Approximate Sketches

BRIAN TSAN, University of California, Merced, USA
ASOKE DATTA, University of California, Merced, USA
YESDAULET IZENOV, University of California, Merced, USA
FLORIN RUSU, University of California, Merced, USA

Sketches are single-pass small-space data summaries that can quickly estimate the cardinality of join queries. However, sketches are not directly applicable to join queries with dynamic filter conditions — where arbitrary selection predicate(s) are applied — since a sketch is limited to a fixed selection. While multiple sketches for various selections can be used in combination, they each incur individual storage and maintenance costs. Alternatively, exact sketches can be built during runtime for every selection. To make this process scale, a high-degree of parallelism — available in hardware accelerators such as GPUs — is required. Therefore, sketch usage for cardinality estimation in query optimization is limited. Following recent work that applies transformers to cardinality estimation, we design a novel learning-based method to approximate the sketch of any arbitrary selection, enabling sketches for join queries with filter conditions. We train a transformer on each table to estimate the sketch of any subset of the table, i.e., any arbitrary selection. Transformers achieve this by learning the joint distribution amongst table attributes, which is equivalent to a multidimensional sketch. Subsequently, transformers can approximate any sketch, enabling sketches for join cardinality estimation. In turn, estimating joins via approximate sketches allows tables to be modeled individually and thus scales linearly with the number of tables. We evaluate the accuracy and efficacy of approximate sketches on queries with selection predicates consisting of conjunctions of point and range conditions. Approximate sketches achieve similar accuracy to exact sketches with at least one order of magnitude less overhead.

CCS Concepts: • **Information systems → Query optimization**.

Additional Key Words and Phrases: cardinality estimation, database sketch, synopsis, neural networks

ACM Reference Format:
Brian Tsan, Asoke Datta, Yesdaulet Izenov, and Florin Rusu. 2024. Approximate Sketches. *Proc. ACM Manag. Data* 2, 1 (SIGMOD), Article 66 (February 2024), 24 pages. https://doi.org/10.1145/3639321

1 INTRODUCTION

Cardinality estimation is fundamental to cost-based query optimization, enabling fast query execution in databases. However, query optimization is a time-sensitive process and accurate estimation cannot be afforded at the cost of low query latency. In the popular open-source PostgreSQL database management system [25], cardinality estimation defaults to univariate histograms and tracking the most common values per column. Although simple, these versatile statistics are generally applicable to many queries. However, for join queries, cardinality estimation methods notoriously assume that data are independent and uniform, which is often unrealistic. Sketches are lightweight probabilistic data structures that can quickly estimate the cardinality of joins without such strong assumptions. However, their rigid requirements make them less general than histograms.

Authors' addresses: Brian Tsan, University of California, Merced, 5200 Lake Rd, Merced, CA, USA, 95343, btsan@ucmerced.edu; Asoke Datta, University of California, Merced, 5200 Lake Rd, Merced, CA, USA, 95343, adatta2@ucmerced.edu; Yesdaulet Izenov, University of California, Merced, 5200 Lake Rd, Merced, CA, USA, 95343, yizenov@ucmerced.edu; Florin Rusu, University of California, Merced, 5200 Lake Rd, Merced, CA, USA, 95343, frusu@ucmerced.edu.

ACM 2836-6573/2024/2-ART66
https://doi.org/10.1145/3639321

Count-Min [6] and Fast-AGMS [5] are hash-based sketches that use hash functions to efficiently summarize the distribution of their underlying data—a selection. This selection is defined prior to creating the sketch and cannot be changed afterwards, limiting the queries that each sketch is applicable to. Despite this, sketches appeal as efficient to create and maintain, needing just a single pass over the data. This lends towards their suitability for streaming data, where data might only be observable once. Additionally, sketches are quick to infer and produce estimates. For example, the cardinality estimate for the join of two selections can be inferred by a dot product between a pair of vectors, their sketches.

Yang et al. [32] applied the ubiquitous transformer [28] to model the distribution of relational data, thus training an extremely accurate cardinality estimator. This method, titled "NeuroCard: One Cardinality Estimator for All Tables", requires prior knowledge of the join relationships between all tables in the database. This allows the transformer to be trained on the full outer join of all tables, thus modeling the joint distribution of all tables and columns. As discussed in section 6, a number of research papers follow this direction — modeling the outer join between tables, rather than the individual tables themselves. Though provably effective, it remains to be seen how such methods can scale to large databases.

Our motivation is twofold. Sketches are efficient but are not directly applicable to queries outside of their specific selections. Meanwhile, transformers can accurately estimate the cardinality of any selection, but current methods require prior knowledge of the join schema for a database, as well as precomputing its full outer join. Our work combines both, mixing the strengths and eliminating the weaknesses of each individual method. By using transformers to individually model each table, rather than the full outer join, it is possible to approximate the sketch of any arbitrary selection in a table. Sketches, in turn, allow transformers to perform join cardinality estimation without training on the full outer join.

By individually modeling tables, our approximate sketching method scales linearly with each table in the database. In experiments, we use two datasets derived from the Join Order Benchmark [20] to evaluate the cardinality estimation accuracy of our approximate sketches compared to actual sketches and NeuroCard [32]. We also evaluate their efficacy when integrated in PostgreSQL's query optimizer, to analyze the difference between modeling tables individually or together as their full outer join. We observe only slight differences in performance between approximate and actual sketches. Source code is available at [27].

Our main technical contributions are as follows:

- An approximation method for Count-Min sketches of arbitrary filter conditions, including point and range predicates. This approximation replaces upstream sketch creation and can be applied to other downstream sketch applications, including two-way and multi-way joins.
- A method for converting Count-Min sketches to Fast-AGMS sketches, which is applicable to both approximate and actual sketches. Classically, Count-Min sketch estimates are upper bounds, but converting to Fast-AGMS allows unbiased estimation.
- A novel upper-bounded variant of our approximation formula that is computationally more tractable. In this work, we propose 4 variations of approximate sketches: Count-Min or Fast-AGMS, and upper-bounded or exact.
- A comprehensive evaluation on JOB-light and JOB-light-ranges showing that our approximations are on par with actual sketches. To the best of our knowledge, this is the first evaluation of Count-Min sketches on these workloads, particularly as unbiased estimators.

2 PROBLEM DEFINITION

In databases, cardinality refers to the number of records returned by a query. Our goal is to use sketches to estimate the cardinality of any query, specifically for join queries with arbitrary point and range selection predicate(s). This setting is identical to the one in [29]. Join queries are notoriously difficult to estimate the cardinality of and query optimizers may simplify it as the Cartesian product between selections:

$$|\sigma(A) \bowtie \sigma(B)| \leq |\sigma(A)| \times |\sigma(B)|$$

The resulting product greatly overestimates the cardinality of the join between selections $\sigma(A)$ and $\sigma(B)$. Sketches may more accurately estimate it as the inner product of two vectors projected from the selections using a hash function, $h(\sigma(A)) = a$ and $h(\sigma(B)) = b$:

$$|\sigma(A) \bowtie \sigma(B)| \approx a \cdot b$$

However, these sketches may not be readily available if the selections are not specified beforehand. Additionally, sketches are inapplicable to other selections, so an application may have to keep multiple sketches—one for every selection. The cost of maintaining these sketches can quickly add up, especially for multidimensional data — there can be an intractable number of selections to sketch.

This brings us to our problem: obtaining sketches a and b for their respective selections $\sigma(A)$ and $\sigma(B)$, which can be any arbitrary conjunction of point and range predicates. As mentioned, the naive solution of maintaining multiple sketches is undesirable. Moreover, such a solution [29] was only proposed for primitive AGMS sketches [1, 2], which are not based on hashing. The only alternative is sketching the selection at query execution time [16], which trades the cost of maintaining sketches that might not ever be used in favor of the on-demand cost of exactly sketching only relevant selections. We go further and avoid the sketching process by approximately generating the sketch as the output of a transformer model.

We find that any Count-Min sketch can be expressed in terms of probabilities that can be estimated from a transformer trained on a superset of the sketch selection, i.e., the entire table. More exactly, the Count-Min sketch a of the selection $\sigma(A)$ can be expressed as the following:

$$a = p|\sigma(A)|$$

where p is a probability vector that describes the probability mass distribution of the sketch of $\sigma(A)$. Scaling this probability vector by the size of $\sigma(A)$ is equivalent to the Count-Min sketch of the selection. The crux of our work is proposing a method for training transformers to output p for any arbitrary selection. Additionally, since $|\sigma(A)|$ is not exactly known prior to query execution, we explain how our model can estimate it.

3 PRELIMINARIES

3.1 Query Optimization

Query optimization in databases relies on cardinality estimation to predict and minimize the runtime of queries. The cardinality of a query and the subqueries in its execution plan is strongly indicative of its runtime. For example, consider a query joining 3 selections $\sigma(A)$, $\sigma(B)$, and $\sigma(C)$. Which join should be executed first — should $\sigma(A) \bowtie \sigma(B)$ or $\sigma(B) \bowtie \sigma(C)$ go first? Additionally, since a join is a commutative operation, $\sigma(A) \bowtie \sigma(C)$ can also go first. Either way, the query will have the same result that is $\sigma(A) \bowtie \sigma(B) \bowtie \sigma(C)$.

While a query may have many correct execution plans, most are suboptimal in terms of runtime. Accurate cardinality estimation is vital to guiding the query optimizer's search for an optimal plan. An optimizer enumerates potential plans, evaluating each with a cost function based on their

cardinality. Cardinality estimation must be performed quickly, to avoid having a negative impact on query latency. In practice, cardinality estimation methods implemented in databases tends towards being fast and simple.

3.2 PostgreSQL

PostgreSQL is the premier database system featured in research literature. Being both open-source and relatively mature, its simple yet effective cardinality estimator is a classic baseline [12]. By default, PostgreSQL's cardinality estimator relies on maintaining small univariate histograms and tracking the most common values per column. These are inexpensive and can be applied to various predicates and data types. Since these statistics are univariate or per column, cardinality estimation in PostgreSQL assumes that columns are statistically independent. When this assumption (often) does not hold, it may lead to inaccurate cardinality estimation. While PostgreSQL supports multivariate histograms, they are not used by default due to their exponential size complexity.

3.3 Sketches

Sketch algorithms process data to create compact data structures, called sketches, which can efficiently summarize data in logarithmic space. This diverse class of algorithms are characterized as *pass efficient*, since sketches are created in a single pass. This lends to their formulation as stream processing algorithms, where the insertions and deletions to a database can be considered a stream. Like histograms, sketches summarize data in a fixed amount of memory, asymptotic to the full data. However, various sketches are specialized for different purposes, such as estimating the number of distinct values [9], testing membership [3], or estimating frequencies [2]. We focus on two hash-based frequency sketches, Count-Min [6] and Fast-AGMS [5]. Despite being univariate, they can accurately estimate the cardinality of joins without assuming independence, though they are not without limitations.

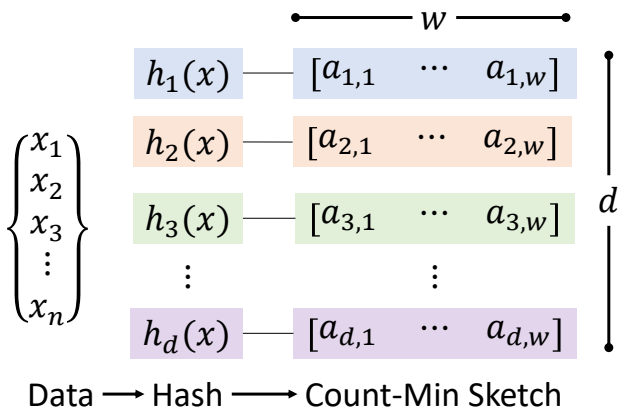


Fig. 1. Count-Min sketch with a $d \times w$ structure. Depth d refers to the number of independent sketches. Width w refers to the number of counters in each sketch.

3.4 Count-Min

The Count-Min sketch is a hash-based sketch that can estimate the cardinality of point, range, and join queries. It can achieve an expected error ϵ while only using space proportional to $1/\epsilon$. The sketch can be considered a vector $a \in \mathcal{R}^w$, initialized from zero. For each element in a stream,

the sketch is updated by a hash function $h : \mathcal{R} \rightarrow [1, w]$. For example, inserting an element x increments the corresponding *counter* in vector a such that $a_{h(x)} \leftarrow a_{h(x)} + 1$. Likewise, deleting an element subtracts from the corresponding counter. It follows that the cardinality of a query for a point x is upper-bounded by $a_{h(x)}$. To tighten this bound, d independent Count-Min sketches can be kept, each with their own hash function, as depicted in Figure 1. The upper-bound is then the minimum of the independent estimates:

$$|\sigma_x| \leq \min \left\{ a_{1,h_1(x)} \dots a_{d,h_d(x)} \right\} \tag{1}$$

Since data is hashed, the Count-Min sketch requires additional specialization for range queries. Before hashing, first consider which interval(s) the element belongs to. For example, the element x belongs to the $\lfloor x/32 \rfloor$-th interval of size 32. A Count-Min sketch for size 32 intervals would then be updated by hashing $\lfloor x/32 \rfloor$ instead of x. Such a sketch infers tighter upper-bounds of range queries, especially for closed ranges of sizes up to 32. Additionally, sketches of various interval sizes can efficiently handle large ranges. Any interval of size n can be exactly partitioned into $O(\log n)$ dyadic intervals [11] which are disjoint intervals sized 2^i and can be uniquely identified by the quotient of x divided by 2^i:

$$D^i(x) = \left\lfloor \frac{x}{2^i} \right\rfloor \tag{2}$$

The cardinality of a join query is estimated by the dot product between Count-Min sketches that share the same hash function. The cardinality of the join between streams A and B is upper-bounded by the dot product of their sketches a and b, respectively. Intuitively, the elements in A whose hash is i may only join with elements in B with the same hash, resulting in the upper-bound of $a_i b_i$. Rather than an upper-bound, Count-Min can perform the unbiased estimation presented as Count-Mean-Min by Deng et al. [7], where sketches a and b are *denoised* before computing their dot product:

$$a'_i = \frac{w-1}{w} \left(a_i - \bar{a} + \frac{a_i}{w} \right) \tag{3}$$

The denoised Count-Min sketch of a is denoted as a'. It is computed by deducting the average of all other counters – besides itself – from each counter. Deng et al. [7] intuitively explained that this "removes the noise and returns the residue" in each counter. Afterwards, the dot product $a' \cdot b'$, is an unbiased cardinality estimate of $A \bowtie B$. Since the estimates are no longer an upper-bound, the median of d unbiased estimates should be taken as the final estimate.

3.5 Fast-AGMS

Like Count-Min, Fast-AGMS is also a hash-based sketch that can estimate cardinalities of point and join queries. Whereas Count-Min only uses one hash function per sketch, Fast-AGMS uses two: one $h : \mathcal{R} \rightarrow [1, w]$ for determining which counter to update and another $\xi : \mathcal{R} \rightarrow \pm 1$ to decide whether to add or subtract. For example, inserting an element x updates a counter in vector a such that $a_{h(x)} \leftarrow a_{h(x)} + \xi(x)$. Cardinality estimation is performed identically to Count-Min, except the resulting estimates are inherently unbiased.

While Count-Min and Fast-AGMS sketches are univariate, their cardinality estimation for join queries do not assume that data is statistically independent. By using hash functions, sketches instead assume that elements with the same hash can join together. However, being univariate, a sketch cannot be applied to selections that are subsets of its own data. Although a sketch can be applied to a larger selection, since every sketched element is included, the same guarantee cannot

be made if the sketch is applied to a smaller selection. As such, sketches are ideally made for specific selections, rather than for the full data.

3.6 Transformers

Rather than maintain sketches for various selections, our work uses transformers to approximate the sketch of any arbitrary selection. Transformers output conditional probabilities, which sketches can be seen as a function of. This class of neural networks is based on attention mechanisms [28], which generally consists of multiple *heads* that each learn to extract relevant features from inputs before aggregating them. These state-of-the-art models have become ubiquitous, even outside of natural language processing (NLP).

In NLP, the masked language modeling problem is particularly relevant to our work. Masked language modeling uses models for the task of filling in the blank or **MASK** token. For example, consider the sentence "*I went to the grocery store for milk*" where each word can be represented by an embedding. Embeddings are defined as vectors which each correspond to their own specific concept — in this case a specific word — that they represent. We replace the embedding of "*milk*" with a special **MASK** embedding that indicates an omitted word. The **MASK** embedding is used to represent missing or unknown values and can be used to train models to fill in those values. We can train a transformer model to predict the omitted — *masked* — word in the sequence by using the original sequence as the target label. The transformer outputs a probability distribution over the words in its dictionary of embeddings. This probability distribution takes the form of a probability vector of size m, where m is the number of words represented in its embedding dictionary. This is conditioned on other words in the input e.g. $P\left(X_8 | x_1 = \text{I}, x_2 = \text{went}, \ldots, x_7 = \text{for}\right) \in \mathcal{R}^m$ which denotes a probability vector for the 8^{th} *masked* word conditioned on the other known words.

3.7 NeuroCard

Transformers were applied to cardinality estimation before, specifically for point and range queries in Naru and then extended to joins by Yang et al. in NeuroCard [32]. These methods utilize self-supervised models as to learn the joint distribution amongst columns. In Naru, Yang et al. [33] trains a unidirectional transformer on an entire table, treating each row as a sequence of columns. For each row, the transformer predicts the value of the first column x_1, then the value of the second x_2 given x_1, then the value of the third x_3 given x_1 and x_2, and so on. At each step, the transformer outputs the probability distribution of a column conditioned on prior columns. By using the chain rule, the product of these conditional probabilities is a joint probability equal to the selectivity of a point query:

$$\begin{aligned} |\sigma_{x_1 \wedge x_2 \wedge \cdots \wedge x_n}| &\propto P(x_1, x_2, \ldots, x_n) \\ &\propto P(x_1) P\left(x_2 | x_1\right) \ldots P\left(x_n | x_1, x_2, \ldots\right) \end{aligned} \tag{4}$$

Point predicates are denoted as $x_1, \ldots, x_n$ whose point value is represented by an embedding. When a column has no predicate, it can be considered an unknown — a *wildcard* — and represented by the **MASK** embedding. Without the **MASK** embedding, the transformer would otherwise infer a probability for each possible value of the wildcard column — summing these probabilities marginalizes the wildcard from the joint probability. To efficiently handle wildcards, Naru implements a sampling algorithm to reduce the number of inferences. This sampling algorithm also handles range predicates, i.e., the sum of probabilities for all values within a range equals its selectivity.

For joins, Yang et al. proposed NeuroCard, which simply extends Naru beyond a single table. NeuroCard trains a single unidirectional transformer on the full outer join of all tables in the database. To remain tractable, only a small sample of the full outer join is used for training. This

method has clear drawbacks, such as requiring prior knowledge of the join schema and having to potentially model extremely high dimensional data, which may not always be viable. However, if successful, the model can then treat join queries as a query on a view of the full outer join. In cardinality estimation methods, each additional join in a query typically incurs significant error, but this seems not to be the case for NeuroCard, which treats the full outer join as a single table.

NeuroCard demonstrates that transformers can be highly accurate cardinality estimators, even for joins. However, it may not always be feasible to train on the full outer join or expect a single model to learn its joint distribution. As a more scalable alternative, our work eschews the join schema and trains just one model per table, relying on sketches to handle joins.

4 APPROXIMATE SKETCHES

4.1 Notation

We describe the notation in this section. Firstly, d and w are reserved for the depth and width of sketches, respectively. A hash function $h_d : \mathcal{R} \rightarrow \{1, \ldots, w\}$ may have a subscript to denote that it belongs to one of d independent sketches. If a hash function is without a subscript, then the context assumes it corresponds to a singular sketch.

The inputs to our models are embeddings — vectors which represent all possible hash values. Additionally, there is also a **MASK** embedding vector, which does not represent a hash value, but instead the lack of any. The **MASK** embedding is required to obtain the probability distribution of a particular column. All embeddings are randomly initialized prior to training.

It is assumed that the output of our models represent the probability distribution of hashes — specifically a probability vector denoted as $P(h(X)) \in \mathcal{R}^w$, where capital X is a specific column or attribute. When the column is capitalized, e.g., X, then $P(h(X)) \in \mathcal{R}^w$ is a vector of w probabilities that represents the distribution of the hashes of X. We abuse notation to let $h(X)$ refer to the multiset of hashes of the values in X e.g., $\{h(x) \forall x \in X\}$. Otherwise, lower-case x refers to a specific value and $P(h(x)) \in [0, 1]$ is the scalar probability of the hash $h(x)$ being in the multiset of hashes. This is equivalent to a point predicate for the equality $X = x$ and a upper-bound of its selectivity $P(x) \leq P(h(x))$ Note that the selectivity of the point predicate is referred to as $P(x)$ while implying that it is estimated from our model as $P(h(x))$.

When there are multiple point predicates $\{x_1, \ldots, x_n\}$ their joint selectivity — the joint probability of satisfying all predicates — may be referred as $P(x_1, \ldots, x_n)$ when n is given. Otherwise, their joint selectivity may also be denoted as $P\ (x_k\ \forall k)$ when there can be an arbitrary number of predicates. The subscript variables i, j, and k subscript an arbitrary predicate or hash function.

4.2 Model

We use bidirectional transformers [8] to approximate sketches. It differs from the unidirectional transformer in NeuroCard, which estimates a column's probability distribution conditioned only on predicates of *preceding* columns. The order of columns is fixed prior to training, so a unidirectional transformer cannot estimate conditional probability distributions for columns that precedes a predicate column. Therefore, they cannot be used to approximate sketches of selections with predicates subsequent to the sketch column — the column whose hashes are used to update the sketch.

In contrast, bidirectional transformers can estimate a column's probability distribution conditioned on predicates of both preceding and subsequent columns. For any given row, a bidirectional transformer predicts the values of all masked columns at once, whereas unidirectional transformers predicts for each column one by one, conditioning the estimate of each column on those

prior to it. For example, consider a table T containing columns $(X_1, \ldots, X_n)$ and also its predicates $x_1, \ldots, x_n$ represented by the embeddings of their point values or the **MASK** embedding for wildcard predicates. Taking those embeddings as input, a bidirectional transformer outputs a conditional probability for every wildcard, conditioned on every embedding that is not a wildcard. In the case where only x_1 is a wildcard, then the model outputs a single conditional probability vector $P(X_1 \mid x_2, \ldots, x_n)$ that describes the probability mass distribution of $\Pi_{X_1}\left(\sigma_{x_2 \wedge \cdots \wedge x_n}(X)\right)$ — the values of X_1 within the predicate selection. When every predicate is a wildcard, then the model instead outputs a probability vector for each column: $P(X_1), \ldots, P(X_n)$. As such, the order of columns does not prevent our models from approximating the sketch of any arbitrary column and selection. In practice, even a simple multilayer perceptron may estimate a probability conditioned on both left and right contexts. However, we choose to use bidirectional transformers because of its ubiquity in masked language modeling

As transformers uses embedding vectors to represent discrete values such as hashes, we cannot directly apply our models to range predicates, as ranges are continuous. Instead, we focus on point queries. Range queries can then be handled by discretizing intervals and treating range predicates as multiple point predicates.

4.3 Training Data

To train, we preprocess each table with hash functions. For each row of a table, the hash function $h : \mathcal{R} \rightarrow \{1, \ldots, w\}$ is applied to each column. Thus we replace each individual element with its hash value. We also initialize embeddings for all possible hash values — corresponding to w embedding vectors per column. Embeddings are not shared between columns, as to not assume that their hashes have the same distribution.

When approximating d independent sketches, we repeat the process for each hash function. This produces $d \times n$ hashes, where n is the number of columns — each column is hashed d times. We concatenate these hashes and treat it as a single row. Again, embeddings are not shared between columns nor hash functions. This allows the model to learn the joint probability distribution of all of a table's hashes:

$$P\left(h_i(X_j)\ \forall j < n\ \forall i < d\right) \tag{5}$$

Intuitively, multiple hash functions can represent exponentially more distinct values. A single hash function can only represent w distinct values, whereas d hash functions can represent w^d distinct values. In total, we create $w \times d \times n$ embeddings per table, each representing a hash value.

During training, each row is considered a single training sample and all preprocessing e.g., hashing, can be performed on the fly. As in masked language modeling, columns are randomly masked — their embeddings are replaced by a **MASK** embedding. The model's objective is to predict the masked hash value, by outputting a probability vector of size w where the i-th element is the probability of the masked hash value being i. By running gradient descent over an entire table, the model eventually learns the joint probability distribution of (the hashes of) its columns.

After training, we can infer the model for approximate sketches of arbitrary selections. While point queries and range queries do not require explicitly require approximating sketches from the model, its cardinality estimates are equivalent to Count-Min estimates. Only join queries require explicitly approximating sketches, whose inner products estimate the cardinality of joins.

4.4 Point Queries

Point queries are handled similarly to Naru, by estimating its selectivity as a joint probability, factorized as the product of conditional probabilities. Unlike Naru, which directly learns the

distribution of columns, our models learn the distribution of their hashes. Additionally, using a bidirectional model means joint probabilities can factorized in any order via the chain rule in Equation 4.

Consider a point query on a n-column table T and predicates with point values $(x_1, \ldots, x_n)$ on columns $(X_1, \ldots, X_n)$. Its cardinality equals the size of T times its selectivity — equal to the joint probability of satisfying all predicates:

$$|\sigma_{x_1 \wedge \cdots \wedge x_n}| = |T| \times P\left(x_1, \ldots, x_n\right) \tag{6}$$

This joint probability can be estimated from a model trained on table T. The point values for each predicate are converted to $n \times d$ hashes $\{h_i(x_j)\ \forall j < n\ \forall i < d\}$ which are each represented by their corresponding embedding. Since bidirectional models can factorize joint probabilities in any order, let i and j be arbitrarily chosen and replace the embedding of $h_i(x_j)$ with the **MASK** embedding, masking it. Wildcards — columns without predicates — default to the **MASK** embedding.

The model takes the embeddings of those point values and outputs conditional probability vectors, $P\left(h_i(x_j) \mid h_k(x_l)\ \forall k\ \forall l \neq j\right) \in [0, 1]$, one for each masked column hash, e.g., $h_i(x_j)$. Masking $h_i(x_j)$ indicates the model to output a probability vector for the distribution of $h_i(X_j)$ conditioned on other predicates — the $h_i(x_j)$-th element of this vector is an upper-bound of the scalar probability of satisfying predicate $X_j = x_j$ given that all other predicates $X_l = x_l$ are satisfied. The process is repeated with arbitrary i and j until all predicates are masked once, estimating a conditional probability for each predicate. Their product is the joint factorization:

$$\begin{aligned} P\left(x_1, \ldots, x_n\right) &\leq P\left(h_i(x_j)\ \forall j < n\ \forall i < d\right) \\ &\leq \prod_{i,j}^{d,n} P\left(h_i(x_j) \mid h_k(x_l)\ \forall k\ \forall l > j\right) \end{aligned} \tag{7}$$

This estimates (an upper-bound of) the selectivity of all predicates. Scaling by the size of the table $|T|$ produces the cardinality estimate of the point query. Since our model is trained on hashes, the resulting cardinality estimate is an approximation of a Count-Min estimate, hence an upper-bound. However, in practice, the model may produce errors such that the estimate is not an upper-bound.

year			
2^3	2^2	2^1	2^0
[2000, 2008)	⊃ [2004, 2008)	⊃ [2006, 2008)	⊃ 2006
[2008, 2016)	⊃ [2012, 2016)	⊃ [2012, 2014)	⊃ 2012
[2008, 2016)	⊃ [2012, 2016)	⊃ [2012, 2014)	⊃ 2012
[2008, 2016)	⊃ [2012, 2016)	⊃ [2012, 2014)	⊃ 2013
[1984, 1992)	⊃ [1984, 1988)	⊃ [1984, 1986)	⊃ 1984
[2000, 2008)	⊃ [2004, 2008)	⊃ [2004, 2006)	⊃ 2004

Table 1. Dyadic intervals containing the values of a continuous column, *year*. Interval sizes are user-specified and treated as another column for training.

4.5 Range Queries

Since embeddings represent discrete values, we discretize ranges. We adopt dyadic intervals as our discrete units, following their original proposal in Count-Min [6]. We propose a general algorithm

to decompose ranges into (dyadic) intervals, which are each be treated as a point. This allows us to reduce range queries into *dyadic range* queries which are effectively point queries. For our model to handle dyadic range queries, we augment our training data to include dyadic intervals.

As shown in Table 1, we create *virtual* columns during training that correspond to various dyadic interval sizes e.g., $2^1, 2^2, \ldots, 2^k$. Virtual columns are treated the same as other columns, though they did not originally exist in our table. For a numerical column value x, a virtual column for interval size 2^i will contain $D^i(x)$ which uniquely identifies the 2^i sized interval containing x. Each numerical column may have its own corresponding virtual columns. As before, virtual columns are hashed and assigned embeddings like other columns.

Any range can be decomposed into a *minimal dyadic cover*, which is the unique set of dyadic intervals that exactly covers the given range. A range of size n can be decomposed to $O\left(\log_2 n\right)$ dyadic intervals. The exact number of intervals in the decomposition depends on how many and what interval sizes the model is trained with. Large interval sizes reduce ranges into fewer point queries, but increase the number of embeddings to train.

A decomposition may still involve exceedingly many intervals, especially when the model is trained with smaller interval sizes. Furthermore, not every interval may be relevant to a range query. Intervals which contain frequent points can be considered more informative than others. The frequency of points within an interval can be estimated via univariate statistics such as histograms. Naru implements a *progressive sampling* algorithm that exploits their model to infer the frequency of points in a range. We implement Algorithm 1 to decompose ranges into closed intervals while using a frequency estimator to prune less relevant intervals.

Our algorithm decomposes a range $[\alpha, \beta]$ by finding a cover with the largest interval size and iterates on refining the left and right-most intervals of the cover. Large intervals are preferred over small ones — each interval is treated as a point. A large interval can only be replaced by smaller intervals that are less *noisy*. An interval's noise is its *difference* or the frequency of its points that are not within the original range $[\alpha, \beta]$. Frequency is inferred from a given function f, which can be our model or a simple histogram.

Algorithm 1 Decompose Range

Input: Range $[\alpha, \beta]$,
Interval Sizes $\mathcal{S}$,
Sampling Limit k,
Frequency $f : [a, b] \rightarrow$ number of rows in $[a, b]$
Output: Intervals $\mathcal{I}$
Initialize set $\mathcal{I} \leftarrow$ intervals of size $\max \mathcal{S}$ covering $[\alpha, \beta]$
Sort $\mathcal{S}$ in descending order
for remaining sizes $s \in \mathcal{S}$ **do**
  $[a, b) \leftarrow$ interval of size s containing left bound α
  Calculate noise $\leftarrow f([a, \alpha)) + f([\beta, b))$
  if noise < noise of left-most interval $\in \mathcal{I}$ **then**
    Replace left-most interval with intervals of size s
  end if
  Repeat process for right-most interval
end for
Sort $\mathcal{I}$ in descending order of $f([a, b))$
return top-k intervals in $\mathcal{I}$

To illustrate the decomposition, consider the following example. Given the range $[1, 10]$, decompose it into dyadic intervals of size 2, 4, and 8. Assume that the frequency of any value is 1. First, we cover the range using size 8 intervals:

$$[1, 10] \approx [0, 8) \cup [8, 16) \tag{8}$$

The left-bound cannot be better covered by $[0, 2)$ or $[0, 4)$, so we do not replace the left-most interval. The right-most interval is noisy, because it has the difference $[11, 16)$. We can reduce the difference, by replacing the right-most interval with one of size 4:

$$[1, 10] \approx [0, 8) \cup [8, 12) \tag{9}$$

The right-most interval now has the improved difference $[11, 12)$. We do not replace it with intervals of size 2, $[8, 10)$ and $[10, 12)$, since its difference would not improve. The final decomposition consists of just 2 dyadic intervals, with respective sizes 8 and 4. They are represented as values $D^3(0)$ and $D^2(8)$ in virtual columns, and can be queried as points via Equation 7.

4.6 Join Queries

For point and range queries, there has been no need to *materialize* a sketch. However, join queries involve multiple selections that must each have a sketch to estimate the cardinality of their join.

Approximate sketches of arbitrary selections can be inferred from the models corresponding to the table of each selection. A Count-Min sketch, vector a, consists of w counters which are incremented according to its hash function $h : \mathcal{R} \rightarrow [1, w]$ applied to a particular column — the sketch column — of a selection. The i-th counter, a_i, represents the number of rows from the selection whose sketch column value has the hash i. This can be equivalent to the product of the cardinality of the selection and the conditional probability of a row having the hash i given that it satisfies the selection predicates:

$$a_i = |T| P\left(x_k \; \forall k\right) \times P\left(h(x_j) = i \mid x_k \; \forall k\right) \tag{10}$$

Let x_j denote values in the sketch column and $\{x_k \; \forall k\}$ denote the set of k selection predicates. Additionally, $h(x_j)$ must equal i, in order to update a_i. The cardinality of the selection equals the joint selectivity of all selection predicates $P\left(x_k \; \forall k\right)$ scaled by the number of rows in the table $|T|$. The entire sketch can therefore be expressed as a conditional probability vector times the cardinality of the selection:

$$a = P\left(h(X_j) \mid x_k \; \forall k\right) P\left(x_k \; \forall k\right) |T| \tag{11}$$

In Equation 11, both the conditional probability vector and the selectivity can be inferred from the model. Selectivity can be estimated via Equation 7. The conditional probability vector is estimated by providing the **MASK** embedding for $h(X_j)$, indicating that the model should output the distribution of $h(X_j)$ conditioned on the predicates — the embeddings of the hashes of all predicate values are input to the model.

When the selection predicates include a range, the query is decomposed into multiple point queries via Algorithm 1. Each point query partitions the original selection into *sub*-selections. The sum of sketches from every sub-selection is the sketch of the original selection, via the linear property of Count-Min sketches, i.e., the sum of sketches of the sub-selections equals the sketch of the union of those sub-selections:

An example of this type of selection is shown in Figure 2. For the range predicate $x \in [0, 10]$ and the point predicate $y = 2$, our goal is the sketch of column Z on the selection. Assume our model uses the dyadic interval sizes of 4 and 8. First, decompose any ranges. Decomposing the

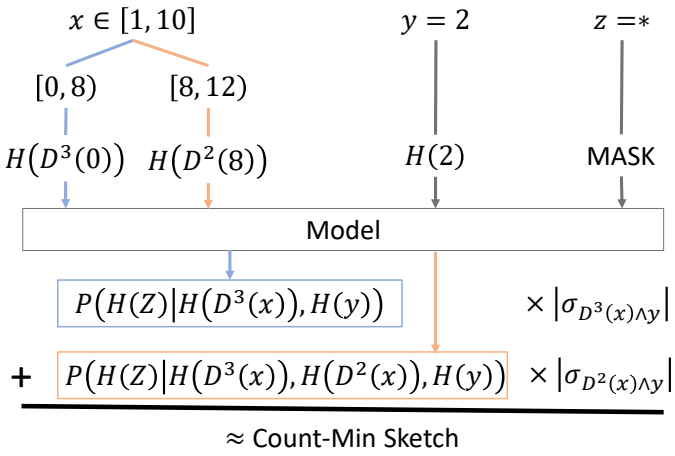


Fig. 2. A selection query defined by a range on x and a point on y. Decomposing the range produces two *dyadic ranges* whose approximate sketches are summed up.

range on x produces 2 sub-selections: $\{x \in [0, 8), y = 2\}$ and $\{x \in [8, 12), y = 2\}$. Then, map each predicate value to its embedding via hashing. To be hashed, dyadic intervals of size 2^i are mapped to a natural number using Equation 2, which is the input embedding to the model along with the **MASK** embedding for the sketch column Z. The selection $\{x \in [8, 12), y = 2\}$ can also include an embedding for the condition $x \in [8, 16)$ without being incorrect — its embeddings represent $h(D^3(8))$, $h(D^2(8))$, and $h(2)$ for both dyadic intervals having x and the point $y = 2$. The model outputs a probability vector for the distribution of the hash of z conditioned on the predicates. To approximate the Count-Min sketch, scale each probability vector by the cardinality of their sub-selection, which can be estimated (7). Finally, add the sketch of each sub-selection for the sketch of their union.

After approximating the sketch of each selection, the cardinality of their join can be estimated from the sketches. For 2-way joins, the classical sketch join estimate is the dot product of sketch vectors. However, for joins of more than 2 selections, the sketch estimate is not well-defined in literature. Izenov et al. [16] proposed the COMPASS merge method for multi-way join size estimation with Fast-AGMS sketches, which involves successively *merging* sketches. We adapt this method, combined with Count-Mean-Min [7], for unbiased estimates of multi-way joins.

To illustrate, consider 3 selections $\{A, B, C\}$ and their respective sketches $\{a_1, \ldots, a_d\}$, $\{b_1, \ldots, b_d\}$, and $\{c_1, \ldots, c_d\}$ which use the same set of d hash functions. To estimate the cardinality of the 3-way join, first designate one of the selections as the *center*. Let b be the center and pick any two of its sketches: b_i and b_j such that $i \neq j$. Merge b_i and b_j by taking the element-wise minimum (absolute value) of each element in the vectors, keeping the signs of each element. We denote the merged sketch as b_{ij}. Then, element-wise multiply a_i, b_{ij}, and c_j, sum up all elements, and return the absolute value as the estimate:

$$|a \bowtie b \bowtie c| = \sum a_i \circ b_{ij} \circ c_j \,, \quad i \neq j \tag{12}$$

Repeat this estimation for all combinations of i and j, taking the median as the final estimate. This method is extends to n-way star joins by merging $n - 1$ sketches, e.g., 2 sketches are merged in this

3-way join example. Intuitively, the element-wise minimum merging alleviates the overestimation of multiplying multiple sketches. Although originally proposed for Fast-AGMS sketches, we apply it to unbiased [7] Count-Min sketches.

4.7 Upper-Bounded Sketches

We propose a simplification of our approximate sketch formula (11) to more quickly approximate sketches. In Equation 11, the joint selectivity of k predicates is the joint probability $P\left(x_k\ \forall k\right)$. Recall that this can be factorized as the product of k conditional probabilities:

$$P\left(x_k\ \forall k\right) = P\left(x_1 \mid ..., x_k\right) P\left(x_2 \mid \ldots, x_k\right) \ldots P\left(x_k\right)$$

where each conditional probability can be estimated from the model. However, each estimation linearly increases the time spent on model inference. Estimation error for each conditional probability may even compound when multiplied. Thus, reducing the number of estimates may improve inference time with negligible error.

The inequality $P\left(x_k\ \forall k\right) \leq \min\left\{P\left(x_k\right)\ \forall k\right\}$ expresses the joint selectivity as upper-bounded by the selectivity of the single most selective predicate. As such, the Count-Min sketch has an upper-bound that replaces the joint selectivity with the minimum selectivity of all predicates:

$$\begin{aligned} a &= P\left(H(X_j) \mid x_k\ \forall k\right) P\left(x_k\ \forall k\right) |T| \\ &\leq P\left(H(X_j) \mid x_k\ \forall k\right) \min\left\{P\left(x_k\right)\ \forall k\right\} |T| \end{aligned} \tag{13}$$

Approximating this upper-bounded sketch only requires estimating the conditional probability vector and the selectivity of a single predicate. A conditional probability vector can be estimated by just inferring the model once. The selectivity of a single predicate, an unconditional probability, does not necessarily have to be inferred from the model. Rather, even the simple univariate histograms already used in databases would suffice. Thus, sketches have an upper-bounded approximation which only requires inferring the model a single time.

4.8 Count-Min as Fast-AGMS

Recall that the differentiating characteristic of Fast-AGMS sketch is that they can either increment or decrement their counters. When inserting an element into the Fast-AGMS sketch, a hash function $h : \mathcal{R} \rightarrow \{1, \ldots, w\}$ determines which counter to update. Another hash function $\xi : \mathcal{R} \rightarrow \pm 1$ determines whether to increment or decrement that counter. Consider inserting an element x. If $\xi(x) > 0$, then increment the $h(x)$-th counter. Otherwise, if $\xi(x) < 0$, then decrement that counter instead. This can be seen as keeping two vectors of w counters — one corresponding to counters which are only incremented and the other for counters which are only decremented. The Fast-AGMS sketch can be computed by element-wise adding these two vectors. We can revise our hash function to create Count-Min sketches that can also be converted to Fast-AGMS sketches:

$$H(x) = \begin{cases} h(x) & \xi(x) = 1 \\ w + h(x) & \xi(x) = -1 \end{cases} \tag{14}$$

Our proposed combined hash function uses both h and ξ to update a vector of $2w$ counters. This hash function $H : \mathcal{R} \rightarrow \{1, \ldots, w\}$ updates the first w counters for elements whose $\xi(x) > 0$. Otherwise, elements whose $\xi(x) < 0$ are mapped to the last w counters. Using this combined hash function to train our model, our approximate Count-Min sketch can be converted to a half-width Fast-AGMS sketch by subtracting the last half of the vector from the first half. Converting to Fast-AGMS may be desirable when an unbiased estimate is preferred. Deng et al. [7] proposed a

Table	Columns	Rows
movie_info_idx	2	1,380,035
title	8	2,528,312
movie_companies	3	2,609,129
movie_keyword	2	4,523,930
movie_info	2	14,835,720
cast_info	3	36,244,344

Table 2. Tables in JOB-light-ranges with their number of rows and relevant columns.

similar method for unbiased estimation using Count-Min sketches, while preserving their width. We experimentally investigate both alternatives.

5 EXPERIMENTAL EVALUATION

5.1 Workloads

Sketches and their approximate counterparts are evaluated on 2 workloads derived from the Join Order Benchmark [20] or JOB. The original JOB workload consists of 113 realistic join queries based on data from the *Internet Movie Database* or IMDB [15]. It was designed to evaluate the efficacy of query optimization methods, such as cardinality estimation, for actual databases. The derived workloads are commonly used [12] for evaluating *learned* model-based cardinality estimators:

(1) JOB-light [18] consists of 70 queries from JOB without string predicates and joins only up to 5 tables — strings often escape the scope of *learned* cardinality estimation research and some models only support a fixed maximum number of joins.
(2) JOB-light-ranges [32] has 1000 queries generated following the same patterns as JOB-light but features range predicates on multiple attributes, whereas JOB-light only had one attribute for ranges.

For each query, we derive their *subqueries* on which we evaluate the accuracy of cardinality estimation methods. These subqueries are all the possible joined in each query e.g., $A \bowtie B \bowtie C$ has 3 subqueries: $A \bowtie B$, $A \bowtie C$, and $B \bowtie C$. There are 696 and 10230 total subqueries in JOB-light and JOB-light-ranges, respectively. Both workloads involve the same set of tables, shown in Table 2.

5.2 Metrics

Cardinality estimation accuracy is measured using the standard *Q-error* metric [23] calculated as the ratio between the estimate $\hat{Y}$ and the target Y:

$$\text{Q-error} = \frac{\max\left\{Y, \hat{Y}\right\}}{\min\left\{Y, \hat{Y}\right\}} \tag{15}$$

A large Q-error does not necessarily mean the cardinality estimates are insufficient for effective query optimization. Instead, some research [24] show that cardinality estimates with poor Q-error may still produce optimal plans in PostgreSQL.

To evaluate the efficacy of our methods in query optimization, we integrate their cardinality estimates into PostgreSQL 13.1 and evaluate query plans using the *P-error* metric [12]. P-error is defined as the ratio between the PostgreSQL plan cost (*PPC*) of the plan created based on estimated

cardinalities $\hat{C}$ and that of the plan based on true cardinalities C:

$$\text{P-error} = \frac{PPC\left(P(\hat{C}), C\right)}{PPC\left(P(C), C\right)} \tag{16}$$

where $P(\hat{C})$ denotes the plan generated from the estimated cardinalities. Its PostgreSQL plan cost, denoted by $PPC(P(\hat{C}), C)$, is the cost of executing that plan according to its true cardinalities e.g., as shown via PostgreSQL's `EXPLAIN ANALYZE` command. The closer P-error is to 1, the more optimal the plan is according to the PostgreSQL cost model.

5.3 Methods

We compare exact – computed only over data that satisfies the filter conditions – Count-Min and Fast-AGMS sketches to their approximations. The same model can approximate both Count-Min and Fast-AGMS sketches, either as an exact approximation (Equation 11) or a simplified upper-bounded approximation (Equation 13):

- Count-Min sketches, whether exact or approximate, are used with the *count-mean-min* estimation method [7] to produce unbiased cardinality estimations.
- Fast-AGMS sketches inherently produce unbiased estimates, but we only evaluate Fast-AGMS sketches that are half the widths of their Count-Min counterparts, as per how we derive Fast-AGMS sketches from Count-Min sketches.
- NeuroCard uses a single model trained on the full outer join of the entire database, in order to handle join queries on any subset of the full outer join schema. In contrast with our bidirectional transformers, we compare with the pre-trained unidirectional transformer from Yang et al. [32] with a sampling limit of 512.

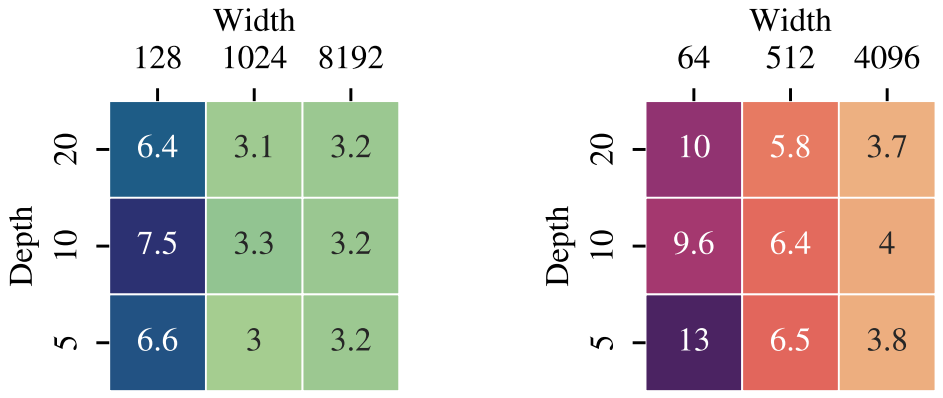
For experiments, we create the exact Count-Min and Fast-AGMS sketches of required selections on demand. These sketches are idealized in the sense that they exactly match the relevant selections in any given query. Realistically, such sketches are unlikely to be readily available, due to the high upkeep or upfront computational cost. In practice, it is more efficient to maintain sketches periodically, similar to histograms in PostgreSQL, which may be slightly out of date. Our goal is to approximate sketches of arbitrary selections while having fewer restrictions than exact sketches.

Our models are trained similarly to NeuroCard's but applied to each individual table rather than the full outer join. By modeling joins via the sketches of individual tables, we can apply approximate sketches to any database and workload without prior knowledge of the join relationships between tables. We compare with NeuroCard to evaluate the extent to which sketches can model joins rather than a single model trained on the full outer join.

Though we compare against exact sketches and NeuroCard, approximate sketches are not expected to be more accurate than either. Intuitively, approximations should not outperform the exact sketches, nor should independent models of individual tables more accurately learn inter-table correlations than a single model of the full outer join. However, we find that approximate sketches closely follow and may sometimes surpass the accuracy of exact sketches.

5.4 Experimental Setup

As typical, training large deep learning models requires sufficiently capable hardware. For training, we use 8 NVIDIA® Tesla® P40 GPUs, though even one would already suffice to speedup training. For inference, in which we only estimate the cardinality of one query at a time, we use a single GPU. Our hardware also includes 2 Intel® Xeon® E5-2699 v4 CPUs and 256 GB RAM memory. To isolate the impact of our cardinality estimation methods, we configure PostgreSQL to use up to 128 GB of memory and run single-threaded, minimizing the effect of disk access operations and parallelism on



(a) Count-Min

(b) Fast-AGMS

Fig. 3. Q-error of exact sketches on JOB-light. Larger width reduces error significantly while larger depth has negligible impact on the error.

query performance. We create indexes on all join keys in our database, allowing the query optimizer to produce diverse plans with indexed operations. Otherwise, without indexes, PostgreSQL's query optimizer may produce similar costing plans regardless of cardinality estimation accuracy.

5.5 Model Training

For each of the 6 tables in our workloads, we train 3 bidirectional transformer models [8] to approximate Count-Min sketches of width 128, 1024, and 8192. The depth of approximate sketches is kept a constant 5. Increasing the depth would heavily inflate the number of embeddings while yielding marginal benefit, as seen in Figure 3. The 128 and 1024 width models use the same hyperparameters, only differing in their number of embeddings. The model with width 8192 has larger hyperparameters, to accommodate more embeddings.

Model	Width	Parameters	Rows/sec
Approx. Sketch	128	321,217	7896
Approx. Sketch	1024	379,457	7153
Approx. Sketch	8192	6,945,025	1628
NeuroCard	–	5,416,064	4364

Table 3. Training rate of approximate sketches (bidirectional transformers) with various widths compared with NeuroCard's unidirectional transformer.

We train each model for 10 epochs and report the rate of training for the largest table, *cast_info*, as the average rows processed per second. We compare these rates in Table 3. A model's total training time primarily scales with its number of parameters and the number of rows to process. On the smallest table, *movie_info_idx*, an epoch may take less than 3 minutes to complete, whereas *cast_info* requires 76 minutes for even our smallest model.

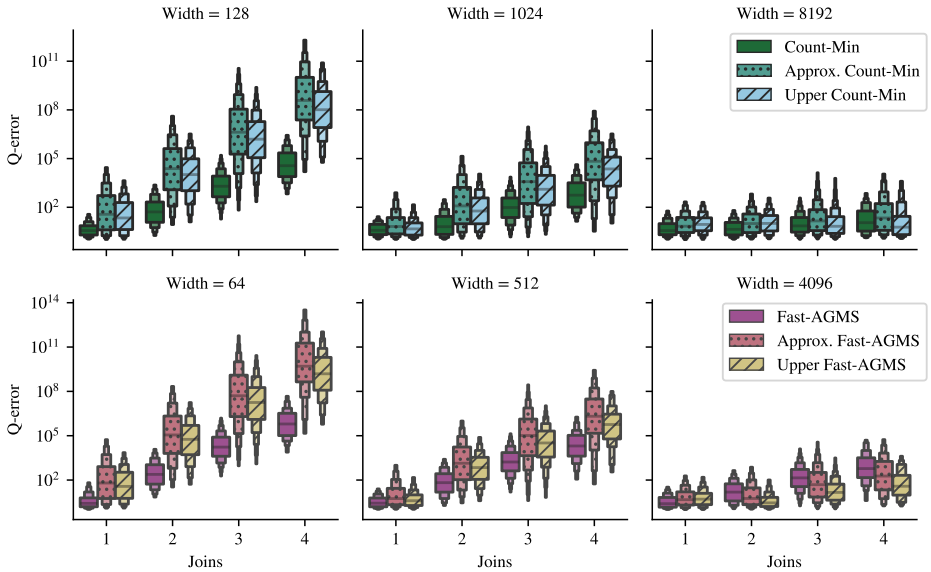


Fig. 4. Distribution of Q-error on JOB-light-ranges subqueries for (approximate) sketches.

For comparison, we include the rate of training for NeuroCard's unidirectional transformer, which is trained on a small sample of the full outer join of all tables. Both our models and NeuroCard's are trained with mini-batch stochastic gradient descent and the same per-device batch size of 512. As expected, the smaller models can be trained at a faster rate, but we note that the rate of training does not scale linearly with the number of model parameters. Although NeuroCard uses 23% fewer parameters than our largest model, it actually has more than twice the training rate. We surmise this discrepancy is partially due to implementation differences, such as hashing on the fly during training.

5.6 Model Inference

In Table 4, we report the estimation time for exact sketches, approximate sketches, and NeuroCard on the 10,230 subqueries in JOB-light-ranges. We compare our approximate sketches to exact Count-Min sketches. The estimation time for exact sketches includes the time to create them in-memory, as exact sketches of exact selections are unlikely to readily exist in a realistic setting. This loosely compares to methods such as COMPASS [16] which creates sketches on-demand by *pushing down* selection predicates. Since our approximate sketch models are trained with a depth of 5, we use the same depth for the exact sketches. We also include the estimation time for NeuroCard, comparing our one-model-per-table estimator to a single-model estimator.

Exact Count-Min sketches are created in-memory by pushing down predicates to obtain each table's relevant selection. Our pushdown implementation queries for each selection, which are then processed to create the exact Count-Min sketches. The end-to-end estimation time for exact sketches is largely determined by the pushdown operation, regardless of the sketch width. This may require entire seconds to create all of the required sketches for a given query. Note that COMPASS reports using GPUs to accelerate sketch creation by a factor of roughly 10, which is not reflected in our reported times.

Estimator	Width	Time (ms)			
		Mean	25%	50%	75%
Count-Min	128	786.20	227.25	590.25	1,118.84
Count-Min	1024	778.02	228.64	595.50	1,116.32
Count-Min	8192	758.83	217.22	573.35	1,092.46
Approx. Sketch	128	5.98	2.80	4.43	7.77
Approx. Sketch	1024	6.41	2.91	4.70	8.30
Approx. Sketch	8192	6.52	3.23	4.99	8.44
NeuroCard	–	120.34	98.60	122.71	138.51

Table 4. Estimation time for learned and pushdown sketches.

The estimation time for our approximate sketch method includes the time from approximating each sketch to using those sketches for cardinality estimation. For any join query, the sketch of each selection being joined can be approximated from our models in a single inference. Even for range selections, which are partitioned via our decomposition algorithm, the sketches of its sub-selections can be estimated together in a single *batched* inference. Thus, the estimation time for approximate sketches scales linearly with the number of tables being sketched, i.e., the number of models to infer from. After approximating the sketches, the cardinality of joins can be estimated as the inner-product between sketches or the multi-way join estimation process from COMPASS. We observe that the estimation times for approximate sketches increases insignificantly for larger widths, as model operations take advantage o GPU parallelism.

Compared to approximate sketches, NeuroCard has a longer estimation time due to its sampling algorithm. It handles ranges by sampling points within the range and estimating the cardinality of each point. NeuroCard samples only the most relevant points as predicted by its model. This requires many model inferences, hence its higher estimation time. Although approximate sketching requires multiple models, each model only requires a single inference. We report estimation times using NeuroCard's unidirectional transformer with a sampling limit of 512.

5.7 Approximate Sketch Accuracy

By comparing their Q-error, we determine whether approximate sketches are on par with exact sketches. Figure 4 shows the Q-error distribution for sketches of various widths on JOB-light-ranges' 10,230 subqueries, separated by number of joins. Approximate sketches can be over a hundred times worse than the exact Count-Min and Fast-AGMS sketches at widths 128 and 64, respectively. However, this disparity is greatly diminished for wider sketches. For the widest widths tested, the Q-error between exact and approximate sketches is indistinguishable. As such, approximate sketches can be a viable alternative to exact sketches, particularly since exact sketches can be impractical to readily have for any arbitrary selection.

We observe that our upper-bounded approximations are generally more accurate than their more exact approximations, for all sketch widths and number of joins. For our widest sketches, upper-bounded approximations may be even more accurate than exact sketches. This tendency can be explained if our models are prone to underestimation, which would then be corrected in our upper-bounded approximation formula (Equation 13).

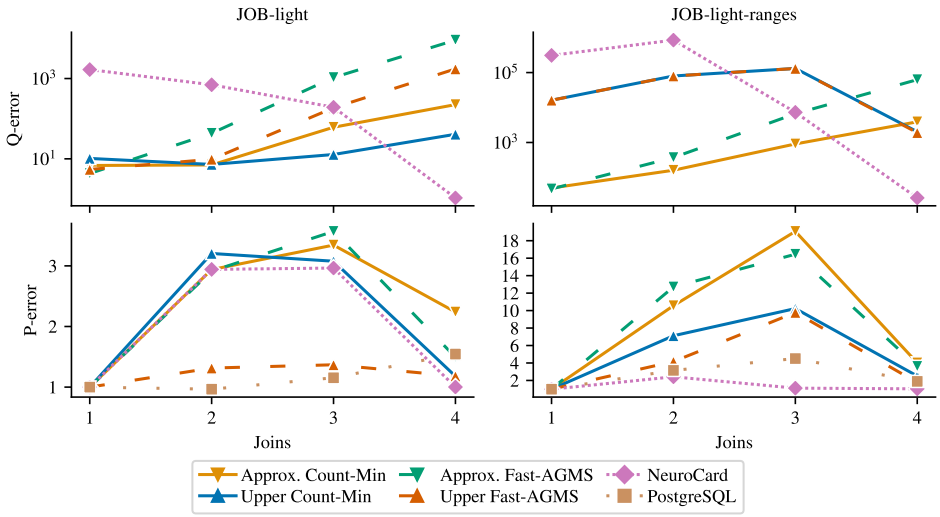


Fig. 5. Average Q-error and P-error by number of joins on JOB-light and JOB-light-ranges.

5.8 Cardinality Estimation

We evaluate the efficacy of approximate sketches as a cardinality estimator for the PostgreSQL database system. For each query, PostgreSQL's query optimizer expects a cardinality estimate for each of its subqueries e.g., for the 70 queries of JOB-light, PostgreSQL requires the cardinality estimates of its 696 subqueries. As such, we apply each of our compared methods to estimate cardinalities for the subqueries of both our workloads.

In Figure 5, we plot each method's average Q-error and P-error by number of joins on both workloads which features up to 4 joins per query. We evaluate both approximate Count-Min and Fast-AGMS sketches of widths 8192 and 4096, respectively. Additionally, both upper-bounded (Equation 13) and exact approximations (Equation 11) are evaluated. NeuroCard is included in evaluations as a single model of the full outer join, in contrast with our *one-model-per-table* methodology. We configure NeuroCard to use a high sampling limit of 8000, to maximize its accuracy.

As expected, approximate sketches generally become less accurate with more joins, due to compounding error from the sketches of additional selections. Our upper-bounded approximation formula eschews all but the most selective predicate in a selection, rather than the joint selectivity of all predicates, which is used in our exact approximations. As such, upper-bounded approximations have better Q-error on JOB-light than on JOB-light-ranges, which features more predicates. However, for queries with 4 joins in JOB-light-ranges, upper-bounded approximations are yet again more accurate — even more than for queries with just a single join. This suggests our upper-bounded approximation is more robust to errors from the model's estimation or that our models tend to underestimate, which is corrected in our upper-bounded approximation formula. Overall, upper-bounded approximations are more accurate than our more exact approximations.

NeuroCard behaves oppositely to approximate sketches — its Q-error improves with each additional join. As a model trained on the full outer join, NeuroCard treats join queries as a query over the view of the full outer join — in which the join query returns a single selection. Thus, unlike approximate sketches, NeuroCard does not produce separate estimates for individual selections, avoiding compounding error. Rather, each additional join helps informs the model, i.e., the input to

NeuroCard would otherwise contain more wildcards. NeuroCard falls short of expectations and only becomes more accurate than approximate sketches for queries with the highest number of joins.

5.9 Plan Cost

Whereas Q-error is the ratio between a cardinality estimate and the true cardinality, P-error is the ratio between the cost of the query plan determined using cardinality estimates and the cost of the *optimal* query plan determined using true cardinalities. As shown in Figure 5, the P-error for queries with just a single join is always 1, since there is only one possible join order. For queries with 2 or more joins, it would be intuitive to expect that the P-errors of approximate sketches and NeuroCard would follow their Q-errors. However, we instead find support for claims that poor Q-error is not particularly indicative of poor plans [24]. In JOB-light, our upper-bounded approximation of Fast-AGMS generally has better P-error than other methods, despite having worse Q-error than approximate Count-Min sketches. Likewise, in JOB-light-ranges, NeuroCard has better P-error than approximate sketches, despite having worse Q-error for queries with less than 4 joins. Q-error was proposed as a loose upper-bound to P-error [23] which entails that inaccurate cardinality estimators can still produce optimal plans, as evident in our evaluation.

We also report the P-error for PostgreSQL — a standard baseline — with its own cardinality estimates based on univariate histograms and heavy hitters. In our comparison, PostgreSQL is unique in combining two different type of statistics instead of using a deep learning model. For both workloads, it outperforms all other methods in our comparison, except NeuroCard on JOB-light-ranges. Given its performance and low requirements, e.g., no training or prior knowledge of joins, there is little margin for improvement over PostgreSQL that would warrant the application of current *learned* cardinality estimators. This matches findings by Yizenov et al. [16] in which PostgreSQL outperforms Fast-AGMS sketches for queries with less than 10 joins. However, superior sketch estimation methods might still be proposed, that can also utilize our approximate sketch, besides COMPASS's join estimation method.

Overall, PostgreSQL consistently achieves impressive P-error. Otherwise, upper-bounded approximate Fast-AGMS has the best P-error on JOB-light, despite Count-Min sketches having better Q-error. On JOB-light-ranges, NeuroCard has the best P-error, despite having the worst Q-error on queries with less than 3 joins. Despite its lack of correlation to Q-error, we find that P-error is clearly reflected in runtime.

5.10 Query Runtime

In Table 5, we report the total runtime for both JOB-light and JOB-light-ranges. For each workload, we measure the time for executing each of its queries — 70 in JOB-light and 991 in JOB-light-ranges. The estimation time for each cardinality estimation method is measured separately and consists of estimating the cardinality for every subquery derived from each workload. We evaluate the same methods compared in Figure 5, in which approximate Count-Min and Fast-AGMS sketches use widths 8192 and 4096, respectively. For NeuroCard, we report its estimation time when it is configured with a sampling limit of 512. Note that all approximate sketches, whether Count-Min or Fast-AGMS, and an upper-bounded approximation or not, are generated by the same model inference — a single output of our model can be used to approximate a Count-Min sketch, via either our exact or upper-bounded formula, which can then also be converted to a Fast-AGMS sketch. As such, the estimation times for all approximate sketches listed in Table 5 share the same estimation time for workload. Especially because the estimation procedure, e.g., a dot product, after approximating the sketches is too short to be significant, even in milliseconds. This is also the case for PostgreSQL's estimation procedure.

Estimator	Total runtime (minutes)			
	JOB-light		JOB-light-ranges	
	Query	Est.	Query	Est.
Approx. Count-Min	90.0	0.23	809.4	1.11
Upper Count-Min	91.8	0.23	685.8	1.11
Approx. Fast-AGMS	88.8	0.23	709.8	1.11
Upper Fast-AGMS	92.4	0.23	664.2	1.11
NeuroCard	90.6	1.17	625.2	20.46
PostgreSQL	101.4	–	648.6	–

Table 5. Total query and estimation runtime.

On JOB-light, we observe that the total query execution time for each workload does not clearly follow Q-error nor P-error. For example, approximate Fast-AGMS has the worst overall Q-error on JOB-light yet still achieves the best query execution time on the workload. However, the difference in query execution time is slight — there is less than a 4% improvement between the slowest and fastest *learned* cardinality estimation method. Meanwhile, all methods result in 10% faster query execution time than PostgreSQL.

On JOB-light-ranges, query execution times perfectly matches the P-error shown in Figure 5. This is likely because the workload is simply larger, according to the law of large numbers. As such, NeuroCard has the best query execution time, followed closely by PostgreSQL. This is still the case, even when including NeuroCard's hefty estimation time, which brings its runtime to within 1% of PostgreSQL's runtime. The best performing approximate sketch, our upper-bounded Fast-AGMS approximation, achieves a total runtime within 3% of PostgreSQL's, whereas our Count-Min approximation has a whopping 25% worse runtime.

Overall, our approximate sketch method clearly approaches the accuracy and performance of NeuroCard. Though unexpected, our upper-bounded approximation formula is an especially effective cardinality estimator. This shows that sketches can enable modeling individual tables and producing join cardinality estimates as effectively as a model trained on the full outer join of all tables, which may not be generally viable.

6 RELATED WORK

Although our work focuses on Count-Min and Fast-AGMS, there are many other sketches with various applications. The HyperLogLog [9] method approximates the number of distinct elements, by hashing them to bit strings and estimating as 2 raised to the maximum number of leading zero bits. Bloom filters [3] uses multiple hash functions to test the membership of specific elements — an element does not exist in the data, if the corresponding bits it would hash to are zero. The AMS sketch [2] estimates the variance of data as approximated by the variance of counters in a Fast-AGMS sketch — Fast-AGMS is derived from AGMS [1] which, in turn, is derived from the AMS sketch. Though sketches have known statistical guarantees, there is not a single *best* sketch for any purpose, since their accuracy depends on both their memory constraint and the actual distribution of the data. Our approximate sketch method can likely be adapted for many varieties of sketches.

If transformers were only trained for up to one epoch, e.g., a single pass over a stream, they could be considered sketches themselves. As research progresses on the efficient training and inference of transformers, it is not unthinkable that such models might eventually be integrated into databases. Although large transformer models are often applied in few-shot or even zero-shot

settings [4], this is predominately for NLP problems. Fine-tuning pre-trained models is unlikely to be applicable to our task, since the structure and patterns of data within a database is unlikely to match that of natural language and even another database. Instead, we look at using extreme batch sizes and carefully adjusting the step size [34] to be a plausible general-purpose optimization for faster training. Also, much of the data used to train deep learning models may be pruned away without degraded performance [26], in line with the observation of Yang et al. [32] that a few million records, less than a thousandth of the full outer join, sufficed to train NeuroCard.

Although it is compelling to think that transformers or other deep learning models might eventually become integrated into databases, there are still many reason it would currently be impractical. Optimizing large models with gradient descent practically requires powerful hardware, such as GPUs and TPUs [17]. Such accelerators may not be readily available or even cost-efficient, depending on the application. *Query-driven* models [21] and those that learn from actual query execution [31] suffer from the cold-start problem, where there is not yet (enough) data to learn from. Training such models may require synthesizing and executing large amounts of queries. *Data-driven* models, such as NeuroCard, avoid this but suffer having prerequisites that limit their practicality. For example, graphical models in DeepDB [13] and derivations [30] require prior analyses of the database to determine an optimal model structure. This assumes there is already data whose distributions are not likely to change frequently. Later work [35] addresses this with an incremental update strategy that conserves the original model structure while adapting to data shifts.

By training one model per table, we avoid restricting our method to a specific join schema — we can estimate the cardinality of any join query, even nonsensical joins between non-join keys. Other work leveraging neural networks to improve sketches typically refine the actual sketches themselves, thus sharing the same restrictions as actual sketches but having improved statistical accuracy. Hsu et al. [14] proposes using recurrent neural networks to classify *heavy hitters* in streams and avoid inserting those elements into the Count-Min sketch — it is known that its error is often due to collision with heavy hitters and that *skimming* [10] these away can improve estimation accuracy. Mitzenmacher [22] uses a neural network classifier as a preliminary filter, allowing a subsequent Bloom filter to veto the neural network's prediction and ensure there are no false negatives. Similarly, although not a sketch, Kraska et al. [19] uses extremely simple neural networks to predict the position of records in databases, effectively replacing indexes.

7 CONCLUSIONS AND FUTURE WORK

We address the challenge of applying query predicates to sketches, using transformers to approximate the sketch of any arbitrary selection. Approximate sketches can estimate the cardinality of any join query, even between non-key columns, with similar performance to actual sketches. By training one model per table and modeling joins via sketches, we avoid necessitating prior knowledge of the database join schema, a common pitfall in *learned* cardinality estimators. Our work focuses on Count-Min and also relate it to Fast-AGMS, but these are not the only sketches that can be approximated. Our method is versatile in being applicable to as many queries as the sketches that we approximate. However, it is limited by the capabilities of the model that approximates the sketch — the larger the sketch to approximate, the larger the model that must be trained. Investigating more efficient models may address this shortcoming. Bidirectional transformers are not the only model capable of estimating conditional probabilities and simpler models may suffice, depending on the complexity of the data. Moving away from deep learning models would also reduce hardware requirements as to no longer require GPU for training and inference.

ACKNOWLEDGMENTS

This research was supported by NSF award number 2008815.

REFERENCES

[1] Noga Alon, Phillip B. Gibbons, Yossi Matias, and Mario Szegedy. 1999. Tracking Join and Self-Join Sizes in Limited Storage. In *Proceedings of the Eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (Philadelphia, Pennsylvania, USA) *(PODS '99)*. Association for Computing Machinery, New York, NY, USA, 10–20. https://doi.org/10.1145/303976.303978

[2] Noga Alon, Yossi Matias, and Mario Szegedy. 1999. The Space Complexity of Approximating the Frequency Moments. *J. Comput. System Sci.* 58, 1 (1999), 137–147. https://doi.org/10.1006/jcss.1997.1545

[3] Burton H. Bloom. 1970. Space/Time Trade-Offs in Hash Coding with Allowable Errors. *Commun. ACM* 13, 7 (jul 1970), 422–426. https://doi.org/10.1145/362686.362692

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[5] Graham Cormode and Minos N. Garofalakis. 2005. Sketching Streams Through the Net: Distributed Approximate Query Tracking. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi (Eds.). ACM, 13–24. http://www.vldb.org/archives/website/2005/program/paper/tue/p13-cormode.pdf

[6] Graham Cormode and S. Muthukrishnan. 2005. An Improved Data Stream Summary: The Count-Min Sketch and Its Applications. *J. Algorithms* 55, 1 (apr 2005), 58–75. https://doi.org/10.1016/j.jalgor.2003.12.001

[7] Fan Deng and Davood Rafiei. 2006. New Estimation Algorithms for Streaming Data: Count-min Can Do More. https://webdocs.cs.ualberta.ca/~drafiei/papers/cmm.pdf

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL]

[9] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. 2007. HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. In *AofA: Analysis of Algorithms (DMTCS Proceedings, Vol. DMTCS Proceedings vol. AH, 2007 Conference on Analysis of Algorithms (AofA 07))*, Philippe Jacquet (Ed.). Discrete Mathematics and Theoretical Computer Science, Juan les Pins, France, 137–156. https://doi.org/10.46298/dmtcs.3545

[10] Sumit Ganguly, Minos Garofalakis, and Rajeev Rastogi. 2004. Processing Data-Stream Join Aggregates Using Skimmed Sketches. In *Advances in Database Technology - EDBT 2004*, Elisa Bertino, Stavros Christodoulakis, Dimitris Plexousakis, Vassilis Christophides, Manolis Koubarakis, Klemens Böhm, and Elena Ferrari (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 569–586.

[11] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. 2002. How to Summarize the Universe: Dynamic Maintenance of Quantiles. In *Proceedings of the 28th International Conference on Very Large Data Bases* (Hong Kong, China) *(VLDB '02)*. VLDB Endowment, 454–465.

[12] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, Zhengping Qian, Jingren Zhou, Jiangneng Li, and Bin Cui. 2021. Cardinality Estimation in DBMS: A Comprehensive Benchmark Evaluation. *Proc. VLDB Endow.* 15, 4 (dec 2021), 752–765. https://doi.org/10.14778/3503585.3503586

[13] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: Learn from Data, Not from Queries! *Proc. VLDB Endow.* 13, 7 (mar 2020), 992–1005. https://doi.org/10.14778/3384345.3384349

[14] C Hsu, P Indyk, D Katabi, and A Vakilian. 2019. Learning-based frequency estimation algorithms. In *Intl Conf. on Learning Representations*. 20 pages. https://doi.org/paper/2019_c_hsu_iclr

[15] IMDB. 2023. Internet Movie Database. https://www.imdb.com/

[16] Yesdaulet Izenov, Asoke Datta, Florin Rusu, and Jun Hyung Shin. 2021. COMPASS: Online Sketch-Based Query Optimization for In-Memory Databases. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 804–816. https://doi.org/10.1145/3448016.3452840

[17] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell,

Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, and Jonathan Ross. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. https://arxiv.org/pdf/1704.04760.pdf

[18] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. 2018. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. arXiv:1809.00677 [cs.DB]

[19] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2017. The Case for Learned Index Structures. *CoRR* abs/1712.01208 (2017). arXiv:1712.01208 http://arxiv.org/abs/1712.01208

[20] Viktor Leis, Bernhard Radke, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. 2018. Query optimization through the looking glass, and what we found running the Join Order Benchmark. *VLDB J.* 27, 5 (2018), 643–668. https://doi.org/10.1007/s00778-017-0480-7

[21] Jie Liu, Wenqian Dong, Qingqing Zhou, and Dong Li. 2021. Fauce: Fast and Accurate Deep Ensembles with Uncertainty for Cardinality Estimation. *Proc. VLDB Endow.* 14, 11 (jul 2021), 1950–1963. https://doi.org/10.14778/3476249.3476254

[22] Michael Mitzenmacher. 2018. A Model for Learned Bloom Filters, and Optimizing by Sandwiching. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 462–471.

[23] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. 2009. Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors. *Proc. VLDB Endow.* 2, 1 (aug 2009), 982–993. https://doi.org/10.14778/1687627.1687738

[24] Parimarjan Negi, Ryan Marcus, Andreas Kipf, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. 2021. Flow-Loss: Learning Cardinality Estimates That Matter. *Proc. VLDB Endow.* 14, 11 (jul 2021), 2019–2032. https://doi.org/10.14778/3476249.3476259

[25] PostgreSQL. 2023. PostgreSQL Documentation 13. https://www.postgresql.org/docs/13/row-estimation-examples.html

[26] Ben Sorscher, Robert Geirhos, Shashank Shekhar, Surya Ganguli, and Ari S. Morcos. 2023. Beyond neural scaling laws: beating power law scaling via data pruning. arXiv:2206.14486 [cs.LG]

[27] Brian Tsan. 2024. Approximate Sketches. https://github.com/Btsan/ApproximateSketch.

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *CoRR* abs/1706.03762 (2017). arXiv:1706.03762

[29] David Vengerov, Andre Cavalheiro Menck, Mohamed Zait, and Sunil P. Chakkappen. 2015. Join Size Estimation Subject to Filter Conditions. *Proc. VLDB Endow.* 8, 12 (aug 2015), 1530–1541. https://doi.org/10.14778/2824032.2824051

[30] Ziniu Wu and Amir Shaikhha. 2020. BayesCard: A Unified Bayesian Framework for Cardinality Estimation. *CoRR* abs/2012.14743 (2020). arXiv:2012.14743 https://arxiv.org/abs/2012.14743

[31] Zongheng Yang, Wei-Lin Chiang, Sifei Luan, Gautam Mittal, Michael Luo, and Ion Stoica. 2022. Balsa: Learning a Query Optimizer Without Expert Demonstrations. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD/PODS '22)*. Association for Computing Machinery, New York, NY, USA, 931–944. https://doi.org/10.1145/3514221.3517885

[32] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2020. NeuroCard: One Cardinality Estimator for All Tables. *Proc. VLDB Endow.* 14, 1 (sep 2020), 61–73. https://doi.org/10.14778/3421424.3421432

[33] 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, 279–292.

[34] Yang You, Jing Li, Sashank J. Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2020. Large Batch Optimization for Deep Learning: Training BERT in 76 minutes. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=Syx4wnEtvH

[35] Rong Zhu, Ziniu Wu, Yuxing Han, Kai Zeng, Andreas Pfadler, Zhengping Qian, Jingren Zhou, and Bin Cui. 2021. FLAT: Fast, Lightweight and Accurate Method for Cardinality Estimation. *Proc. VLDB Endow.* 14, 9 (may 2021), 1489–1502. https://doi.org/10.14778/3461535.3461539

Received July 2023; revised October 2023; accepted November 2023