Convolution and Cross-Correlation of Count Sketches Enables Fast Cardinality Estimation of Multi-Join Queries

MIKE HEDDES, University of California, Irvine, USA
IGOR NUNES, University of California, Irvine, USA
TONY GIVARGIS, University of California, Irvine, USA
ALEX NICOLAU, University of California, Irvine, USA

With the increasing rate of data generated by critical systems, estimating functions on streaming data has become essential. This demand has driven numerous advancements in algorithms designed to efficiently query and analyze one or more data streams while operating under memory constraints. The primary challenge arises from the rapid influx of new items, requiring algorithms that enable efficient incremental processing of streams in order to keep up. A prominent algorithm in this domain is the *AMS sketch*. Originally developed to estimate the second frequency moment of a data stream, it can also estimate the cardinality of the equi-join between two relations. Since then, two important advancements are the *Count sketch*, a method which significantly improves upon the sketch update time, and secondly, an extension of the AMS sketch to accommodate multi-join queries. However, combining the strengths of these methods to maintain sketches for multi-join queries while ensuring fast update times is a non-trivial task, and has remained an open problem for decades as highlighted in the existing literature. In this work, we successfully address this problem by introducing a novel sketching method which has fast updates, even for sketches capable of accurately estimating the cardinality of complex multi-join queries. We prove that our estimator is unbiased and has the same error guarantees as the AMS-based method. Our experimental results confirm the significant improvement in update time complexity, resulting in orders of magnitude faster estimates, with equal or better estimation accuracy.

CCS Concepts: • **Information systems → Query optimization**; • **Theory of computation → Streaming models**; **Sketching and sampling**.

Additional Key Words and Phrases: Cardinality Estimation, Sketching, Synopsis Data Structures

ACM Reference Format:
Mike Heddes, Igor Nunes, Tony Givargis, and Alex Nicolau. 2024. Convolution and Cross-Correlation of Count Sketches Enables Fast Cardinality Estimation of Multi-Join Queries. *Proc. ACM Manag. Data* 2, 3 (SIGMOD), Article 129 (June 2024), 26 pages. https://doi.org/10.1145/3654932

1 INTRODUCTION

The analysis of streaming data has amassed considerable attention, driven by the increasing demand for real-time data processing, and the remarkable advancements in algorithms that enable efficiently querying and analyzing data streams under memory constraints. Streaming data refers to data that is received sequentially and is often too large to be stored in its entirety, hence requiring algorithms that can process the data on-the-fly [5, 24]. Efficiently providing answers to queries over streaming data is vital in numerous application environments, including recommendation systems [12, 13, 28], smart cities [6, 22], network traffic monitoring [18, 20], natural language processing [25], and

Authors' addresses: Mike Heddes, University of California, Irvine, Irvine, CA, USA, mheddes@uci.edu; Igor Nunes, University of California, Irvine, Irvine, CA, USA, igord@uci.edu; Tony Givargis, University of California, Irvine, Irvine, CA, USA, givargis@uci.edu; Alex Nicolau, University of California, Irvine, Irvine, CA, USA, anicolau@uci.edu.

ACM 2836-6573/2024/6-ART129
https://doi.org/10.1145/3654932

analysis of market data in financial systems [9, 40, 44]. Our focus on the streaming data setting stems from its generality. Streaming algorithms are not only effective in streaming settings but also seamlessly extend their applicability to non-streaming scenarios. In this work, we present a novel approach to the problem of estimating a crucial collection of complex queries within the general streaming data framework depicted in Figure 1 and elaborated upon below.

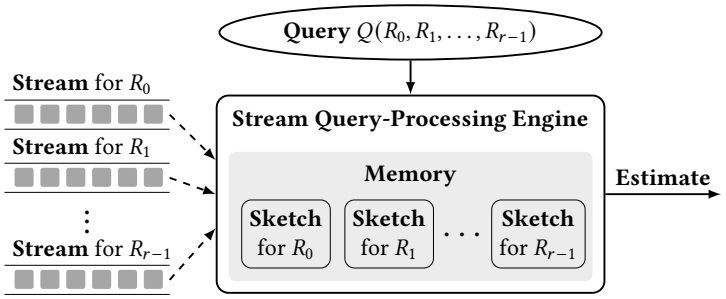


Fig. 1. Streaming query-processing scheme

The rise of data-intensive applications has created a need for data structures that can handle massive volumes of data efficiently. This context motivated the emergence of *synopsis data structures* [23], a family of data structures designed to represent large quantities of data with sublinear space complexity, which is imperative in the given context. Examples include random samples, histograms, wavelets and sketches [17], all of which are actively being researched as a means of analyzing and querying streaming data [1, 15, 17, 32]. These algorithms operate by generating a compressed representation of the original data, which can then be utilized to estimate a specific property or a set thereof. For example, the popular Bloom Filter [7] is widely used for membership testing, while the Count-Min sketch [19] is commonly used for frequency estimation. Both of these methods are examples of sketches. Besides their supported queries, various factors differentiate sketching methods, including sketch size, sketch initialization time, update time, and inference time [14] (see Section 2 for details). These characteristics serve as catalysts for diverse research avenues and are crucial to consider when utilizing or developing a sketching method that is tailored to a specific use case.

Aside from basic statistical properties such as count, sum, and mean, much useful information from a data stream is derived from its frequency distribution, or histogram. This becomes particularly relevant when we need to compare or estimate functions across multiple data sets, such as the number of shared items. Frequency-based sketches are a class of sketching methods specifically designed for estimating functions of the frequency vector. Among these, the *AMS* (Alon-Matias-Szegedy) sketch [4], also known as *Tug-of-War* sketch, stands out as a prime example, renowned for its established reputation of being both simple and remarkably effective in a wide array of applications. The AMS sketch was initially introduced to estimate the second frequency moment of a data stream, but it was later demonstrated to also estimate the cardinality of any equi-join between two relations [3].

Interestingly, it turns out that many important functions on the frequency vector can be expressed as the cardinality of an equi-join. This equivalence is an important driver behind the development of sketches, often seen as an approximate query processing (AQP) technique [32]. One particularly relevant use case is estimating the join cardinality, which is crucial for query optimizers to efficiently assess the cost of candidate physical join plans. The challenge of determining an appropriate join

order is a highly researched problem in the field of databases [11, 30], and the methods employed typically rely on cardinality estimates as the primary input [31].

Two significant breakthroughs emerged a few years after the introduction of the AMS sketch. First, Charikar et al. [10] proposed the *Count sketch*, which divides estimates into "buckets" instead of computing the mean of multiple independent and identically distributed (i.i.d.) estimates. This approach makes the sketch more accurate for skewed data and dramatically speeds up its updates [42, 45]. Second, Dobra et al. [20] proposed a generalization of the AMS sketch that enables the cardinality estimation of multi-join queries, thus considerably expanding the algorithm's applicability.

Although both methods have gained popularity for their respective advantages, the existing literature has highlighted the task of integrating all these benefits into a unified approach as a challenging and unresolved problem [14, 29]. The specific challenge lies in effectively handling multi-join queries with fast updates. This challenge becomes even more significant when considering the prevalence of such multi-joins, as they constitute the majority of queries (see Section 4.1). The difficulty of combining the Count sketch with the AMS-based multi-join query estimation method arises from the use of binning, as we will discuss in Section 3, yet binning is essential for achieving the benefits of the Count sketch.

To address this, we propose a new sketch that combines insights from both Charikar et al. [10] and Dobra et al. [20]. The proposed method relies on the intuitive observation that the operation used to merge single-item AMS sketches to form sketches of tuples, the Hadamard product, is incongruous with the sparse nature of the Count sketch. In essence, when two Count sketches undergo the Hadamard product, the resulting sketch will likely lose information due to the sparsity of the Count sketches.

The core innovation of our approach lies in employing circular convolution instead of the Hadamard product for counting tuples in a data stream. We show that, unlike the Hadamard product, this operation ensures the preservation of information from the operands in the resulting Count sketch. This is complemented by incorporating circular cross-correlation in the estimation procedure. Our method not only exhibits superior estimation accuracy when applied to real data and queries, but also operates within the same memory constraints. Moreover, we have significantly improved the time complexity of the sketch update process, enabling estimates to be computed orders of magnitude faster. We prove that our estimator is unbiased and offers error guarantees equivalent to Dobra et al. [20]. Importantly, our method does not require prior knowledge of the data distribution. Our empirical findings support the practical applicability of the proposed method, underscoring its significant advancement in addressing the aforementioned open problem.

2 BACKGROUND

This section provides the necessary background and introduces the key methodologies and notation used in this work. For an overview of the notation see Table 1. These concepts and methodologies set the foundation for the introduction of our proposed method.

2.1 Streaming data

The focus of this paper is on *streaming data* analysis, a prominent application area for synopsis data structures [23], which involves real-time processing of data that arrives at a high frequency. Streaming data naturally arises in many big data applications, including network traffic monitoring [18, 20], recommendation systems [12, 13, 28], natural language processing [25], smart cities [6, 22], and analysis of market data in financial systems [9, 40, 44]. Algorithms for streaming data are designed to handle data that can only be observed once, in arbitrary order, as it continuously arrives [20].

Table 1. Notation

Symbol	Definition
$[n] = \{0, 1, \dots, n-1\}$	Domain of items
(i, Δ)	Tuple of item and frequency change
$\boldsymbol{f}, \boldsymbol{g} \in \mathbb{R}^n$	Frequency vectors
$\boldsymbol{\Pi} \in \mathbb{R}^{m \times n}$	Random matrix
$\boldsymbol{c} \in \mathbb{R}^m$	Vector of counters, i.e., the sketch
$s_j : [n] \to \{-1, +1\}$	Random sign function
$h_j : [n] \to [m]$	Random bin function
$R_0, R_1, \dots, R_{r-1}$	Database relations
$Q(R_0, R_1, \dots, R_{r-1})$	Query over relations
$u, v \in [w]$	Joined attribute names (vertices)
$\{u, v\} \in E$	Join from all joins (edges)
$u \in \Omega(R_k)$	Joined attribute u of relation R_k
$v \in \Gamma(u)$	Joined attribute v with u
$\Psi(u) \in [w - r + 1]$	Join graph component of attribute u
$I_k = [n] \times \cdots \times [n]$	Domain of relation R_k
$\mathcal{F}_k(i)$	Frequency of tuple i in relation R_k
$X, \mathbb{E}[X]$	Estimate and expected value
ϵ, δ	Error bound and confidence
$y, \hat{y}$	True and predicted cardinality

Consequently, these algorithms must be highly efficient in processing each input, while utilizing limited memory resources, to keep up with the rapid influx of new data.

By presenting our method within the streaming data setting, we establish its applicability to a broad range of scenarios. This is because streaming algorithms are also applicable when multiple data accesses or a specific access order are allowed. Inversely, algorithms that require multiple data accesses or a specific access order, like many learning-based methods, clearly do not apply in a streaming data setting. Even when a streaming algorithm is not strictly necessary, optimizing for fewer data accesses remains advantageous because it minimizes potentially costly I/O operations.

We formulate the problem as follows, based on Cormode and Muthukrishnan [19]: consider a vector $\boldsymbol{f}(t) \in \mathbb{R}^n$, which is assumed too large to be stored explicitly and is therefore presented implicitly in an incremental fashion. Starting as a zero vector, $\boldsymbol{f}(t)$ is updated by a stream of pairs (i_t, Δ_t) which increments the i_t-th element by Δ_t, meaning that $f_{i_t}(t) = f_{i_t}(t-1) + \Delta_t$, while the other dimensions remain unchanged. The *items* i_t are members of the *domain*[1] $[n] = \{0, 1, \dots, n-1\}$; $\Delta_t \in \mathbb{R}$ are the *changes in frequency* and $\boldsymbol{f}(t)$ is called the *frequency vector*. At any time t, a *query* may request the computation of a function on $\boldsymbol{f}(t)$. Specific streaming settings are further classified by their type of updates, as follows:

- **cash-register:** $\Delta_t > 0$ on every update;
- **strict turnstile:** for some updates Δ_t can be negative, but $f_i(t) \geq 0$ for all i and t;
- **general turnstile:** both updates and entries of the vector $\boldsymbol{f}(t)$ can assume negative values at any time $t > 0$.

The algorithms we discuss utilize synopsis data structures to efficiently handle data streams, eliminating the need to explicitly store and compute over $\boldsymbol{f}(t)$ to answer a set of supported queries.

[1]Without loss of generality, we can assume $i \in [n]$ [20, 21].

In the following section, we will introduce a group of sketching techniques known as *linear sketches*. This family of methods, which includes the approach proposed in this paper, is designed to support the most general streaming setting, i.e., the general turnstile.

2.2 Linear sketching

Sketching techniques are a popular set of methods for dealing with streaming data and approximate query processing [14]. Both the baselines and the method proposed in this paper are *linear* sketches, meaning that the summaries they generate can be represented as a *linear transformation* of the input. In contrast, the Bloom filter [7] serves as a classic example of a *non*-linear sketch.

Formally, for a given vector $\boldsymbol{x} \in \mathbb{R}^n$, we define a linear sketch as a vector obtained by $\boldsymbol{\Pi x}$, where $\boldsymbol{\Pi} \in \mathbb{R}^{m \times n}$ is some random matrix, and $m \ll n$. The linearity of the transformation offers notable advantages [14]: it allows for processing items in any order and combining different sketches through addition. This enables efficient handling of data and supports map-reduce style processing of large data streams. In every sketching method, the random matrix is thoughtfully designed to enable the estimation of one or multiple functions over $\boldsymbol{x}$, utilizing only its "summary" captured by $\boldsymbol{\Pi x}$, thereby eliminating the necessity for accessing $\boldsymbol{x}$ itself. When sketching techniques are used in a streaming scenario they are often referred to as *frequency-based sketches*, where the input vector $\boldsymbol{x}$ is the frequency vector $\boldsymbol{f}(t)$ defined in Section 2.1. Hereafter, we will omit the time argument from the frequency vector for brevity.

At this point, one naturally wonders: how can we transform the vector $\boldsymbol{f}$ of size n, which is already considered too large, using a matrix $\boldsymbol{\Pi}$ that is even larger with size mn? Streaming algorithms cleverly represent the matrix $\boldsymbol{\Pi}$ succinctly using hash functions, enabling them to generate just the column of $\boldsymbol{\Pi}$ that is needed to add a given item. Further details on this process will be provided later. Furthermore, it is crucial to ensure that the sketch is efficiently updated as $\boldsymbol{f}$ changes, i.e, as new items stream in. This can be achieved by making $\boldsymbol{\Pi}$ sparse, as we will explore shortly. The effectiveness and versatility of a sketching method primarily relies on the following key properties [14]:

- **Sketch size:** The total number of counters and random seeds required by the sketch, determined by the parameter m.
- **Initialization time:** The time it takes to initialize the sketches. Typically, this involves simply setting a block of memory to zeros, and sampling the random seeds for the hash functions.
- **Update time:** In streaming settings, algorithms must keep pace with the high influx of items. The update time determines the highest item throughput rate that can be sustained.
- **Inference time:** The time it takes to compute an estimate from the generated sketches.
- **Accuracy:** It is crucial to understand the accuracy of an estimator for a given memory budget (limiting the sketch size) and throughput requirement (constraining the update time).
- **Supported queries:** Each sketch is designed to enable estimation of a specific set of functions on the input vector. Typically, a query-specific procedure needs to be performed on the sketch to approximate the value of a particular function.

In the following, we will describe some important sketching methods from the existing literature that have particularly space- and time-efficient ways of representing and computing $\boldsymbol{\Pi f}$.

2.3 AMS sketch

The *AMS* sketch, also referred to as the *Tug-of-War* or *AGMS* sketch, is a pioneering technique for frequency-based sketching that was first introduced by Alon, Matias, and Szegedy (AMS) [4]. The method was originally proposed as a way to estimate the *second frequency moment* F_2 of a data stream, where $F_2 = \|\boldsymbol{f}\|_2^2 = \sum_{i=0}^{n-1} f_i^2$ and $\boldsymbol{f}$ is the frequency vector of the stream as defined

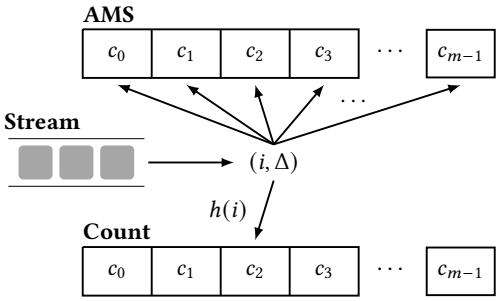


Fig. 2. Comparison of the AMS and Count sketches performing a sketch update for an item in the stream.

in Section 2.1. The AMS sketch is represented by a vector $\mathbf{c}$, containing $m = O(1/\epsilon^2)$ counters c_j, for $j \in [m]$, where $0 < \epsilon < 1$ is the relative error bound. The counters are i.i.d. random variables obtained by $c_j = \sum_{i=0}^{n-1} f_i s_j(i)$, where each $s_j : [n] \to \{-1, +1\}$ is drawn from a family of 4-wise independent hash functions (see Definition 2.1). These hash functions are used to compute the random projection $\mathbf{\Pi} \boldsymbol{f}$ without representing $\mathbf{\Pi}$ explicitly, since $\Pi_{j,i} = s_j(i)$. To establish a confidence level δ, one can take the median of $O(\log 1/\delta)$ independent estimates. The overall method then requires only $O((1/\epsilon^2) \log 1/\delta)$ counters. Taking the median of i.i.d. estimates, sometimes called the "median trick", is universal among sketching methods because it is an effective way to rapidly improve the confidence level of an estimate by the Chernoff Bound [4]. We will, therefore, revisit this concept in the subsequent discussions of other methods.

Definition 2.1 (k-wise independence [37, 48]). *A family of hash functions $H = \{h : [n] \to [m]\}$ is said to be k-wise independent if for any k distinct items $x_0, \ldots, x_{k-1}$ the hashed values $h(x_0), \ldots, h(x_{k-1})$ are independent and uniformly distributed in $[m]$.*

In their original work, Alon et al. [4] showed that $\frac{1}{m}\langle \mathbf{\Pi} \boldsymbol{f}, \mathbf{\Pi} \boldsymbol{f} \rangle$ is an unbiased estimator of F_2. In fact, it can be demonstrated more generally that for any two vectors $\boldsymbol{f}$ and $\boldsymbol{g}$, the normalized inner product of their AMS sketches $\frac{1}{m}\langle \mathbf{\Pi} \boldsymbol{f}, \mathbf{\Pi} \boldsymbol{g} \rangle$ is an unbiased estimator for their inner product $\langle \boldsymbol{f}, \boldsymbol{g} \rangle$ [3]. Notably, when $\boldsymbol{f}$ and $\boldsymbol{g}$ correspond to the frequency vectors of a given attribute of two database relations, this estimated value corresponds to the equi-join size of these relations over that attribute. Theorem 2.1 formally states the expectation, and bounds the variance of the AMS sketch.

Theorem 2.1 (AMS sketch). *For any vectors $\boldsymbol{f}, \boldsymbol{g} \in \mathbb{R}^n$ and a random matrix $\mathbf{\Pi} \in \mathbb{R}^{m \times n}$ constructed by 4-wise independent hash functions $s_j : [n] \to \{-1, +1\}$ for $j \in [m]$ and $\Pi_{j,i} = s_j(i)$, we have:*

$$\mathbb{E}\left[\frac{\langle \mathbf{\Pi} \boldsymbol{f}, \mathbf{\Pi} \boldsymbol{g} \rangle}{m}\right] = \langle \boldsymbol{f}, \boldsymbol{g} \rangle, \quad \text{and} \quad \mathrm{Var}\left(\frac{\langle \mathbf{\Pi} \boldsymbol{f}, \mathbf{\Pi} \boldsymbol{g} \rangle}{m}\right) \leq \frac{2}{m} \|\boldsymbol{f}\|_2^2 \|\boldsymbol{g}\|_2^2$$

Proof. See Lemma 4.4 of Alon et al. [3]. □

2.4 Count sketch

In order to ensure fast sketch updates, i.e., sublinear with respect to its size m, it is desirable for sketching methods to use a *sparse* linear transformation $\mathbf{\Pi}$ when processing input vectors. However, note that the AMS sketch requires changes in all m counters for each update. Consequently, the accuracy of the estimator is constrained by the need to maintain a throughput that corresponds to the rate of incoming items as well as the available memory budget.

The *Count sketch* is another linear sketching method that emerged after AMS and overcomes this limitation by allowing the update time to be independent of the sketch size. It achieves this

by employing a technique called the "hashing trick," [14, 49] which ensures that only one counter per estimate is modified during each update. As a result, the Count sketch improves the update time complexity from $O((1/\epsilon^2)\log 1/\delta)$ to just $O(\log 1/\delta)$, while maintaining not only the same error bounds, but also the same space and inference time complexities as the AMS sketch. The AMS sketch and Count sketch update procedures are compared in Figure 2. Although the Count sketch was originally introduced for the *heavy hitters* problem [10], the same hashing trick can be applied to speed up the AMS sketch, which is commonly known as the *Fast-AMS sketch* [16].

The value of each counter in the Count sketch is given by $c_j = \sum_{i=0:h(i)=j}^{n-1} f_i s(i)$, where $h : [n] \to [m]$ is a random bin function drawn from a family of 2-wise independent hash functions, and $s : [n] \to \{-1, +1\}$ is again a random sign function drawn from a family of 4-wise independent hash functions. The corresponding random matrix $\mathbf{\Pi}$ is specified by $\Pi_{j,i} = s(i)\,\mathbf{1}(h(i) = j)$. Notice that $\mathbf{\Pi}$ has only one non-zero value per column, making it highly sparse. Also, the Count sketch requires only one sign and bin hash function per estimate, regardless of the required precision, resulting in a further reduction in memory usage. An estimate is obtained by taking the inner product between sketches. Theorem 2.2 formally states the expectation and bounds the variance of the Count sketch.

Theorem 2.2 (Count sketch). *For any vectors $\boldsymbol{f}, \boldsymbol{g} \in \mathbb{R}^n$ and a random matrix $\mathbf{\Pi} \in \mathbb{R}^{m \times n}$ constructed by 4-wise independent hash function $s\colon [n] \to \{-1, +1\}$ and 2-wise independent hash function $h\colon [n] \to [m]$ with $\Pi_{j,i} = s(i)\,\mathbf{1}(h(i) = j)$, we have:*

$$\mathbb{E}[\langle \mathbf{\Pi}\boldsymbol{f}, \mathbf{\Pi}\boldsymbol{g}\rangle] = \langle \boldsymbol{f}, \boldsymbol{g}\rangle, \quad \text{and} \quad \mathrm{Var}(\langle \mathbf{\Pi}\boldsymbol{f}, \mathbf{\Pi}\boldsymbol{g}\rangle) \leq \frac{2}{m}\|\boldsymbol{f}\|_2^2\|\boldsymbol{g}\|_2^2$$

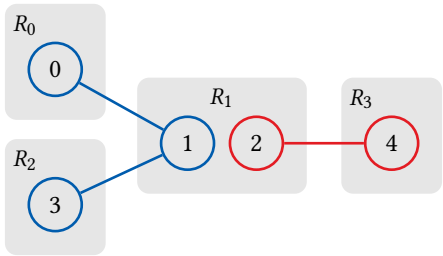
Proof. See Appendix 22 of Weinberger et al. [49] and Lemma 4.4 of Alon et al. [3]. □

The Count sketch has also been shown to outperform the AMS sketch in estimation precision for skewed data distributions [42]. This is because the Count sketch is able to separate out the few high frequency components with high probability. This is an important trait, as it has been widely acknowledged in the literature that the majority of real-world data distributions exhibit a skewed nature [20, 31, 34, 41, 51]. We further discuss this topic in Section 4.1.

2.5 Extensions to the Count sketch

The Count sketch was originally introduced for single-dimensional data which is represented by the frequency vector $\boldsymbol{f}$. More recently, however, several extensions to the Count sketch have been proposed which enable its usage for higher-dimensional data represented by a frequency *tensor* $\boldsymbol{\mathcal{F}}$ instead [33]. The most notable extensions are the *Tensor sketch* [39] and the *Higher-Order Count (HOC) sketch* [43]. Both methods set out to reduce the computational complexity of machine learning applications. The Tensor sketch was used to approximate the polynomial kernel, but finds its origin in estimating matrix multiplication [37]. The HOC sketch was introduced to compress the training data or neural network parameters, in order to speed up training and inference processes.

For each incoming item of the stream, both methods start by encoding all axes separately using independent instances of the Count sketch. They differ in the way these individually sketched axes are combined: the Tensor sketch employs circular convolution (see Definition 2.2), generating a sketch vector, whereas the HOC sketch utilizes the tensor product to produce a sketch tensor of the same order but with reduced dimensions compared to $\boldsymbol{\mathcal{F}}$. The use of the tensor product ensures that axis information about the sketched data is preserved, at the cost of an exponential increase in sketch size with the order of the tensor. The circular convolution, in contrast, preserves the dimensionality of the sketch vector for any tensor order, i.e., it maps tensors to vectors.




```
SELECT COUNT(*) FROM R0, R1, R2, R3
WHERE R0.0 = R1.1 AND R2.3 = R1.1 AND R3.4 = R1.2
```

Fig. 3. Example join graph and corresponding SQL query. Additional attributes in each relation, not involved in the join, are omitted for clarity.

In the context of databases, COMPASS [29] uses HOC sketches to estimate the cardinality of multi-join queries. They additionally propose a method to approximate HOC sketches by merging Count sketches. While they show promising results for query optimization, their estimation method lacks theoretical error guarantees. Moreover, Section 4.2 will show that, in practice, our proposed method achieves significantly higher estimation accuracy. The Tensor sketch (see Definition 2.3) is the most related to our method, however, our method and application are novel and solves an important open problem in the streaming and databases community, as will be detailed in Section 3.

Definition 2.2 (Circular convolution). *The circular convolution $\boldsymbol{x} * \boldsymbol{y}$ of any two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{C}^m$ is a vector with elements given by $(\boldsymbol{x} * \boldsymbol{y})_j = \sum_{i=0}^{m-1} x_i y_{(j-i) \bmod m}$ for all $j \in [m]$.*

Definition 2.3 (Tensor sketch [37, 39]). *Consider any order d tensor $\mathcal{F} \in \mathbb{R}^{n^d}$ and random matrix $\boldsymbol{\Pi} \in \mathbb{R}^{m \times n^d}$ constructed by 2-wise independent hash functions $h_k \colon [n] \to [m]$ and 4-wise independent hash functions $s_k \colon [n] \to \{-1, +1\}$ for each axis $k \in [d]$. Let $\Pi_{j,i} = S(i)\, \mathbf{1}(H(i) = j)$ with the following decomposable hash functions:*

$$H(i) = \left(\sum_{k=0}^{d-1} h_k(i_k) \right) \bmod m, \quad S(i) = \prod_{k=0}^{d-1} s_k(i_k)$$

Then, the Tensor sketch is given by $\boldsymbol{\Pi}\mathcal{F}$.

2.6 Multi-join with AMS sketches

Another significant advancement in linear sketching techniques emerged around the same period as the Count sketch. Dobra et al. [20] proposed a generalization of the AMS sketch that enables the estimation of complex multi-join aggregate queries, such as count and sum queries. These estimates are useful for big data analytics and query optimization [31]. The proposed method addresses the scenario where a query $Q(R_0, R_1, \ldots, R_{r-1})$ involves multiple relations R_k for $k \in [r]$. An example is illustrated in Figure 3, which provides an intuitive visualization of this type of complex query as a disconnected, undirected graph. In this abstraction, each vertex corresponds to an attribute, edges represent the joins between them, and attributes are grouped to form relations.

The technique generates sketches for each relation by iterating over all the *tuples* i in each relation once. This iterative traversal of the tuples aligns with the streaming data scenario described in Section 2.1, enabling the sketches to be created on-the-fly as the relations are updated. The

sketch $\boldsymbol{c}_k$ for relation R_k is given by:

$$\mathrm{c}_{k,j} = \sum_{i \in I_k} \mathcal{F}_k(i) \prod_{u \in \Omega(R_k)} \prod_{v \in \Gamma(u)} s_{j,\{u,v\}}(i_u) \tag{1}$$

where we use the following notation: i represents a tuple that belongs to the domain I_k of relation R_k; I_k is the cross product of item domains $[n] \times \cdots \times [n]$ for each joined attribute of relation R_k; $\mathcal{F}_k(i)$ gives the frequency of tuple i in relation R_k; i_u denotes the value in tuple i for attribute u; u is an attribute from the set of joined attributes of R_k in the query, denoted as $\Omega(R_k)$ (for example, $\Omega(R_1) = \{1, 2\}$ in Figure 3); v is an attribute from the set of attributes joined with u, denoted as $\Gamma(u)$ (for example, $\Gamma(1) = \{0, 3\}$ in Figure 3). We represent a join between two attributes with $\{u, v\}$. Both $u, v \in [w]$ are from the set of all joined attributes. Our notation assumes that all attributes are globally unique, which can easily be achieved in practice, for instance, by concatenating the relation and attribute names. Moreover, following Dobra et al. [20], we assume that joins are non-cyclic, a self-join is thus represented as a join with a fictitious copy of the relation. It is worth noting that the copy does not need to be physically created, this is done solely to simplify the notation. Note also that the functions Γ and Ω are defined for a specific query Q. We omit this dependence in the notation for brevity, as it is evident from their definitions.

Once the sketches are created, a query estimate is derived by performing the element-wise multiplication of the sketches, often referred to as the *Hadamard product* of sketches. This is followed by calculating the mean over the counters. Formally, this can be expressed as: $X = \frac{1}{m} \sum_{j=0}^{m-1} \prod_{k=0}^{r-1} c_{k,j}$, where X is an unbiased estimate of the cardinality of query Q. The expectation and variance of X are formally stated in Theorem 2.3.

Theorem 2.3. *Given an acyclic query of relations R_k for $k \in [r]$, let Equation 1 provide the sketches for each relation and $X = \frac{1}{m} \sum_{j=0}^{m-1} \prod_{k=0}^{r-1} c_{k,j}$ the cardinality estimate of the query, then we have:*

$$\mathbb{E}[X] = \sum_{i \in I_0 \times \cdots \times I_{r-1}} \mathcal{F}_0(i) \cdots \mathcal{F}_{r-1}(i) \prod_{\{u,v\} \in E} \mathbf{1}(i_u = i_v)$$

$$\mathrm{Var}(X) \leq \frac{1}{m} 3^{r-1} \prod_{k=0}^{r-1} \|\boldsymbol{\mathcal{F}}_k\|_2^2$$

Proof. In Lemmas 3.1 and 3.2 of Dobra et al. [20] similar results are presented, albeit with a slightly looser bound on the variance. However, we were unable to locate the proof for their claims. Therefore, we provide the proof for the presented theorem in Appendix A for the sake of completeness. □

From the perspective of the Count sketch extensions discussed in Section 2.5, this method can be interpreted as a similar generalization but for the AMS sketch. It can thus be seen as one of the first methods to generalize sketching for tensor data, although this aspect was not explicitly mentioned in the original work.

2.7 Other related work

In this section, we discuss other recent work in cardinality estimation. The Pessimistic Estimator [8] is an interesting sketching technique which provides an upper bound of the cardinality. They show that it improves upon the cardinality estimator within PostgreSQL. However, the practical use of the method is limited due to its lengthy estimation time [26], at times exceeding the query execution time, as also mentioned by the authors.

Since the inception of sketching, a number of techniques have been proposed that complement the aforementioned sketches. Among these techniques, the Augmented Sketch [41] and the JoinSketch [47] aim to improve the accuracy of sketches for skewed data by separating the high- from the low-frequency items in the sketch. The counters of the high-frequency items are explicitly represented in an additional data structure, thereby preventing them from causing high estimation error due to hash collisions. Another notable technique is the Pyramid sketch [51], which employs a specialized data structure that dynamically adjusts the number of allocated bits for each counter, preventing overflows in the case of high-frequency items. It is important to note that these techniques are proposed as complementary tools, compatible with a variety of sketching methods, including the one proposed in this work.

Recently, there has been a parallel effort aimed at harnessing the power of machine learning for cardinality estimation. Among the various approaches, the most promising ones are data-driven methods that build query-independent models to estimate the joint probability of tuples [26]. Notable examples of such techniques include DeepDB [27], BayesCard [50], NeuroCard [52], and FLAT [53]. While machine learning-based cardinality estimation techniques have been receiving increasing attention, they still face important limitations: these methods are presently viable only in scenarios where supervised training is feasible, their accuracy has not consistently lived up to expectations [36], and they prove impractical in situations with frequent data updates, such as streaming settings, due to their high cost of model updates [26]. In Section 4.4, we demonstrate that our proposed method not only avoids these performance limitations but also achieves significantly higher accuracy compared to the machine learning techniques.

3 METHOD

In this section, we present our method to solve the longstanding challenge of integrating the key advantages of the Count sketch, such as its efficient update mechanism and superior accuracy when handling skewed data, into a method that effectively estimates the cardinality of multi-join queries. Devising such a method is recognized as a challenging task, as underscored not only by the author of the Count-Min sketch [14] but also by other recent work in the field [29]. This acknowledgment highlights the importance of our contribution. The importance of solving this problem is further underscored when considering the prevalence of multi-join queries. In fact, an analysis of two sets of widely recognized benchmarking queries, as discussed in Section 4.1, reveals that multi-joins constitute approximately 97% of all their queries [26, 31].

3.1 Key insight for preserving information

We start with an intuitive discussion on the main insight behind the proposed method, using the example illustrated in Figure 4. The main challenge of combining the Count sketch with the method proposed by Dobra et al. [20] lies in the inability to effectively merge Count sketches using the Hadamard product. Dobra et al. [20] create the sketch for a tuple as the Hadamard product of the AMS sketches for each value in the tuple. As discussed earlier, the advantages of the Count sketch stem from its sparsity. However, as illustrated in the left part of the figure, it is precisely this characteristic that results in a loss of information, with high probability, when combining sketches with the Hadamard product. Essentially, due to the sparsity, the non-zero entry in each sketch is highly likely to appear at a different position, causing the result of the element-wise multiplication to yield a zero vector, devoid of information.

To address this issue, the core concept behind our method, as depicted in the right part of Figure 4, involves the utilization of circular convolution paired with circular cross-correlation (see Definitions 2.2 and 3.1) during inference. With circular convolution, the resulting sketch has the product of the non-zero entries in the bin that corresponds to the sum of the non-zero indices, modulo m.

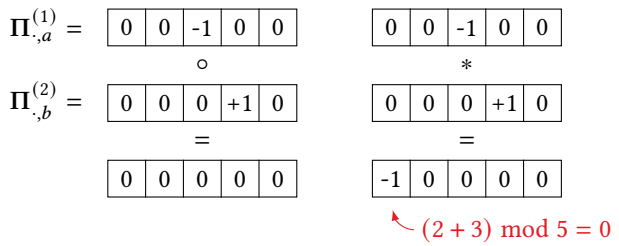


Fig. 4. Comparison of the Hadamard product (left) and circular convolution (right) on two single-item Count Sketches. The resulting sketch represents the 2-tuple (a, b).

Unlike the Hadamard product, this operation guarantees that the information is preserved. We will delve deeper into this intuition and formalize it in the subsequent sections. Crucially, the circular convolution of single-item Count sketches can be computed in $O(1)$ time, with respect to the sketch size m. This means that the sketch of a stream can be updated in constant time for each arriving tuple, in contrast to the $O(m)$ time required by the AMS sketch with the Hadamard product. As we will show empirically in Section 4.3, this translates to sketch updates that are orders of magnitude faster.

Definition 3.1 (Circular cross-correlation). *The circular cross-correlation $\boldsymbol{x} \star \boldsymbol{y}$ of any two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{C}^m$ is a vector with elements given by $(\boldsymbol{x} \star \boldsymbol{y})_j = \sum_{i=0}^{m-1} \overline{x}_i y_{(j+i) \bmod m}$ for all $j \in [m]$, where $\overline{x}_i$ denotes the complex conjugate of x_i.*

3.2 General formulation by example

In order to facilitate a better understanding of the proposed method, we begin by demonstrating it through an illustrative example query. By showcasing the estimation process, we aim to provide valuable insights into the inner workings of our method. Following this, we formally present the method through its pseudocode. Furthermore, we prove that our method is an unbiased cardinality estimator for the previously defined family of multi-join queries and provide bounds on the estimation error. A general overview of our estimation procedure is provided in Algorithm 1. In the following description, we shall use the example query from Figure 3.

Algorithm 1 General estimation procedure

Input: Relations R_k for $k \in [r]$, query $Q(R_0, R_1, \ldots, R_{r-1})$, and sketch size m.
Output: Estimate X

```
s ← SampleSignHashes(Q, m)
h ← SampleBinHashes(Q, m)
for each relation R_k do
    c_k ← CreateSketch(R_k, s, h, Q, m)
end for
X ← GetQueryEstimate(c_0, c_1, ..., c_{r-1}, Q, m)
```

Initialization. Our method starts by initializing the necessary hash functions and counters. We sample an independent random *sign function* $s_{\{u,v\}} \colon [n] \to \{-1, +1\}$, drawn from a family of 4-wise independent hash functions for every join $\{u, v\} \in E$, represented by an edge in Figure 3. Moreover, each graph component is assigned an independent random *bin function* $h_{\Psi(u)} \colon [n] \to [m]$, drawn

from a family of 2-wise independent hash functions. Here, $\Psi(u)$ represents the graph component to which attribute u belongs. A graph component comprises a set of attributes connected by joins, forming a subgraph that is not part of any larger connected subgraph. In the example, there are two graph components: $\{0, 1, 3\}$ and $\{2, 4\}$, identified by the edge colors in Figure 3. A bin function is shared within a graph component because all the attributes that form a graph component must, by definition, be joined on equal values. Note that the equal values are mapped to the same bin by using the same bin function. Lastly, for each relation, a zero vector of m counters is initialized.

Sketching. Once the counters and hash functions are initialized, the tuples from each relation stream are processed. When a tuple streams in, it is mapped to a single sign and bin, derived from the signs and bins of the joined attributes. To determine the sign of a tuple, all the signs of the joined attributes are multiplied together. For instance, tuple i from relation R_1 is hashed as follows: $s_{\{1,3\}}(i_1)s_{\{1,0\}}(i_1)s_{\{2,4\}}(i_2)$, where i_u is the value for attribute u, and $\{u, v\}$ denotes the join between attributes u and v. This is because R_1 has two joined attributes, and one of those is joined twice. Formally, the sign of a tuple i from relation R_k is given by:

$$S_k(i) = \prod_{u \in \Omega(R_k)} \prod_{v \in \Gamma(u)} s_{\{u,v\}}(i_u) \tag{2}$$

To determine the bin of the tuple, the bin indices of all the joined attributes are summed, followed by taking the modulo m. Continuing our example, we have: $(h_{\Psi(1)}(i_1) + h_{\Psi(2)}(i_2)) \bmod m$ because R_1 has two joined attributes. The bin index of a tuple i from relation R_k is formally given by:

$$H_k(i) = \left(\sum_{u \in \Omega(R_k)} h_{\Psi(u)}(i_u) \right) \bmod m \tag{3}$$

Subsequently, the sign of the tuple multiplied by the change of frequency is added to the counter at the bin index of the tuple.

The sketching process for a tuple is equivalent to circular convolution between the sketches for each joined attribute value in the tuple. Since the individual sketches have only one non-zero value, the result of the circular convolution also has one non-zero value, which can be computed in constant time with respect to the sketch size, as explained earlier. The pseudocode for the general sketch creation procedure is provided in Algorithm 2. The sketch $\boldsymbol{c}_k$ for relation R_k is formally stated as follows:

$$c_{k,j} = \sum_{i \in I_k \,:\, H_k(i) = j} \mathcal{F}_k(i) S_k(i) \tag{4}$$

where $\mathcal{F}_k(i)$ denotes the frequency of tuple i in relation R_k. The tuple i is from the domain $I_k = [n] \times \cdots \times [n]$ of relation R_k.

Inference. Upon creating the sketches for each relation, we can proceed to estimate the query's cardinality by combining sketches using either the Hadamard product or circular cross-correlation. The computation consists of summations over the sketch size for each graph component. The sketch for each relation is indexed by the sums of the graph components that have an attribute in that relation. Sketches of relations with multiple joined attributes will, therefore, also have multiple indices, and these are summed to obtain the final index. The sketch values inside the sums are all multiplied. An estimate of the example query is then obtained as follows: $\sum_{j_0=0}^{m-1} \sum_{j_1=0}^{m-1} c_{0,j_0} c_{2,j_0} c_{1,(j_0+j_1) \bmod m} c_{3,j_1}$ because there are two graph components, and R_1 is part of both. This can be factorized as the Hadamard product between sketches $\boldsymbol{c}_0$ and $\boldsymbol{c}_2$ whose result is circular cross-correlated with $\boldsymbol{c}_1$, followed by another Hadamard product with $\boldsymbol{c}_3$, and lastly a summation over the elements. A

Algorithm 2 Sketch creation procedure

```
1:  function CreateSketch(R_k, s, h, Q, m)
2:      c_k ← 0                                          ▷ Size: m
3:      for each tuple (i, Δ) in R_k do                  ▷ The stream of tuples
4:          x = 1                                        ▷ For accumulation of signs
5:          j = 0                                        ▷ For accumulation of bins
6:          for each attribute u in Ω(R_k) do
7:              j ← j + h_{Ψ(u)}(i_u)
8:              for each attribute v in Γ(u) do
9:                  x ← s_{{u,v}}(i_u) x
10:             end for
11:         end for
12:         j ← j mod m
13:         c_{k,j} ← c_{k,j} + xΔ
14:     end for
15:     return sketch c_k
16: end function
```

cardinality estimate X is formally obtained as follows:

$$X = \sum_{j \in J} \prod_{k=0}^{r-1} c_{k,G_k(j)}, \quad \text{with} \quad G_k(j) = \left(\sum_{u \in \Omega(R_k)} j_{\Psi(u)} \right) \bmod m \tag{5}$$

where J is the cross product of bin domains $[m] \times \cdots \times [m]$ for each graph component.

Computing the estimates naively using Equation 5 has an exponential time complexity with the number of graph components. However, this can be improved significantly by factorizing the problem. We can then rely on the fact that circular cross-correlation can be computed efficiently in $O(m \log m)$ time using the fast Fourier transform (FFT). To perform this efficient estimation process methodically, one starts by selecting any joined attribute, say attribute 4 in our example. The process now aggregates all the sketches towards attribute 4 to obtain the estimate. This is implemented as a depth first traversal of the join graph with attribute 4 as the root node of a rooted tree and attributes 0 and 3 as the leaves. The general procedure for combining sketches to efficiently compute an estimate of the query is provided in Algorithm 3. In the pseudocode, we ensure that the recursion only moves away from any selected root attribute $o \in [w]$ by keeping track of the visited nodes V. The functions (I)FFT denote the (inverse) fast Fourier transform. This procedure reduces the inference time complexity to $O(rm \log m)$, that is, nearly linear with respect to the sketch size.

3.3 Analysis

Now that we have discussed the estimation procedure, we present a theoretical analysis of the proposed method. We show that it is an unbiased estimator for the cardinality of multi-join queries in Theorem 3.1, and provide guarantees on the estimation error. Lastly, we provide the time complexity for each estimation stage.

Algorithm 3 Estimation procedure

```
function GetQueryEstimate(c_0, c_1, ..., c_{r-1}, Q, m)
    o ← AnyJoinedAttribute(Q)                          ▷ The root attribute
    V ← {}                                             ▷ Global set of visited attributes
    X ← SumElements(CombineSketches(o, V, m))
    return estimate X
end function
function CombineSketches(u, V, m)
    R_k ← RelationOf(u)
    x ← c_k                                            ▷ Sketch of relation R_k
    add(V, u)                                          ▷ Adds attribute u to the visited set
    ▷ Recurse through the other attributes in the relation.
    for each attribute u' in Ω(R_k) \ {u} do
        add(V, u')
        a ← 1                                          ▷ Size: m
        ▷ By the definition of Ω there is at least one iteration.
        for each attribute v in Γ(u') do
            a ← CombineSketches(v, V, m) ∘ a
        end for
        ▷ Efficient circular cross-correlation
        x ← IFFT(conj(FFT(a)) ∘ FFT(x))
    end for
    ▷ Recurse over the attributes joined with the current.
    for each attribute v in Γ(u) \ V do
        x ← CombineSketches(v, V, m) ∘ x
    end for
    return intermediate sketch x
end function
```

Theorem 3.1. *Given an acyclic query of relations R_k for $k \in [r]$, let Equation 4 provide the sketch for each relation and Equation 5 the cardinality estimate X of the query, then we have:*

$$\mathbb{E}[X] = \sum_{i \in I_0 \times \cdots \times I_{r-1}} \mathcal{F}_0(i) \cdots \mathcal{F}_{r-1}(i) \prod_{\{u,v\} \in E} \mathbf{1}(i_u = i_v)$$

$$\mathrm{Var}(X) \leq \frac{1}{m} 3^{r-1} \prod_{k=0}^{r-1} \|\mathcal{F}_k\|_2^2$$

Proof. We present the proof in Appendix A. □

Using the Chebyshev inequality and the upper bound on the variance from Theorem 3.1, we can bound the absolute estimation error by $\epsilon > 0$ with $m \geq 3^r \epsilon^{-2} \prod_{k=0}^{r-1} \|\mathcal{F}_k\|_2^2$. Furthermore, to guarantee the error with probability at most $1 - \delta$, one selects the median of $l = O(\log 1/\delta)$ i.i.d. estimates by the Chernoff bound [4]. The exponential term 3^r indicates that accurately estimating queries involving many relations quickly becomes infeasible. It remains an open problem whether this exponential dependence can be improved. However, as we will show in the following section, for moderate sized queries, involving up to 6 relations, our estimation accuracy constitutes a significant improvement over the baselines.

Table 2. Comparison of time complexity by stage

Method	Initialization	Update	Inference
AMS	$O(rlm)$	$O(rlm)$	$O(rlm)$
COMPASS (partition)	$O(lm^r)$	$O(rl)$	$O(lm^r)$
COMPASS (merge)	$O(rlm)$	$O(rl)$	$O(rlm^r)$
Ours	$O(rlm)$	$O(rl)$	$O(rlm \log m)$

In Table 2, we present the time complexity of each estimation stage, comparing our method with the AMS-based technique by Dobra et al. [20] and the two variations of COMPASS (partition and merge) [29]. The symbols r, l, and m denote the number of relations, medians, and the sketch size, respectively. The update time of our method for each incoming tuple is remarkably efficient, with a time complexity of only $O(r \log 1/\delta)$. The efficient update time complexity, independent of the estimation error ϵ, enables the sketching of high-throughput streams even when requiring a high level of accuracy. While COMPASS also has fast updates, its exponential dependence on r during inference limits its practical use even for moderately sized queries. Our method achieves fast updates yet introduces only an additional $\log m$ term during the inference stage, compared to the AMS baseline. This slight increase in inference time is negligible when considering the substantial improvement in update time. For instance, our experiments go up to $m = 10^6$ with $l = 5$, which means that our method achieves roughly 10^6 times faster sketch updates, while having only $\log_2(10^6) \approx 20$ times slower inference. As a result, our method effectively minimizes the overall estimation time in various crucial scenarios. These claims are further supported by our empirical results, which are detailed in Section 4.

3.4 Integration with query optimizers

The quintessential application of our proposed method is the cardinality estimator within query optimizers. Query optimizers use a plan enumeration algorithm to find a good join order. The cost of a join order dependents upon the sizes of the intermediate results. The cardinality estimator's role is to provide the estimates for these intermediate sizes [30]. Each intermediate cardinality can be expressed as a sub-query which our method can estimate. The sketches for all the evaluated sub-queries can be created in a single pass over the data. Typically, each sub-query requires its own sketches; however, in cases where an attribute is joined multiple times, the sketches can be reused for each join involving that attribute. For example, to decide the join order of joins $\{0, 1\}$ and $\{1, 3\}$ in Figure 3, the sketch for attribute 1 can be reused for attributes 0 and 3. In Section 4.5, we demonstrate the improvement in query execution time after integrating our proposed cardinality estimator into the query optimizer of PostgreSQL.

4 EXPERIMENTS

In this section, we conduct an empirical analysis to evaluate the effectiveness of our estimator and compare it to various baseline approaches. These baselines include the AMS-based method proposed by Dobra et al. [20] and the two variations of COMPASS [29], namely partition and merge. Our primary objective is to assess the accuracy of our estimator against the baselines for a specified memory budget. Furthermore, we compare the initialization, sketching, and inference times of our method with those of the baselines. Secondly, we compare the estimation error and execution time of our method with the four data-driven machine learning techniques discussed in Section 2.7:

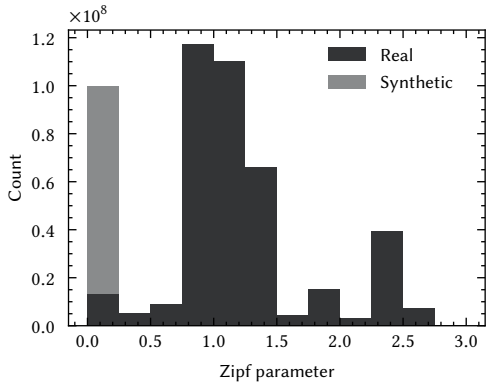


Fig. 5. Total number of entries across all columns of both the STATS and IMDB databases, grouped by the best fit Zipf parameter of each column. Synthetic entries refer to the id and md5sum columns, which are unique by design, the real entries include all other columns.

DeepDB [27], BayesCard [50], NeuroCard [52], and FLAT [53]. Finally, we evaluate the impact of our cardinality estimator on the query execution time of PostgreSQL.

We implemented both the sketching baselines and our method using the PyTorch tensor library [38]. The hash functions are implemented using efficient random polynomials over a Mersenne prime field [2]. All experiments were conducted on an internal cluster of Intel Xeon Gold 6148 CPUs. Each experiment utilized a single CPU and 24 GB of memory. The source code for the experiments, extended results, and cardinality estimates are available online[2].

4.1 Databases and queries

The limitations of synthetic databases in accurately reflecting real-world performance have been widely acknowledged in the literature [26, 31]. To address this concern, our experiments were conducted using two established benchmarking databases containing real data: the IMDB database [31], which encompasses information on movies, actors, and their associated production companies, and the STATS database [26], which comprises user-contributed content from the Stats Stack Exchange network. To provide insights into the characteristics of the databases, Table 3 presents statistics detailing their sizes. Additionally, Figure 5 showcases the distribution skewness of the database entries, highlighting the significant skewness often observed in real data [20, 31, 34, 41, 51]. Our experimentation covers all the 146 STATS-CEB and 70 JOB-light queries, in addition to the 3299 associated sub-queries from the cardinality estimation benchmark [26]. These queries collectively represent a diverse range of real-world workloads.

Table 3. Database size statistics

Database	Relations	Tuples	Storage size
IMDB	21	74.2M	3.88 GB
STATS	8	1.03M	39.6 MB

The key feature of our proposed method is its ability to efficiently estimate multi-join queries. As previously mentioned, our motivation for this capability is rooted in the prevalence of such queries

[2]Source code: https://github.com/mikeheddes/fast-multi-join-sketch

in real-world scenarios. To further substantiate this motivation, we conducted an analysis of all the queries in both the cardinality estimation benchmark [26] and the join order benchmark [31]. The results, displayed in Table 4, indicate that 44% of the relations in the queries are involved in multiple joins, with single joins being the most common at 57%. Notably, 97% of the queries contain at least one relation which participates in multiple joins. These statistics underscore the significance of supporting multi-join queries to effectively address the majority of real-world query scenarios.

Table 4. Percentage of relations among all queries by their number of joins, and the percentage of queries by their relation with the maximum number of joins.

Joins	1	2	3	4	5+
Relations	57%	12%	9%	9%	12%
Queries	3%	24%	30%	20%	23%

Following Izenov et al. [29], in the experiments the filter predicates of each query are processed during ingestion of the tuples from their respective relation streams. In many streaming algorithms, the query is assumed to be known in advance of the stream. Consequently, filtering the tuples at the time of ingestion offers advantages in terms of performance and accuracy. This approach eliminates the need to update the sketch for tuples that do not satisfy the filters. In addition, the estimation accuracy is significantly improved as some data is already filtered out from the sketches [29]. However, it is worth mentioning that sketching methods, including the one presented, are also capable of handling filter predicates during inference. This can be achieved by treating the filters as joins with imaginary tables, a technique employed, for example, by Cormode [14] and Vengerov et al. [46]. Lastly, in our experiments, we report the median of $l = 5$ i.i.d. estimates as the cardinality estimate for all sketching methods.

4.2 Estimation accuracy

We first compare the estimation accuracy of the different sketching methods in terms of the absolute relative error, defined as $|y-\hat{y}|$ divided by $\max(y, 1)$, where y is the true cardinality and $\hat{y}$ its estimate. This metric aligns with the formulation of the theoretical error bound outlined in Section 3.3. In Figure 6, we present the median and the 95th percentile of the error at varying sketch sizes. The statistics are derived from 30 repetitions, each with distinct random initializations. The results are presented for a representative subset of the queries, necessitated by space constraints, but detailed results for all 216 queries can be found online[2].

It is important to note that the experiments for AMS and COMPASS (merge) do not extend to the highest memory usage levels. This limitation arises due to the extensive time required to run the AMS-based experiments, which increases exponentially with each additional data point, rendering their execution quickly infeasible. Additionally, COMPASS (merge) encountered memory constraints during the inference stage, as its memory demand grows exponentially with the number of joins. Notice that the depicted memory usage excludes intermediate representations during inference, thus understating the actual memory required for COMPASS (merge).

Upon analysing the results, we observe that the proposed method delivers comparable or lower error rates across all queries, often demonstrating orders of magnitude greater accuracy. The proposed method achieves zero error on many queries with large sketches. In realistic sketching applications, a margin of error is acceptable; therefore, sketches ranging from 1 to 10 MB could be employed for the STATS-CEB and JOB-light queries. For certain queries, like JOB-light 37 and

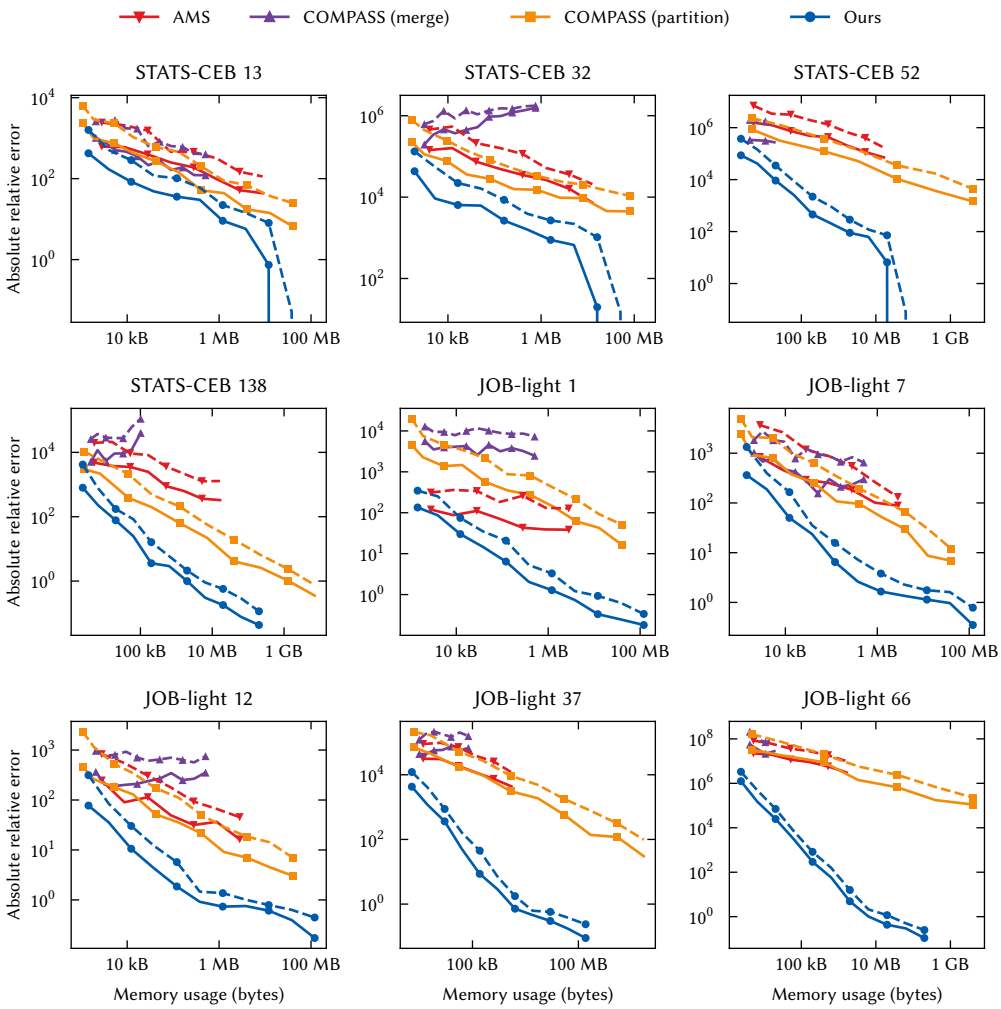


Fig. 6. Absolute relative error of our method compared to the baselines at varying sketch sizes. Solid lines represent the median error and dashed lines denote the 95th percentile. The memory usage includes the counters of the sketches and the random seeds for the hash functions, but excludes the space needed for the intermediate inference calculations.

66, our method not only achieves significantly lower error but also demonstrates a more rapid reduction in error with each increase in memory.

To assess the rate at which our method improves the estimation error compared to the baselines, Figure 7 presents kernel density estimates of the slopes derived from the absolute relative error results for all 216 queries. These slopes are obtained by least-squares linear regression of the data for each method and query in log-log space. This means that the slope represents the exponent of a power law relationship, where the error at memory usage m is given by am^k, with k as the slope and a as the error at $m = 1$.

Remarkably, our method exhibits a significantly faster reduction in error, highlighting its ability to achieve high accuracy with substantially less memory compared to the baselines. Considering

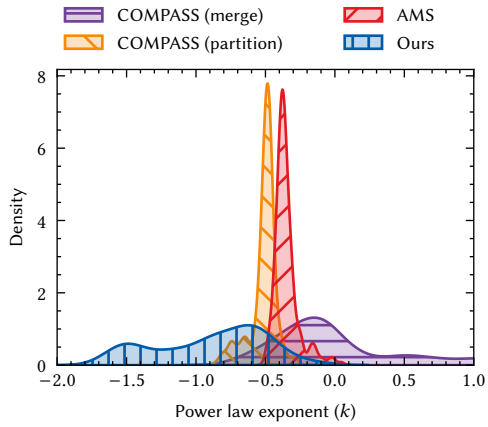


Fig. 7. Kernel density estimates of the power law exponents from the absolute relative error plots for all 216 queries. A higher concentration on the left signifies a greater improvement in accuracy as the memory budget increases.

that the real data in our experiments is skewed, as indicated in Figure 5, we speculate that our method successfully inherits and expands upon the advantages associated with the Count sketch, particularly its effectiveness in handling skewed data. In the context of multi-joins, our method capitalizes on these benefits, demonstrating its ability to compute accurate cardinality estimates.

4.3 Execution times

In the second set of experiments, we look into the execution time of our method and how it compares to the baselines across varying sketch sizes. In Figure 8, we present the execution times for each stage of the estimation process: initialization, sketching, and inference. The figure shows the best fit of a Gaussian process regression to the experimental results from all queries, totaling 232,323 experiments. It also contains data for the learning-based methods which will be discussed in the following section. The individual timing results for all queries are provided online[2].

While the initialization time exhibits a similar trend for all methods, in sketching time there is a notable disparity between AMS and the other methods. This disparity directly reflects the difference in update time complexity. The methods also differ in initialization time complexity, but this is not visible in the timing results because they are plotted with respect to their memory usage rather than the sketch size. That is, for a given sketch size COMPASS (partition) allocates more memory, but for a given memory budget, all methods have similar initialization time. Among the inference times, our method demonstrates remarkable overall efficiency. Even for the most complex queries with the largest sketch size, our method computes its estimate within ten seconds. In contrast, the AMS-based method requires hours to compute estimates, with the majority of that time spent on sketching, all while delivering higher error rates.

To further validate the fast update time of our method, we assessed the maximum stream throughput for all baselines, as outlined in Table 5. The memory usage includes the counters of the sketches and the random seeds for the hash functions. The results were obtained by performing linear least-squares regression on the throughput measurements for each method on all queries. For smaller sketch sizes, the AMS-based method can achieve a throughput similar to the Count sketch-based approaches. However, when working with larger sketch sizes required for high estimation

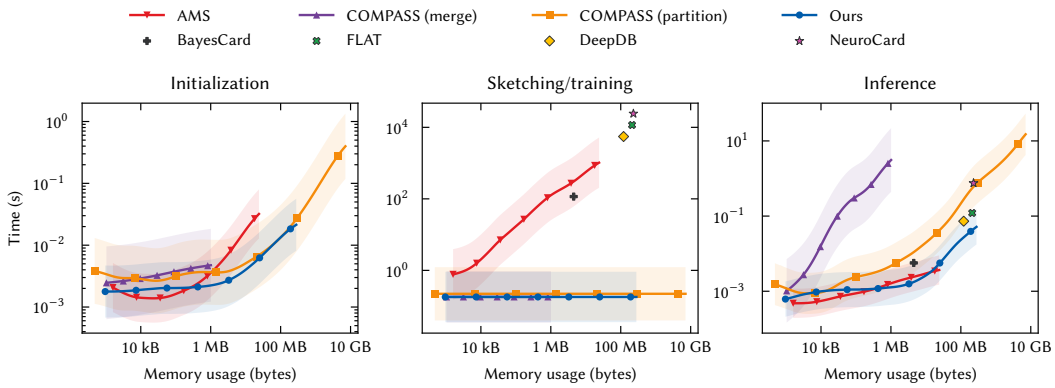


Fig. 8. Execution times for the three stages (initialization, sketching/training, and inference) of the baselines and our method at varying sketch/model sizes. The plots show the best fit of a Gaussian process regression of the data in log-log space, along with standard deviation. The memory usage includes the counters of the sketches and the random seeds for the hash functions. The data for BayesCard [50], FLAT [53], DeepDB [27], and NeuroCard [52] is obtained from Han et al. [26].

accuracy, the AMS-based method becomes limited to handling just a few hundred tuples per second. This significant limitation severely restricts the practical usability of AMS in streaming scenarios.

Table 5. Throughput in tuples processed per second

Memory usage	1 kB	10 kB	100 kB	1 MB	10 MB
AMS	5.2M	576k	63.5k	7.0k	774
COMPASS (partition)	5.9M	5.9M	5.9M	5.9M	5.9M
COMPASS (merge)	6.3M	6.2M	6.0M	5.8M	5.6M
Ours	7.0M	6.8M	6.6M	6.5M	6.3M

4.4 Comparison with learning-based methods

In this set of experiments, we compare the cardinality estimation performance of our proposed method with the four data-driven machine learning techniques discussed in Section 2.7. Specifically, we compare their execution time and estimation quality. To evaluate the quality of cardinality estimates, we employ the q-error metric, defined as $\max(y/\hat{y}, \hat{y}/y)$ if $\hat{y} > 0$ and ∞ otherwise [35]. Figure 9 presents the cumulative distribution function of the q-error for all 3299 sub-queries, showing the fraction of queries that were estimated within a certain q-error. Our method was configured with $m = 1{,}000{,}000$ bins, resulting in an average estimation time of 0.30 seconds and consuming an average of 137 MB of memory. To maintain consistency with the results of the learning methods, as obtained from the cardinality estimation benchmark [26], our method estimated the cardinality of each sub-query only once. We provide our cardinality estimates for the sub-queries together with the source code[2] to facilitate further comparisons with the proposed method and to ensure reproducibility.

The results presented in Figure 9 highlight the superior performance of the proposed method, which delivers error-free estimates for approximately 70% of the sub-queries. Furthermore, our method achieves q-error values of less than 2 for about 95% of the sub-queries. In contrast, even

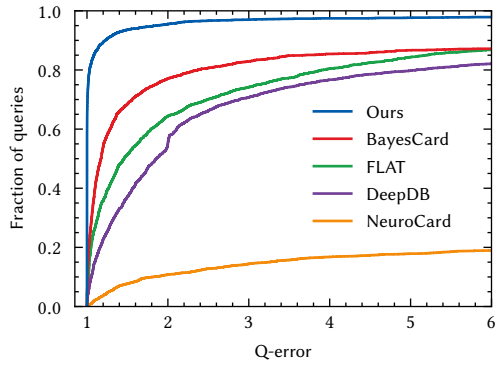


Fig. 9. Cumulative distribution function of the q-error for the STATS-CEB and JOB-light sub-queries. The legend follows the ordering of the lines in the figure.

the best-performing learning-based method, BayesCard, achieves this level of accuracy for less than 80% of the sub-queries. These findings demonstrate the remarkable estimation accuracy of our proposed estimator. Moreover, our sketching approach provides theoretical guarantees on the estimation error which are lacking for the learning-based methods.

Our proposed method is also notably efficient when compared to the learning-based methods. As depicted in Figure 8, the training phase of the learning-based methods requires 3 to 5 orders of magnitude more time than creating our sketches. In addition, Table 5 shows that our method can handle over 6 million updates per second. This is because our method simply adds another item to the sketch. BayesCard, on the other hand, takes 12 seconds to process an update [26], and the other learning-based methods take several minutes. Consequently, these learning-based methods prove impractical for scenarios involving frequent updates.

4.5 PostgreSQL query execution time

In the last set of experiments, we evaluate the impact of our cardinality estimator on the query execution time of PostgreSQL. We utilized the evaluation setup devised by Han et al. [26], they modified PostgreSQL to enable the injection of cardinality estimates for all the sub-queries of the STATS-CEB and JOB-light queries. We compare our method against PostgreSQL's own cardinality estimator as well as the aforementioned learning-based methods. Table 6 shows the total execution time for all the queries. The injected sub-query cardinality estimates are those reported in Section 4.4. In the results, PostgreSQL refers to the default PostgreSQL cardinality estimator, and True Cardinality denotes an oracle method with access to the actual intermediate sizes.

Our proposed method achieved the lowest total execution time with an improvement of 43% compared to PostgreSQL. On STATS-CEB, our method improves over PostgreSQL by 48%, falling just short of FLAT, which showed an improvement of 55%. On JOB-light, our method achieved equivalent execution time to the True Cardinality, while most other learning-based methods, with the exception of BayesCard, do not improve over PostgreSQL. These results underscore the practical advancement enabled by our proposed method due to its superior estimation accuracy, in addition to its exceptional efficiency.

5 CONCLUSION

We have introduced a new sketching method that significantly enhances the cardinality estimation for multi-join queries. The proposed approach provides fast update times, which remain constant

Table 6. PostgreSQL plan execution time and percentage improvement using different cardinality estimators.

Method	STATS-CEB		JOB-light		Total	
PostgreSQL	4.04 h	0%	1.08 h	0%	5.13 h	0%
True Cardinality	1.78 h	56%	0.84 h	23%	2.62 h	49%
BayesCard	2.42 h	40%	0.87 h	20%	3.29 h	36%
FLAT	**1.82 h**	**55%**	1.73 h	-60%	3.55 h	31%
DeepDB	2.24 h	45%	1.81 h	-67%	4.05 h	21%
NeuroCard	4.55 h	-13%	2.51 h	-132%	7.06 h	-38%
Ours	2.09 h	48%	**0.83 h**	**23%**	**2.92 h**	**43%**

irrespective of the required estimation accuracy. This crucial feature allows for efficient processing of high-throughput streams, all the while delivering superior estimation accuracy compared to state-of-the-art baseline approaches. This is substantiated by our bound on the estimation error, as well as our empirical findings. Our results underscore the practical suitability of the proposed method for applications such as query optimization and approximate query processing, surpassing the capabilities of previous methods. The presented method successfully addresses the longstanding challenge of integrating the key advantages of the Count sketch with the AMS-based method for multi-join queries.

A MEAN AND VARIANCE

In this appendix, we present the proofs for Theorems 2.3 and 3.1.

Proof for Theorem 2.3. By the definitions of X and $\boldsymbol{c}_{k,j}$, with $I = I_0 \times \cdots \times I_{r-1}$, and using the linearity of expectation, we get:

$$\mathbb{E}[X] = \frac{1}{m} \sum_{j=0}^{m-1} \sum_{i \in I} \mathbb{E}\left[\prod_{k=0}^{r-1} \mathcal{F}_k(i) \prod_{u \in \Omega(R_k)} \prod_{v \in \Gamma(u)} s_{j,\{u,v\}}(i_u) \right]$$

By construction of the sketches, the sign functions are independent across different joins and there are exactly two occurrences of each sign function, one for each end of a join edge. We can thus write:

$$\mathbb{E}[X] = \frac{1}{m} \sum_{j=0}^{m-1} \sum_{i \in I} \left(\prod_{k=0}^{r-1} \mathcal{F}_k(i) \right) \prod_{\{u,v\} \in E} \mathbb{E}\left[s_{j,\{u,v\}}(i_u) s_{j,\{u,v\}}(i_v) \right]$$

Since $\mathbb{E}[s(a)s(b)] = \mathbf{1}(a = b)$, we get the desired expectation.

For the variance, all the dimensions of the sketches are i.i.d., and since $\mathrm{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$, for any $j \in [m]$, we have:

$$\mathrm{Var}(X) = \frac{1}{m} \mathrm{Var}\left(\prod_{k=0}^{r-1} c_{k,j} \right) \leq \frac{1}{m} \mathbb{E}\left[\left(\prod_{k=0}^{r-1} c_{k,j} \right)^2 \right]$$

By the definition of $\boldsymbol{c}_{k,j}$ and the linearity of expectation, we have:

$$\mathrm{Var}(X) \leq \frac{1}{m} \sum_{i \in I} \sum_{i' \in I} \mathcal{F}_0(i) \cdots \mathcal{F}_{r-1}(i) \mathcal{F}_0(i') \cdots \mathcal{F}_{r-1}(i') \, \mathbb{E}\left[\prod_{k=0}^{r-1} \prod_{u \in \Omega(R_k)} \prod_{v \in \Gamma(u)} s_{j,\{u,v\}}(i_u) s_{j,\{u,v\}}(i'_u) \right]$$

Taking into account that each sign function occurs twice (now four times since the value is squared), we obtain:

$$\mathrm{Var}(X) \leq \frac{1}{m} \sum_{i \in I} \sum_{i' \in I} \mathcal{F}_0(i) \cdots \mathcal{F}_{r-1}(i) \mathcal{F}_0(i') \cdots \mathcal{F}_{r-1}(i') \prod_{\{u,v\} \in E} \mathbb{E}\left[s_{j,\{u,v\}}(i_u) s_{j,\{u,v\}}(i'_u) s_{j,\{u,v\}}(i_v) s_{j,\{u,v\}}(i'_v) \right]$$

By the four-wise independence of the sign functions, the expected value is one if there are two equal pairs or if all values are equal, and zero otherwise. Therefore, we have that:

$$\begin{aligned} &\mathbb{E}\left[s_{j,\{u,v\}}(i_u) s_{j,\{u,v\}}(i'_u) s_{j,\{u,v\}}(i_v) s_{j,\{u,v\}}(i'_v) \right] \\ &= \mathbf{1}\big((i_u = i_v \wedge i'_u = i'_v) \vee (i_u = i'_u \neq i_v = i'_v) \vee (i_u = i'_v \neq i_v = i'_u) \big) \end{aligned}$$

For each join, there are three disjunctions which are conjoined over all the joins. By the distributivity property of conjunction over disjunction, there are thus $3^{|E|}$ disjunctions in total. Since the queries are acyclic, $|E| = r - 1$. The variance is bound by the sum over all disjunctions, where intersections are counted double. Using the Cauchy–Schwarz inequality, each disjunction itself is bound by the product of squared frequency norms. This gives the desired upper bound on the variance. □

Proof for Theorem 3.1. By the definitions of X and S_k, with $I = I_0 \times \cdots \times I_{r-1}$, and the linearity of expectation, we have:

$$\mathbb{E}[X] = \sum_{j \in J} \sum_{i \in I} \mathcal{F}_0(i) \cdots \mathcal{F}_{r-1}(i) \, \mathbb{E}\left[\prod_{k=0}^{r-1} \mathbf{1}(H_k(i) = G_k(j)) \prod_{u \in \Omega(R_k)} \prod_{v \in \Gamma(u)} s_{\{u,v\}}(i_u) \right]$$

By the definitions of H_k and G_k, since the sign and bin functions are independent, and using again the observation that each sign function occurs twice, we can write:

$$\begin{aligned} \mathbb{E}[X] = &\sum_{i \in I} \mathcal{F}_0(i) \cdots \mathcal{F}_{r-1}(i) \left(\prod_{\{u,v\} \in E} \mathbf{1}(i_u = i_v) \right) \\ &\sum_{j \in J} \mathbb{E}\left[\prod_{k=0}^{r-1} \mathbf{1}\left(\sum_{u \in \Omega(R_k)} h_{\Psi(u)}(i_u) - j_{\Psi(u)} \equiv 0 \pmod m \right) \right] \end{aligned}$$

By isolating the case where all $i_u = i_v$, all the attribute values in the same graph component must be equal. Since each independent bin function is thus called with only one distinct value, the expected value of the bin functions is $m^{-(w-r+1)}$, which is exactly the reciprocal of $|J|$, giving the desired expectation.

For the variance, we have that $\mathrm{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$, where:

$$\begin{aligned} \mathbb{E}[X]^2 = &\sum_{i \in I} \sum_{i' \in I} \left(\prod_{k=0}^{r-1} \mathcal{F}_k(i) \mathcal{F}_k(i') \right) \prod_{\{u,v\} \in E} \mathbf{1}(i_u = i_v \wedge i'_u = i'_v) \\ \mathbb{E}[X^2] = &\sum_{i \in I} \sum_{i' \in I} \left(\prod_{k=0}^{r-1} \mathcal{F}_k(i) \mathcal{F}_k(i') \right) \left(\prod_{\{u,v\} \in E} \mathbb{E}\left[s_{\{u,v\}}(i_u) s_{\{u,v\}}(i'_u) s_{\{u,v\}}(i_v) s_{\{u,v\}}(i'_v) \right] \right) \\ &\sum_{j \in J} \sum_{j' \in J} \mathbb{E}\left[\prod_{k=0}^{r-1} \mathbf{1}(H_k(i) = G_k(j)) \, \mathbf{1}(H_k(i') = G_k(j')) \right] \end{aligned}$$

The expected value of the sign functions is stated in the proof for Theorem 2.3. If we consider only the first disjunction, then we get:

$$\mathbb{E}[X]^2 \sum_{j\in J}\sum_{j'\in J}\mathbb{E}\left[\prod_{k=0}^{r-1}\mathbf{1}(H_k(i)=G_k(j))\,\mathbf{1}(H_k(i')=G_k(j'))\right]$$

which is equal to $\mathbb{E}[X]^2$, because when $i_u = i_v \wedge i'_u = i'_v$, all graph components must have one or two distinct values each. In the case of one distinct value, $j_q = j'_q$ for that graph component q. Thus, the expected value per graph component is either $1/m^2$ or $\mathbf{1}(j_q = j'_q)/m$, making the sums over J equal to 1. Now, let us consider everything but $\mathbb{E}[X]^2$ in $\mathbb{E}[X^2]$. There must be at least one occurrence of either $i_u = i'_u \neq i_v = i'_v$ or $i_u = i'_v \neq i_v = i'_u$. Either way, $j_q = j'_q$ for that graph component q while there are still at least two distinct values. The sums over J is thus $\leq 1/m$. We then get that:

$$\mathbb{E}[X^2] = \mathbb{E}[X]^2 + Y \implies \mathrm{Var}(X) \leq \frac{1}{m}\mathbb{E}[X]^2 + Y$$

$$\mathrm{Var}(X) \leq \frac{1}{m}\sum_{i\in I}\sum_{i'\in I}\mathcal{F}_0(i)\cdots\mathcal{F}_{r-1}(i)\mathcal{F}_0(i')\cdots\mathcal{F}_{r-1}(i')$$
$$\prod_{\{u,v\}\in E}\mathbf{1}\big(i_u = i_v \wedge i'_u = i'_v\big) + \mathbf{1}\big(i_u = i'_u \neq i_v = i'_v\big) + \mathbf{1}\big(i_u = i'_v \neq i_v = i'_u\big)$$

which is the same expression of the variance bound as the one for Theorem 2.3. The final steps of the variance bound are thus equivalent, resulting in the desired upper bound. □

REFERENCES

[1] Charu C Aggarwal and Philip S Yu. 2007. A survey of synopsis construction in data streams. *Data streams: models and algorithms* (2007), 169–207.

[2] Thomas Dybdahl Ahle, Jakob Tejs Bæk Knudsen, and Mikkel Thorup. 2020. The Power of Hashing with Mersenne Primes. *arXiv preprint arXiv:2008.08654* (2020).

[3] 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 Dymposium on Principles of Database Systems*. 10–20.

[4] Noga Alon, Yossi Matias, and Mario Szegedy. 1996. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM Symposium on Theory of computing (STOC)*. 20–29.

[5] Shivnath Babu and Jennifer Widom. 2001. Continuous queries over data streams. *ACM SIGMOD Record* 30, 3 (2001), 109–120.

[6] Alain Biem, Eric Bouillet, Hanhua Feng, Anand Ranganathan, Anton Riabov, Olivier Verscheure, et al. 2010. IBM infosphere streams for scalable, real-time, intelligent transportation services. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. 1093–1104.

[7] Burton H Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13, 7 (1970), 422–426.

[8] Walter Cai, Magdalena Balazinska, and Dan Suciu. 2019. Pessimistic cardinality estimation: Tighter upper bounds for intermediate join cardinalities. In *Proceedings of the 2019 International Conference on Management of Data*. 18–35.

[9] Lei Cao, Qingyang Wang, and Elke A Rundensteiner. 2014. Interactive outlier exploration in big data streams. *Proceedings of the VLDB Endowment* 7, 13 (2014), 1621–1624.

[10] Moses Charikar, Kevin Chen, and Martin Farach-Colton. 2002. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*. Springer, 693–703.

[11] Surajit Chaudhuri. 1998. An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. 34–43.

[12] Chen Chen, Hongzhi Yin, Junjie Yao, and Bin Cui. 2013. Terec: A temporal recommender system over tweet stream. *Proceedings of the VLDB Endowment* 6, 12 (2013), 1254–1257.

[13] Zhida Chen, Gao Cong, and Walid G Aref. 2020. STAR: A distributed stream warehouse system for spatial data. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2761–2764.

[14] Graham Cormode. 2011. Sketch techniques for approximate query processing. *Foundations and Trends® in Databases* (2011), 15.

[15] Graham Cormode. 2022. Current Trends in Data Summaries. *ACM SIGMOD Record* 50, 4 (2022), 6–15.
[16] Graham Cormode and Minos Garofalakis. 2005. Sketching streams through the net: Distributed approximate query tracking. In *Proceedings of the 31st international conference on Very large Data Bases (VLDB)*. 13–24.
[17] Graham Cormode, Minos Garofalakis, Peter J Haas, Chris Jermaine, et al. 2011. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends® in Databases* 4, 1–3 (2011), 1–294.
[18] Graham Cormode, Flip Korn, Shanmugavelayutham Muthukrishnan, and Divesh Srivastava. 2003. Finding hierarchical heavy hitters in data streams. In *Proceedings 2003 VLDB Conference*. Elsevier, 464–475.
[19] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.
[20] Alin Dobra, Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. 2002. Processing complex aggregate queries over data streams. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*. 61–72.
[21] Sumit Ganguly, Minos Garofalakis, and Rajeev Rastogi. 2004. Tracking set-expression cardinalities over continuous update streams. *The VLDB Journal* 13, 4 (2004), 354–369.
[22] Nikos Giatrakos, Alexander Artikis, Antonios Deligiannakis, and Minos Garofalakis. 2017. Complex event recognition in the big data era. *Proceedings of the VLDB Endowment* 10, 12 (2017), 1996–1999.
[23] Phillip B Gibbons and Yossi Matias. 1999. Synopsis data structures for massive data sets. *External Memory Algorithms* 50 (1999), 39–70.
[24] Anna C Gilbert, Yannis Kotidis, Shanmugavelayutham Muthukrishnan, and Martin Strauss. 2001. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *VLDB*, Vol. 1. 79–88.
[25] Amit Goyal, Hal Daumé III, and Graham Cormode. 2012. Sketch algorithms for estimating point queries in nlp. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*. 1093–1103.
[26] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, et al. 2021. Cardinality estimation in DBMS: a comprehensive benchmark evaluation. *Proceedings of the VLDB Endowment* 15, 4 (2021), 752–765.
[27] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: learn from data, not from queries! *Proceedings of the VLDB Endowment* 13, 7 (2020), 992–1005.
[28] Yanxiang Huang, Bin Cui, Wenyu Zhang, Jie Jiang, and Ying Xu. 2015. Tencentrec: Real-time stream recommendation in practice. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 227–238.
[29] 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*. 804–816.
[30] Hai Lan, Zhifeng Bao, and Yuwei Peng. 2021. A survey on advancing the dbms query optimizer: Cardinality estimation, cost model, and plan enumeration. *Data Science and Engineering* 6 (2021), 86–101.
[31] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really? *Proceedings of the VLDB Endowment* 9, 3 (2015), 204–215.
[32] Kaiyu Li and Guoliang Li. 2018. Approximate query processing: What is new and where to go? A survey on approximate query processing. *Data Science and Engineering* 3 (2018), 379–397.
[33] Yipeng Liu, Jiani Liu, Zhen Long, and Ce Zhu. 2022. Tensor Sketch. *Tensor Computation for Data Analysis* (2022), 299–321.
[34] Nishad Manerikar and Themis Palpanas. 2009. Frequent items in streaming data: An experimental evaluation of the state-of-the-art. *Data & Knowledge Engineering* 68, 4 (2009), 415–430.
[35] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. 2009. Preventing bad plans by bounding the impact of cardinality estimation errors. *Proceedings of the VLDB Endowment* 2, 1 (2009), 982–993.
[36] Magnus Müller. 2022. Selected problems in cardinality estimation. (2022).
[37] Rasmus Pagh. 2013. Compressed matrix multiplication. *ACM Transactions on Computation Theory (TOCT)* 5, 3 (2013), 1–17.
[38] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).
[39] Ninh Pham and Rasmus Pagh. 2013. Fast and scalable polynomial kernels via explicit feature maps. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 239–247.
[40] Gordon J Ross, Dimitris K Tasoulis, and Niall M Adams. 2011. Nonparametric monitoring of data streams for changes in location and scale. *Technometrics* 53, 4 (2011), 379–389.
[41] Pratanu Roy, Arijit Khan, and Gustavo Alonso. 2016. Augmented sketch: Faster and more accurate stream processing. In *Proceedings of the 2016 International Conference on Management of Data*. 1449–1463.

[42] Florin Rusu and Alin Dobra. 2008. Sketches for size of join estimation. *ACM Transactions on Database Systems (TODS)* 33, 3 (2008), 1–46.
[43] Yang Shi and Animashree Anandkumar. 2019. Higher-order count sketch: dimensionality reduction that retains efficient tensor operations. *arXiv preprint arXiv:1901.11261* (2019).
[44] Michael Stonebraker, Uğur Çetintemel, and Stan Zdonik. 2005. The 8 requirements of real-time stream processing. *ACM Sigmod Record* 34, 4 (2005), 42–47.
[45] Mikkel Thorup and Yin Zhang. 2004. Tabulation based 4-universal hashing with applications to second moment estimation. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*. 615–624.
[46] David Vengerov, Andre Cavalheiro Menck, Mohamed Zait, and Sunil P Chakkappen. 2015. Join size estimation subject to filter conditions. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1530–1541.
[47] Feiyu Wang, Qizhi Chen, Yuanpeng Li, Tong Yang, Yaofeng Tu, et al. 2023. JoinSketch: A Sketch Algorithm for Accurate and Unbiased Inner-Product Estimation. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.
[48] Mark N Wegman and J Lawrence Carter. 1981. New hash functions and their use in authentication and set equality. *Journal of computer and system sciences* 22, 3 (1981), 265–279.
[49] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature hashing for large scale multitask learning. In *Proceedings of the 26th annual international conference on machine learning*. 1113–1120.
[50] Ziniu Wu, Amir Shaikhha, Rong Zhu, Kai Zeng, Yuxing Han, and Jingren Zhou. 2020. Bayescard: Revitilizing bayesian frameworks for cardinality estimation. *arXiv preprint arXiv:2012.14743* (2020).
[51] Tong Yang, Yang Zhou, Hao Jin, Shigang Chen, and Xiaoming Li. 2017. Pyramid sketch: A sketch framework for frequency estimation of data streams. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1442–1453.
[52] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2020. NeuroCard: one cardinality estimator for all tables. *Proceedings of the VLDB Endowment* 14, 1 (2020), 61–73.
[53] Rong Zhu, Ziniu Wu, Yuxing Han, Kai Zeng, Andreas Pfadler, Zhengping Qian, et al. 2021. FLAT: fast, lightweight and accurate method for cardinality estimation. *Proceedings of the VLDB Endowment* 14, 9 (2021), 1489–1502.

Received October 2023; revised January 2024; accepted February 2024