Deep Unsupervised Cardinality Estimation

Zongheng Yang[1], Eric Liang[1], Amog Kamsetty[1], Chenggang Wu[1], Yan Duan[3], Xi Chen[1,3], Pieter Abbeel[1,3], Joseph M. Hellerstein[1], Sanjay Krishnan[2], Ion Stoica[1]

[1]UC Berkeley [2]University of Chicago [3]covariant.ai

[1]{zongheng,ericliang,amogkamsetty,cgwu,pabbeel,hellerstein,istoica}@berkeley.edu
[2]skr@uchicago.edu [3]{rocky,peter}@covariant.ai

ABSTRACT

Cardinality estimation has long been grounded in statistical tools for density estimation. To capture the rich multivariate distributions of relational tables, we propose the use of a new type of high-capacity statistical model: deep autoregressive models. However, direct application of these models leads to a limited estimator that is prohibitively expensive to evaluate for range or wildcard predicates. To produce a truly usable estimator, we develop a Monte Carlo integration scheme on top of autoregressive models that can efficiently handle range queries with dozens of dimensions or more.

Like classical synopses, our estimator summarizes the data without supervision. Unlike previous solutions, we approximate the joint data distribution without any independence assumptions. Evaluated on real-world datasets and compared against real systems and dominant families of techniques, our estimator achieves single-digit multiplicative error at tail, an up to 90× accuracy improvement over the second best method, and is space- and runtime-efficient.

PVLDB Reference Format:
Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Xi Chen, Pieter Abbeel, Joseph M. Hellerstein, Sanjay Krishnan, and Ion Stoica. Deep Unsupervised Cardinality Estimation. *PVLDB*, 13(3): 279-292, 2019.
DOI: https://doi.org/10.14778/3368289.3368294

1. INTRODUCTION

Cardinality estimation is a core primitive in query optimization [42]. One of its main tasks is to accurately estimate the *selectivity* of a SQL predicate—the fraction of a relation selected by the predicate—without actual execution. Despite its importance, there is wide agreement that the problem is still unsolved [26,28,36]. Open-source and commercial DBMSes routinely produce up to 10^4−10^8× estimation errors on queries over a large number of attributes [26].

The fundamental difficulty of selectivity estimation comes from condensing information about data into summaries [18]. The predominant approach in database systems today is to collect single-column summaries (e.g., histograms and sketches), and to combine these coarse-grained models assuming column independence. This represents one end of the spectrum, where the summaries are fast to construct and cheap to store, but compounding errors occur due to the coarse information and over-simplifying independence assumptions. On the other end of the spectrum, when given the *joint data distribution* of a relation (the frequency of each unique tuple normalized by the relation's cardinality), perfect selectivity "estimates" can be read off or computed via integration over the distribution. However, the *joint* is intractable to compute or store for all but the tiniest datasets. Thus, traditional selectivity estimators face the hard tradeoff between the amount of information captured and the cost to construct, store, and query the summary.

Proceedings of the VLDB Endowment, Vol. 13, No. 3
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3368289.3368294

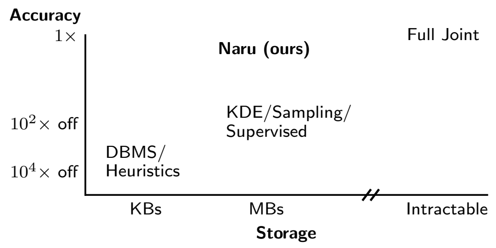


Figure 1: Approximating the joint data distribution in full, Naru enjoys high estimation accuracy and space efficiency.

An accurate and compact *joint approximation* would allow better design points in this tradeoff space (Figure 1). Recent advances in deep unsupervised learning have offered promising tools in this regard. While it was previously thought intractable to approximate the data distribution of a relation in its full form [7, 14], *deep autoregressive models*, a type of density estimator, have succeeded in modeling high-dimensional data such as images, text, and audio [40, 48–50]. However, these models only estimate *point densities*—in query processing terms, they only handle equality predicates (e.g., "what is the fraction of tuples with price equal to \$100?"). Full-featured selectivity estimation requires handling not only equality but also range predicates (e.g., "what fraction of tuples have price less than \$100 and weight greater than 10 lbs?"). Naive estimation of the range density by integrating over the query region requires summing up an enormous number of points. In an 11-dimensional table we consider, a challenging range query has 10^{10} points in the query region, which would take more than 1,000 hours to sum over by a naive enumeration scheme. A full-featured selectivity estimator, therefore, requires new techniques beyond the state of the art.

In this paper, we show that selectivity estimation can be performed with high accuracy by using deep autoregressive models. We first show how relational data—including both numeric and categorical attributes—can be mapped onto these models for effective selectivity estimation of equality predicates. We then introduce a new Monte Carlo integration technique called *progressive sampling*, which efficiently estimates range queries even at high dimensionality. By leveraging the availability of conditional probability distributions provided by the model, progressive sampling steers the sampler into regions of high probability density, and then corrects for the induced bias by using importance weighting. This technique extends the state of the art in density estimation, with particular applicability to our problem of general-purpose selectivity estimation. Our scheme is effective: a thousand samples suffice to accurately estimate the aforementioned 10^{10}-point query.

To realize these ideas, we design and implement Naru (Neural Relation Understanding), a selectivity estimator that approximates the joint data distribution in its full form, without any column independence assumptions. Approximating the joint in full not only provides superior accuracy, but also frees us from specifying what combinations of columns to build synopses on. We further propose optimizations to efficiently handle wildcard predicates, and to encode and decode real-world relational data (e.g., supporting various datatypes, small and large domain sizes). Combining our integration scheme with these practical strategies results in a highly accurate, compact, and functionality-rich selectivity estimator based on deep autoregressive models.

Just like classical synopses, Naru summarizes a relation in an unsupervised fashion. The model is trained via statistically grounded principles (maximum likelihood) where no supervised signals or query feedback are required. While *query-driven* estimators are optimized with respect to a set of training queries (i.e., "how much error does the estimator incur on these queries?"), Naru is optimized with respect to the underlying data distribution (i.e., "how divergent is the estimator from the data?"). Being data-driven, Naru supports a much larger set of queries and is automatically robust to query distributional shifts. Our evaluation compares Naru to the state-of-the-art unsupervised and supervised techniques, showing Naru to be the only estimator to achieve worst-case *single-digit multiplicative errors* for challenging high-dimensional queries.

In summary, we make the following contributions:

1. We show deep autoregressive models can be used for selectivity estimation (§2, §3), and propose optimizations to make them suitable for relational data (§4).

2. To handle challenging range queries, we develop *progressive sampling*, a Monte Carlo integration algorithm that efficiently estimates range densities even with large query regions (§5.1). We augment it with a novel optimization, *wildcard-skipping* (§5.2), to handle wildcard predicates. We also propose information-theoretic column orderings (§5.3) to reduce estimation variance.

3. We extensively evaluate on real datasets against 8 baselines across 5 different families (heuristics, real DBMSes, sampling, statistical methods, deep supervised regression). Our estimator Naru achieves up to orders-of-magnitude better accuracy with space usage ~1% of data size and ~5−10ms of estimation latency (§6).

2. PROBLEM FORMULATION

Consider a relation T with attribute domains $\{A_1, \dots, A_n\}$. Selectivity estimation seeks to estimate the fraction of tuples in T that satisfy a particular predicate, $\theta : A_1 \times \cdots \times A_n \to \{0, 1\}$. We define the selectivity to be $\mathsf{sel}(\theta) := |\{\boldsymbol{x} \in T : \theta(\boldsymbol{x}) = 1\}|/|T|$.

The *joint data distribution* of the relation, defined to be

$$P(a_1, \dots, a_n) := f(a_1, \dots, a_n)/|T|$$

is closely related to the selectivity, where $f(a_1, \dots, a_n)$ is the number of occurrences of tuple $(a_1, \dots, a_n)$ in T. It forms a valid probability distribution since integrating it over the attribute domains yields a value of 1. Thus, exact selectivity calculation is equivalent to integration over the joint:

$$\mathsf{sel}(\theta) = \sum_{a_1 \in A_1} \cdots \sum_{a_n \in A_n} \theta(a_1, \dots, a_n) \cdot P(a_1, \dots, a_n).$$

In this work, we consider finite relation T and hence its empirical domains A_i are finite. Therefore summation is used in the integration calculation above.

2.1 Approximating the Joint via Factorization

Given the joint, exact selectivity "estimates" can be calculated by integration. However, the number of entries in the joint—and thus the maximum number of points needed to be summed over in the integration—is $|P| = \prod_{i=1}^{n} |A_i|$, a size that grows exponentially in the number of attributes. Real-world tables with a dozen or so columns can easily have a theoretic joint size of 10^{20} and upwards (§6). In practice, it is possible to bound this number by $|T|$, the number of tuples in the relation, by not storing any entry with zero occurrence. Algorithmically, to scale construction, storage, and integration to high-dimensional tables, joint approximation techniques seek to *factorize* [15] the joint into some lower-dimensional representation, $\widehat{P} \approx P$.

Classical 1D histograms [42] use the simplest factorization, $\widehat{P}(A_1, \cdots, A_n) \approx \prod_{i=1}^{n} \widehat{P}(A_i)$, where independence between attributes is assumed. The $\widehat{P}(A_i)$'s are materialized as histograms that are cheap to construct and store. Selectivity estimation reduces to calculating per-column selectivities and combining by multiplication,

$$\mathsf{sel}(\theta) \approx \left(\sum_{a_1 \in A_1} \theta_1(a_1)\widehat{P}(a_1)\right) \times \cdots \times \left(\sum_{a_n \in A_n} \theta_n(a_n)\widehat{P}(a_n)\right)$$

where each θ_i is predicate θ projected to each attribute (assuming here θ is a conjunction of single-attribute filters).

Richer factorizations are possible and are generally more accurate. For instance, Probabilistic Relational Models [13, 14] from the early 2000s leverage the conditional independence assumptions of Bayesian Networks (e.g., joint factored into smaller distributions, $\{\widehat{P}(A_1|A_2, A_3),\ \widehat{P}(A_2), \widehat{P}(A_3)\}$). Dependency-Based Histograms [7] use decomposable interaction models and rely on partial independence between columns (e.g., $\widehat{P}(A_1, A_2, A_3) \approx \widehat{P}(A_1)\widehat{P}(A_2, A_3)$). Both methods are marked improvements over 1D histograms since they capture more than single-column interactions. However, the tradeoff between richer factorizations and costs to store or integrate is still unresolved. Obtaining selectivities becomes drastically harder due to the integration now crossing multiple attribute domains. Most importantly, the approximated joint's precision is compromised since some forms of independence are still assumed.

In this paper, we consider the richest possible factorization of the joint, using the product rule:

$$\widehat{P}(A_1, \cdots, A_n) = \widehat{P}(A_1)\widehat{P}(A_2|A_1)\cdots\widehat{P}(A_n|A_1, \ldots, A_{n-1})$$

Unlike the previous proposals, the product rule factorization is an *exact* relationship to represent a distribution. It makes no independence assumptions and captures all complex interactions between attributes. Key to this goal is that the factors, $\{\widehat{P}(A_i|A_1, \ldots, A_{i-1})\}$, need not be materialized; instead, they are calculated on-demand by a neural network, a high-capacity universal function approximator [11].

2.2 Problem Statement

We estimate the selectivities of queries of the following form. A query is a conjunction of single-column boolean predicates, over arbitrary subsets of columns. A predicate contains an attribute, an operator, and a literal, and is read as $A_i \in R_i$ (attribute i takes on values in valid region R_i). Our formulation includes the usual $=, \neq, <, \leq, >, \geq$ predicates, the rectangular containment $A_i \in [l_i, r_i]$, or even IN clauses. For ease of exposition, we use *range* to denote the valid region R_i or, for the whole query, the composite valid region $R_1 \times \cdots \times R_n$. We assume the *domain* of each column, A_i, is finite: since a real dataset is finite, we can take the empirically present values of a column as its finite domain.

We make a few remarks. First, disjunctions of such predicates are supported via the inclusion-exclusion principle. Second, our formulation follows a large amount of existing work on this topic [7,14,17,35,38] and, in some cases, offers more capabilities. Certain prior work requires each predicate be a rectangle [17,22] or columns be real-valued [17,24]; our "region" formulation supports complex predicates and does not make these assumptions. Lastly, the relation under estimation can either be a base table or a join result.

3. DEEP AUTOREGRESSIVE MODELS

3.1 Overview

Naru uses a deep autoregressive model to approximate the joint distribution. We overview the statistical features they offer and how those relate to selectivity estimation.

Access to point density $\widehat{P}(\boldsymbol{x})$. Deep autoregressive models produce point density estimates $\widehat{P}(\boldsymbol{x})$ after training on a set of n-dimensional tuples $T = \{\boldsymbol{x}_1, \ldots\}$ with the unsupervised maximum likelihood objective. Many network architectures have been proposed in recent years, such as masked multi-layer perceptrons (e.g., MADE [12], ResMADE [9]) or masked self-attention networks (e.g., Transformer [50]).

Access to conditional densities $\{\widehat{P}(x_i|\boldsymbol{x}_{<i})\}$. Additionally, autoregressive models also provide access to all conditional densities present in the product rule:

$$\begin{aligned}\widehat{P}(\boldsymbol{x}) &= \widehat{P}(x_1, x_2, \cdots, x_n)\\ &= \widehat{P}(x_1)\widehat{P}(x_2|x_1)\cdots\widehat{P}(x_n|x_1, \ldots, x_{n-1})\end{aligned}$$

Namely, given input tuple $\boldsymbol{x} = (x_1, \cdots, x_n)$, one can obtain from the model the n conditional density estimates, $\{\widehat{P}(x_i|\boldsymbol{x}_{<i})\}$. The model can be architected to use any ordering(s) of the attributes (e.g., (x_1, x_2, x_3) or (x_2, x_1, x_3)). In our exposition we assume the left-to-right schema order (§5.3 discusses heuristically picking a good ordering).

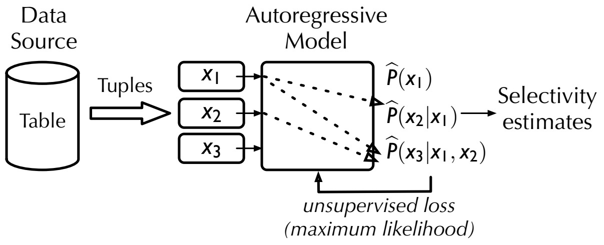


Figure 2: Overview of the estimator framework. Naru is trained by reading data tuples and does not require supervised training queries or query feedback, just like classical synopses.

Naru chooses autoregressive models for selectivity estimation for two important reasons. First, autoregressive models have shown superior modeling precision in learning images [40,49], audio [48], and text [50]. All these domains involve correlated, high-dimensional data akin to a relational table. Second, as we will show in §5.1, access to conditional densities is critical in efficiently supporting range queries.

3.2 Autoregressive Models for Relational Data

Naru allows any autoregressive model $\mathcal{M}$ to be plugged in. In general, such model has the following functional form:

$$\mathcal{M}(\boldsymbol{x}) \mapsto \left[\widehat{P}(X_1), \widehat{P}(X_2|x_1), \cdots, \widehat{P}(X_n|x_1, \ldots, x_{n-1})\right] \quad (1)$$

Namely, one tuple goes in, a list of conditional density distributions comes out, each being a *distribution* of the ith attribute conditioned on previous attributes. (The *scalars* required to compute the point density, $\{\widehat{P}(x_i|\boldsymbol{x}_{<i})\}$, are read from these conditional distributions.) How can a neural net $\mathcal{M}$ attain the autoregressive property, e.g., that $\widehat{P}(X_3|x_1, x_2)$ only depends on, or "sees", the information from the first two attribute values (x_1, x_2) but not anything else?

Information masking is a common technique used to implement autoregressive models [12,49,50]; here we illustrate the idea by constructing an example architecture for relational data. Suppose we assign each column i its own compact neural net, whose input is the aggregated information about *previous* column values $\boldsymbol{x}_{<i}$. Its role is to use this context information to output a distribution over its own domain, $\widehat{P}(X_i|\boldsymbol{x}_{<i})$. Consider a `travel_checkins` table with columns `city`, `year`, `stars`. Assume the model is given the input tuple, $\langle$`Portland`, `2017`, `10`$\rangle$. First, column-specific encoders $E_{\texttt{col}}()$ transform each attribute value into a numeric vector suitable for neural net consumption, $[E_{\texttt{city}}(\texttt{Portland}), E_{\texttt{year}}(\texttt{2017}), E_{\texttt{stars}}(\texttt{10})]$. Then, appropriately aggregated inputs are fed to the per-column neural nets $\mathcal{M}_{\texttt{col}}$:

$$\begin{gathered}\mathbf{0} \to \mathcal{M}_{\texttt{city}}\\ E_{\texttt{city}}(\texttt{Portland}) \to \mathcal{M}_{\texttt{year}}\\ \oplus\,(E_{\texttt{city}}(\texttt{Portland}), E_{\texttt{year}}(\texttt{2017})) \to \mathcal{M}_{\texttt{stars}}\end{gathered}$$

where $\oplus$ is the operator that aggregates information from several encoded attributes. In practice, this aggregator can be vector concatenation, a set-invariant *pooling* operator (e.g., elementwise sum or max), or even self-attention [50].

Notice that the first output, from $\mathcal{M}_{\texttt{city}}$, does not depend on any attribute values (its input **0** is arbitrarily chosen). The second output depends only on the attribute value from `city`, and the third depends only on both `city` and `year`. Therefore, the three outputs can be interpreted as

$$\left[\widehat{P}(\texttt{city}), \widehat{P}(\texttt{year}|\texttt{city}), \widehat{P}(\texttt{stars}|\texttt{city}, \texttt{year})\right]$$

Thus, autoregressiveness is achieved via such input masking.

Training these model outputs to be as close as possible to the true conditional densities is done via maximum likelihood estimation. Specifically, the *cross entropy* [11] between the data distribution P and the model estimate $\widehat{P}$ is calculated over all tuples in relation T and used as the loss:

$$\mathcal{H}(P, \widehat{P}) = -\sum_{\boldsymbol{x} \in T} P(\boldsymbol{x}) \log \widehat{P}(\boldsymbol{x}) = -\frac{1}{|T|} \sum_{\boldsymbol{x} \in T} \log \widehat{P}(\boldsymbol{x}) \quad (2)$$

It can be fed into a standard gradient descent optimizer [20]. Lastly, the Kullback-Leibler divergence, $\mathcal{H}(P, \widehat{P}) - \mathcal{H}(P)$, is the *entropy gap* (in bits-per-tuple) incurred by the model. A lower gap indicates a higher-quality density estimator; thus, it serves as a monitoring metric during and after training.

4. ESTIMATOR CONSTRUCTION

We now discuss practical issues in constructing Naru.

4.1 Workflow

Figure 2 outlines the workflow of building a Naru estimator. After specifying a table T to build an estimator on, batches of random tuples from T are read to train Naru. In practice, a snapshot of the table can be saved to external storage so normal DBMS activities are not affected. Neural network training can be performed either close to the data (at periods of low activity) or offloaded to a remote process.

For a batch of tuples, Naru encodes each attribute value using column-specific strategies (§4.2). The encoded batch then gets fed into the model to perform a gradient update step. Our evaluation (§6.4) empirically observed that one pass over data is sufficient to achieve a high degree of accuracy (e.g., outperforming real DBMSes by 10−20×), and more passes are beneficial until model convergence.

Appends and updates may cause statistical staleness. Naru can be fine-tuned on the updated relation to correct for this, as we show in §6.8.3. Further, if new data comes in per-day partitions, then each partition can train its own Naru model. Efficient incremental model update is an important topic worthy of detailed study, which we defer to future work.

Joins. The estimator does not distinguish between the type of table it is built on. To build an estimator on a joined relation, either the entire joined relation can be pre-computed and materialized, or multi-way join operators [51, 52] and samplers [2, 27] can be used to produce batches of tuples on-the-fly. Given access to tuples from the joined result, no changes are needed to the estimator framework. Once trained, the estimator supports queries that filter any column in the joined relation. This treatment follows prior work [19, 31, 35] and is conceptually clean.

4.2 Encoding and Decoding Strategies

Naru models a relation as a high-dimensional discrete distribution. The key challenge is to *encode* each column into a form suitable for neural network consumption, while preserving the column semantics. Further, each column's output distribution $\widehat{P}(X_i|\boldsymbol{x}_{<i})$ (a vector of scores) must be efficiently *decoded* regardless of its datatype or domain size.

For each column Naru first obtains its domain A_i either from user annotation or by scanning. All values in the column are then dictionary-encoded into integer IDs in range $[0, |A_i|)$. For instance, the dictionary can be `Portland` $\mapsto 0$, `SF` $\mapsto 1$, etc. For a column with a natural order, e.g., numerics or strings, the domain is sorted so that the dictionary order follows the column order. Overall, this pre-processing step is a lossless transformation (i.e., a bijection).

Next, column-specific encoders $E_{\texttt{col}}()$ encode these IDs into vectors. The ML community has proposed many such strategies before; we make sensible choices by keeping in mind a few characteristics specific to relational datasets:

Encoding small-domain columns: one-hot. For such a column $E_{\texttt{col}}()$ is set to *one-hot encoding* (i.e., indicator variables). For instance, if there are a total of 4 cities, then the encoding of `SF` is $E_{\texttt{city}}(1) = [0, 1, 0, 0]$, a 4-dimensional vector. The small-domain threshold is configurable and set to 64 by default. This encoding takes $O(|A_i|)$ space per value.

Encoding large-domain columns: embedding. For a larger domain, the one-hot vector wastes space and computation budget. Naru uses embedding encoding in this case. In this scheme—a preprocessing step in virtually all natural language processing tasks—a learnable embedding matrix of type $\mathbb{R}^{|A_i| \times h}$ is randomly initialized, and $E_{\texttt{col}}()$ is simply row lookup into this matrix. For instance, $E_{\texttt{year}}(4) \mapsto$ row 4 of embedding matrix, an h-dimensional vector. The embedding matrix gets updated during gradient descent as part of the model weights. Per value this takes $O(h)$ space (Naru defaults h to 64). This encoding is ideal for domains with a meaningful semantic distance (e.g., cities are similar in geo-location, popularity, relation to its nation) since each dimension in the embedding vector can learn to represent each such similarity.

Decoding small-domain columns. Suppose domain A_i is small. In this easy case, the network allocates an *output layer* to compute a *distribution* $\widehat{P}(X_i|\boldsymbol{x}_{<i})$, which is a $|A_i|$-dimensional vector of probabilities used for selectivity estimation. We use a fully connected layer, $\mathsf{FC}(F, |A_i|)$, where F is the hidden unit size. For example, for a city column with three values in its domain, the output distribution may be $[\mathsf{SF} = 0.2; \mathsf{Portland} = 0.5; \mathsf{Waikiki} = 0.3]$. During optimization, the training loss seeks to minimize the divergence of this output from the data distribution.

Decoding large-domain columns: embedding reuse. If the domain is large, however, using a fully connected output layer $\mathsf{FC}(F, |A_i|)$ would be inefficient in both space and compute. Indeed, an `id` column in a dataset we tested on has a large domain size of $|A_i| = 10^4$, inflating the output layer beyond typical scales.

Naru solves this problem by an optimization that we call "embedding reuse". In essence, we replace the potentially large output layer $\mathsf{FC}(F, |A_i|)$ with a much smaller version, $\mathsf{FC}(F, h)$ (recall that h is the typically small embedding dimensions; defaults to 64). This immediately yields a saving ratio of $|A_i|/h$. The goal of decoding is to take in inputs

$\boldsymbol{x}_{<i}$ and output $|A_i|$ probability scores over the domain. With the shrunk-down output layer, inputs $\boldsymbol{x}_{<i}$ would pass through the net arriving at an h-dimensional feature vector, $H \subseteq \mathbb{R}^{1\times h}$. We then calculate HE_i^T, where $E_i \subseteq \mathbb{R}^{|A_i|\times h}$ is the *already-allocated* embedding matrix for column i, obtaining a vector $\mathbb{R}^{1\times|A_i|}$ that can be interpreted as the desired scores after normalization. We have thus decoded the output while cutting down the cost of compute and storage. This scheme has proved effective in other large-domain tasks [39].

4.3 Model Choice

As discussed, any autoregressive model can be plugged in, taking advantage of Naru's encoding/decoding optimizations as well as querying capabilities (§5). We experiment with three representative architectures: (A) Masked Autoencoder (MADE) [12], a standard multi-layer perceptron with information masking to ensure autoregressiveness; (B) ResMADE [9], a simple extension to MADE where residual connections are introduced to improve learning efficiency; and (C) Transformer [50], a class of self-attentional models driving recent state-of-the-art advances in natural language processing [8, 54]. Table 7 compares the tradeoffs of these building blocks. We found that, under similar parameter count, more advanced architectures (B, C) achieve better entropy gaps; however, the smaller entropy gaps do not automatically translate into better selectivity estimates and the computational cost can be significantly higher (for C).

5. QUERYING THE ESTIMATOR

Once an autoregressive model is trained, it can be queried to compute selectivity estimates. Assume a query $\mathsf{sel}(\theta) = P(X_1 \in R_1, \ldots, X_n \in R_n)$ asking for the selectivity of the conjunction, where each range R_i can be a point (equality predicate), an interval (range predicate), or any subset of the domain (IN). The calculation of this density is fundamentally summing up the probability masses distributed in the cross-product region, $R = R_1 \times \cdots \times R_n$.

We first discuss the straightforward support for equality predicates, then move on to how Naru solves the more challenging problem of range predicates.

Equality Predicates. When values are specified for *all* columns, estimating conjunctions of these equality predicates is straightforward. Such a point query has the form $P(X_1 = x_1, \ldots, X_n = x_n)$ and requires only a single forward pass on the point, $(x_1, \ldots, x_n)$, to obtain the sequence of conditionals, $[\widehat{P}(X_1 = x_1), \widehat{P}(X_2 = x_2|X_1 = x_1), \ldots, \widehat{P}(X_n = x_n|X_1 = x_1, \ldots, X_{n-1} = x_{n-1})]$, which are then multiplied.

Range Predicates. It is impractical to assume a workload that only issues point queries. With the presence of any range predicate, or when some columns are not filtered, the number of points that must be evaluated through the model becomes larger than 1. (In fact, it easily grows to an astronomically large number for the majority of workloads we considered.) We discuss two ways in which Naru carries out this operation. *Enumeration* exactly sums up the densities when the queried region R is sufficiently small:

$$\mathsf{sel}(X_1 \in R_1, \ldots, X_n \in R_n) \approx \sum_{x_1 \in R_1} \cdots \sum_{x_n \in R_n} \widehat{P}(x_1, \ldots, x_n).$$

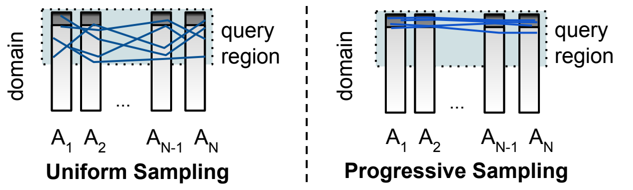


Figure 3: The intuition of progressive sampling. Uniform samples taken from the query region have a low probability of hitting the high-mass sub-region of the query region, increasing the variance of Monte Carlo estimates. Progressive sampling avoids this by sampling from the estimated data distribution instead, which naturally concentrates samples in the high-mass sub-region.

When the region R is deemed too big—almost always the case in the datasets and workloads we considered—we instead use a novel approximate technique termed *progressive sampling* (described next), an unbiased estimator that works surprisingly well on the relational datasets we considered.

Lastly, queries with out-of-domain literals can be handled via simple rewrite. For example, suppose year's domain is $\{2017, 2019\}$. A range query with an out-of-domain literal, say "year < 2018", can be rewritten as "year ≤ 2017" with equivalent semantics. For equality predicates with out-of-domain literals, Naru simply returns a cardinality of 0. Hereafter we consider in-domain literals and valid regions.

5.1 Range Queries via Progressive Sampling

The queried region $R = R_1 \times \cdots \times R_n$ in the worst case contains $O(\prod_i D_i)$ points, where $D_i = |A_i|$ is the size of each attribute domain. Clearly, computing the likelihood for an exponential number of points is prohibitively expensive for data/queries with even moderate dimensions. Naru proposes an approximate integration scheme to address this challenge.

First attempt (Figure 3, left). The simplest way to approximate the sum is via uniform sampling. First, sample $\boldsymbol{x}^{(i)}$ uniformly at random from R. Then, query the model to compute $\widehat{p}_i = \widehat{P}(\boldsymbol{x}^{(i)})$. By naive Monte Carlo, for S samples we have $\frac{|R|}{S}\sum_{i=1}^{S}\widehat{p}_i$ as an unbiased estimator to the desired density. Intuitively, this scheme is randomly throwing points into target region R to probe its average density.

To understand the failure mode of uniform sampling, consider a relation T with n correlated columns, with each column distribution skewed so that 99% of the probability mass is contained in the top 1% of its domain (Figure 3). Take a query with range predicates selecting the top 50% of each domain. It is easy to see that uniformly sampling from the query region will take in expectation $1/(0.01/0.5)^n = 1/0.02^n$ samples to hit the high-mass region we are integrating over. Thus, the number of samples needed for an accurate estimate increases exponentially in n. Consequently, we find that this sampler collapses catastrophically in the real-world datasets that we consider. It has the worst errors among all baselines in our evaluation.

Progressive sampling (Figure 3, right). Instead of uniformly throwing points into the region, we could be more selective in the points we choose—precisely leveraging the power of the trained autoregressive model. Intuitively, a sample of the first dimension $x_1^{(i)}$ would allow us to *"zoom in" into the more meaningful region of the second dimension.*

Algorithm 1 Progressive Sampling: estimate the density of query region $R_1 \times \cdots \times R_n$ using S samples.

```
1: function ProgressiveSampling($S; R_1, \ldots, R_n$)
2:     $\widehat{P} = 0$
3:     for $i = 1$ to $S$ do                              ▷ Batched in practice
4:         $\widehat{P} = \widehat{P} + \text{Draw}(R_1, \ldots, R_n)$
5:     return $\widehat{P}/S$

6: function Draw($R_1, \ldots, R_n$)                      ▷ Draw one tuple
7:     $\widehat{p} = 1$
8:     $\boldsymbol{s} = \mathbf{0}_n$                     ▷ The tuple to fill in
9:     for $i = 1$ to $n$ do
10:        Forward pass through model: $\mathcal{M}(\boldsymbol{s})$
11:        $\widehat{P}(X_i|\boldsymbol{s}_{<i})$ = the $i$-th model output     ▷ Eq. 1
12:        Zero-out probabilities in slots $[0, D_i) \setminus R_i$
13:        Re-normalize, obtaining $\widehat{P}(X_i|X_i \in R_i, \boldsymbol{s}_{<i})$
14:        $\widehat{p} = \widehat{p} \times \widehat{P}(X_i \in R_i|\boldsymbol{s}_{<i})$
15:        Sample $s_i \sim \widehat{P}(X_i|X_i \in R_i, \boldsymbol{s}_{<i})$
16:        $\boldsymbol{s}[i] = s_i$
17:    return $\widehat{p}$                          ▷ Density of the sampled tuple $s$
```

This more meaningful region is exactly described by the second conditional output from the autoregressive model, $\widehat{P}(X_2|x_1^{(i)})$, a distribution over the second domain given the first dimension sample. We can obtain a sample of the second dimension, $x_2^{(i)}$, from this space instead of from Unif(R_2). This sampling process continues for all columns. To summarize, progressive sampling consults the autoregressive model to steer the sampler into the high-mass part of the query region, and finally compensating for the induced bias with importance weighting.

Example. We show the sampling procedure for a 3-filter query. Drawing the i-th sample for query $P(X_1 \in R_1, X_2 \in R_2, X_3 \in R_3)$:

1. Forward $\mathbf{0}$ to get $\widehat{P}(X_1)$. Compute and store $\widehat{P}(X_1 \in R_1)$ by summing. Then draw $x_1^{(i)} \sim \widehat{P}(X_1|X_1 \in R_1)$.
2. Forward $x_1^{(i)}$ to get $\widehat{P}(X_2|x_1^{(i)})$. Compute and store $\widehat{P}(X_2 \in R_2|x_1^{(i)})$. Draw $x_2^{(i)} \sim \widehat{P}(X_2|X_2 \in R_2, x_1^{(i)})$.
3. Forward $(x_1^{(i)}, x_2^{(i)})$ to get $\widehat{P}(X_3|x_1^{(i)}, x_2^{(i)})$. Compute and store $\widehat{P}(X_3 \in R_3|x_1^{(i)}, x_2^{(i)})$.

The summation and sampling steps are fast since they are only over single-column distributions. This is in contrast to integrating or summing over all columns at once, which has an exponential number of points. The product of the three stored intermediates,

$$\widehat{P}(X_1 \in R_1) \cdot \widehat{P}(X_2 \in R_2|x_1^{(i)}) \cdot \widehat{P}(X_3 \in R_3|x_1^{(i)}, x_2^{(i)}) \quad (3)$$

is an unbiased estimate for the desired density. By construction, the sampled point satisfies the query ($x_1^{(i)}$ is drawn from range R_1, $x_2^{(i)}$ from R_2, and so forth). It remains to show that this sampler is approximating the correct sum:

Theorem 1. *Progressive Sampling estimates are unbiased.*

The proof only uses basic probability rules and is deferred to our online technical report. Algorithm 1 shows the pseudocode for the general n-filter case. For a column that does not have an explicit filter, it can in theory be treated as having a wildcard filter, i.e., $R_i = [0, D_i)$. We describe our more efficient treatment, *wildcard-skipping*, in §5.2. Our evaluation shows that the sampler can cover both low and high density regions, and handles challenging range queries for large numbers of columns and joint spaces.

Progressive sampling bears connections to sampling algorithms in graphical models. Notice that the autoregressive factorization corresponds to a complex graphical model where each node i has all nodes with indices $< i$ as its parents. In this interpretation, progressive sampling extends the *forward sampling with likelihood weighting* algorithm [23] to allow variables taking on *ranges of values* (the former, in its default form, allows equality predicates only).

5.2 Reducing Variance: Wildcard-Skipping

Naru introduces *wildcard-skipping*, a simple optimization to efficiently handle wildcard predicates. Instead of sampling through the full domain of each wildcard column in a query, $X_i \in *$, we could restrict it to a special token, $X_i = \mathsf{MASK}_i$. Intuitively, MASK_i signifies column i's *absence* and essentially marginalizes it. In our experiments, wildcard-skipping can reduce the variance of worst-case errors by several orders of magnitude (§6.6).

During training, we perturb each tuple so that the training data contains MASK tokens. We uniformly sample a subset of columns to *mask out*—their original values in the tuple are discarded and replaced with corresponding $\mathsf{MASK}_{\text{col}}$. For an n-column tuple, each column has a probability of w/n to be masked out, where $w \sim \text{Unif}[0, n)$. The output target for the cross-entropy loss still uses the original values.

5.3 Reducing Variance: Column Ordering

Naru models adopt a single ordering of columns during construction (§4). However, different orderings may have different sampling efficiency. For instance, having `city` as the first column and setting it to `Waikiki` focuses on records only relevant to that city, a data region supposedly much narrower than that from having `year` as the first column.

Empirically, we find that these heuristic orders work well:

1. MutInfo: successively pick column X_i that maximizes the *mutual information* [6] between all columns chosen so far and itself, $\arg\max_i I(X_{\text{chosen}}; X_i)$.
2. PMutInfo: a variant of the above that maximizes the *pairwise* mutual information, $\arg\max_i I(X_{\text{last}}; X_i)$.

Intuitively, maximizing $I(X_{\text{chosen}}; X_i)$ corresponds to finding the next column with *the most information already contained in the chosen columns.* For both schemes, we find that picking the column with the maximum marginal entropy, $\arg\max_i H(X_i)$, as the first works well. Interestingly, on our datasets, the Natural ordering (left-to-right order in table schema) is also effective. We hypothesize this is due to human bias in placing important or "key"-like columns earlier that highly reduce the uncertainty of other columns.

Lastly, we note that *order-agnostic training* has been proposed in the ML literature [12,47,54]. The idea is to train the same model on more than one order, and at inference time invoke a (presumably seen) order most suitable for the query. This is a possible future optimization for Naru, though in the preliminary experiments we did not find the performance benefits on top of our optimizations significant.

6. EVALUATION

We answer the following questions in our evaluation:

1. How does Naru compare to state-of-the-art selectivity estimators in accuracy (§6.2)? Is it robust (§6.3)?
2. How long does it take to train a Naru model to achieve a useful level of accuracy (§6.4)?
3. Naru requires multiple inference passes to produce a selectivity estimate. How does this compare with the latency of other approaches (§6.5)?
4. How do wildcard-skipping and column orderings affect accuracy and variance (§6.6)? How does accuracy change with model choices and sizes (§6.7)?

Lastly, a series of microbenchmarks are run to understand Naru's limits (§6.8).

6.1 Experimental Setup

We compare Naru against predominant families of selectivity estimation techniques, including estimators in real databases, heuristics, non-parametric density estimators, and supervised learning approaches (Table 2). To ensure a fair comparison between estimators, we restrict each estimator to a fixed *storage budget* (Table 1). For example, for the Conviva-A dataset, Naru's model must be less than 3MB in size, and the same restriction is held for all estimators for that dataset when applicable.

6.1.1 Datasets

We use real-world datasets with challenging characteristics (Table 1). The number of rows ranges from 10K to 11.6M, the number of columns ranges from 11 to 100, and the size of the joint space ranges from 10^{15} to 10^{190}:

DMV [44]**.** Real-world dataset consisting of vehicle registration information in New York. We use the following 11 columns with widely differing data types and domain sizes (the numbers in parentheses): record_type (4), reg_class (75), state (89), county (63), body_type (59), fuel_type (9), valid_date (2101), color (225), sco_ind (2), sus_ind (2), rev_ind (2). Our snapshot contains 11,591,877 tuples. The exact joint distribution has a size of 3.4×10^{15}.

Conviva-A. Enterprise dataset containing anonymized user activity logs from a video analytics company. The table corresponds to 3 days of activities. The 15 columns contain a mix of small-domain categoricals (e.g., error flags, connection types) as well as large-domain numerical quantities (e.g., various bandwidth numbers in kbps). Although the domains have a range (2–1.9K) similar to DMV, there are many more numerical columns with larger domains, resulting in a much larger joint distribution (10^{23}).

Conviva-B. A small dataset of 10K rows and 100 columns also from Conviva, with a joint space of over 10^{190}. Though this dataset is trivial in size, this enables the use of an emulated, perfect-accuracy model for running detailed robustness studies (§6.8).

6.1.2 Estimators

We next discuss the baselines listed in Table 2.

Real databases. Postgres and DBMS-1 represent the performance a practitioner can hope to obtain from a real DBMS. Both rely on classical assumptions and 1D histograms, while the latter additionally contains cross-column correlation statistics. Every column has a histogram and associated statistics built. Postgres is tuned to use a maximum amount of per-column bins (10,000). For DBMS-1, one invocation of stats creation with all columns specified only builds a histogram on the first column; we therefore invoke stats creation several times so that all columns are covered.

Independence assumption. Indep scans each column to obtain perfect per-column selectivities and combines them by multiplication. This measures the inaccuracy solely attributed to the independence assumption.

Multi-dimensional histogram. We compare to MHIST, an N-dimensional histogram. We use *MaxDiff* as our partition constraint, *Value* (V) as the sort parameter, and *Area* (A) as the source parameter [37]. We use the MHIST-2 algorithm [38] and the *uniform spread* assumption [37] to approximate the value set within each partition. According to [38], the resulting MaxDiff(V,A) histogram offers the most accurate and robust performance compared to other state-of-the-art histogram variants.

Bayesian Network. We use a Chow-Liu tree [4] as the Bayesian Network, since empirically it gave the best results for the allowed space. Since the size of conditional probability tables scales with the cube of column cardinalities, we use equal-frequency discretization (to 100 bins per column) to bound the space consumption and inference cost. Lastly, we apply the same progressive sampler to allow this estimator to support range queries (it does not support range queries out of the box); this ensures a fair comparison between the use of deep autoregressive models and this approach.

Kernel density estimators & Sampling. In the non-parametric sampling regime, we evaluate a uniform sampler and a state-of-the-art KDE-based selectivity estima-

Table 1: List of datasets used in evaluation. "Dom." refers to per-column domain size. "Joint" is number of entries in the exact joint distribution (equal to the product of all domain sizes). "Budget" is the storage budget we allocated to all evaluated estimators, when applicable, relative to the in-memory size of the corresponding original tables.

Dataset	Rows	Cols	Dom.	Joint	Budget
DMV	11.6M	11	2–2K	10^{15}	1.3% (13MB)
Conviva-A	4.1M	15	2–1.9K	10^{23}	0.7% (3MB)
Conviva-B	10K	100	2–10K	10^{190}	N/A

Table 2: List of estimators used in evaluation.

Type	Estimator	Description
Heuristic	Indep	A baseline that multiplies *perfect* per-column selectivities.
Real System	Postgres	1D stats and histograms via independence/uniformity assumptions.
Real System	DBMS-1	Commercial DBMS: 1D stats plus inter-column unique value counts.
Sampling	Sample	Keeps p% of all tuples in memory. Estimates a new query by evaluating on those samples.
MHIST	MHIST	The MaxDiff(V,A) histogram [37].
Graphical	BayesNet	Bayes net (Chow-Liu tree [4]).
KDE	KDE	Kernel density estimation [17, 19].
Supervised	MSCN	Supervised deep regression net [22].
Deep AR	Naru	(Ours) Deep autoregressive models.

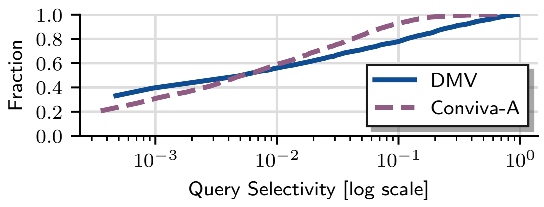


Figure 4: Distribution of query selectivity (§6.1.3).

tor [17, 19]. Sample keeps a set of p% of tuples uniformly at random from the original table. In accordance with our memory budget for each dataset, p is set to 1.3% for DMV and 0.7% for Conviva-A. KDE [17] attempts to learn the underlying data distribution by averaging Gaussian kernels centered around random sample points. The number of sample points is chosen in accordance with our memory budget: 150K samples for DMV and 28K samples for Conviva-A. The bandwidth for KDE is computed via Scott's rule [41]. The bandwidth for KDE-superv is initialized in the same way, but is further optimized through query feedback from 10K training queries. We modify the source code released by the authors [30] in order to run it with more than ten columns.

Supervised learning. We compare to a recently proposed supervised deep net-based estimator termed *multi-set convolutional network* [22], or MSCN. We apply the source code from the authors [21] to our datasets. As it is a *supervised* method, we generate 100K training queries from the same distribution the test queries are drawn, ensuring their "representativeness". The net stores a materialized sample of the data. Every query is run on the sample to get a bitmap of qualifying tuples—this is used as an input additional to query features. We try three variants of the model, all with the same hyperparameters and all trained to convergence: MSCN-base uses the same setup reported originally [22] (1K samples, 100K training queries) and consumes 3MB. We found that MSCN's performance is highly dependent on the samples, so we include a variant with 10× more samples (MSCN-10K: 10K samples, 100K train queries), consuming 13MB (satisfying DMV's budget only). We also run MSCN-0 that stores no samples and uses query features only.

Deep unsupervised learning (ours). We train one Naru model for each dataset. All models are trained with the unsupervised maximum likelihood objective. Unless stated otherwise, we employ wildcard-skipping and the natural column ordering. Sizes are reported without any compression of network weights:

- DMV: masked autoencoder (MADE), 5 hidden layers (512, 256, 512, 128, 1024 units), consuming 12.7MB.
- Conviva-A: MADE, 4 hidden layers with 128 units each, consuming 2.5MB. The embedding reuse optimization with $h = 64$ is used (§4.2).

For timing experiments, we train and run the learning methods (KDE, MSCN, Naru) on a V100 GPU. Other estimators are run on an 8-core node and vectorized when applicable.

6.1.3 Workloads

Query distribution (Figure 4). We generate multidimensional queries containing both range and equality predicates. The goal is to test each estimator on a wide spectrum of target selectivities: we group them as high (>2%), medium (0.5%–2%), and low ($\leq$ 0.5%). Intuitively, all solutions should perform reasonably well for high-selectivity queries, because dense regions require only coarse-grained modeling capacity. As the query selectivity drops, the estimation task becomes harder, since low-density regions require each estimator to model details in each hypercube. True selectivities are obtained by executing the queries on Postgres.

The query generator is inspired by prior work [22]. Instead of designating a few fixed columns to filter on, we consider the more challenging scenario where filters are randomly placed. First, we draw the number of (non-wildcard) filters $5 \leq f \leq 11$ uniformly at random. We always include at least five filters to avoid queries with very high selectivity, on which all estimators perform similarly well. Next, f distinct columns are drawn to place the filters. For columns with domain size $\geq$ 10, the filter operator is sampled uniformly from $\{=, \leq, \geq\}$; for columns with small domains, the equality operator is picked—the intention is to avoid placing a range predicate on categoricals, which often have a low domain size. The filter literals are then chosen from a random tuple sampled uniformly from the table, i.e., they follow the data distribution. For example, a valid 5-filter query on DMV is "(fuel_type = GAS) $\wedge$ (rev_ind = N) $\wedge$ (sco_ind = N) $\wedge$ (valid_date $\geq$ 2018-03-23) $\wedge$ (color = BK)". Overall, the queries span a wide range of selectivities (Figure 4).

Accuracy metric. We report accuracy by the multiplicative error [22, 26, 28] (also termed "Q-error"), the factor by which an estimate differs from the actual cardinality:

$$\text{Error} := \max(\textit{estimate}, \textit{actual}) / \min(\textit{estimate}, \textit{actual})$$

We lower bound the estimated and actual cardinalities at 1 to guard against division by zero. In line with prior work [7, 26], we found that the multiplicative error is much more informative than the *relative* error, as the latter does not fairly penalize small cardinality estimates (which are frequently the case for high-dimensional queries). Lastly we report the errors in quantiles, with a particular focus at the tail. Our results show that all estimators can achieve low median (or mean) errors but with greatly varying performance at the tail, indicating that mean/median metrics do not accurately reflect the hard cases of the estimation task.

6.2 Estimation Accuracy

In summary, Tables 3 and 4 show that not only does Naru match or exceed the best estimator across the board, it excels in the *extreme tail of query difficulty*—that is, worst-case errors on low-selectivity queries. For these types of queries, Naru achieves orders of magnitude better accuracy than classical approaches, and up to 90× better tail behavior than query-driven (supervised) methods.

The same Naru model is used to estimate all queries on a dataset, showing the robustness of the model learned. We now discuss these macrobenchmarks in more detail.

6.2.1 *Results on* DMV

Overall, Naru achieves the best accuracy and robustness across the selectivity spectrum. In the tail, it outperforms MHIST by 691×, DBMS-1 by 114×, un-tuned (tuned) MSCN by 115× (33×), BayesNet by 70×, Sample by 47×, and KDE-superv by 21×. We next discuss takeaways from Table 3.

Table 3: Estimation errors on DMV. Errors are grouped by true selectivities and shown in percentiles computed from 2,000 queries.

Estimator	High ((2%, 100%])				Medium ((0.5%, 2%])				Low (≤ 0.5%)			
	Median	95th	99th	Max	Median	95th	99th	Max	Median	95th	99th	Max
Indep	1.12	1.55	46.1	2566	1.25	46.6	1051	$8 \cdot 10^4$	1.35	225	2231	$2 \cdot 10^4$
Postgres	1.12	1.55	46.3	2608	1.25	45.5	1070	$8 \cdot 10^4$	1.36	227	2287	$2 \cdot 10^4$
DBMS-1	1.45	3.36	5.80	12.6	2.72	6.38	9.29	10.1	5.28	83.0	417	917
Sample	**1.00**	**1.02**	**1.03**	**1.05**	1.02	1.05	**1.07**	**1.10**	1.12	43.2	98.1	377
KDE	10.9	1502	$1 \cdot 10^4$	$2 \cdot 10^5$	48.0	$2 \cdot 10^4$	$9 \cdot 10^4$	$2 \cdot 10^5$	38.0	3191	$2 \cdot 10^4$	$5 \cdot 10^4$
KDE-superv	1.40	3.81	4.91	13.3	1.53	4.36	8.12	16.9	1.95	30.0	98.0	175
MSCN-base	1.17	1.42	1.47	1.58	1.14	1.65	2.53	3.96	2.95	32.5	85.6	921
MSCN-0	16.8	92.4	195	285	8.89	80.4	344	471	4.79	67.1	169	6145
MSCN-10K	1.04	1.10	1.12	1.16	1.04	1.12	1.19	1.23	1.51	14.2	33.7	264
MHIST	1.70	2.65	4.11	9.20	1.65	6.00	15.1	21.1	2.81	70.3	352	5532
BayesNet	1.01	1.07	1.12	1.44	1.05	1.18	1.26	1.40	3.41	16.1	79	561
Naru-1000	1.01	1.03	1.05	1.28	**1.01**	1.06	1.15	1.27	**1.03**	1.44	2.51	**8.00**
Naru-2000	1.01	1.03	1.04	1.16	**1.01**	**1.04**	1.09	1.38	**1.03**	**1.41**	**2.18**	**8.00**

Table 4: Estimation errors on Conviva-A. Errors grouped by true selectivities and shown in percentiles computed from 2,000 queries.

Estimator	High ((2%, 100%])				Medium ((0.5%, 2%])				Low (≤ 0.5%)			
	Median	95th	99th	Max	Median	95th	99th	Max	Median	95th	99th	Max
DBMS-1	1.75	5.25	7.77	9.12	3.93	13.6	19.9	31.3	8.63	176	636	4737
Sample	**1.02**	**1.06**	**1.09**	**1.11**	**1.04**	**1.14**	**1.18**	**1.23**	1.18	49.3	218	696
KDE	105	$2 \cdot 10^5$	$5 \cdot 10^5$	$8 \cdot 10^5$	347	$6 \cdot 10^4$	$8 \cdot 10^4$	$8 \cdot 10^4$	224	$1 \cdot 10^4$	$2 \cdot 10^4$	$2 \cdot 10^4$
KDE-superv	1.99	7.97	14.5	33.6	2.04	8.44	17.0	49.7	2.76	74.5	251	462
MSCN-base	1.14	1.27	1.36	1.48	1.15	1.55	2.26	57.7	2.05	20.3	84.1	370
MHIST	1.54	5.24	11.2	$1 \cdot 10^4$	2.22	10.5	30.3	$3 \cdot 10^4$	6.71	792	4170	$8 \cdot 10^4$
BayesNet	1.15	1.75	2.05	2.70	1.31	2.70	4.95	47.6	1.67	14.8	78.0	998
Naru-1000	**1.02**	1.11	1.18	1.37	1.05	1.17	1.27	1.40	1.10	1.71	3.01	185
Naru-2000	**1.02**	1.10	1.16	1.28	1.05	1.17	1.27	1.38	**1.09**	1.66	3.00	58.0
Naru-4000	**1.02**	1.10	1.17	1.21	**1.04**	1.15	1.27	1.36	**1.09**	**1.57**	**2.50**	**4.00**

Independence assumptions lead to orders of magnitude errors. Estimators that assume full or partial independence between columns produce large errors, regardless of query selectivity or how good per-column estimates are. These include Indep, Postgres, and DBMS-1, whose tail errors are in the $10^3 - 10^5\times$ range. Naru's model is powerful enough to avoid this assumption, leading to better results.

MHIST outperforms Indep and Postgres by over an order of magnitude. However, its performance is limited by the linear partitioning and uniform spread assumptions.

BayesNet also does quite well, nearly matching supervised approaches, but is still significantly outperformed by Naru due to the former's uses of lossy discretization and conditional independence assumptions.

Worst-case errors are much harder to be robust against. All estimators perform worse for low-selectivity queries or at worst-case errors. For instance, in the high-selectivity regime Postgres's error is a reasonable 1.55× at 95th, but becomes 1682× worse at the maximum. Also, Sample performs exceptionally well for high and medium selectivity queries, but drops off considerably for low selectivity queries where the sample has no hits. MSCN struggles since its supervised objective requires more training data to cover all possible low-selectivity queries. Naru yields much lower (single-digit) errors at the tail, showing the robustness that results from directly approximating the joint.

KDE struggles with high-dimensional data. KDE's errors are among the highest. The reason is that, the bandwidth vector found is highly sub-optimal despite tunings, due to (1) a large number of attributes in DMV, and (2) discrete columns fundamentally do not work well with the notion of "distance" in KDE [17]. The method must rely on query feedback (KDE-superv) to find a good bandwidth.

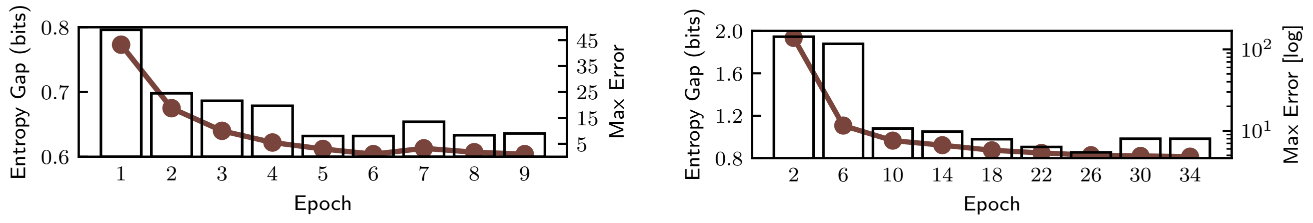
MSCN heavily relies on its materialized samples for accurate prediction. Across the spectrum, its accuracy closely approximates Sample. MSCN-10K has 3× better tail accuracy than MSCN-base due to access to 10× more samples, despite having the same network architecture and trained on the same 100K queries. Both variants' accuracies drop off considerably for low-selectivity queries, since, when there are no hits in the materialized sample, the model relies solely on the query features to make "predictions". MSCN-0 which does not use materialized samples performs much worse, obtaining a max error of 6145×.

6.2.2 *Results on* Conviva-A

Based on DMV results, we keep only the promising baselines for this dataset. Table 4 shows that Naru remains best-in-class for a dataset with substantially different columns and a much larger joint size.

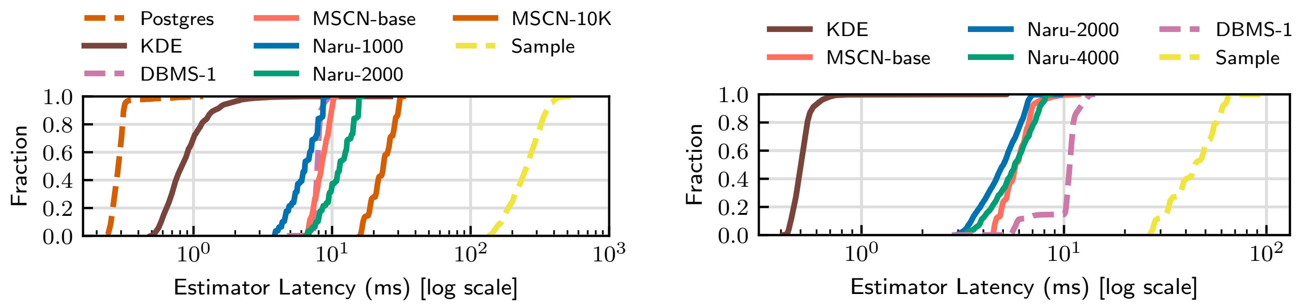
For this dataset, most estimators produce larger errors. This is because Conviva-A has a much larger joint space. DBMS-1, MHIST, BayesNet, and KDE-superv exhibit 5×, 14×, 1.8×, and 2.6× worse max error than before respectively. MSCN-base's max error in the medium-selectivity regime is also 14× worse. As a non-parametric method covering the full joint space, Sample remains a robust choice.

For Naru, since the sampler needs to cross more domains and a much larger joint space, Naru-1000 becomes insufficient to provide single-digit error in all cases. However, a modest scaling of the number of samples to 4K decreases the worst-case error back to single-digit levels. This suggests that the approximated joint is sufficiently accurate, and that the key challenge lies in *extracting its information.*



(a) DMV **(b)** Conviva-A

Figure 5: Training time vs. quality (§6.4). Dotted lines show divergence from data; bars show max estimation errors.



(a) DMV **(b)** Conviva-A

Figure 6: Estimator latency (§6.5). Learning methods are run on GPU; other estimators are run on CPU (dashed lines).

6.3 Robustness to Out-of-Distribution Queries

Our experiments thus far have drawn the filter literals (query centers) from the data. However, a strong estimator must be robust to out-of-distribution (OOD) queries where the literals are drawn from the entire joint domain, which often result in no matching tuples. Table 5 shows results on select estimators on 2K OOD queries on DMV, where 98% have a true cardinality of zero. The supervised MSCN-10K suffers greatly (e.g., median is now 23×, up from the 1.51× in Table 3) because it was trained on a set of in-distribution queries; at test time, out-of-distribution queries confuse the net. KDE-superv, a sampling-based approach, finds no hits in its sampled tuples, and therefore appropriately assigns zero density mass for all queries.

Table 5: Robustness to OOD queries. Errors from 2,000 queries.

Estimator	Median	95th	99th	Max
MSCN-10K	23	96	151	417
KDE-superv	1.00	1.00	3.67	163
Sample	1.00	1.00	2.00	116
Naru-2000	1.00	1.00	1.26	4.00

Since Naru approximates the data distribution, it correctly learns that out-of-distribution regions have little or no density mass, outperforming KDE by 40× and MSCN by 104×.

6.4 Training Time vs. Quality

Compared to supervised learning, Naru is efficient to train: no past queries are required; we only need access to a uniform random stream of tuples from the relation. We also find that, surprisingly, it only takes a few epochs of training to obtain a sufficiently powerful Naru estimator.

Figure 5 shows how two quality metrics, entropy gap and estimation error, change as training progresses. The metrics are calculated after each epoch (one pass over the data) finishes. An epoch takes about 75 seconds and 50 seconds for DMV and Conviva-A, respectively. The number of progressive samples is set to 2K for DMV and 8K for Conviva-A.

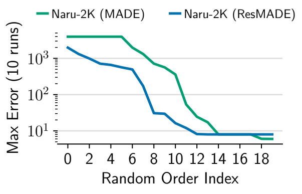
Observe that Naru quickly converges to a high goodness-of-fit both in terms of entropy gap and estimation quality. For DMV where a larger Naru model is used, 1 epoch of training suffices to produce the best estimation accuracy compared to all baselines (Table 3, last column). For Conviva-A, 2 epochs yields the best-in-class quality and about 15 epochs yields the quality of single-digit max error.

6.5 Estimation Latency

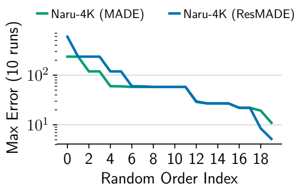
Figure 6 shows Naru's estimation latency against other baselines. On both datasets Naru can finish estimation in around 5-10ms on a GPU, which is faster than scanning samples (Sample and MSCN) and is competitive with DBMS-1. We note the caveat that latencies for Postgres and DBMS-1 include producing an entire plan for each query.

Naive progressive sampling requires as many model forward passes as the number of attributes in the relation. With Naru's wildcard-skipping optimization (§5.2), however, we can skip the forward passes that generate distributions for the wildcard columns. Hence, the number of forward passes required is the number of non-wildcard columns in each query. This effect manifests in the slightly slanted nature of Naru's CDF curves—queries that only touch a few columns are faster to estimate than those with a larger number of columns. Latency tail is also well-behaved: on DMV, Naru-1000's median is at 6.4ms vs. max at 9.4ms; on Conviva-A, Naru-2000's median is at 5.0ms vs. max at 9.7ms.

Naru's estimation latency can be further minimized by engineering. Naru's sampler is written in Python code and a general-purpose deep learning framework (PyTorch); the resultant control logic overhead from interpretation can be removed by using hand-optimized native code. Orthogonal techniques such as half-precision, i.e., 32-bit floats quantized into 16-bit floats, would shrink Naru's compute cost by half.

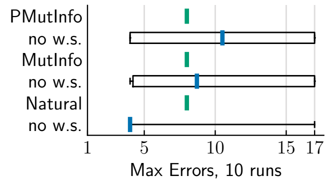


(a) DMV

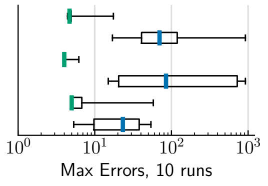


(b) Conviva-A

Figure 7: Variance of random orders. Each dataset's 2000-query workload is run 10 times; the maximum of these 10 max errors are shown. Order indices sorted by descending max error.



(a) DMV



(b) Conviva-A

Figure 8: Wildcard-skipping and heuristic orders. These orders have much lower variance than random orders. Ablation of wildcard-skipping is shown. Distributions of 10 max errors are plotted; whiskers denote min/max and bold bars denote medians.

Table 6: Larger model sizes yield lower entropy gap. Here we only consider scaling the hidden units of a MADE model.

Architecture	Size (MB)	Entropy gap, 5 epochs
$32 \times 32 \times 32 \times 32$	0.6	4.23 bits per tuple
$64 \times 64 \times 64 \times 64$	1.1	2.25 bits per tuple
$128 \times 128 \times 128 \times 128$	2.7	1.01 bits per tuple
$256 \times 256 \times 256 \times 256$	3.8	0.84 bits per tuple

Table 7: Comparison of autoregressive building blocks (§4.3). Max error calculated by running DMV's 2000-query workload 10 times (Naru-2000). FLOPs is the number of floating point operations required per forward pass per input tuple.

	Params	FLOPs	Ent. Gap	Max Error
MADE	3.3M	6.7M	0.59	8.0×
ResMADE	3.1M	6.2M	0.56	8.0×
Transformer	2.8M	35.5M	0.54	8.2×

6.6 Variance Analysis

Effect of random orders. Figure 7 shows the estimation variance of randomly sampled column orders. We sample 20 random orders and train a Naru model on each, varying the autoregressive building block. The result shows that the choice of ordering does affect estimation variance in the extreme tail. However, we found that on 99%-tile or below, almost all orderings can reach single-digit errors.

Effect of wildcard-skipping (§5.2) and heuristic orders (§5.3). Figure 8 shows that the information-theoretic orders have much lower variance than randomly sampled ones. The left-to-right order (Natural) is also shown for comparison. We also find that wildcard-skipping is critical in reducing max error variance by up to several orders of magnitude (e.g., MutInfo's max drops from 10^3 to < 10).

6.7 Autoregressive Model Choice and Sizing

In Table 6, we measure the relationship between model size and entropy gap on Conviva-A. While larger model sizes yield lower entropy gaps, Figure 5 shows that this can yield diminishing returns in terms of accuracy.

Table 7 compares accuracy and (storage and computation) cost of three similarly sized autoregressive building blocks. The results suggest that ResMADE and regular MADE are preferable due to their efficiency. We expect the Transformer—a more advanced architecture—to excel on datasets of larger scale.

6.8 Understanding Estimation Performance

Naru's accuracy depends critically on two factors: (1) the accuracy of the density model; and (2) the effectiveness of progressive sampling. This section seeks to understand the interplay between the two components and how each contributes to estimation errors. We do this by running microbenchmarks against the Conviva-B dataset, which has only 10K rows but has 100 columns for a total joint space of 10^{190}. The small size of the dataset makes it possible to run queries against an emulated *oracle model* with perfect accuracy by scanning the data. This allows us to isolate errors introduced by density estimation vs. progressive sampling.

6.8.1 *Robustness to Increasing Model Entropy Gap*

One natural question is: how accurate does the density model have to be? One metric of modeling accuracy is *the fraction of total probability mass assigned to observed data tuples.* For example, a randomly initialized model will assign equal probability mass to all points in the joint space of tuples. As training proceeds, it learns to assign higher probability to tuples actually present in the relation. Under the simplifying assumption that all relation tuples are unique (as they are in Conviva-B), we can quantify this fraction as follows. Suppose the model has an entropy gap of 2 bits; then, the fraction of probability mass assigned to the relation, f, satisfies $-\log_2 f = 2$, which leads to $f = 25\%$.

Figure 9 shows that Naru achieves the best performance with a model entropy gap of 0-2 bits. A gap of lower than 0.5 bits does not substantially improve performance. This means that for the best accuracy, the model must assign between $25 - 100\%$ of the probability mass to the empirical data distribution. Surprisingly, Naru still outperforms baselines with up to 10 bits of entropy gap, which corresponds to less than $\approx 0.1\%$ probability mass assigned. We hypothesize that the range queries make such modeling errors less critical, because density errors of individual tuples could even out when estimating the density of the region as a whole.

6.8.2 *Robustness to Increasing Column Counts*

While the datasets tested in macrobenchmarks have a good number of columns, using Conviva-B we test how well progressive sampling scales to 10× as many dimensions. Figure 10 shows that while the number of columns does significantly increase the variance of estimates, the number of progressive samples required to mitigate this variance remains tractable. A choice of 1000 sample paths produces reasonable worst-case accuracies for up to 100 columns, and 10000 sample paths improves on that by a modest factor.

6.8.3 *Robustness to Data Shifts*

Lastly, we study how Naru reacts to data shifts. We partition DMV by a date column into 5 parts. We then ingest each partition in order, emulating the common practice of "1

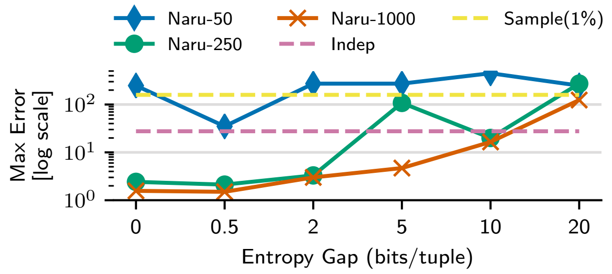


Figure 9: Accuracy of Naru as an artificial entropy gap is added to an oracle model for Conviva-B projected to the first 15 columns. 50 queries are drawn from the same distribution as in the macrobenchmarks. Naru has the best accuracy for an entropy gap of less than 2 bits, though remains competitive up to a surprisingly large gap of 10 bits. Variance of progressive sampling decreases dramatically when moving from 50 to 250 to 1000 samples.

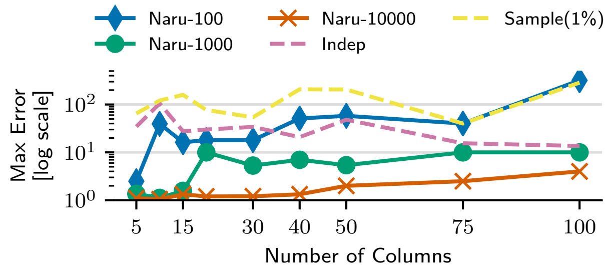


Figure 10: Accuracy of Naru as we add more columns from Conviva-B. We again use an oracle model (with 0 bits of entropy gap) and 50 randomly generated queries. The number of predicates covers at most 12 columns. The number of progressive sample paths required to accurately query the model increases modestly with the number of columns, but remains tractable even as the joint data space reaches over 10^{190} (at 100 columns).

new partition per day". Each estimator is built after seeing the first partition. After a new ingest, we test the previously built estimators on queries that touch all data ingested so far. The same query generator as macrobenchmarks is used where the filters are drawn from tuples in the first partition (true selectivities computed on all data ingested so far).

Table 8: Robustness to data shifts. Errors from 200 queries.

Partitions Ingested	1	2	3	4	5
Naru, refreshed: max	2.0	2.0	2.0	2.0	2.0
90%-tile	1.20	1.14	1.12	1.14	1.15
Naru, stale: max	2.0	40.3	47.5	52.9	53.5
90%-tile	2.0	2.4	3.4	4.4	5.5

Table 8 shows the results of (1) Naru, no model updates, (2) Naru, with gradient updates on each new ingest. The model architecture is the same as in Table 3; 8,000 progressive samples are used since we are interested in learning how much imprecision or staleness presents in the model itself and not the effectiveness of information extraction. The results show that, Naru is able to handle queries on new data with reasonably good accuracy, even without having seen the new partitions. The model has learned to capture the underlying data correlations so the degradation is graceful.

7. RELATED WORK

Naru builds upon decades of rich research on selectivity estimation and this section cannot replace comprehensive surveys [5]. Below, we highlight the most related areas.

Joint approximation estimators. Multidimensional histograms [16,33,37,38] can been seen as coarse approximations to the joint data distribution. Probabilistic relational models (PRMs) [14] rely on a Bayes Net (conditional independence DAG) to factor the joint into materialized conditional probability tables. Tzoumas *et al.* [46] propose a variant of PRMs optimized for practical use. Dependency-based histograms [7] make partial or conditional independence assumptions to keep the approximated joint tractable (factors stored as histograms). Naru belongs to this family and applies recent advances from the deep unsupervised learning community. Naru does not make *any* independence assumptions; it directly models the joint distribution and lazily encodes all product-rule factors in a universal function approximator.

Query-driven estimators are supervised methods that take advantage of past or training queries [3]. ISOMER [43] and STHoles [1] are two representatives that adopt feedback to improve histograms. LEO [45] and CardLearner [53] use feedback to improve selectivity estimation of future queries. Heimel *et al.* [17] propose query-driven KDEs; Kiefer *et al.* [19] enhance them to handle joins. Supervised learning regressors [10,22,29], some utilizing deep learning, have also been proposed. Naru, an unsupervised data-driven synopsis, is orthogonal to this family. Our evaluation shows that full joint approximation yields accuracy much superior to two supervised methods.

Machine learning in query optimizers. Naru can be used as a drop-in replacement of selectivity estimator used in ML-enhanced query optimizers. Ortiz *et al.* [34] learns query representation to predict cardinalities, a *regression* rather than our *generative* approach. Neo [31], a learned query optimizer, approaches cardinality estimation *indirectly*: embeddings for all attribute values are first pretrained; later, a network takes them as input and additionally learns to correct or ignore signals from the embeddings. This proposal, as well as reinforcement learning-based join optimizers (DQ [25], ReJOIN [32]), may benefit from Naru's improved estimates.

8. CONCLUSION

We have shown that deep autoregressive models are highly accurate selectivity estimators. They approximate the data distribution without any independence assumptions. We develop a Monte Carlo integration scheme and associated variance reduction techniques that efficiently handle challenging range queries. To the best of our knowledge, these are novel extensions to autoregressive models. Our estimator, Naru, exceeds the state-of-the-art in accuracy over several families of estimators.

Naru can be thought of as an unsupervised *neural synopsis*. In contrast to supervised learning-based estimators, Naru enjoys drastically more efficient training since there is no need to execute queries to collect feedback—it only needs to read the data. Learning directly from the underlying data allows Naru to answer a much more general set of future queries and makes it inherently robust to shifts in the query workload. Our approach is non-intrusive and can serve as an opt-in component inside an optimizer.

9. REFERENCES

[1] N. Bruno, S. Chaudhuri, and L. Gravano. STHoles: A multidimensional workload-aware histogram. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, SIGMOD '01, pages 211–222, New York, NY, USA, 2001. ACM.

[2] S. Chaudhuri, R. Motwani, and V. Narasayya. On random sampling over joins. In *ACM SIGMOD Record*, volume 28, pages 263–274. ACM, 1999.

[3] C. M. Chen and N. Roussopoulos. Adaptive selectivity estimation using query feedback. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, SIGMOD '94, pages 161–172, New York, NY, USA, 1994. ACM.

[4] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467, 1968.

[5] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1-3):1–294, 2011.

[6] T. M. Cover and J. A. Thomas. *Elements of information theory.* John Wiley & Sons, 2012.

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

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[9] C. Durkan and C. Nash. Autoregressive energy machines. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1735–1744, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[10] A. Dutt, C. Wang, A. Nazi, S. Kandula, V. Narasayya, and S. Chaudhuri. Selectivity estimation for range predicates using lightweight models. *PVLDB*, 12(9):1044–1057, 2019.

[11] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning.* Springer series in statistics New York, 2001.

[12] M. Germain, K. Gregor, I. Murray, and H. Larochelle. MADE: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889, 2015.

[13] L. Getoor, N. Friedman, D. Koller, and B. Taskar. Learning probabilistic models of relational structure. In *ICML*, volume 1, pages 170–177, 2001.

[14] L. Getoor, B. Taskar, and D. Koller. Selectivity estimation using probabilistic models. In *ACM SIGMOD Record*, volume 30, pages 461–472. ACM, 2001.

[15] G. Grimmett, D. Stirzaker, et al. *Probability and random processes.* Oxford university press, 2001.

[16] D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi. Selectivity estimators for multidimensional range queries over real attributes. *The VLDB Journal*, 14(2):137–154, 2005.

[17] M. Heimel, M. Kiefer, and V. Markl. Self-tuning, gpu-accelerated kernel density models for multidimensional selectivity estimation. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1477–1492, New York, NY, USA, 2015. ACM.

[18] R. Kaushik, J. F. Naughton, R. Ramakrishnan, and V. T. Chakravarthy. Synopses for query optimization: A space-complexity perspective. volume 30, pages 1102–1127, New York, NY, USA, Dec. 2005. ACM.

[19] M. Kiefer, M. Heimel, S. Breß, and V. Markl. Estimating join selectivities using bandwidth-optimized kernel density models. *PVLDB*, 10(13):2085–2096, 2017.

[20] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[21] A. Kipf. Github repository, learnedcardinalities. `github.com/andreaskipf/learnedcardinalities`, 2019. [Online; accessed March, 2019].

[22] A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper. Learned cardinalities: Estimating correlated joins with deep learning. In *CIDR 2019, 9th Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 13-16*, 2019.

[23] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques.* MIT press, 2009.

[24] F. Korn, T. Johnson, and H. Jagadish. Range selectivity estimation for continuous attributes. In *Proceedings. Eleventh International Conference on Scientific and Statistical Database Management*, pages 244–253. IEEE, 1999.

[25] S. Krishnan, Z. Yang, K. Goldberg, J. Hellerstein, and I. Stoica. Learning to optimize join queries with deep reinforcement learning. *arXiv preprint arXiv:1808.03196*, 2018.

[26] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann. How good are query optimizers, really? *PVLDB*, 9(3):204–215, 2015.

[27] V. Leis, B. Radke, A. Gubichev, A. Kemper, and T. Neumann. Cardinality estimation done right: Index-based join sampling. In *CIDR*, 2017.

[28] V. Leis, B. Radke, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann. Query optimization through the looking glass, and what we found running the join order benchmark. *The VLDB Journal*, pages 1–26, 2018.

[29] H. Liu, M. Xu, Z. Yu, V. Corvinelli, and C. Zuzarte. Cardinality estimation using neural networks. In *Proceedings of the 25th Annual International Conference on Computer Science and Software Engineering*, pages 53–59. IBM Corp., 2015.

[30] M. Heimel. Bitbucket repository, feedback-kde. `bitbucket.org/mheimel/feedback-kde`, 2019. [Online; accessed March, 2019].

[31] R. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh,

T. Kraska, O. Papaemmanouil, and N. Tatbul. Neo: A learned query optimizer. *PVLDB*, 12(11):1705–1718, 2019.

[32] R. Marcus and O. Papaemmanouil. Deep reinforcement learning for join order enumeration. In *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, aiDM'18, pages 3:1–3:4, New York, NY, USA, 2018. ACM.

[33] M. Muralikrishna and D. J. DeWitt. Equi-depth multidimensional histograms. In *ACM SIGMOD Record*, volume 17, pages 28–36. ACM, 1988.

[34] 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*, DEEM'18, pages 4:1–4:4, New York, NY, USA, 2018. ACM.

[35] Y. Park, S. Zhong, and B. Mozafari. Quicksel: Quick selectivity learning with mixture models. *arXiv preprint arXiv:1812.10568*, 2018.

[36] M. Perron, Z. Shang, T. Kraska, and M. Stonebraker. How I learned to stop worrying and love re-optimization. In *35th IEEE International Conference on Data Engineering, ICDE 2019*, 2019.

[37] V. Poosala, P. J. Haas, Y. E. Ioannidis, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, SIGMOD '96, pages 294–305, New York, NY, USA, 1996. ACM.

[38] V. Poosala and Y. E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB*, volume 97, pages 486–495, 1997.

[39] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *URL https://openai.com/blog/better-language-models*, 2019.

[40] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. PixelCNN++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[41] D. W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization.* John Wiley & Sons, Inc., 1992.

[42] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, pages 23–34. ACM, 1979.

[43] U. Srivastava, P. J. Haas, V. Markl, M. Kutsch, and T. M. Tran. ISOMER: Consistent histogram construction using query feedback. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 39–39. IEEE, 2006.

[44] State of New York. Vehicle, snowmobile, and boat registrations. catalog.data.gov/dataset/vehicle-snowmobile-and-boat-registrations, 2019. [Online; accessed March 1st, 2019].

[45] M. Stillger, G. M. Lohman, V. Markl, and M. Kandil. LEO-DB2's learning optimizer. In *VLDB*, volume 1, pages 19–28, 2001.

[46] K. Tzoumas, A. Deshpande, and C. S. Jensen. Lightweight graphical models for selectivity estimation without independence assumptions. *PVLDB*, 4(11):852–863, 2011.

[47] B. Uria, I. Murray, and H. Larochelle. A deep and tractable density estimator. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages I–467–I–475. JMLR.org, 2014.

[48] A. Van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. WaveNet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[49] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in neural information processing systems*, pages 4790–4798, 2016.

[50] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[51] T. L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In *ICDT*, 2014.

[52] S. D. Viglas, J. F. Naughton, and J. Burger. Maximizing the output rate of multi-way join queries over streaming information sources. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 285–296. VLDB Endowment, 2003.

[53] C. Wu, A. Jindal, S. Amizadeh, H. Patel, W. Le, S. Qiao, and S. Rao. Towards a learning optimizer for shared clouds. *PVLDB*, 12(3):210–222, Nov. 2018.

[54] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019.