Pre-training Summarization Models of Structured Datasets for Cardinality Estimation

Yao Lu, Srikanth Kandula, Arnd Christian König, Surajit Chaudhuri
Microsoft
project-iris@microsoft.com

ABSTRACT

We consider the problem of pre-training models which convert structured datasets into succinct summaries that can be used to answer cardinality estimation queries. Doing so avoids per-dataset training and, in our experiments, reduces the time to construct summaries by up to 100×. When datasets change, our summaries are incrementally updateable. Our key insights are to use multiple summaries per dataset, use learned summaries for columnsets for which other simpler techniques do not achieve high accuracy, and that analogous to similar pre-trained models for images and text, structured datasets have some common frequency and correlation patterns which our models learn to capture by pre-training on a large and diverse corpus of datasets.

PVLDB Reference Format:
Yao Lu, Srikanth Kandula, Arnd Christian König, Surajit Chaudhuri. Pre-training Summarization Models of Structured Datasets for Cardinality Estimation. PVLDB, 15(3): 414 - 426, 2022.
doi:10.14778/3494124.3494127

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at http://yao.lu/iris.

1 INTRODUCTION

We consider using dataset summaries to quickly and approximately answer cardinality estimation (CE) queries. There is substantial prior work [15, 16, 20, 21, 26, 28, 42–44] on this problem especially in the context of query optimizers (QO). In practical systems, the state-of-the-art is to use histograms [36] and sketches [21].

Recent works learn *dataset-specific* models to answer CE queries [27, 34, 39, 45, 50, 51]. Training these models is costly since some require thousands of training examples for each dataset; each training example is a (predicate, cardinality) pair obtained from history or computed over the dataset. Moreover, these models do not evolve with the dataset; when rows are appended or edited, the models must be retrained [27]. The large cost to build (and, when needed, retrain) these models makes data ingestion an expensive operation, especially on databases that host many relations. In this work, we seek to answer CE queries efficiently *without* per-dataset training.

We also note that today's query optimizers make hundreds of CE calls to optimize a query [41]; some prior learned models need a few milliseconds per CE call aided by a GPU [39, 51]. In this work, we seek to provide CE well within a millisecond without GPUs. Doing so allows the technique to be used in practical database servers.

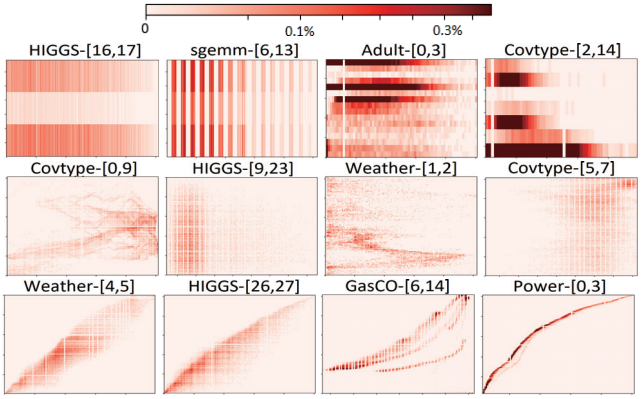

Figure 1: Illustrating some frequency and correlation patterns; for datasets in Table 4, we show heat-plots with the axes corresponding to the ranges of two columns and point darkness indicating joint frequency.

Inspired by similar models of images and text documents (e.g., ResNet[32] and BERT [24], respectively), we seek to pre-train models which convert any previously unseen dataset into summaries that in turn can be used to answer CE queries. Pre-training on a large corpus of datasets, intuitively, is feasible because there are common patterns in the frequency and correlation distributions of multi-attribute datasets [22, 24]. Figure 1 shows patterns in some column pairs from real-world datasets.

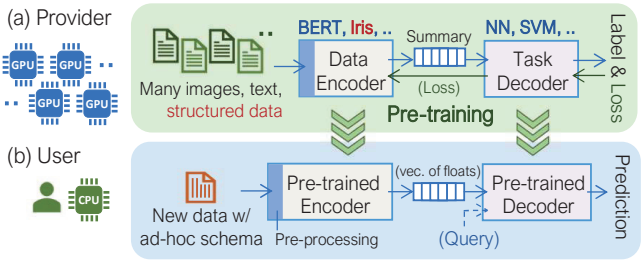

Figure 2: Building and applying pre-trained models. Iris **provides off-the-shelf models for structured datasets.**

Our solution, Iris[1], consists of an *encoder* that applies a learned function on each input row, a *summary* that is a linear function over the encoder outputs, and a *decoder* that answers queries using

[1] short for **I**ncrementable **R**epresentations of multi-d**I**mensional **S**ets

Proceedings of the VLDB Endowment, Vol. 15, No. 3 ISSN 2150-8097.
doi:10.14778/3494124.3494127

Table 1: Quantization and normalization are necessary to train and apply a model on diverse inputs: note that capturing the diversity of structured datasets is more complex than the corresponding case of image and text datasets.

	Images	Text (word model)	Structured data
Data size	Arbitrary (# pixels)	Arbitrary (# words)	Arbitrary (# rows)
Data unit	Pixel = 3 RGB octets	Word from a known vocabulary	Row has multiple # columns of varying types and sizes
Normalization	Resize to W×H pixels	Avg over sections, each w/ n words padded w/ white-space	See §4 for details
Quantization	Not needed	Tokenize into $\in [0, v_t]$ using word-to-vec	See §4 for details
Input to ML model	W×H×3 octets	$n \log_2 v_t$	See §4 for details
Order dependence	Sensitive to pixel location	Sensitive to word location	Independent of row order
Correlation	Pixel co-occurrence	Word co-occurrence	Multi-dimensional correlation

the summary. As shown in Figure 2, a model provider pre-trains summarization models once, offline, over a large corpus of structured datasets. The models apply off-the-shelf on a new dataset without further training; we will show that Iris builds summaries quickly and that the summaries are small, can be queried efficiently, and provide accurate cardinality estimates.

To be incrementally updateable, Iris constrains the space of summarization models to only be simple linear functions per row. But can such simple functions deliver accurate cardinality estimates when using only small summaries? Moreover, structured datasets have a rich variety in schema, frequency distributions and correlations; can we pre-train models that generalize across such a rich variety? One key insight that we use in Iris is simple in hindsight: capturing correlation between column pairs, triples etc. suffices when (1) most predicates use only a few columns or (2) low-dimensional correlations may approximate well the higher-dimensional relations; our results will show that this holds often in practice. Hence, Iris maintains multiple summaries per dataset – one for each correlated column subset and uses a novel scheme to quantize low-dimensional columnsets of arbitrary schemas into fixed-size vectors. Our pre-trained models transform these fixed-size vectors into summaries of a given size. We also apply model selection to decide how best to summarize a given set choosing between histograms, sparse representations and learned Iris summaries.

We are unaware of any pre-trained models that summarize structured datasets. BERT and ResNet [24, 32] are the closest analogues. Some sketches [15, 16, 21, 28] apply incrementally per row and can answer CE queries quickly but the functions to encode rows and decode answers are defined by experts whereas Iris uniquely learns these functions from data. Some prior works also use multiple summaries [23, 30, 47]; Iris is novel in its use of learned summaries and different types of summaries for different columnsets.

In summary, our work has the following contributions:

- We show how to pre-train and use summarization models including (1) pre-training Iris encoders to generate summaries, and (2) pre-training Iris decoders that answer CE queries using the summaries. The summaries can be incrementally computed or updated and all models execute efficiently using SIMD on a CPU.
- A pre-processor that takes as input a relation and determines which column subsets to jointly summarize and which summarization method to use per column subset. Doing so crucially lets us support relations with many columns and ad-hoc schemas using a small number of models. The pre-processor also reduces storage and computation costs for pre-training.
- For cardinality estimation, we compare Iris with sampling, histograms, and state-of-the-art methods that learn a model for each dataset [27, 34, 37, 50, 51]. Over various datasets and predicates, results show that Iris offers a similar or better accuracy at a small storage budget while substantially reducing the latency to build summaries compared to the ML-based counterparts.

2 OVERVIEW OF IRIS

In this section, we discuss the scope of Iris, our goals, and key aspects of how we build and use summaries.

Scope: Iris supports queries which can be represented as:

SELECT `COUNT(*)` **FROM** T
WHERE $l_1 \leq \texttt{Col}_1 \leq u_1$ **AND** ... $l_i \leq \texttt{Col}_i \leq u_i$ ...,

where T is a table and column Col_i can be of numeric or categorical types. Equality, one-sided predicates and predicates over a subset of columns also belong in the above class.[2] Some generalizations also follow directly, e.g., using multiple calls for disjunctions.

This scope is largely similar to prior ML-based CE [27, 34, 37, 50, 51] but we note some key aspects: Iris does not support joins unlike some prior works [34, 37]; Iris supports categorical columns but some prior works do not [27, 37] and finally group-by's and other relational operations are not supported by any of these works.

Requirements: Iris seeks to improve CE accuracy within the constraints of today's production database systems, i.e., use summaries that are about as large as the per-column histograms that are maintained by production databases today and respond to CE calls in about as much time as used by production cardinality estimators without GPUs. Also, importantly, Iris aims to reduce the costs to build summaries by using models that are pre-trained offline and Iris's summaries should be incrementally updateable.

Many recent works that train a model per dataset [27, 37, 51] do not update incrementally. Even on the other aspects, our results will show that prior works do not provide pareto improvements [42]; e.g., better accuracy typically requires more storage, or a longer estimation latency or a higher construction cost. Prior works require tens to hundreds of minutes to train a model per dataset [27, 34, 39, 51], build models as large as tens of MBs [39, 51] and take a few milliseconds per cardinality estimate using a GPU [39]. In contrast, a CE call takes 0.1-1ms [27] in today's production databases without GPUs. Moreover, the memory footprint to store summaries and the build costs per dataset will pose a large burden on production servers which host thousands of datasets.

Summarization challenges and solutions: Pre-training summarization models for datasets poses some novel challenges:

[2] For equality, set $l_i = u_i$; for one-sided ranges, set $l_i = \min(\text{Col}_i)$ or $u_i = \max(\text{Col}_i)$; for columns that are not in the predicate, set $l_i = \min \text{Col}_i$ and $u_i = \max \text{Col}_i$.

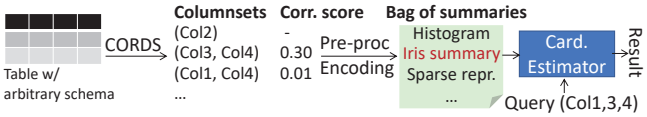

Figure 3: Using multiple summaries and choosing different techniques for different columnsets.

First, image and text models [24, 32] are sensitive to the context and location of a pixel or a word but, since structured datasets are multi-sets, row order should have no effect on the dataset summaries. In Iris, summaries are a simple linear function per-row: $\frac{1}{|S|}\sum_{s\in S}\psi(s)$, where ψ is a learned function for each row s in a multi-set S. It is easy to see that such summaries are mergeable and updateable. ψ can apply incrementally and in parallel over different data subsets while using a small memory footprint.

Next, to generalize to diverse inputs, ML models must convert arbitrary datasets (or images or documents) into fixed-size vectors that a learned function can apply over (see Table 1). We find that quantizing structured datasets is more complex than images and text [31] because summarization models must handle tables with different numbers of rows, different schemas and columns with different ranges and distinct counts. We discuss our method in §3.3.

Third, jointly summarizing columns that are either uncorrelated or only weakly correlated wastes space and computation. Hence, we leverage prior work [35] to identify low-dimensional correlated column subsets and maintain multiple summaries: one for each correlated column subset. Doing so offers more accurate cardinality estimates compared to a single summary over all columns at a given storage or computational budget. Using multiple summaries also lowers training complexity by balancing generalization and specification; we neither train a single summarization model for all schemas (which, due to the richness of schemas, will require a very large model or has high error) nor do we train a different model for each possible schema (which will require training too many models). Instead, we train a small number of summarization models and use them to accommodate a wide variety of relations.

Finally, we use model selection to decide how best to summarize each columnset (e.g., least error in CE at a given resource budget) because we find that histograms and sparse representations are better suited than learned summaries for some columnsets.

Pre-training overview: With Iris, a model provider uses a large corpus of datasets to pre-train a few different Iris models which are parameterized by the input size (i.e., the number of bits that a column set is quantized into) and the output size (i.e., the size of the summary). The pre-training cost amortizes over the many datasets that the same pre-trained model can be used for.

Overview of summarizing a new dataset: As shown in Figure 3, we first identify columnsets that are correlated (§3.1) because jointly summarizing the values of such columnsets leads to more succinct summaries and more accurate answers for predicates over the columnsets. We prefer to jointly summarize columnsets which have fewer columns and high correlation between columns. Next, we apply a simple model selection process to pick the appropriate summarization technique (§3.2) for each columnset from among a few different methods including multi-d histograms, sparse representations and Iris models. Intuitively, we pick the simplest method that can achieve good accuracy given a storage budget and only use the learned Iris models when simpler methods do not provide accurate CE. Thus, effectively, we maintain a *bag of summaries* per dataset and each column is covered by at least one summary.

Estimating cardinality using a bag of summaries is not trivial because the predicate columns can be covered in more than one summary. Consider a case where summaries exist for {Col1, Col2} and {Col2, Col3} and the predicate is $p_1 \wedge p_2 \wedge p_3$; here p_i is a predicate on Col$i$. We have two choices: evaluate $p_1 \wedge p_2$ on the former multi-column summary and p_3 on the latter or evaluate p_1 on the former and $p_2 \wedge p_3$ on the latter. It is unclear which choice would result in a more accurate estimate. We use a heuristic which preferentially uses summaries that cover more correlated columns and is cognizant of imprecision in the cardinality estimates (§3.4).

3 DATA SUMMARIZATION

Here, we discuss how to bring our CE solution to bear on an arbitrarily wide table; we defer discussing the training and use of Iris models to §4. In §3.1, we discuss the correlation analyses which determines which column subsets to jointly summarize. In §3.2, we discuss the model selection process which identifies an appropriate summarization technique for each column subset. In §3.3, we discuss the quantization and normalization that we use before building summaries. Finally, §3.4 discusses how we estimate the cardinality of a given query predicate using a bag of summaries. Table 2 summarizes the notations used in this paper.

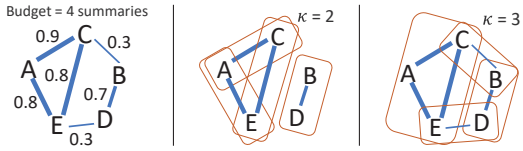

Figure 4: Choosing columnsets to jointly summarize. Left: each node is a column and edge widths denote correlation scores. Middle and right: two greedy choices.

3.1 Picking column subsets to summarize

A table with d columns has $2^d - 1$ column subsets; thus, jointly summarizing every subset is expensive (in terms of storage and compute) and may record redundant information. For example, columns that are independent of others can be summarized individually. Our goal is to capture as much of the inter-column correlations as possible given resource constraints.

We use CORDS [35] to analyze correlations between columns of a dataset. CORDS uses a small sample[3] to estimate the *mean-square contingency* between a *p*air of columns. We implement CORDS and compute the mean-square contingency value between all column pairs which we call *correlation scores*.

Using these pairwise correlation scores, we pick column subsets to summarize as follows: on a graph where columns are nodes and edges are weighted based on the correlation score, pick a clique

[3] a few thousand rows picked uniformly at random and independent of the table size

Table 2: Notations used in the paper.

Notation	Meaning
n, d	Row and column counts in the relation.
κ	Clique size in correlation analysis (§3.1).
$\epsilon_0, \epsilon_1, \epsilon_2$	Cutoffs used in model selection (§3.2).
ℓ	Size of quantized vector and the input to a summarization model.
η	Size of an Iris summary.
$q()$	Quantization func. to replace column value with identifier (§3.3).
ξ	Max. size (#bits) of a column's quantized value (§3.3).
ψ	Pre-trained summarization model (§4).
ϕ_c, ϕ_r	Learned column and row embeddings in the Iris encoder (§4.1).
ϕ_1, ϕ_2, ϕ_3	Learned functions in the Iris decoder (§4.1).

of no more than κ nodes which has the largest normalized edge weight of all such cliques. While storage budget remains, for each picked clique, we build a multi-column summary and remove all of the picked edges before proceeding to pick a new clique. All columns that remain un-picked when storage space is used up are summarized using 1-d histograms. Figure 4 shows an example.

Our results will show that the above logic is a good trade-off between storage size, model training costs and the ability to capture multi-column correlations which affect the accuracy of cardinality estimates. Note that a column may appear in more than one summary. Estimates of higher dimensional correlations, if available, will improve our choice of which columnsets to jointly summarize; however computing these estimates with a small sample is non-trivial and orthogonal to our work and so we only use the pair-wise correlations estimated by CORDS [35]. Note also that the complexity of the above logic is $O(d^{\kappa+1})$ given constant κ. To keep costs small, we use $\kappa = 2$ unless otherwise specified; that is, Iris builds summaries only for column pairs. Summarizing larger columnsets can use less storage or improve accuracy but incurs higher model training cost (due to more parameters); as we will show in our experiments, summaries of column pairs suffice for Iris to offer good results on a large collection of datasets (Table 5) including those that have higher dimensional correlations. We will also show that summaries can be small (e.g., each is below 1KB) and the overall storage budget is also small (e.g., between 0.5× to 2× of the storage space used to maintain summaries in today's production systems).[4]

3.2 Model selection for summarization

For each column subset chosen above, we determine how to summarize so as to achieve accurate cardinality estimates. Recall that every column appears in at least one summary. Our model selection process below executes on the same samples used by CORDS above and is based in part on the correlation score s_k of column subset k and the number of distinct values $\#\mathrm{DV}_k$.[5]

Sparse We use a sparse representation when $\#\mathrm{DV}_k < \epsilon_0$; that is, record a set of tuples (r, m) where r is a distinct row and m is the frequency with which r occurs in the dataset.

AVI If $s_k < \epsilon_1$, we assume that the columns are nearly independent and only use per-column histograms.

Hist If $s_k \in [\epsilon_1, \epsilon_2]$, we maintain a multi-d histogram.

Iris For all of the other cases, we maintain Iris summaries.

[4]Most systems maintain 1-d histograms per column [5, 6, 8].

[5]For simplicity, we approximate $\#\mathrm{DV}_k$ with the number of non-zero buckets in the 2-d histogram that CORDS builds when computing the correlation score for columnpair k.

Algorithm 1: Assign quantization budget to columns

Input : d columns, ℓ bits for quantization, H_i estimate of #DVs in column i, ξ max #buckets that can be allocated on a column.

Output: $\mathbf{T} = \{t_j\}, j = 1 \ldots d$, denotes t_j bits assigned to column j; $\sum t \leq \ell$.

1 $T \leftarrow \{1\} \times d$;

2 Let $\mathrm{Acc}(x) \leftarrow 1 - \frac{0.5}{2^{x+1}}$;

3 **while** $\sum t < \ell$ **do**

4 $\quad j \leftarrow arg\max_{i:2^{t_i+1} < \min(\mathrm{H_i},\, \xi)} (\mathrm{Acc}(t_i) + \epsilon \mathrm{H}_i)$;

5 $\quad j \leftarrow 0$ if j undefined;

6 $\quad t_j \leftarrow t_j + 1$ // allocate next quantization bit to column j;

7 **end**

Intuitively, the above process uses the simplest summary that can achieve desired accuracy given a storage size; in particular, we use Iris summaries only when other simpler methods are not appropriate. In more detail, when a column pair has very few distinct rows (below ϵ_0), the sparse representation explicitly stores all frequencies and offers perfect selectivity estimates. We choose ϵ_0 based on the per-summary size. Next, for column pairs that have modest correlation, we use a multi-d histogram[6] which offers more accurate selectivity estimates. We use Iris for all of the remaining cases; that is, when the column sets are correlated and have a larger number of distinct rows.

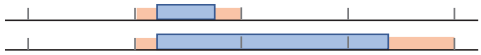

Figure 5: Range predicates (in blue) incur an uncertainty of a half of a bucket's width (region in orange) on average.

3.3 Quantization for rows

For models to generalize, as noted in Table 1, we must quantize inputs into fixed-size vectors. For example, images of different kinds are converted to a fixed number of pixels [32]. Quantizing datasets is a challenge because we must convert each row into a fixed-size vector regardless of the ranges, types and distinctness of column values. Some prior works use one-hot encoding [37, 45] to train models per-dataset which does not generalize to unseen schemas. Our quantization method assigns to each column a certain number of buckets and then replaces column values with the index of the bucket that contains the value. If the desired size of the quantized vector is ℓ bits, then the product of the number of buckets assigned to each column must be 2^ℓ; the key is how best to allocate buckets among the columns so as to reduce quantization error. Our method is shown in Algorithm 1. Our insight is that, under certain simplifying assumptions,[7] the quantization error for a column which we estimate as the expected cardinality error on a range-predicate is as shown on line #2 of Algorithm 1. Figure 5 illustrates this for some examples; when a column is assigned x bits, each bucket occupies $\frac{1}{2^x}$'th of the range and the expected relative accuracy on range predicates is $1 - \frac{0.5}{2^x}$. Building on this idea, in line #4, Algorithm 1 greedily allocates each bit to the column that is most likely to improve the accuracy of range predicates; here, H_i is an estimate of distinct count in column i; H_i can be obtained at a low cost using

[6]We reuse ideas from §3.3 to pick the boundaries for the multi-d histogram so as to optimize accuracy given a storage budget.

[7]frequency distribution is uniform over distinct values, all columns are equally likely to appear in range predicates etc.

Algorithm 2: CE with possibly overlapping summaries

Input : Query q, bag of summaries $\mathcal{B} = \{(C_i, \mathcal{S}_i)\}$
Output: Estimated cardinality card.

1 $\forall i,\ \mathcal{D}_i \leftarrow C_i \cap \text{ColsOf}(q)$ // columns in both summary and predicate
2 Relevant summaries, $\mathcal{R} \leftarrow \{b_i \in \mathcal{B} | \mathcal{D}_i \neq \emptyset\}$;
3 For $b_i \in \mathcal{R}$, $\text{sel}_i, \text{vprec}_i \leftarrow \text{Eval}(\mathcal{S}_i, \text{Adjust}(q, \mathcal{S}_i, \mathcal{D}_i))$;
4 card $\leftarrow 1$; covcols $\leftarrow \emptyset$;
5 **for** b_i *in ascending order of* vprec_i, sel_i, $|\mathcal{D}_i - \text{covcols}|$ **do**
6 $\quad \widehat{\text{sel}_i} \leftarrow \text{Eval}(\mathcal{S}_i, \text{Adjust}(q, \mathcal{S}_i, \mathcal{D}_i - \text{covcols}))$
7 $\quad \text{card} \leftarrow \text{card} * \widehat{\text{sel}_i}$
8 $\quad \text{covcols} \leftarrow \text{covcols} \cup \mathcal{D}_i$
9 **end**

GEE estimates [18] over the CORDS samples (§3.1) or by querying HLL sketches maintained by production systems [6, 8, 9]. We defer some details of Algorithm 1 such as how we compute the bucket boundaries for each column to [7]. This algorithm can improve by using the actual frequency distribution and/or knowledge of the range predicates in the workload but, as we will show in our experiments, Algorithm 1 is simple, fast and already quite useful.

3.4 CE using a bag of summaries

Given a bag of summaries constructed for different column pairs, how to estimate the cardinality of a query predicate? The challenge here is that the predicate columns can be covered by more than one summary. Consider an example where summaries exist for {Col1}, {Col2} and {Col1, Col2}, the last being a multi-column summary. For a predicate over columns {Col1, Col2}, intuitively, using the multi-column summary would lead to a more accurate answer since this summary encodes correlations between the columns. Now consider a more complex case where summaries exist for {C1, C2}, {C3, C4}, {C1, C3} and {C2, C4} and a predicate that uses all four columns. There are many choices here including using just the first two summaries or just the last two summaries but which of these choices would lead to a more accurate answer? Optimizing directly for accuracy is non-trivial because it is unclear how to relate accuracy to the information that is available to make this choice (i.e., the summaries and correlation scores between columns). Our heuristic approach, shown in Algorithm 2, preferentially uses summaries that are more precise, cover more predicate columns, and is cognizant of imprecision in the cardinality estimates.

Considering Algorithm 2 in more detail, in line#2, we pick summaries that cover at least one of the predicate columns. In line#3, we compute the selectivity from each picked summary. Since a summary may contain some but not all of the predicate columns, Adjust() picks clauses in q that use at least one column covered by the summary; furthermore, for any summary column not covered by these clauses, Adjust() adds a clause from the minimum to the max value of that column. The Eval() function in line#3 evaluates the adjusted predicate on each summary, and we discuss this further in §4.2. The latency of an Eval call is in micro-seconds and multiple calls can execute in parallel. In line#3, we also estimate the precision variance (vprec) based on the model that generates each cardinality estimate; specifically, we assess sparse representations, histograms and Iris summaries as generating estimates with precision variance 0, 1 and 2 respectively where lower value indicates better precision variance. We also *lazily* evaluate the summaries that have higher precision variance (such as Iris models) only when the more precise and faster summaries such as sparse representations do not cover all of the predicate columns. Finally, we assemble the combined cardinality estimate in lines#5–#8 by iterating over the relevant summaries in ascending lexicographic order of prec_i, sel_i and $|\mathcal{D}_i - \text{covcols}|$; the last term prefers summaries that cover columns not covered by previous summaries. We will show that the overall CE latency is sub-millisecond so that production query optimizers can use Iris without increasing the overall QO latency.

4 TRAINING AND USING IRIS MODELS

We describe how to train and use the learned summarization models in four parts – how to summarize a rowset (§4.1), how to update the summaries used by Iris (§4.3), how to answer a cardinality estimation query using a summary (§4.2), and how to train the summarization models (§4.4). Table 2 shows our notation.

4.1 Using Iris models to summarize a rowset

The summary of a multiset S which consists of n rows each having d columns is defined as:

$$\mathcal{S} \triangleq \frac{1}{n} \sum_{s \in S} \psi(s) \text{ where} \tag{1}$$

$$\psi(s_i) \triangleq \phi_r \left(\text{Reorder} \left(\left[\phi_c \left(q(c_i^1) \right), \ldots, \phi_c \left(q(c_i^d) \right) \right] \right) \right). \tag{2}$$

Here, ψ is a learned model that is computed by:

(1) using the quantization function q, discussed in §3.3, to replace the value of each column with an identifier,
(2) applying a learned function ϕ_c on each column's identifier,
(3) reordering the results by the number of bits assigned to the corresponding column, and
(4) applying another learned function ϕ_r on the resulting vector to generate a new vector.

The top left part of Figure 6 illustrates these steps. We call ϕ_c and ϕ_r the column and row embedding functions respectively. Recall that quantization converts each column into at most ξ bits and each row to ℓ bits; hence, ϕ_r takes ℓ bits as input and outputs at most η bits which is the size of the summary (Table 2). Reorder achieves a form of schema invariance since regardless of where a column may be in the schema of the relation, the most distinctive columns will appear at the same input position for ϕ_r. Note that the summary (Equation 1) is a sum over row computations and is independent of the order of input rows. Adding or removing rows, and hence updating a row's contents, are local incremental operations; ψ need only apply on the rows that change. Hence, it is easy to see that the above method is parallelizable per row.

Model choice: We use shallow NNs as shown in Table 3 to represent the ϕ_r and ϕ_c functions and learn them end-to-end. ϕ_c is an embedding layer [4] and ϕ_r is a few fully connected layers followed by non-linear activation. By applying only on quantized vectors, we transform arbitrary structured data into a known universe; prior work [52] proves that such functions can succinctly encode sets wherein each element is drawn from a known universe.

Observe that we use the same per-column function on all columns and the same per-row function on rows from any dataset; doing so keeps pre-training costs (#parameters) small and is crucial to

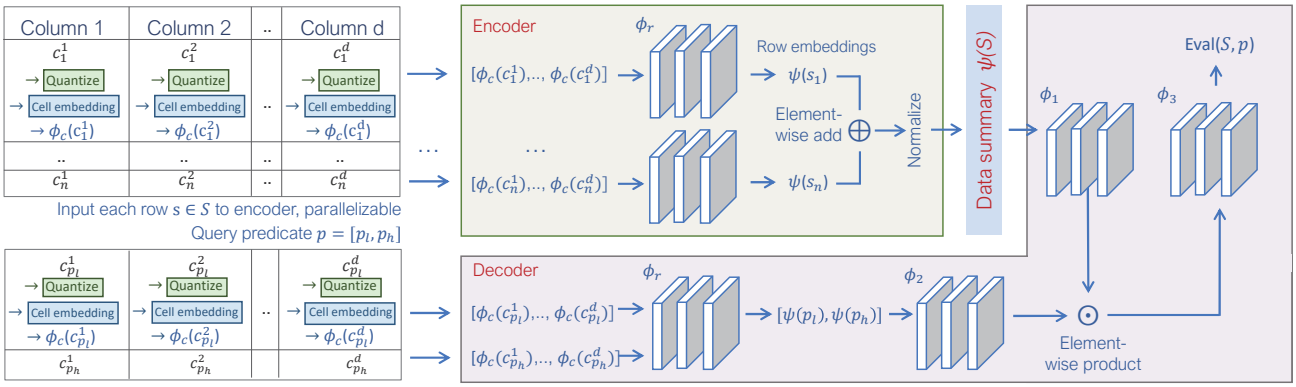

Figure 6: Expanding on Figure 2, the workflow of how we use pre-trained Iris models to summarize datasets (encoder above) and to estimate the cardinality of query predicates (decoder below).

Table 3: Model specifications. We show embedding (E) and fully connected layers (FC) with input → output #neurons.

Layer	ϕ_c	ϕ_r	ϕ_1	ϕ_2	ϕ_3
1	E: $\xi \times 64$	FC: $64d \rightarrow \eta$	FC: $\eta \rightarrow \eta$	FC: $2\eta \rightarrow \eta$	FC: $\eta \rightarrow 64$
2	-	ELU	ELU	ELU	ELU
3	-	dropout:0.3	dropout:0.3	dropout:0.3	dropout:0.3
4	-	-	-	-	FC: $64 \rightarrow 1$

generalize to unseen datasets. Normalization, in Equation 1, simply divides the total over all rows by the number of rows (n).

Parameter choice: How to pick appropriate values of ℓ (the effective quantized size of a columnset) and η (the size of the output of ϕ_r and the size of the summary)? We will show results in §5 for different values of these parameters which represent a trade-off between training complexity, storage overhead and accuracy. Intuitively, the larger these values are, the larger is the summary and the model size (e.g., the number of parameters in the learned function ϕ_r), resulting in a larger training overhead. At the same time, larger values allow more aspects of the underlying frequency and multi-column correlations to be captured, resulting in potentially better summaries but also a higher chance of overfitting. Our findings in §5 show that using small models, Iris can achieve comparable or better accuracy compared to alternate techniques.

Intuition: We believe that the per-column and per-row functions, ϕ_c and ϕ_r respectively, capture intra- and inter-column characteristics. When learning word embeddings, the analogous characteristics are word frequency and co-occurrence probabilities (e.g., how often 'VLDB' and 'SIGMOD' co-occur). As is the case with other learned models [27, 37, 38], today's model explanation techniques do not allow us to precisely understand what the learned functions capture. Focusing on summarizing small quantized columnsets, we believe, is key to training well because there are fewer kinds of frequency and correlation patterns in this narrower case than if were to aim to summarize arbitrary relations that can be of any schema and have any numbers of columns. We believe that using learned models to only summarize columnsets for which other simpler methods such as histograms do not offer good accuracy also helps by further reducing the kinds of patterns that the pre-trained summarization models must capture. Some other works use a similar insight to decompose complex functions into a few wavelets [48], local sparse codes [40] or to apply compressed sensing [13]. Finally, similar to BERT [24] and ResNet [32], we pre-train our summarization models on thousands of columnsets extracted from a diverse corpus and use augmentations to further improve variety. We believe each of these aspects is crucial to achieve good accuracy with pre-trained summarization models.

4.2 Answering queries over Iris summaries

For a two-sided predicate $p = [p_l, p_h]$, i.e., $\bigwedge_i p_l^i \leq c_i \leq p_h^i$, its cardinality estimate using summary $\mathcal{S}$ is computed as:

$$\text{Eval}(\mathcal{S}, p) \triangleq \phi_3\left(\phi_1\left(\mathcal{S}\right) \odot \phi_2\left(\left[\psi(p_l), \psi\left(p_h\right)\right]\right)\right), \tag{3}$$

where $\odot$ denotes an element-wise product. In words, we apply three learned models– ϕ_1, ϕ_2 and ϕ_3– on the dataset summary as well as on $\psi(p)$ as shown in Figure 6 bottom.

Latency: We implement Eval calls using SIMD on server-class CPUs and show in §5.2.4 that the latency with Iris is comparable to that of production estimators.

Intuition: Equation 3 can be thought of as learning a multi-attribute CDF whose value is the fraction of rows that fall in-between two input points. Importantly, here, the input is a summary of the multi-set instead of the actual relation; using only the summary makes the function easier to evaluate. Also, intuitively, using the same summarization function ψ on the the low and high values of the predicate p_l, p_h is appropriate since doing so projects each predicate to the same space as the dataset rows; ϕ_2 further transforms these two predicate embeddings into a single vector of the same dimensionality as the data summary. Our choice of neural structures– specifically, ϕ_1, ϕ_2, ϕ_3 and a dot product to combine with the encoded predicate– resembles structures that work well in question answering and dialog systems [12, 52].

4.3 Incrementally handling dataset changes

Iris handles inserts, deletes and changes to the rows in a relation as follows: if multi-set S with n rows has a learned summary $\mathcal{S}$, adding or deleting a row s_i or changing s_i to a new value s_j leads to the following summaries: $\frac{n\mathcal{S}+\psi(s_i)}{n+1}$, $\frac{n\mathcal{S}-\psi(s_i)}{n-1}$ and $\mathcal{S} + \frac{\psi(s_j)-\psi(s_i)}{n}$ respectively where ψ is the encoder function from Equation 2 and

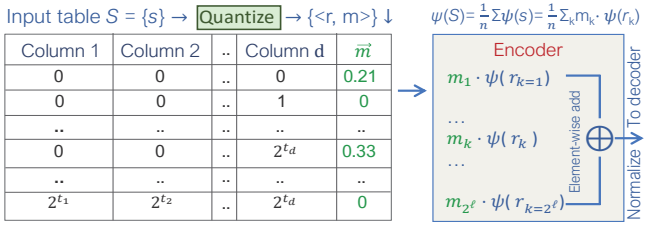

Figure 7: A portion of the training pipeline for Iris models.

Table 4: Datasets to pre-train Iris. R: real, C: categorical.

Name	#cols	#rows	#DV	Name	#cols	#rows	#DV
	R/C	*n*	min/med/max		R/C	*n*	min/med/max
Higgs	28/0	11M	3/7K/290K	KDD99	34/5	5M	2/89/14K
SUSY	19/0	5M	2/1M/2.1M	PRSA	16/2	430K	5/645/35K
Gasmeth	18/0	4.2M	10/15K/18K	Retail	5/3	410K	32/4K/20K
Gastemp	20/0	3.8M	4K/9K/4M	Covtype	10/9	580K	67/456/6K
hepmass	29/0	3.5M	2/742K/1M	Sgemm	10/4	240K	58K/58K/58K
Weather	7/0	3.5M	365/1K/9K	Adult	6/8	30K	16/92/22K
PAMPA2	54/0	2.9M	13/1M/2M	YearPred	90/0	520K	89/.5M/.6M
HTsensor	11/0	930K	50K/.2M/.5M	Power	7/0	2.1M	32/210/4K
WECs	49/0	300K	.2M/.2M/.3M	Census90	0/50	2.4M	2/6/2K

Figure 6. We incrementally update the other summaries that Iris uses in the obvious way – specifically, for the per-column or per-column pair histograms (noted as AVI and Hist respectively), bucket boundaries do not change and bucket frequencies increase or decrease as needed; for Sparse summaries, which record frequencies of the frequent tuples, we update the frequencies and add/drop the least frequent tuple if necessary to stay within storage size. Note that we use the same method in §4.2 to answer queries over updated datasets; Equation 3 applies on the updated summary $\mathcal{S}$.

4.4 Pre-training Iris models

Iris' learned models resemble a conditional auto-encoder [12]; the summarization models are *encoders* which convert large tables into succinct summaries and the models that compute cardinality estimates over these summaries are *conditional decoders* which output the cardinality (fraction of rows that match a predicate, see Equation 3). Iris learns these models end-to-end in the normal way [31]. Specifically, the training corpus consists of a large number of columnsets and corresponding to each a large number of query predicates and their cardinality. In each training epoch, we pick a batch of columnsets and queries and, after applying the encoder and decoder models, compute gradients to update the models. The loss function is $\left|\log \frac{\text{true card.}}{\text{est. card.}}\right|$.Table 3 describes aspects of the functions to be learned; we use fully connected layers followed by exponential linear unit (ELU) activation [19].

Corpus: We pre-train Iris models over the datasets listed in Table 4 and test on the hold-out datasets shown in Table 5. The datasets contain a rich variety of multi-column correlations, column types and distributions of distinct counts and frequency. From each dataset, we randomly pick columnpairs and generate two-sided predicates to pre-train by (1) picking the lower and upper ends of each predicate clause uniformly at random, (2) picking a midpoint and width for each predicate clause uniformly at random, (3) sampling the midpoint from data and uniformly distribute the width or (4) exponentially distribute the width. Note that these datasets and predicates subsume and extend those considered in prior works [27, 34, 39, 45, 51]. Overall, we use 30K distinct columnpairs to pre-train Iris models.

Different inputs for training: Recall from §3.3 and §4.2 that we quantize each table row and the query predicate into a fixed size input because the learned models require fixed size inputs. We require an additional normalization for training because, the *entire dataset* is the training input rather than individual rows. To convert entire datasets into a fixed size, since each quantized row occupies at most ℓ bits, we represent an entire dataset by appending to each distinct quantized row a new column that indicates the normalized frequency with which that row appears in the dataset (see Figure 7). Each dataset hence translates to 2^ℓ floats – one float to encode the frequency of each distinct quantized value. The figure also shows a subtle modification where we multiply the summary generated for a distinct quantized row by the normalized frequency with which it occurs. Thus, the summary becomes a weighted sum of the per-item summaries, i.e., $\psi(S) = \frac{1}{n}\sum_k m_k \cdot \psi(r_k)$, where r_k is a distinct quantized row and m_k is its frequency.

Adding variety: Prior works have shown that variety in the training corpus improves generalization [22, 24, 32]. To add variety while pre-training Iris models, besides using a diverse collection of datasets and query predicates, we use the following augmentations:

- On randomly chosen columns, transform the identifiers by adding a random offset (i.e., id $\leftarrow$ (id + rand)%2^{t_j}), by mirroring (id $\leftarrow 2^{t_j}$ − id), or by multiplying a random factor (id $\leftarrow$ int(id · rand)%2^{t_j}). Recall from §3.3 that column j is allocated t_j bits and hence the quantized ids are integers in $[0, 2^{t_j} - 1]$.
- Mixing the m vectors of different datasets from the same training batch to mimic synthetically generated datasets that are concatenations of portions of different datasets. As a special case, we also use random sub-portions of actual datasets.
- To the normalized frequency column, add or multiply with a uniform random noise and then re-normalize.

Other augmentations can also be used and we will show that these augmentations significantly improve model quality in §5.

5 EVALUATION

We evaluate Iris against state-of-the-art techniques for cardinality estimation (CE) with the following goals.

- Q1 Is it beneficial to use Iris models off-the-shelf to estimate cardinality on tables unseen at pre-training? Does Iris compare favorably to state-of-the-art techniques on the accuracy and latency of CE and the storage and construction costs per dataset?
- Q2 We tease apart the usefulness of various aspects of Iris by comparing with alternates on a large number of relations.
- Q3 We conduct a sensitivity analyses of Iris by sweeping values of relevant parameters.
- Q4 We measure the costs to update Iris summaries when datasets change as well as to construct summaries in parallel.

5.1 Experiment Setup

Datasets. We use the datasets in Table 5 that are unseen during pre-training to test cardinality estimation. Note that the tables have rich schemas – different numbers and types of columns as well

as a variety of frequency and correlation patterns.[8] Recall that our model selection process in §3.2 uses learned models only on some columnpairs; from among all such columnpairs across the datasets in Table 5, we pick 16 columnpairs at random to evaluate some aspects in more detail. In the following, we use *test relation* to denote each full dataset in Table 5 as well as their projections on the randomly picked columnpairs from these datasets.

Predicates. The testing queries used in our experiments have a similar or larger scope than those used in prior work [27, 51]. For each test relation, we generate query predicates as follows: first, for a relation with d columns, we pick d_p predicate columns uniformly at random where $d_p \in [1, d]$ is also picked uniformly at random. Next, for real-valued predicate columns, we pick range predicates whose *mid-point* is the value of a row sampled uniformly-at-random and whose *width* is uniformly randomly sampled between the min and max values of that column.[9] For columns with categorical values, we pick point predicates which equal the value of a row sampled uniformly-at-random from the relation. We pick 1K queries per full dataset and 300 for each of the 16 columnpairs.

Metrics. We measure the following metrics:
Storage used, estimated as follows: for ML-based approaches, we count the number of floats (neurons) in the models. For histograms, we measure the storage of all buckets. Many storage systems compactly store categorical columns as dictionaries [6, 8, 9]; we assume no extra space for such dictionaries among all baselines.

Query latency: We measure the time to obtain a cardinality estimate using one thread on a server-class machine without GPU support.

Accuracy: For each predicate, we measure the q_θ error defined as: if $g, \hat{g}$ are the estimated and actual cardinalities, $q_\theta(g, \hat{g}) = \max(\frac{\max(g,\theta)}{\max(\hat{g},\theta)}, \frac{\max(\hat{g},\theta)}{\max(g,\theta)})$. This is a widely-used metric [27, 37, 51] which measures the relative error after skipping values below θ; we use $\theta = 10$ similar to [27]. Per relation, we report the geometric mean of q_θ over all predicates (GMQ) as well as the 95^{th} percentile. Intuitively, the geometric mean is more robust to outliers; a q_θ value of $1 + x$ indicates that the relative error in cardinality estimates is $100 * x\%$; and, perfect estimates have a value of 1.

Building costs: We measure the latency on a single CPU thread to construct the model or summary for each relation. For methods that learn a new model per dataset, the latency includes obtaining the training examples and training the model. Since the pre-trained models for BERT and ResNet are publicly available, we expect that pre-trained summarization models for datasets will also be freely available, e.g., from our artifact repository; hence a database admin can directly apply these models on their datasets.

Model training metrics: We also report the costs to pre-train the summarization models (e.g., latency, resource usage) and analyze the cost-vs-accuracy trade-off for different training choices and hyper-parameters.

Baselines. We compare Iris against the following baselines:

[8] We support all columns in these tables except the `comment` column in LineItem which is a long string with a high distinctness. Recall that Iris supports both numeric and categorical columns; strings with low distinctness are treated as categorical columns.
[9] *width* can be zero implying a point predicate.

Table 5: Datasets that are held-out or unseen in pre-training.

Table Name	#Cols.		#Rows	#Distinct Values
	Real	Cat.	n	min/median/max
DMV [10]	7	13	11M	2/12/200K
TPCH-LineItem [11]	11	4	6M	2/50/1.5M
Poker [25]	0	11	1M	4/10/13
IMDB-CastInfo [3]	6	1	36M	11/2.3M/26M
Airline-OnTime [2]	66	17	440K	2/130/6620

Sampling: For each test relation and given a storage size, we sample rows uniformly at random up to the given storage size. We report aggregate metrics over 20 trials.

Histograms from `DBMS-x`: We use the built-in CE estimator in a commercial DBMS which relies on per-column histograms. We obtain the selectivity of the predicate on each column and account for correlations using three different heuristics [27]: Attribute Value Independence (xAVI) which assumes independence between columns, Exponential Backoff (xEBO) which combines up to four most-selective predicate columns with diminishing impact [27], and taking just the minimum selectivity across per-column predicates (xMinSel). Each histogram has up to 200 buckets and uses 4KB space per column.

Learned Models: We compare with four different approaches that build models on each relation: (1) *LM* [27] which builds Gradient Boosting Trees that take as input the query predicate, the xAVI, xMinSel and xEBO estimates from DBMS-x and outputs cardinality estimates; we also compare with *LM-* models which are identical to *LM* except for using only the query predicate as the input; (2) *MSCN* [37] which learns a more complex network over the query predicate and existence bits on a sample; (3) *Naru* [51] which builds auto-regressive models to capture marginal probability distributions from the raw data; (4) *DeepDB* [34] which builds Sum-Product Networks (SPNs) to capture the distributions and correlations among different columns. Note that the latter two schemes directly train over features from the actual dataset whereas the first two are mostly workload-driven.

For each above technique, we use code shared by the authors. For techniques that use training queries [27, 37], we offer as many training queries as recommended by the authors of these techniques; training queries are generated using the same process above as the test queries. Furthermore, to obtain models at different storage sizes, we vary model parameters such as the number of neurons, number of layers etc. as recommended by the authors of each technique in personal communication. Specifically, for DeepDB, we binary search the `rdc_threshold` knob; for MSCN, we use a sample that is half of the storage budget and vary the network size; for Naru, we use the default settings of binary input and one-hot output encodings and vary the hidden layer sizes.

Multi-dimensional histograms: We also compare with: (1) *STholes* [16] which adapts the boundaries of multi-d histograms based on query workload. We use the code shared by the authors of [16] and train with 10K predicates generated using the same method as the test predicates. (2) *K-d tree*[14] which partitions multi-dimensional data by alternatively picking cuts on different dimensions so as to minimize the frequency difference between the two cut portions. We build k-d trees of varying depth (based on available storage budget); each tree memorizes the cuts and the frequency at each leaf.

Table 6: Accuracy of CE at a storage budget of 1×, i.e., 4KB per column in each relation.

Dataset	Space	Accuracy (= GMQ / 95th percentile) of cardinality estimates computed using different techniques									
	KB	Sampling	xAVI	xMinSel	xEBO	Iris	LM-	LM[27]	MSCN[37]	Naru[51]	DeepDB[34]
DMV	80	3.52/150.91	1.61/27.82	2.55/213.70	1.75/28.53	1.60/22.58	1.78/12.75	X	10.79/208.0	X	X
TPCH-Lineitem	60	2.33/19.54	2.13/10.44	145.08/351K	51.42/6.4K	1.67/4.63	2.41/9.01	X	36.13/3.4K	X	X
Poker	44	1.57/20.78	1.03/1.20	518.26/7.7K	97.07/943	1.03/1.19	1.85/6.95	X	25.28/32K	6.4K/401K	1.14/2.0
IMDB-CastInfo	28	1.85/81.39	2.38/25.51	3.39/37.46	2.12/15.81	2.49/43.19	2.43/21.93	X	3.6K/26M	X	X
Airline-OnTime	332	1.28/5.50	1.30/7.22	28.54/940.07	2.84/72.80	1.33/7.51	1.55/7.59	X	3.59/407	X	X

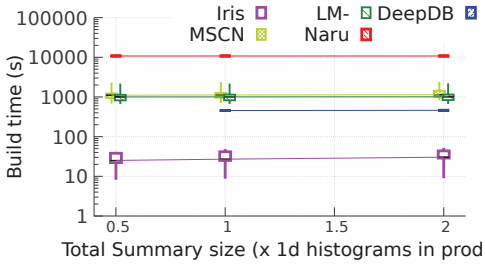

Figure 8: Time to build summaries of different sizes on the datasets from Table 5 (single thread; no GPU). Table 9 and Table 10 show the one-time costs for the pre-training.

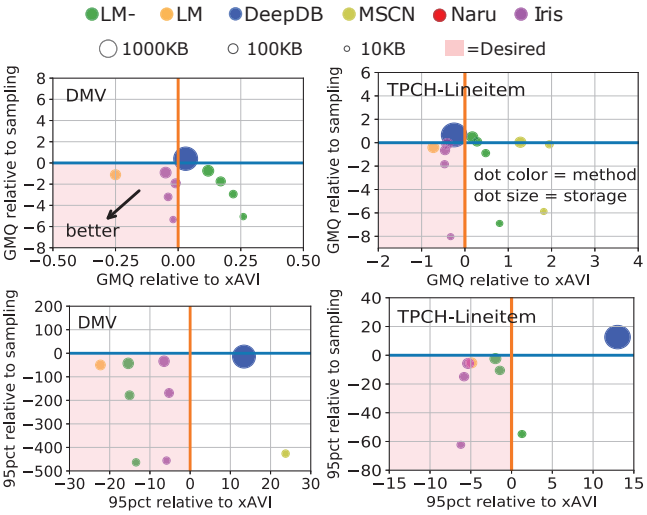

Figure 9: The relative change in accuracy (GMQ and 95 percentile) for different CE techniques over two simple production baselines (Sampling and xAVI); we show summary sizes that are $\frac{1}{4}$×–2× of the storage used in production systems.

Implementation details: We train Iris models using Keras and TensorFlow on a server with 4x Nvidia Tesla V100 GPUs, each with 16GB memory. We use 64 epochs in the training with Adam optimizer; each has 1280 steps with batch size 2K. Learning rate is $1e^{-3}$. In addition to the summaries from §3.2, Iris also maintains per dataset: a small sub-sample (e.g., of the CORDS input), small per-column histograms (e.g., of the bucketization input [7]) and bucket boundaries (see Table 8). Using these we compute an upper bound and a lower estimate of cardinality and clip the result of Equation 3. Since GPUs may not be available in a typical data platform, we use CPU-only code to construct summaries and answer cardinality estimation queries. The ϕ_r and ϕ_c functions used to compute summaries are a dictionary lookup over the row and column value respectively; the dictionary has $\ell \cdot \eta$ floats and is in-memory. The decoder, which answers CE queries, is in C++ and uses SIMD.

5.2 Efficacy of CE on entire datasets

5.2.1 Time to build summaries. Figure 8 shows that Iris is faster, by one to two orders of magnitude, relative to prior works that build models on each dataset. To normalize across datasets that have different sizes, we measure the space used by summaries as a multiple of the space that is used by production systems today. For example, 1× storage equals the size of the per-column histograms maintained by production systems (~ 4KB histograms per column). The candlesticks show the average latency to build summaries for each of the datasets in Table 5 on a single CPU thread. Iris is faster because pre-trained summarizing models apply on each dataset without additional training. We note a few details. First, we defer discussing how the latency breaks down, the effects of parallelizing various parts and handling data updates to §5.2.3 and §5.4. Next, Naru [51] and DeepDB [34] have fewer data points because they successfully build models only on a few of the datasets at the given storage sizes; the latency to obtain cardinality estimates increases with summary size and our goal is to not adversely increase the overall query optimization latency. Finally, Figure 8 shows the build time for the LM- variant described in §5.1; this is an underestimate of the build time for the actual model LM [27] which uses cardinality estimates from DBMS-x. We also ignore the time to find hyperparameters to ensure that the LM, MSCN, Naru and DeepDB models fit within the target summary size for each dataset. To sum, using pre-trained summarization models significantly reduces the computation needed before cardinality estimates are available.

5.2.2 Accuracy of cardinality estimates: Table 6 shows the geometric mean of q_θ error (GMQ) and the 95th percentile achieved on the datasets from Table 5 when the total summary size is held equal to the size of the 1d histograms that are maintained by production systems today. For many different summary sizes, Figure 9 depicts the accuracy of different techniques; the corresponding detailed results are in Table 6 and [7]. We find that techniques that learn dataset specific models – LM, MSCN, Naru and DeepDB – are either unable to work within the available space (marked as X in Table 6) or have worse accuracy than sampling and xAVI. We discuss the reasons in more detail in the next paragraph. In general, when the summary size is large, the error caused by Sampling converges quickly [43] and at 2× storage Sampling provides the best or close-to-best accuracy. Overall, these results show that Iris has comparable or better accuracy in many configurations and, can be used alongside or as a

Table 7: Breaking down the Iris build time on different datasets and storage budgets (in seconds, on one CPU).

Dataset	DMV		Airline	
Storage Budget	1×(80KB)	2×(160KB)	1×(332KB)	2×(664KB)
CORDS	12.2	12.2	30.6	30.6
Iris - Quantization	6.1	7.1	9.6	10.3
Iris - Summarization	4.3	4.7	3.7	4.3
Other models	2.1	2.4	4.0	4.4
Total (s)	24.7	26.4	47.9	49.6

replacement of alternative techniques especially at small summary sizes which translate into quick cardinality estimates.

Now, we further discuss the results from Table 6.

- Poker's columns are mostly independent; thus, per-column histograms (e.g., with xAVI) achieve good accuracy. Iris's model selection discerns independence and Iris only builds per-column histograms and sparse representations for Poker (see Table 8). Note that Iris's per-column histograms are smaller and simpler (only have frequency per bucket whereas xAVI also has distinct counts and uses proprietary heuristics [6, 8, 9]).
- LM cannot build models at 1× summary size because the per-column histograms it uses to compute input features (xAVI, xMinSel and xEBO estimates from DBMS-x) themselves take 1× storage. As the table shows, Iris outperforms LM- in three of the five datasets including Poker where LM- is unable to discern independence. At a 2× summary size (see [7]), LM improves over LM- and achieves the best accuracy on Lineitem and DMV but does not lead on other datasets.
- Naru and DeepDB are unable to build models on the examined datasets at many summary sizes because the column embedding and sum-product networks that they build respectively require much larger storage [34, 51]. For example, a Naru model that handles 13 of the 20 columns from the DMV dataset is 13MB, 162× the size of the summaries used in production systems.
- MSCN maintains samples per dataset which use up half of the summary space and perhaps as a result the trained models have worse accuracy than the relatively simpler LM- models.
- The CastInfo dataset requires special attention because both LM- and Sampling outperform Iris. We find that this dataset has stronger correlation among he columns which Iris fails to capture well since we only summarize column pairs. To verify this issue, we built five additional summaries on column triples with $\kappa = 3$ using the same pre-trained models – specifically, on every triple where all three column pairs had a high correlation score from CORDS. The resultant bag of summaries has better accuracy on CastInfo: the GMQ (and 95th percentile) change from 2.49 (43.19) in Table 6 to 2.21 (29.1); the five additional 3-d summaries fit within the 1× storage budget, due to leftover storage after the initial model selection (Table 8). Our findings are that (a) pairwise summaries still offer a sizable improvement because they capture the marginal correlations and (b) building learned summaries on larger columnsets (e.g., column triples) improves accuracy because they capture the correlation better and avoid the imprecise combination of estimates from multiple summaries (line#5 in Algorithm 2). Naively extending to this case however increases pre-training costs; we leave careful choice of summarizing larger columnsets to future work.

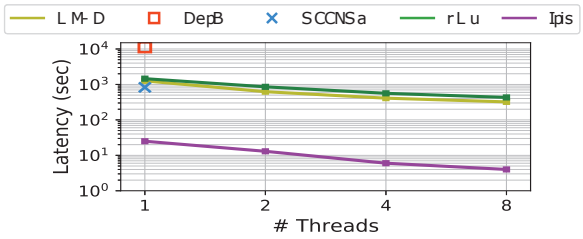

Figure 10: Latency to build summaries in parallel (DMV, 1×).

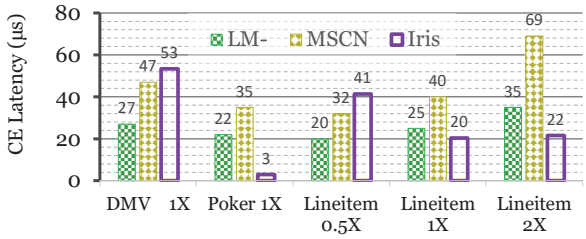

Figure 11: Average latency in μs per card. estimate for a few example datasets and summary sizes.

Slicing results by predicate cardinality: We see that, in general, more selective predicates have lower accuracy to the estimates (results in [7] and analyses in [43]). The variance of sample-based estimates increases with predicate selectivity and histogram-based techniques are more inaccurate for predicates which fall within one histogram bucket. Iris is competitive across the cardinality range.

5.2.3 Breaking down buildtime and parallelization: Table 7 breaks down the total time that Iris incurs to build summaries. As the table shows, running CORDS to assess correlation between column pairs takes the longest amount of time. The time to quantize and summarize increases slightly with storage budget since Iris can build more summaries at larger storage budgets (see Table 8). These steps can all be parallelized – CORDS can run in parallel per column pair and different threads can summarize different data partitions. Figure 10 shows how the latency to build summaries changes with multiple threads. For prior works, we find that training corpus collection is easier to parallelize than model training; in particular, the Naru, DeepDB codebase intertwines multiple aspects in a complex way and are nontrivial to parallelize. As the figure shows, Iris achieves a nearly linear reduction in latency (from ~ 25s with one thread to ~ 4s with eight threads).

5.2.4 Cardinality estimation latency: Figure 11 shows the average latency to obtain a cardinality estimate from various models on a single CPU thread; we use SIMD support where possible. Prior work shows that cardinality estimators in production DBMS have latency between 0.1ms to 1ms [27] depending on the complexity of the predicate. As the figure shows, Iris and some alternatives offer comparable latency. LM [27] is not shown in the figure but its latency can be thought of as the latency of LM- in the figure plus the latency to obtain additional cardinality estimates from DBMS-x which are used as inputs by the LM models. Naru, DeepDB also does not appear in Figure 11 because we could not build these models for many of the datasets and summary sizes shown here for the

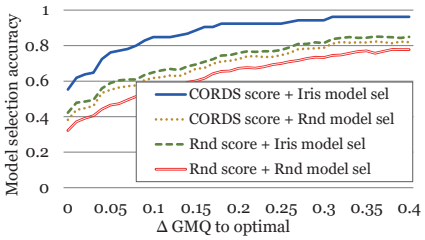

Figure 12: On Lineitem with 1× storage, for different model selection schemes, we show their accuracy to pick summaries that are within ΔGMQ (x-axis) to optimal.

Table 8: The number of column pairs which receive a particular summary type from Iris's model selection for different datasets and storage budgets. *including overhead per column: 0.5KB 1-d histogram, 0.125KB bucket boundaries and 0.5KB small samples (§5.1).

Dataset	Storage budget*	Max size per summ.	Number of summ.		
			Iris	Sparse	Hist.
DMV	80KB (1×)	2KB	15	53	0
TPCH-Lineitem	30KB (0.5×)	1KB	6	22	6
TPCH-Lineitem	60KB (1×)	2KB	6	24	17
TPCH-Lineitem	120KB (2×)	4KB	3	28	10
Poker	44KB (1×)	2KB	0	55	0
IMDB-CastInfo	28KB (1×)	2KB	1	0	5
Airline-OnTime	332KB (1×)	2KB	0	314	0

reasons that were discussed in §5.2.2; at larger summary sizes and on other datasets, we find these models have latency in the range of a few milliseconds as reported elsewhere [34, 51]. Such latencies will noticeably delay query optimization time. Figure 11 also shows that the CE latency can depend on the number of Iris summaries used - each decoder inference takes 32μs with SIMD which is 1-2 orders of magnitude slower than evaluating Hist or Sparse; since we use larger summary sizes at a larger storage budget (Table 8), the choices of model selection can be nuanced. In some cases (e.g., on Poker), Iris offers much faster cardinality estimates by evaluating only the simpler summaries (Hist or Sparse).

5.2.5 Memory footprint: The memory footprint to build Iris summaries and evaluate cardinality estimates is smaller than that of other learned models. Iris's summaries build in one pass over the data and the memory consumption is just the row sample used by CORDS (§3.1) and the dictionary for ϕ_r and ϕ_c (§4.1, §5.1) which is about a few hundred KBs. In comparison, prior learned models must store the entire training corpus in memory and use more memory to train models. Naru, DeepDB also keeps a large sample of the dataset in memory to estimate the ground-truth for training predicates. In all, we observe a footprint of a few hundred MBs for the prior works; that is, Iris uses roughly 1000× less memory to build summaries. To provide cardinality estimates, the memory footprint needed is similar across all techniques (about tens of KBs per dataset), since we explicitly equalize the summary space used by the various techniques.

5.3 Understanding the components of Iris

5.3.1 Model Selection. Recall from §3.2 that Iris uses different techniques to summarize different columnsets. Here, we evaluate the efficacy and outcomes of the selection process. Figure 12 demonstrates model selection on Lineitem with 1× storage. Specifically, we show the chance of picking summaries that are within a given delta to optimal; we vary ΔGMQ on the x-axis which corresponds to the GMQ of the picked summary minus that of the optimal choice. Compared to using random correlation scores and/or picking summarization techniques randomly, Iris appears better at model selection. For example, with ΔGMQ=0 (i.e., matching optimal), Iris is correct in 57% of the cases while randomly scoring and picking is only correct in 32% of the cases. When Iris does not match optimal, the difference is often small – for 82% of the choices, ΔGMQ is below 0.1. Overall, learned summaries are best suited for 19% of all the column pairs while 39% of the columns pairs can use AVI without a significant accuracy degradation. To summarize, Table 8 shows the numbers and types of summaries chosen by Iris across datasets and storage budgets.

Table 9: The error incurred by Iris (GMQ) and by quantization (qGMQ); we also show the sizes and (pre-)training time.

Param. ℓ/η	Size (B)	Accuracy		Training (hours)
		GMQ / 95pct	qGMQ / 95pct	
6/-		-	3.14 / 2619	-
7/-		-	2.58 / 1049	-
10/32	Summary	1.52 / 22.45	1.82 / 109.06	17
10/64	$\propto \eta$;	1.62 / 63.38	1.82 / 109.06	12
10/128	Quant.	1.67 / 92.46	1.82 / 109.06	14
11/128	$\propto 2^\ell$.	1.46 / 11.97	1.69 / 86.51	20

Table 10: The training time and CE accuracy when pre-training different sizes and variety of the training corpus.

Number of raw Training relations	Augmented?	Train. time	ΔGMQ	Δ95pct
3%	✓	20 hrs	+0.16	+22.62
33%	✓	20 hrs	+0.07	+8.07
33%	×	6 hrs	+0.52	+404.62
100%	✓	20 hrs	[1.46]	[11.97]

5.3.2 Ablation study: quantization vs. summarization. We decouple the quantization error (§3.3) in cardinality estimation from using an Iris model. To estimate such error, we answer cardinality estimation queries by using a quantized version of the relation which contains the frequency count for each distinct quantized row; an example of this representation is the table on the left in Figure 7. Using the quantized version, we estimate the cardinality of a two-sided query by summing up the frequency of all tuples that fall within the (quantized) predicates instead of using Eqn. 3. We call this value the qGMQ. Table 9 shows the qGMQ on 16 sub-relations of the test datasets; we see that using more bits to quantize (i.e., larger ℓ) reduces the quantization error but the marginal reduction decreases. We also note that the final GMQ is smaller than qGMQ due to the result clipping by small samples and 1-d histograms (see implementation details in §5.1). The quantized version of a relation is larger in size than the summaries and the learned summarization models can be thought of as compressing 2^ℓ floats to η floats; $\eta \ll 2^\ell$. Smaller summaries are efficient to use and store but may add error in cardinality estimates.

5.3.3 Effects of training with different corpuses. Table 10 shows the training time and accuracy when pre-training summarization models in different ways. As the table shows, training with fewer relations and without the augmentations (§4.4) that add variety

Table 11: Comparing the accuracy when summarizing correlated column pairs with different techniques; see §5.3.4.

Method	GMQ/95pct	
	0.25KB	0.5KB
Iris	1.62 / 63.38	1.46 / 11.97
Sampling	1.93 / 391.4	1.63 / 36.02
ST-Holes	4.33/ 466.8	7.93 / 620.5
Kd-trees	8.77/ 1022	11.10 / 2017

Table 12: Accuracy of cardinality estimates when using different methods to handle updates to the Lineitem table (1× summary size); see §5.4.

Method	Baseline	ΔGMQ / Δ95pct to baseline		
	No change	10% change	15% change	20% change
Ground truth	1.0 / 1.0	+0.13 / + 0.49	+0.19 / +0.78	+0.25 / +1.05
LM- do nothing	2.41 / 9.01	+0.11 / +0.40	+0.16 / +1.08	+0.21 / +2.86
Iris do nothing	1.67 / 4.37	+0.04 / +0.65	+0.07 / +0.69	+0.09 / +0.92
Iris update	1.67 / 4.37	+0.02 / +0.03	+0.04 / +0.15	+0.06 / +0.47
Iris rebuild	1.67 / 4.37	+0.01 / -0.02	+0.03 / +0.03	+0.04 / +0.36

lowers the accuracy of cardinality estimates in GMQ and especially in tail errors. The accuracy values shown are the average over 16 sub-relations from the datasets in Table 5 and are relative to the model trained using the values on the last row. The model training time is similar when using more data because each training epoch is a fixed-size batch sampled from the corpus. The models in Table 10 use hyperparameter values $\ell = 11$ and $\eta = 128$; varying the training corpus has a similar effect on other hyperparameter values.

5.3.4 Micro-benchmarks. For 16 columnpairs picked randomly from the test datasets, as described in §5.1, we compare Iris summaries with sampling, STholes and K-d trees. Table 11 shows that, relative to Iris, these alternatives offer worse accuracy at the same storage budget. Thus, we believe Iris's learned summaries are a useful addition to our model selection process; they offer better accuracy at small storage sizes.

5.4 How does Iris handle updates to datasets?

Table 12 shows the accuracy of cardinality estimates when different fractions of the TPC-H Lineitem table are updated using rows from a skewed generator (i.e., [1] with zipf=2). The table shows results for three variants: *do nothing* uses unchanged summaries, *update* uses summaries incrementally updated as noted in §4.3 and *rebuild* constructs summaries afresh on the updated relations. As the figure shows, incrementally updating Iris summaries leads to substantially better accuracy compared to *do nothing*; the *rebuild* variant is slightly better because rebuilding from scratch can revisit model selection and use more appropriate quantization (e.g., pick bucket boundaries based on the updated relation).

In contrast, other learned models [27, 39, 45, 51] cannot update incrementally and their accuracy (GMQ) degrades more severely when more updates occur; for example, the gaps of GMQ and especially tail error for LM- are more than doubled compared with those for Iris when 20% of rows change.

On the other hand, retraining LM- on the updated relations is as expensive as shown in §8 and Figure 8. While rebuilding Iris is already efficient, incrementally updating Iris's models is even faster at roughly 1.5ns per appended row and 3ns per updated row on one thread of a 3.5GHz desktop CPU.

5.5 Discussion and future work

Customizing pre-trained Iris models: While the pre-trained summarization models are used off-the-shelf in our experiments, fine-tuning the models on individual columnsets or datasets [31] may further improve the CE accuracy. However, unlike the prior case, the compute and storage costs for specification will not amortize across different datasets.

Training efficiency and knowledge transfer. Multi-task training has been shown to be effective for BERT models [24] to train representations end-to-end based on usefulness for different tasks. Similar ideas can apply in the case of pre-training summarization models; pre-training an encoder along side multiple decoders may improve the accuracy of every task and also allow the same summaries to be used for multiple applications [24]. Extending to such scenarios can be interesting future work.

6 RELATED WORK

Learned models for cardinality estimation: We already compared with works that learn a new model per dataset [27, 37, 39, 51]. Iris uses summaries which can be considered as learning from data [39, 51] whereas others only learn transformations from predicates to cardinality [27, 37]. Iris offers several advantages (1) no per-dataset training which reduces the time to build cardinality estimators, (2) building multiple summaries for different columnsets of a dataset instead of a single model over all columns which improves accuracy for the same storage space, (3) support for both categorical and real columns, (4) using SIMD to summarize data and estimate cardinality which achieves low latency without GPUs and, (5) summaries that incrementally respond to data changes and can be built in parallel per row.

Data structures for cardinality estimation: Production 1D histograms [36] record frequency and distinct counts per bucket and use proprietary algorithms to offer more accurate CE. A few works build multiple multi-d histograms to support wide tables [23, 30]; the estimates from various histograms are combined using learned graphical models. ST-Holes [16] which picks bucket boundaries based on queries and refines boundaries progressively. Other works use wavelets [29, 46], graphical models [17, 49] and kernel density estimators [33] for cardinality estimates. The key difference of Iris is that Iris uses pre-trained summarization models and avoids per-dataset training.

7 CONCLUSIONS

We present Iris, a cardinality estimator for multi-dimensional structured data, which decomposes wide tables into correlated column subsets and summarizes different columnsets using different techniques including novel pre-trained summarization models. Uniquely, Iris requires no per-dataset training. Our experiments show that Iris can match the storage size and latency of production cardinality estimators while offering similar or better accuracy, incremental updatability and faster summary construction time relative to methods that train a new model for each dataset. We believe that pre-trained summarization models of structured datasets are broadly applicable beyond cardinality estimation.

REFERENCES

[1] [n.d.]. A new parallel data-generator for zipf-skewed TPC-H. https://bit.ly/36cl8fc.
[2] [n.d.]. Airline dataset. https://relational.fit.cvut.cz/dataset/Airline.
[3] [n.d.]. IMDB dataset. https://https://homepages.cwi.nl/~boncz/job/imdb.tgz.
[4] [n.d.]. Keras Deep Learning Library. http://keras.io.
[5] [n.d.]. The New and Improved Cardinality Estimator in SQL Server 2014. https://bit.ly/2Wku8JV.
[6] [n.d.]. Oracle Histograms. https://bit.ly/37lkFsc.
[7] [n.d.]. Pre-training Summarization Models of Structured Datasets for Cardinality Estimation: Extended Version. http://yao.lu/iris/extended.pdf.
[8] [n.d.]. Snowflake Histograms. https://bit.ly/2Lv1dkf.
[9] [n.d.]. SQL Server Histograms. https://bit.ly/2WpU1YN.
[10] [n.d.]. State of New York Vehicle, snowmobile, and boat registrations. https://catalog.data.gov/dataset/vehicle-snowmobile-and-boat-registrations.
[11] [n.d.]. TPC-H Benchmark. http://www.tpc.org/tpch/.
[12] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. 2015. Vqa: Visual question answering. In *ICCV*.
[13] Richard G Baraniuk. 2007. Compressive sensing [lecture notes]. *IEEE signal processing magazine* 24, 4 (2007), 118–121.
[14] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975), 509–517.
[15] Kevin S. Beyer, Peter J. Haas, Berthold Reinwald, Yannis Sismanis, and Rainer Gemulla. 2007. On synopses for distinct-value estimation under multiset operations. In *SIGMOD*.
[16] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. 2001. STHoles: A Multidimensional Workload-aware Histogram *(SIGMOD)*.
[17] Kaushik Chakrabarti, Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. 2000. Approximate Query Processing Using Wavelets *(VLDB)*.
[18] Moses Charikar, Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. 2000. Towards estimation error guarantees for distinct values. In *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 268–279.
[19] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015).
[20] Graham Cormode, Minos Garofalakis, Peter J. Haas, and Chris Jermaine. 2012. Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. *Found. Trends databases* 4 (Jan. 2012).
[21] Graham Cormode and S. Muthukrishnan. 2005. An Improved Data Stream Summary: The Count-min Sketch and Its Applications. *J. Algorithms* (2005).
[22] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.
[23] 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.
[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[25] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml.
[26] Marianne Durand and Philippe Flajolet. 2003. Loglog Counting of Large Cardinalities. In *ESA*.
[27] Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek Narasayya, and Surajit Chaudhuri. 2019. Selectivity Estimation for Range Predicates Using Lightweight Models. *Proc. VLDB Endow.* 12, 9 (2019).
[28] Philippe Flajolet and G. Nigel Martin. 1985. Probabilistic Counting Algorithms for Data Base Applications. *J. Comput. Syst. Sci.* (1985).
[29] Minos Garofalakis and Phillip B. Gibbons. 2002. Wavelet Synopses with Error Guarantees *(SIGMOD)*.
[30] Lise Getoor, Benjamin Taskar, and Daphne Koller. 2001. Selectivity estimation using probabilistic models. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*. 461–472.
[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.
[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[33] Max Heimel, Martin Kiefer, and Volker Markl. 2015. Self-Tuning, GPU-Accelerated Kernel Density Models for Multidimensional Selectivity Estimation *(SIGMOD)*.
[34] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2019. DeepDB: Learn from Data, not from Queries! *arXiv preprint arXiv:1909.00607* (2019).
[35] Ihab F Ilyas, Volker Markl, Peter Haas, Paul Brown, and Ashraf Aboulnaga. 2004. CORDS: automatic discovery of correlations and soft functional dependencies. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. 647–658.
[36] Yannis Ioannidis. 2003. The History of Histograms (Abridged) *(VLDB)*.
[37] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. 2018. Learned cardinalities: Estimating correlated joins with deep learning. *CIDR* (2018).
[38] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*. 489–504.
[39] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph Hellerstein, and Ion Stoica. 2018. Learning to optimize join queries with deep reinforcement learning. *arXiv preprint arXiv:1808.03196* (2018).
[40] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y Ng. 2007. Efficient sparse coding algorithms. In *Advances in neural information processing systems*. 801–808.
[41] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really?. In *VLDB*.
[42] Beibin Li, Yao Lu, Chi Wang, and Srikanth Kandula. 2021. Cardinality Estimation: Is Machine Learning a Silver Bullet?. In *3rd International Workshop on Applied AI for Database Systems and Applications (AIDB)*.
[43] Beibin Li, Yao Lu, Chi Wang, and Srikanth Kandula. 2021. Q-error bounds of random uniform sampling for cardinality estimation. *arXiv preprint arXiv:2108.02715* (2021).
[44] Gurmeet Singh Manku and Rajeev Motwani. 2002. Approximate Frequency Counts over Data Streams. In *VLDB*.
[45] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A learned query optimizer. *arXiv preprint arXiv:1904.03711* (2019).
[46] Yossi Matias, Jeffrey Scott Vitter, and Min Wang. 1998. Wavelet-based Histograms for Selectivity Estimation *(SIGMOD)*.
[47] Kexin Rong, Yao Lu, Peter Bailis, Srikanth Kandula, and Philip Levis. 2020. Approximate partition selection for big-data workloads using summary statistics. *arXiv preprint arXiv:2008.10569* (2020).
[48] Christopher Torrence and Gilbert P Compo. 1998. A practical guide to wavelet analysis. *Bulletin of the American Meteorological society* 79, 1 (1998), 61–78.
[49] Kostas Tzoumas, Amol Deshpande, and Christian S Jensen. 2011. Lightweight graphical models for selectivity estimation without independence assumptions. *Proceedings of the VLDB Endowment* 4, 11 (2011), 852–863.
[50] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2020. NeuroCard: one cardinality estimator for all tables. *arXiv preprint arXiv:2006.08109* (2020).
[51] 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. *Proceedings of the VLDB Endowment* 13, 3 (2019), 279–292.
[52] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. 2017. Deep sets. In *Advances in neural information processing systems*. 3391–3401.