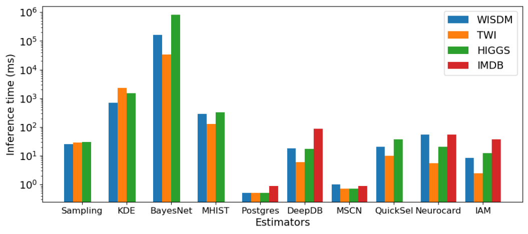


Figure 4: Inference time on each dataset.

Table 6: Model sizes of Neurocard and IAM (MB).

	WISDM	TWI	HIGGS	IMDB
MSCN	2.5	2.5	2.5	2.6
DeepDB	79	30	98	320
Neurocard	1.8	0.9	2.5	11
IAM	1.1	0.66	1.6	6.3

Table 7: Inference time with batch query processing on IMDB (ms per query). OOM refers to out of memory.

batch size	1	64	128
Model_QE	20	0.3	0.18
MSCN	0.9	0.09	0.06
Neurocard	54	14	OOM
IAM	39	9.7	7.3

6.3.2 Batch Query Inference. This experiment is to evaluate if batch query processing for Neurocard and IAM can improve performance. It is known that query-driven estimators are lightweight and typically support batch query processing, and we report the estimation time of Model_QE [10] and MSCN in this experiment for reference. Table 7 shows the estimation time of different batch sizes on IMDB. We make the following observations. 1). IAM can benefit from batch query inference, and IAM takes less than 10ms to estimate a query for batch size 64. Since the model size of IAM is smaller than the size of Neurocard, IAM benefits more from batch query inference than Neurocard. 2) MSCN and Model_QE has better speed up efficiency than Neurocard and IAM . This is because Neurocard and IAM need preprocessing on CPU for each step of progressive sampling. This preprocessing is hard to be handled with batch. Bitmap samples for each query in MSCN are also generated sequentially, and thus the estimation time changes little when batch size increases.

6.4 End-to-End Time

We proceed to evaluate the impact on query optimization since a major application of selectivity estimation is to improve the performance of query optimization. We compare the end-to-end

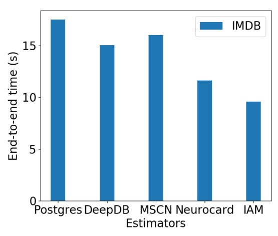


Figure 5: End-to-end time on IMDB.

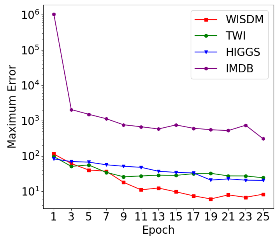


Figure 6: Training epoch vs. max error.

Table 8: Training time (s) on IMDB.

MSCN	DeepDB	Neurocard	IAM
256	636	1169	3522

query execution time of using different estimators. Following the procedure [5], we modify the query optimizer in Postgres so that it accepts external selectivity estimation. We focus on the queries on IMDB, which are more complex and challenging for query optimizers than queries on single table. For each query, modified Postgres collects selectivities of all subqueries provided by different estimators, and use them for query optimization. Figure 5 gives the average end-to-end query execution time of different estimators. Since UAE and Neurocard have similar accuracy on IMDB, we use Neurocard to represent both of them. We have two findings. First, query optimizer could generate better query plans when estimators offer more accurate selectivity estimations. IAM provides the most accurate estimation, and its query execution is the most efficient. Second, the performance gap in terms of estimation accuracy is not the same as that in terms of end-to-end time. For example, IAM outperforms Postgres by more than an order of magnitude on mean accuracy, but only improves on Postgres by 1.83 times in terms of end-to-end time. On the other hand, IAM outperforms Neurocard by 48% on mean accuracy, and improves on Neurocard by 25% in terms of end-to-end time. This is because (1) a query optimizer may give the same query plan even if the estimated selectivities are different and (2) query execution time could be similar even if query plans are different. This implies that improvement of estimators in terms of accuracy may not be able to translate into decrease on query execution time, although better estimations provide better results for approximate query processing application.

6.5 Training Time vs Estimation Error

We proceed to evaluate the impact of training epochs on estimation errors. IAM takes 40 seconds, 189 seconds, 258 seconds and 646 seconds for one training epoch on WISDM, TWI, HIGGS and IMDB, respectively. Figure 6 shows the maximum estimation error versus the number of training epochs on three datasets. We observe that IAM is efficient on training since it can converge to a good estimation performance with a few training epochs. For example, IAM can reach to convergence with about 11, 9, 19 and 7 epoches on WISDM, TWI, HIGGS and IMDB, respectively, as shown in Figure 6.

We report the training time of IAM and other learning based methods in Table 8. MSCN is very fast in training since the size of training queries is much smaller than the size of dataset (10^{12}). Although Neurocard is faster than IAM since IAM learns both GMMs and an AR model, they are in the same order of magnitude.

Table 9: Impact of domain reducing methods on WISDM.

Methods	Error Median	95th	Max	Est. time (ms)
GMM (30)	**1.05**	**1.89**	**14.3**	8.4
Hist (30)	1.19	7.19	10230	6.5
Hist (100)	1.09	2.73	2399	7.3
Hist (1000)	1.08	2.15	62	10.6
Spline (30)	1.33	7.86	10230	8.1
Spline (100)	1.13	2.51	519	8.6
Spline (1000)	1.07	1.93	110	11.7
UMM (30)	1.40	31.6	2981	7.9
UMM (100)	1.13	3.67	105	6.9
UMM (1000)	1.08	2.65	72.8	13.2

Table 10: Impact of domain reducing methods on TWI.

Methods	Error Median	95th	Max	Est. time (ms)
GMM (30)	**1.01**	**2.21**	**26.0**	4.87
Hist (30)	1.13	139	43044	3.56
Hist (100)	1.04	8.52	60797	3.4
Hist (1000)	**1.01**	2.84	28782	16.4
Spline (30)	1.11	22.8	119839	8.5
Spline (100)	1.05	8.47	60907	4.5
Spline (1000)	**1.01**	3.2	26627	17
UMM (30)	1.38	85	73108	4.5
UMM (100)	1.07	10.5	1060	3.7
UMM (1000)	1.07	2.76	351	8.3

6.6 Alternatives for Reducing Domains

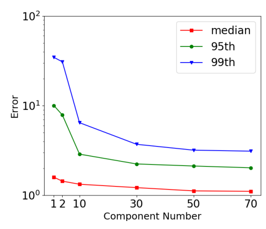
We consider the 3 alternative methods in Section 4.1 for reducing domains and integrate them into IAM. 1) Equi-depth histogram (Hist). 2) Spline based histogram (Spline) [35]. 3) Uniform mixture model (UMM). Following the previous work [35, 41] and to explore the highest accuracy, we use 30, 100 and 1000 components for each alternative method. Tables 9–11 report the accuracy and estimation time of all methods. We observe that 1) when alternative methods using the same number of components with GMM, GMM has better accuracy than other methods in most cases, and 2) when alternative methods using a larger number of components to achieve high accuracy, they have similar median error with GMM, but have much larger max error and longer estimation time than GMM. This is because all alternative methods assume uniform distribution within a component even if they use different methods to construct components. However, most real-world datasets are very skewed. Those methods have to use many components to fit the data. Hence, we use GMM in IAM.

6.7 Varying the Number of Mixture Components

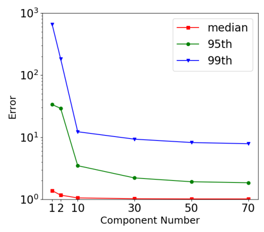
This experiment is to evaluate the effect of the number of components in the GMM on accuracy. As shown in Figure 7, IAM is very robust to the number of the components when the number is larger than 30 on all datasets. Specifically, the accuracy generally becomes better as the number of components is smaller than 30 and more components of mixture models are used. The improvements on accuracy is dramatically when the component number increases in the range of 1–10. However, when the number of components reaches 30, further increasing the number will not improve accuracy much. Table 12 gives the memory usage

Table 11: Impact of domain reducing methods on HIGGS.

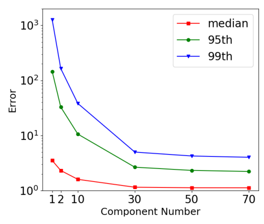
Methods	Error			Est. time (ms)
	Median	95th	Max	
GMM (30)	1.07	1.96	**9.00**	10
Hist (30)	1.2	12.9	18658	12.9
Hist (100)	1.12	3.43	12774	9.1
Hist (1000)	**1.04**	**1.65**	527	25.4
Spline (30)	1.38	11.3	25794	11
Spline (100)	1.22	2.52	527	13.2
Spline (1000)	1.14	2.00	527	23
UMM (30)	2.8	45.4	4398	13
UMM (100)	1.21	3.33	35	11
UMM (1000)	1.12	2.77	11	39



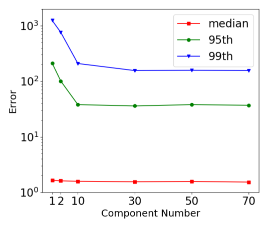
(a) WISDM



(b) TWI



(c) HIGGS



(d) IMDB

Figure 7: Varying the number of components.

Table 12: Model size (MB) of IAM vs. number of components.

#	WISDM	TWI	HIGGS	IMDB
1	0.72	0.51	0.72	6.1
10	1.0	0.56	0.9	6.2
30	1.1	0.66	1.6	6.3
50	1.2	0.77	2.1	6.5
70	1.3	0.88	2.7	6.6

with different numbers of mixture components. We observe that the model size will increase with the increase of the number of components, although accuracy may not improve much beyond a certain level.

7 RELATED WORK

Deep autoregressive models and mixture models. Closest to our work is the deep autoregressive models [9, 13, 43], which are state-of-the-art density estimation models in machine learning. Recent estimators [19, 49, 53, 54] based on the deep autoregressive model have reported impressive accuracy results. However, the accuracy of these estimators will deteriorate significantly in the presence of attributes of large domain size (e.g., continuous attributes), as the search space in query inference will be very large.

To address the challenge, we propose the novel idea of integrating mixture models and an AR model for selectivity estimation. Mixture models use multiple simple probability models to learn data distribution. Typically, mixture models are used for unsupervised clustering and are powerful in modeling data distributions. However, very little research (including QuickSel [37] and DeepSpace [49] as discussed in Section 4) considers mixture models in selectivity estimation.

Other deep learning based estimators. DeepDB [24] uses sum-product networks (SPNs) [38] as the density estimator. DeepDB learns multiple SPNs, each on a subset of correlated tables, and conditional independence is assumed across the SPNs. FLAT [58] improves on DeepDB with no independence assumption. Dutt et al. [11] considers selectivity estimation as a regression problem and uses neural networks to fit the queries. MSCN [29] improves on it to handle categorical attributes and joins by using neural networks to extract features and concatenate them by a convolutional network.

Classical estimators. We next provide a brief review of classical methods by categories. First, sample based methods [16, 32] do selectivity estimation on a set of samples. Performances of these methods are highly related to the sample portion. Second, histogram based methods [1, 4, 8, 15, 31, 34, 39–41, 45, 46] build buckets based on data or queries. They often make conditional independence and uniform distribution assumptions. Third, probabilistic relational models (PRMs) [14, 47] use Bayesian Networks to model the joint distribution, which also rely on conditional independence assumption. PRMs have relatively high accuracy but low efficiency on estimation. Fourth, Kernel Density Estimation (KDE) [22, 26] methods use Gaussian kernel to fit data distribution without any independence assumption. Their performances are not often competitive since adjusting the bandwidth is difficult. Both KDE and mixture models learn data distribution with some simple function. However, there are two essential reasons we use mixture models rather than KDE. 1) KDE needs to sample many tuples as Gaussian kernel, so the reduced domain size could be also very large. 2) KDE is inefficient in query inference, which is also shown in our experiment in Section 6.3.1.

8 CONCLUSIONS AND FUTURE WORK

In this work, we propose a novel model IAM that integrates GMMs and a deep autoregressive model. We utilize the GMMs to reduce large domain sizes of continuous attributes, and thus search space of the deep autoregressive model is reduced dramatically. We also develop an unbiased progressive sampling algorithm for IAM, enabling IAM to perform accurate and fast range query inference. Our experiments demonstrate that IAM achieves more accurate selectivity estimations than the baselines. Compared with a real DBMS and other estimators, IAM has the best performance on query optimization.

In the future, we plan to implement other mixture models in IAM to learn different data distributions. Additionally, it is of interest to extend IAM on other approximate query processing queries, such as AVG and SUM queries.

ACKNOWLEDGMENTS

This research is supported in part by MOE Tier-2 grant MOE2019-T2-2-181 and MOE Tier-1 grant RG114/19(S). This research was conducted when Peizhi Wu was with NTU.

REFERENCES

[1] Ashraf Aboulnaga and Surajit Chaudhuri. 1999. Self-tuning histograms: Building histograms without looking at data. *ACM SIGMOD Record* 28, 2 (1999), 181–192.

[2] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. 2014. Searching for exotic particles in high-energy physics with deep learning. *Nature communications* 5, 1 (2014), 1–9.

[3] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. 2017. *Deep learning*. Vol. 1.

[4] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. 2001. In *SIGMOD*. ACM, 211–222.

[5] Walter Cai, Magdalena Balazinska, and Dan Suciu. 2019. Pessimistic cardinality estimation: Tighter upper bounds for intermediate join cardinalities. In *SIGMOD*. 18–35.

[6] Benjamin Charlier, Jean Feydy, Joan Alexis Glaunès, François-David Collin, and Ghislain Durif. 2021. Kernel operations on the GPU, with autodiff, without memory overflows. *J. Mach. Learn. Res.* 22 (2021), 74:1–74:6.

[7] C. K. Chow and C. N. Liu. 1968. Approximating discrete probability distributions with dependence trees. *IEEE Trans. Inf. Theory* 14, 3 (1968), 462–467.

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

[9] Conor Durkan and Charlie Nash. [n.d.]. Autoregressive Energy Machines. In *ICML*.

[10] Anshuman Dutt, Chi Wang, Vivek Narasayya, and Surajit Chaudhuri. 2020. Efficiently approximating selectivity functions using low overhead regression models. *VLDB* 13, 12 (2020), 2215–2228.

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

[12] Nir Friedman and Moisés Goldszmidt. [n.d.]. Discretizing Continuous Attributes While Learning Bayesian Networks. In *ICML*. 157–165.

[13] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. [n.d.]. MADE: Masked Autoencoder for Distribution Estimation.

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

[15] Dimitrios Gunopulos, George Kollios, Vassilis J Tsotras, and Carlotta Domeniconi. 2005. Selectivity estimators for multidimensional range queries over real attributes. *the VLDB Journal* 14, 2 (2005), 137–154.

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

[17] Shohedul Hasan, Saravanan Thirumuruganathan, Jees Augustine, Nick Koudas, and Gautam Das. 2019. Multi-attribute selectivity estimation using deep learning. *arXiv preprint arXiv:1903.09999* (2019).

[18] Shohedul Hasan, Saravanan Thirumuruganathan, Jees Augustine, Nick Koudas, and Gautam Das. 2020. Deep Learning Models for Selectivity Estimation of Multi-Attribute Queries. In *SIGMOD*. 1035–1050.

[19] Shohedul Hasan, Saravanan Thirumuruganathan, Jees Augustine, Nick Koudas, and Gautam Das. 2020. Deep Learning Models for Selectivity Estimation of Multi-Attribute Queries. In *SIGMOD*. 1035–1050.

[20] Trevor Hastie, Jerome H. Friedman, and Robert Tibshirani. [n.d.]. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.*

[21] M. Heimel. 2019. *Bitbucket repository, feedback-kde*. Retrieved Jun, 2020 from https://bitbucket.org/mheimel/feedback-kde

[22] Max Heimel, Martin Kiefer, and Volker Markl. [n.d.]. Self-tuning, gpu-accelerated kernel density models for multidimensional selectivity estimation.

[23] Benjamin Hilprecht. 2019. *Github repository, deepdb*. Retrieved Aug, 2020 from https://github.com/DataManagementLab/deepdb-public

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

[25] IAM 2021. *Technical report of this paper*. Retrieved Dec, 2021 from https://drive.google.com/drive/folders/1vx3CB5aNAwUVEMMHN7pVw3VDGl6Xb3Am?usp=sharing

[26] Martin Kiefer, Max Heimel, Sebastian Breß, and Volker Markl. 2017. Estimating Join Selectivities using Bandwidth-Optimized Kernel Density Models. *VLDB* 10, 13 (2017), 2085–2096.

[27] Diederik P. Kingma and Jimmy Ba. [n.d.]. Adam: A Method for Stochastic Optimization.

[28] Andreas Kipf. 2018. *Github repository, MSCN*. Retrieved Jul, 2020 from https://github.com/andreaskipf/learnedcardinalities

[29] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. 2019. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In *CIDR*.

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

[31] Lipyeow Lim, Min Wang, and Jeffrey Scott Vitter. 2003. SASH: A self-adaptive histogram set for dynamically changing workloads. In *VLDB*. 369–380.

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

[33] MicrosoftSQL 2017. *Statistics in MicrosoftSQLL*. Retrieved Mar 06, 2020 from https://docs.microsoft.com/en-us/sql/relational-databases/statistics/statistics?view=sql-server-2017#UpdateStatistics

[34] M Muralikrishna and David J DeWitt. 1988. Equi-depth multidimensional histograms. In *SIGMOD*. 28–36.

[35] Thomas Neumann and Sebastian Michel. 2008. Smooth interpolating histograms with error guarantees. In *British National Conference on Databases*. 126–138.

[36] Oracle 2019. *Sampling in Oracle*. Retrieved Dec 09, 2019 from https://www.oracle.com/technetwork/database/bi-datawarehousing/twp-stats-concepts-0218-4403739.pdf

[37] Yongjoo Park, Shucheng Zhong, and Barzan Mozafari. [n.d.]. QuickSel: Quick Selectivity Learning with Mixture Models. In *SIGMOD*. 1017–1033.

[38] Hoifung Poon and Pedro M. Domingos. [n.d.]. Sum-product networks: A new deep architecture. In *ICCV*. 689–690.

[39] Viswanath Poosala, Peter J Haas, Yannis E Ioannidis, and Eugene J Shekita. 1996. Improved histograms for selectivity estimation of range predicates. *ACM Sigmod Record* 25, 2 (1996), 294–305.

[40] Viswanath Poosala and Yannis E Ioannidis. 1997. Selectivity estimation without the attribute value independence assumption. In *VLDB*, Vol. 97. 486–495.

[41] Viswanath Poosala, Yannis E. Ioannidis, Peter J. Haas, and Eugene J. Shekita. 1996. Improved Histograms for Selectivity Estimation of Range Predicates. In *SIGMOD*. 294–305.

[42] PostgreSQL 2018. *Row Estimation in PostgreSQL*. Retrieved Oct 13, 2020 from https://www.postgresql.org/docs/11/row-estimation-examples.html

[43] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[44] Richard A Redner and Homer F Walker. 1984. Mixture densities, maximum likelihood and the EM algorithm. *SIAM review* 26, 2 (1984), 195–239.

[45] P Griffiths Selinger, Morton M Astrahan, Donald D Chamberlin, Raymond A Lorie, and Thomas G Price. 1989. Access path selection in a relational database management system. In *Readings in Artificial Intelligence and Databases*. 511–522.

[46] Michael Stillger, Guy M Lohman, Volker Markl, and Mokhtar Kandil. 2001. LEO-DB2's learning optimizer. In *VLDB*, Vol. 1. 19–28.

[47] Kostas Tzoumas, Amol Deshpande, and Christian S Jensen. 2011. Lightweight graphical models for selectivity estimation without independence assumptions. *VLDB* 4, 11 (2011), 852–863.

[48] Todd L. Veldhuizen. 2014. Triejoin: A Simple, Worst-Case Optimal Join Algorithm. In *ICDT*. 96–106.

[49] Dimitri Vorona, Andreas Kipf, Thomas Neumann, and Alfons Kemper. 2019. Deepspace: Approximate geospatial query processing with deep learning. In *SIGSPATIAL*. 500–503.

[50] Qiang Wang, Yi Shen, and Jian Qiu Zhang. 2005. A nonlinear correlation measure for multivariable data set. *Physica D: Nonlinear Phenomena* 200, 3-4 (2005), 287–295.

[51] Kazuho Watanabe and Sumio Watanabe. 2006. Stochastic complexities of Gaussian mixtures in variational Bayesian approximation. *The Journal of Machine Learning Research* 7 (2006), 625–644.

[52] Gary M. Weiss, Kenichi Yoneda, and Thaier Hayajneh. 2019. Smartphone and Smartwatch-Based Biometrics Using Activities of Daily Living. *IEEE Access* 7 (2019), 133190–133202.

[53] Peizhi Wu and Gao Cong. 2021. A Unified Deep Model of Learning from both Data and Queries for Cardinality Estimation. In *SIGMOD*. 2009–2022.

[54] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2020. NeuroCard: one cardinality estimator for all tables. *VLDB* 14, 1 (2020), 61–73.

[55] Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Xi Chen, Pieter Abbeel, Joseph M Hellerstein, Sanjay Krishnan, and Ion Stoica. 2019. Deep Unsupervised Cardinality Estimation. *VLDB* 13, 3, 279–292.

[56] Ryuichi Ito Yongjoo Park. 2018. *Github repository, quicksel*. Retrieved Dec, 2021 from https://github.com/illinoisdata/quicksel

[57] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. [n.d.]. Random Sampling over Joins Revisited. In *SIGMOD*. 1525–1539.

[58] Rong Zhu, Ziniu Wu, Yuxing Han, Kai Zeng, Andreas Pfadler, Zhengping Qian, Jingren Zhou, and Bin Cui. 2021. FLAT: fast, lightweight and accurate method for cardinality estimation. *VLDB* 14, 9 (2021), 1489–1502.

[59] Chenggang Wu Zongheng Yang. 2020. *Github repository, Neurocard*. Retrieved Dec, 2020 from https://github.com/neurocard/neurocard

[60] Daniel Zwillinger and Stephen Kokoska. 1999. *CRC standard probability and statistics tables and formulae*. Crc Press.