Published: 30 September 2024

RESEARCH-ARTICLE

Enabling Adaptive Sampling for Intra-Window Join: Simultaneously Optimizing Quantity and Quality

XILIN TANG, Renmin University of China, Beijing, China

FENG ZHANG, Renmin University of China, Beijing, China

SHUHAO ZHANG, Nanyang Technological University, Singapore City, Singapore

YANI LIU, Renmin University of China, Beijing, China

BINGSHENG HE, National University of Singapore, Singapore City, Singapore

XIAOYONG DU, Renmin University of China, Beijing, China

Proceedings of the ACM on Management of Data, Volume 2, Issue 4 (October 2024)
https://doi.org/10.1145/3677134
EISSN: 2836-6573

Enabling Adaptive Sampling for Intra-Window Join: Simultaneously Optimizing Quantity and Quality

XILIN TANG* and FENG ZHANG*, Renmin University of China, China
SHUHAO ZHANG, Nanyang Technological University, Singapore
YANI LIU, Renmin University of China, China
BINGSHENG HE, National University of Singapore, Singapore
XIAOYONG DU†, Renmin University of China, China

Sampling is one of the most widely employed approximations in big data processing. Among various challenges in sampling design, sampling for join is particularly intriguing yet complex. This perplexing problem starts with a classical case where the join of two Bernoulli samples shrinks its output size quadratically and exhibits a strong dependency on the input data, presenting a unique challenge that necessitates *adaptive sampling* to guarantee both the quantity and quality of the sampled data. The community has made strides in achieving this goal by constructing offline samples and integrating support from indexes or key frequencies. However, when dealing with stream data, due to the need for real-time processing and high-quality analysis, methods developed for processing static data become unavailable. Consequently, a fundamental question arises: Is it possible to achieve adaptive sampling in stream data without relying on offline techniques?

To address this problem, we propose FreeSam, which couples hybrid sampling with intra-window join, a key stream join operator. Our focus lies on two widely used metrics: *output size*, ensuring quantity, and *variance*, ensuring quality. FreeSam enables adaptability in both the desired quantity and quality of data sampling by offering control on the two-dimensional space spanned by these metrics. Meanwhile, adjustable trade-offs between quality and performance make FreeSam practical for use. Our experiments show that, for every 1% increase in latency limitation, FreeSam can yield a 3.83% increase in the output size while maintaining the level of the estimator's variance. Additionally, we give FreeSam a multi-core implementation and ensure predictability of its latency through both an analytic model and a neural network model. The accuracy of these models is 88.05% and 96.75% respectively.

CCS Concepts: • **Information systems → Stream management**; *Join algorithms*; • **Theory of computation → Sketching and sampling**.

Additional Key Words and Phrases: Data Stream, Stream Sampling, Approximate Join Processing

ACM Reference Format:
Xilin Tang, Feng Zhang, Shuhao Zhang, Yani Liu, Bingsheng He, and Xiaoyong Du. 2024. Enabling Adaptive Sampling for Intra-Window Join: Simultaneously Optimizing Quantity and Quality. *Proc. ACM Manag. Data* 2, 4 (SIGMOD), Article 198 (September 2024), 31 pages. https://doi.org/10.1145/3677134

*Xilin Tang and Feng Zhang are co-first authors. Both authors contributed equally to this research.
†Xiaoyong Du is the corresponding author.

Authors' Contact Information: Xilin Tang, kline@ruc.edu.cn; Feng Zhang, fengzhang@ruc.edu.cn, Renmin University of China, Beijing, China; Shuhao Zhang, Nanyang Technological University, Singapore; Yani Liu, Renmin University of China, Beijing, China, 2021000888@ruc.edu.cn; Bingsheng He, National University of Singapore, Singapore; Xiaoyong Du, Renmin University of China, Beijing, China, duyong@ruc.edu.cn.

ACM 2836-6573/2024/9-ART198
https://doi.org/10.1145/3677134

1 Introduction

Sampling has become increasingly important in approximate processing of big data [23, 37, 60, 64, 65, 77, 80, 87, 95, 97, 104, 109, 129, 130, 136–139]. Within various sampling problems, sampling for join stands out for its unique influnce on the combinatorial procedure of join [18, 44, 59, 112, 122].

Sampling for join techniques often strive to achieve two distinct objectives: 1) quantity, measured by *output size* [28, 29, 120], and 2) quality, typically represented by *variance* [96, 135]. Maximizing output size means producing the largest size of join output using the minimal sample of input relations, thereby enabling efficient delivery of adequate entries for exploratory cases such as pattern mining, visualization, etc [13, 58, 73, 100, 134]. Variance generally denotes the volatility of the estimator of the original join output size (cardinality). It is a popular indicator of accuracy and stability [6, 92] and can easily be turned to measure the accuracy of other statistical estimators as well [24, 59]. Notably, modern applications necessitate the *adaption* of both metrics. For example, representative reporting [30, 32] requests a specific volume of high-quality data samples. To fulfill such demand, the community has introduced techniques such as index-assisted sampling [40, 75, 76, 79], frequency-based sampling [18, 36], and cyclic join handling [19].

Despite these efforts, adaptive sampling for join in the stream scenario received less attention. Without high-speed indexes and prepared presamples, maintaining the low-latency and progressive execution of sampling for join becomes particularly challenging. Meanwhile, the unpredictable distribution of stream data poses obstacles in controlling sampling quality. We target the most fundamental pattern of stream join, binary join, and design sampling in the environment of intra-window join (*IaW⋈*), a prevalent stream join operator that processes an entire window regardless of the window's type [2, 9, 35, 52, 78, 140].

One representative use case is found in the intense IoT streams within traffic management, which demands dual objectives of high-quality approximate estimation and pattern discovery from high-quantity raw data, all within a second-level latency budget [8, 84]. For example, estimating traffic states on urban signalized links involves integrating massive streams from in-vehicle and street detector to provide second-by-second updates of approximation on vehicle accumulation and queue tail location [106]. The variance of the estimation error is directly applied to measure the accuracy of queue tail location estimation. Meanwhile, there is a strong interest in increasing the output size of the raw data from connected vehicles, which facilitates pattern detection of links between locations and vehicles. Therefore, adaptively satisfying these two quality and quantity metrics in this scenario is of practical significance. Despite these benefits, current adaptive sampling for join is not inherently stream-oriented, as its latency could be 13.77× longer on stream compared to on-the-fly methods (detailed in Section 3).

Developing adaptive sampling for stream join faces three major challenges. First, the absence of offline support limits the range of sampling options. For instance, sampling implemented through attribute-to-attribute jumps relies heavily on high-speed indexes [30, 40, 75, 76, 79, 107, 126], which are missing in stream processing. Meanwhile, existing methods typically prioritize a single metric, resulting in compromised control over the other metric. Second, lacking pre-knowledge of stream data hampers the ability of sampling to adjust join results with data features [115, 145]. Prior to the influx of data, the server has no knowledge about its distribution, making it impractical to render adaption by mixing presampled data [94, 105, 118]. In other words, adaptive sampling for stream join calls for an on-the-fly design. Third, the application of sampling for join requires timely prediction. While previous works, such as Quickr [68], have introduced sampling into the query plan, there is still no cost model of the sampling-aware stream join. A usable operator needs to be modeled appropriately to inform the system or user about its cost.

We propose **FreeSam**, an adaptive sampling solution for *IaWJ*, to address the above challenges. First, FreeSam strikes a balance between efficient stream join algorithms and sampling flexibility. It pioneers the possibility of incorporating different sampling techniques into the building and probing phases of symmetric hash join, and combines it with hybrid sampling, redefining the problem of sampling for stream join. Second, without assuming that the data holds a certain distribution or that the future arrival order of the stream is known, FreeSam provides a general solution to adapt sampling for *IaWJ* in the two-dimensional space spanned by quantity and quality. We accomplish the adaption by proposing a new concept, *probe utilization*, which leverages the "discarded" data. By adjusting the probe utilization, FreeSam can vary the number of output tuples without deteriorating the variance of the estimator, thereby supporting the adaption of quantity and quality. Third, to enhance the predictability of the procedure's time consumption, we develop an analytical cost model and a tiny neural network for latency prediction. An analytical model is more explainable but also tends to ignore stream features such as the arrival rate of data, the duplication of keys, or hardware characteristics like the number of cores and memory bandwidth. Therefore, we build a tiny neural network based on this cost model, which can adapt to different circumstances and achieve better accuracy.

We validate the efficacy of FreeSam on three real-world datasets. Experiments show that by varying probe utilization, FreeSam can trade off 3.83% of the output size for every 1% increase in latency and maintain the level of the estimator's variance simultaneously, which achieves the adaption among the two prime metrics. For example, on the dataset Rovio, when fixing a 10^{-3} level variance of the relative estimation error, FreeSam's output size can vary from 9.49×10^5 to 1.35×10^7. This capability enables FreeSam to cater to applications that necessitate the adaption to both quantity and quality. Meanwhile, with the analytical cost model, the average accuracy of latency inference reaches 88.05%, and the tiny neural network improves it to 96.75%. Furthermore, FreeSam is scalable on multi-core and validated through comprehensive studies.

To sum up, we make the following contributions in this paper.

- We propose an adaptive sampling solution for *IaWJ*, called FreeSam. By symmetrically incorporating sampling into hash join phases, it simultaneously achieves flexible sampling and efficient progressive execution.
- We introduce a new concept, probe utilization, to manage the adaption of quantity and quality metrics. We prove its properties formally.
- We develop an analytical cost model and a tiny neural network to infer time consumption. It reveals where sampling takes effect and how to make the inference adapt to diverse system and hardware settings.
- We conduct a comprehensive evaluation to measure the performance of FreeSam and corroborate its sampling quality.

2 Background and Related Work

2.1 Sampling for Join

Sampling for join has attracted continuous attention in databases. Unlike sketch approaches [16, 26, 46, 85, 101, 103, 119, 125], histogram strategies [42, 49, 86, 142], and wavelet methods [25, 48, 133], the complexity of it lies in resolving the different data distribution between the input data and the output of join [30, 43, 63, 81].

Challenges. To exhibit the challenges in sampling for join, we analyze two basic sampling methods. *Bernoulli sampling*, also known as uniform sampling or random sampling [1], samples each tuple in the data independently with the same probability. However, although the independence of each sampled tuple is assured, it cannot guarantee the independent distribution of the join output.

More precisely, when sampling tuples with a q probability, it pronely generates only q^2 of the original outputs [17, 24]. *Universe sampling* [122] samples all tuples whose keys satisfy the universal hash value condition. This introduces higher dependence between tuples when performing joins, which achieves a uniform distribution of the original join result in an ideal situation. However, as pointed out in the study [59], it pronely leads to a larger variance of the estimator as well.

Current solutions. To tackle the challenges, researchers have introduced more data information into the sampling scheme or mixed basic sampling methods [5, 10, 15, 21, 55, 62, 71, 74, 90, 91, 93, 122]. One approach is to utilize indexes to support stratifying, like bifocal sampling [40]. A notable work using indexes is wander join [75, 76], which recognizes index assistance as the multipartite graph information. Another approach of information injection directly applies the key frequency, like end-biased sampling [36]. As to hybrid samplings, works like Bi-Level [53] and SUBS [59] design a multi-layer sampling procedure, and use different sampling methods to filter data in each layer. Also, some approaches incorporate weight into sampling [33, 34, 108, 113]. Some works use multiple techniques, like Two-Level [18]. However, despite these efforts, most of them do not effectively enhance sampling for stream join due to the unpredictability of streams.

2.2 Sampling for Join in Stream

The invocation of low-latency and real-time processing in massive streaming data leads to the use of approximation to alleviate the workload [51, 66, 99, 127]. Among the various big data synopses, sampling is particularly useful with its flexibility and insensitivity to data [144, 145].

Current status. Sampling for join in stream scenarios has received limited attention. In static databases, sampling methods can be distinguished as *online* or *offline* based on when the sample is generated: either during or before the query [18, 68]. Offline methods are not suitable for the streaming situation, because the system has no preparation until the data arrives. Meanwhile, not all online methods are applicable, because there are no overall statistics such as the frequency of each key or the index built before. This explains why most of the methods in Section 2.1 are not pertinent, and why samplers in the latest version of modern stream processing systems, like Flink [39, 89, 124] and Spark [110], still use basic approaches like Bernoulli sampling [1] and reservoir sampling [7, 38, 123]. Therefore, in stream scenarios, traditional solutions hold segregated sampler and joiner, and defer the join until the sampling ends [115, 120].

Despite recent progress in hybrid sampling methods, these approaches predominantly depend on static data conditions, and few are aptly designed for streaming environments. For instance, SamComb [94] necessitates a presampled data pool for output formulation, and Two-Level [18] requires a preliminary scan to determine key-oriented parameters. To our knowledge, the only hybrid approach that is suited for streaming scenarios is UBS [59], which sets only a single parameter during its prescan phase. Therefore, we preset its parameter and adapt it to streaming scenarios as a benchmark in Section 7. Our research aims to establish an inherent connection between the sampler and the joiner. Moreover, to obtain predictable performance and enable query optimization [4, 67, 111], it is desirable to build cost models for samplings in stream processing [20, 141, 143].

3 Motivation

In this section, we explore the reasons why previous static methods fall short in streaming contexts and emphasize the lack of adaptability across different metrics.

Why are previous static methods not applicable? Our initial experiments show that, without specific consideration of stream scenarios, static methods can cause 13.77× larger latency on average. We introduce two state-of-the-art static methods as a comparative benchmark: *Two-level* [18] and *SJoin* [145]. Remarkably, even a preliminary on-the-fly integration of Bernoulli sampling, referred to as *On-the-Fly Bernoulli* in Figure 1, outperforms these methods in terms

of latency. The comparison is conducted on the DEBS dataset, as described in Section 7.1, with varying window lengths.

Observation. Figure 1 shows the elapsed time for the abovementioned methods across varying window lengths. The window here pertains to the time interval required for the input stream to arrive, constituting the initial portion of the elapsed time, which is referred to as the time within the window. It indicates two unique concerns that differentiate sampling for join in streams from static databases. First, progressive processing aims to minimize the need for preprocessing and waiting. For example, although *Two-level* establishs good sampling parameters for each key through an overall pre-scan, its latency remains constant and does not improve as window lengths increase. Second, low-latency feedback necessitates a lightweight procedure. For example, when processing DEBS with a 0.1 sampling rate and a window length of 0s (equivalent to a cold-start database), *SJoin* takes 4.12s, whereas *On-the-Fly Bernoulli* takes only 0.21s, reducing the latency by 95%. The reason is that *SJoin* focuses on constructing synopses using relatively heavy data structures, which targets infrequent updates rather than real-time requirements. Consequently, stream processing necessitates a novel and specialized sampling for join.

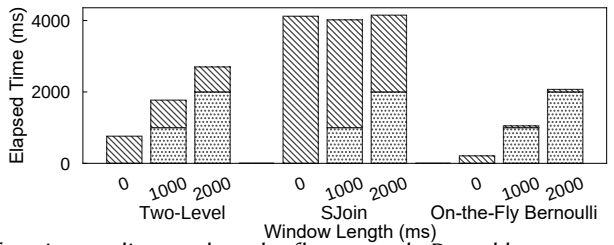


Fig. 1. Comparison of static samplings and on-the-fly approach. Dotted lower segments in bars represent the elapsed time within the windows, while dashed upper segments represent the latency formed by the difference between the total elapsed time and the end of the windows.

Lack of adaptation between metrics. While adaptive sampling has been explored in various ways, such as varying dataflow [35] and changing query filters [94], adapting sampling to metrics remains difficult: For a given sampling scheme, the results obtained on metrics are monotonic functions of the sampling rate, which establishes a one-to-one relationship across different metrics.

Table 1. Comparison of classic methods on different metrics with varying sampling rates on DEBS.

#sampling	Output Size		Variance	
rate	Bernoulli	Universe	Bernoulli	Universe
0.001	495	$\mathbf{1.47}\times10^{5}\checkmark$	$\mathbf{1.25}\times10^{-2}\checkmark$	3.05×10^{-1}
0.01	4.84×10^{4}	$\mathbf{2.84}\times10^{6}\checkmark$	$\mathbf{4.61}\times10^{-4}\checkmark$	1.03×10^{-1}
0.1	4.90×10^{6}	$\mathbf{2.66}\times10^{7}\checkmark$	$\mathbf{2.14}\times10^{-5}\checkmark$	1.21×10^{-2}

Observation. In Table 1, we demonstrate the comparison between Bernoulli sampling and universe sampling on two specific metrics using the DEBS dataset. The results on both metrics exhibit monotonic behavior with respect to the sampling rate, and typically one method performs well on one metric but poorly on the other. For instance, while universe sampling yields a larger output size of 2.84×10^{6}, it has a relatively high variance of the estimator at 1.03×10^{-1}. In terms of geometric average, Bernoulli sampling achieves 6.87×10^{-3} smaller variance than universe sampling, but universe sampling has a 45.56× larger output size. Although hybrid methods like UBS [59] can achieve higher output sizes with the same level of variance as Bernoulli sampling, the one-to-one correspondence between metrics remains unchanged. As this challenge has not yet been resolved, our goal is to fundamentally enable the adjustment of interrelationships between metrics.

4 Problem Formulation

In this section, we show our problem settings in a conceptual way. We summarize the notations used in this paper in Table 2.

Table 2. Notations and descriptions used in this paper.

Notation	Description
$x = (k, v, ts)$	Tuple with three attributes
R, S	Streams to join
$R[k]$	Subset of R with key k
w or $w(t_1, t_2)$	Window
w_R	Stream R on window w
$w_R(t_1, t_2)$ or $w_R(t_2)$	R on timestamp determined window w
$\|w\|$	Window length (ms)
J	Join size
σ	Sampling function
Id	Identity function
$I[State]$	1 if $State$ is true; 0 otherwise
ϵ	Sampling rate
q	Rate of Bernoulli sampling
p	Rate of universe sampling
λ	Probe utilization
$\gamma_{i,j}$	$\sum_k R^i(k) \cdot S^j(k)$
Ψ	Result of a sampling-aware $IaWJ$
$X_{r,s}$	$I[r$ matches $s]$
$\xi_{r,s}$	$I[r$ arrives the system before $s]$ or its probability
$\Xi_{r,s}$	Constant $(\sum \xi_{r,s})$
$\mu_{r,s}$	The average of $\xi_{r,s}$ $(\sum \xi_{r,s}/J)$
$f(x_{r,s}, x_{s,r})$	Functional abbreviation of coefficient
C	Time cost

4.1 Revisiting Stream Join Model

Following the definitions in previous work [3, 82, 83, 88, 102, 116, 117, 140], we define a **tuple** x as an ordered set (k, v, ts), where k, v, and ts represent the key, payload, and timestamp, respectively. An input **stream**, denoted by R or S, is a list of tuples chronologically arriving at the system, e.g., a query processor. The subset with a specific key k of stream R is denoted by $R[k]$. Moreover, in stream processing, we usually operate on a continuous bounded subset of the stream data (i.e., a window), which is defined below.

Definition 1. *For any two arbitrarily given timestamps t_1 and t_2 and a stream R, the **window**, denoted by $w(t_1, t_2)$, is an interval $[t_1, t_2]$. R on w is $\{(k, v, ts) \mid t_1 \leq ts \leq t_2\}$, denoted by $w_R(t_1, t_2)$, and can be abbreviated as $w_R(t_2)$ or w_R without causing ambiguity. We denote $|t_1 - t_2|$, i.e., the **length** of the window, by $|w|$.*

Stream join can be classified into two types depending on how we choose the window. The first is *inter-window join*, also known as sliding window join [45, 69, 131]. It deals with windows overlapping with others and puts its core challenge to enable an incremental update of new tuples. The second is *intra-window join*, abbreviated as $IaWJ$, which focuses on performing join over an entire window [35, 78], irrespective of the window type (i.e., sliding, tumbling, or session) [121]. In this work, the query we study is natural join on two streams, R and S, within a specified time-based window. Additionally, we explore its extension to aggregations. $IaW\tilde{J}$ is chosen as our test case, and stream join in this paper refers to $IaW\tilde{J}$.

Various multicore $IaWJ$ algorithms have been proposed [12, 14, 22, 31, 35, 41, 56, 72, 78, 128]. Among these algorithms, *symmetric hash join (SH$\tilde{J}$)* [128] has been the default (and often the only available) join algorithm used in many state-of-the-art stream processing engines [98]. The gist of it is to interleave building and probing phases in hash join to achieve low processing latency. In view of the superb applicability of SHJ, we adopt it as the integration object of sampling design for stream join.

4.2 Sampling-Aware Intra-Window Join

We now formalize the sampling for *IaWJ* problem. We define the term "*sampling-aware IaWJ*" to encompass the entire process, which begins with the entry of two input streams and concludes with the generation of a single output stream.

For the basic concepts, we denote a **sampling scheme** on a stream R by σ, and the **result of sampling** by $\sigma(R)$. The **sampling rate**, or the ratio of data stored, is denoted by ϵ. Given the stream join settings in Section 4.1, we symmetrically define the sampling-aware *IaWJ* of two streams. The definition considers two sampling schemes for each stream to address its property. The result of sampling-aware *IaWJ* is generated by tuples in stream R probing the stored part of S ($R \bowtie S$), and tuples in S probing the stored part of R ($S \bowtie R$). We describe the sampling of stream R by two sampling schemes of σ_R and σ'_R. σ_R governs the building phase, i.e., whether to store an incoming tuple in R for subsequent use, and σ'_R controls whether to probe the samples stored in the opposite stream S. Notably, σ_R and σ'_R can be different. For S, this description is similar, which is described by σ_S and σ'_S. The following definition elucidates the sampling-aware *IaWJ* considered in this paper.

Definition 2. *Given a window w, input streams R and S, and sampling schemes of $\sigma_R, \sigma_S, \sigma'_R, \sigma'_S$, the formalized definition of the **sampling-aware IaWJ** is as follows, and is abbreviated as Ψ.*

$$\left(\bigcup_{i \in w_S} \sigma_R(w_R(i.ts)) \bowtie \sigma'_S(\{i\})\right) \cup \left(\bigcup_{j \in w_R} \sigma_S(w_S(j.ts)) \bowtie \sigma'_R(\{j\})\right).$$

Example. In Figure 2, we show a conceptual procedure of sampling-aware *IaWJ* using the example of the weather stream in Section 7. Assume that the ticked tuples in Figure 2 (a) are the result of future samplings σ_R and σ_S. They are stored in memory via the building phase when arriving over time. Setting $\sigma'_R = \sigma'_S = Id$ (i.e., σ'_R and σ'_S return whatever they obtain), every tuple probes the opposite stream's sample to produce matches. Notably, this matching process occurs only when a tuple probes the opposing tuple that has arrived earlier and thus has a smaller timestamp (ts). The execution procedure is shown in Figure 2 (b). The shadowed tuples are stored during the building phase according to Figure 2 (a). Lines with arrows mark the join matches generated by probing, such as r_1 and s_1. Snapshots of matches generated over time are shown in Figure 2 (c). At $ts = 2$, only r_1 and s_1 arrive, and the procedure outputs a match (r_1, s_1) totally formed by tuples stored in the memory. At $ts = 3$, r_2 comes, and the procedure outputs a match (r_2, s_1) albeit r_2 is not sampled by σ_R (not stored in memory).

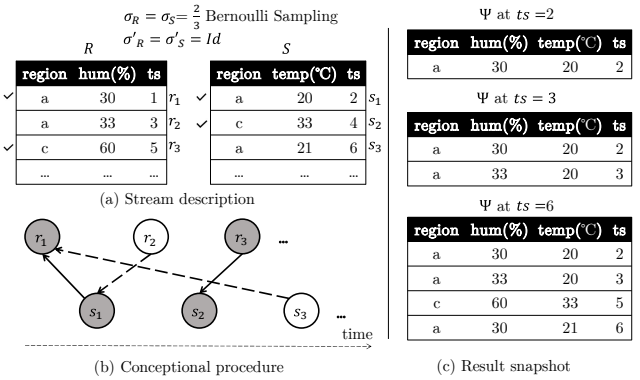


Fig. 2. Example of sampling-aware *IaWJ*. The column "region" represents the key, "hum" and "temp" represent values of humidity and temperature, respectively, and "ts" represents the timestamp that is assumed to be congruous with the order in which tuples arrive at the system in this example.

4.3 Metrics

We now describe the metrics used to measure the quantity and quality of the result, as well as processing performance, which collectively constitute our goal of optimization.

Quantity and quality. We use the most commonly used metrics: *output size* [28, 59, 120] and *variance* [18, 96, 135]. *Output size* refers to the total number of join outputs produced during the procedure. It highlights the delivery of raw data, proved to be especially pertinent in exploratory scenarios where users may possess limited knowledge of incoming data and hope to capture as much information as possible for subsequent utilization. *Variance* represents the variability of the estimator of the original join output size (cardinality). It emphasizes statistical accuracy and stability, and determines the size of the confidence interval [6]. Additionally, variance can be adapted to measure the accuracy of aggregations like `COUNT`, `SUM`, and `AVG` [24]. Our approach integrates both metrics, accommodating a broad range of queries and presenting a comprehensive and versatile strategy for sampling in joins.

Performance. The *latency* of a join match denotes the duration from the arrival of its last input (R or S) to the generation of its join result. Following the previous work [70, 140], we use the quantile worst-case latency (e.g., 95th) as criterion. An acceptable latency is usually the first concern of a streaming procedure.

5 FreeSam

5.1 FreeSam Design

Instantiation. Definition 2 indicates the symmetry of build and probe that lies in sampling-aware *IaWJ*. Using symmetry, we can construct the flow that incorporates two sampling schemes per stream. Furthermore, to prevent wastage of memory usage, it is crucial to ensure that any tuple stored in memory generates join matches; hence, the sampling result of σ' needs to be a superset of the sampling result of σ to prevent tuples occupying memory for doing nothing. Thus, σ' essentially operates on the data discarded by σ. A illustration of the role that sampling plays is shown in Figure 3. Every tuple r in stream R first attempts to pass through σ_R. If it passes σ_R, both building and probing phases process it. If it does not pass σ_R, σ'_R gives it an additional chance to be sampled, and applies only the probing phase when σ'_R samples it. Similarly, stream S follows a symmetrical procedure of it.

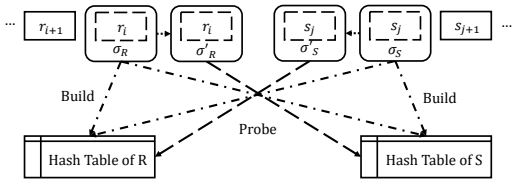


Fig. 3. Conceptual coupling of sampling and *IaWJ*. Dashed lines mean that whether the procedure executes depends on the sampling. The different formats of arrows refer to passing sampling results of different sampling schemes.

Which sampling methods to apply and combine? As discussed in Section 2, hybrid sampling has emerged as a highly promising concept in the realm of database systems. Among various hybrid approaches, the combination of Bernoulli sampling and universe sampling has gained significant traction [18, 53, 59]. The strength of universe sampling lies in its ability to produce output sizes that are generally desirable, while it tends to exhibit higher estimator variance [122]. Conversely, the Bernoulli method often exhibits an opposite behavior [17, 24, 68]. By combining these two methods in a hybrid approach, it becomes possible to mitigate the limitations of each individual sampling technique. Fortunately, these two samplings are still available within the inherent limitations of

the streaming scenario due to their efficient and lightweight properties. In contrast to previous studies that focused on communication complexity [59] or key frequency [18] in databases, we highlight the incorporation of the hybrid approach with the stream join procedure, presenting the first solution for adaptive sampling for *IaWJ*. Based on the sampling functions defined in Section 4.2, we consider a two-step algorithm design: one for σ and the other for σ'.

1) Sampling-aware building phase. The sampling strategy attached to build phase controls which tuples are retained in memory. As a specialization of sampling for join, rather than indiscriminately treating each tuple uniformly, we aim to selectively filter the keys required for building. Therefore, we employ the two-layer σ to respectively control the selection of keys and tuples.

The first layer of σ is universe sampling, responsible for selecting keys to produce a significant fraction of the original join matches. With the parameter $p \in [0, 1]$ and a given perfect hash $h : K \mapsto [0, 1]$, it samples the tuples whose keys k satisfy $h(k) \leq p$. The second layer, Bernoulli sampling, achieves uniformity of tuple-wise selection. With parameter $q \in [0, 1]$, it samples every tuple that has passed the first layer with the same probability q. In practice, we apply a more efficient implementation of it. The method uses the number of skipped tuples between the two sampled tuples. By generating a pseudo-random number u uniformly distributed over $[0, 1]$, based on the inverse transform method [114], the random variable $\lfloor \log u / \log(1-q) \rfloor$ follows a geometric distribution that corresponds to the gap of the inclusions [51]. Denoting the universe and Bernoulli layers as U and B respectively, the probability of a tuple r in R to be sampled is $Pr[r \in \sigma_R(R)] = Pr[r \in U(R)] \cdot Pr[r \in B(R) | r \in U(R)] = pq$.

2) Sampling-aware probing phase. For σ', the crux of σ' designing is to specify those "useful" tuples for probing. To avoid wastage, the first principle is to make it contain σ. Therefore, its first two layers have the same design as σ. Another task is to utilize the tuples not sampled by σ. For those keys not sampled in σ's result, it is impossible for them to generate matches with stored tuples, so σ' should not consider them. Then, to measure the ratio of useful tuples σ' uses, we bring up probe utilization.

Definition 3. *For a given stream R with sampling schemes σ and σ' for building and probing respectively, the **probe utilization** is given by the ratio of tuples in $\sigma'(R)$ that are not in $\sigma(R)$, within the subset of R identified by the key sampled in σ:*

$$\lambda = \frac{\sum_k I[\,|\sigma(R)[k]| > 0\,] \cdot |\sigma'(R)[k] - \sigma(R)[k]|}{\sum_k I[\,|\sigma(R)[k]| > 0\,] \cdot |R[k] - \sigma(R)[k]|}.$$

Guided by this, we add a third sampling layer. The tuples that pass the first universe layer but not the second layer are transferred into an extra Bernoulli sampling. By varying the parameter of the third layer, we can directly control λ. Denoting the extra probing layer as P, the probability a tuple r in R to be sampled is $Pr[r \in \sigma_R(R)] = Pr[r \in U(R)] \cdot (Pr[r \in B(R) | r \in U(R)] + Pr[r \in P(R), r \notin B(R) | r \in U(R)]) = p(q + (1-q)\lambda)$.

Considering both σ and σ'. Combining σ and σ' requests unifying their related parameter, p, for filtering the keys. Given the effective sampling rates of R and S, denoted as ϵ_R and ϵ_S, the sampling rate for a stream R (ϵ_R) is determined by the product of the universe sampling's parameter p_R and the Bernoulli sampling's parameter q_R, that is, $p_R \cdot q_R$. When focusing on p in the first layer, if p_R and p_S are set different, the stream with the larger p may sample the key that is never sampled or joined by the other stream with a lower p, resulting in a significant loss in performance. Therefore, we set $p_R = p_S = p$, and replace q_R and q_S with ϵ_R/p and ϵ_S/p respectively. This design makes the key filtering of σ and σ' unified.

In this subsection, our primary focus is to provide insights into the design of FreeSam and its workflow. The discussion on the procedure for parameter settings shall be covered later in Section 5.6.

Algorithm 1: FreeSam

Input: Streams: R, S; Sampling rate: ϵ_R, ϵ_S;
Hash function for keys: $h : K \mapsto [0, 1]$;
Parameter of the universe layer: p;
Probe utilization: λ_R, λ_S.

```
1  GenGap ← λx[ ⌊log RandomFloat(0,1)/log(1 − x)⌋ + 1]
2  Δ[0...1] ← [GenGap(ϵ_R/p), GenGap(ϵ_S/p)]
3  Δ'[0...1] ← [GenGap(λ_R), GenGap(λ_S)]
4  while streams not end do
5      tuple ← NextTuple(R, S)
6      idx ← I[tuple in R]
7      if h(tuple.key) ≤ p then
8          Δ[idx] ← Δ[idx] − 1
9          Δ[idx]' ← Δ'[idx] − 1
10         if Δ[idx] == 0 then
11             build tuple into the storage of the belonging stream;
12             probe the opposite stream to generate join matches
13             Δ[idx] ← idx ? GenGap(ϵ_S/p) : GenGap(ϵ_R/p)
14             Δ[idx]' ← Δ'[idx] + 1
15         else if Δ'[idx] == 0 then
16             probe the opposite stream to generate join matches
17             Δ'[idx] ← idx ? GenGap(λ_S) : GenGap(λ_R)
```

Algorithm. We show the algorithm of FreeSam in Algorithm 1. In Lines 1-3, the algorithm initializes a lambda function *GenGap* to generate a random gap value with the foregoing geometric approach, and initializes two arrays Δ and Δ' to store the remaining gap values for each stream's sampling schemes σ and σ'. In Lines 4-6, the algorithm fetches tuples continuously from streams R and S, and sets the index idx according to which stream the tuple comes from. In Lines 7-8, the algorithm verifies whether the tuple successfully passes the universe sampling layer and should be taken into account for the sampling in next layer. It then adjusts the gap values in Δ and Δ' accordingly. If Δ reaches 0, the tuple is chosen by σ. Then, Lines 10-14 will build the tuple into the storage of the belonging stream and probe the opposite stream to generate join matches. Otherwise, if Δ' reaches 0, the tuple is chosen by σ', and Lines 15-17 will process it. The algorithm probes the other stream to generate join matches without building the tuple into the storage. The algorithm continues processing tuples until both streams have been fully processed.

5.2 Two Extreme Cases

In this part, we provide a preliminary exploration of the adaptability of FreeSam to the quantity and quality metrics mentioned in Section 4.3. We show this by analyzing two extreme cases of applying FreeSam. In the first case, λ is set to 0, and the results will have the same form as the separated calculation of sampling and join. This case verifies that the algorithm can still be valid for non-streaming problems and provides a baseline. In the second case, we set λ to 1 and examine its maximum possible impact in a random case. Through the comparison of the two cases, we can preliminarily explore the impact and benefits of introducing λ.

Minimum probe utilization. To make the probe utilization minimum, we set λ to 0 and the extra probe is canceled. Meanwhile, if for a stream R, $(\sigma(R) - \sigma'(R)) \neq \emptyset$, then there are tuples that are stored but not utilized. Therefore, we set $\sigma' = Id \circ \sigma$. This setting makes sampling and join decoupled and converts the problem to join on samples.

In terms of the output size metric, the superiority of the algorithm depends on how many join matches are produced in the expectation sense. We denote the original join output size by $J = \sum_k R[k] \cdot S[k]$.

Lemma 1. *Given ϵ_R, ϵ_S, and p, which determine σ_R and σ_S, let $\sigma'_R = Id \circ \sigma_R$ and $\sigma'_S = Id \circ \sigma_S$, making $\lambda_R = \lambda_S = 0$. Then, the expectation of $|\Psi|$ is given by:*

$$E[|\Psi|] = \frac{\varepsilon_R \varepsilon_S}{p} J.$$

The detailed derivation is shown in Appendix A.1.

$E[|\Psi|]$ also provides the facility for building the estimator of original join output size by the Horvitz-Thompson method [57]. Accordingly, the unbiased estimation is $\hat{J} = \frac{p}{\varepsilon_R \varepsilon_S} |\Psi|$. As mentioned in Section 4.3, the variance of the estimator is a significant indicator of its accuracy. Its variance is given by Lemma 2.

Lemma 2. *Given ϵ_R, ϵ_S, and p, which determine σ_R and σ_S, let $\sigma'_R = Id \circ \sigma_R$ and $\sigma'_S = Id \circ \sigma_S$, making $\lambda_R = \lambda_S = 0$. The variance of $\hat{J}$ is as follows:*

$$Var[\hat{J}] = \frac{1-p}{p}\gamma_{2,2} + \frac{p-\epsilon_S}{p\epsilon_S}\gamma_{2,1} + \frac{p-\epsilon_R}{p\epsilon_R}\gamma_{1,2} + \frac{(p-\epsilon_S)(p-\epsilon_R)}{p\epsilon_S\epsilon_R}\gamma_{1,1},$$

where $\gamma_{i,j} = \sum_k R^i[k] \cdot S^j[k]$.

The proof of the variance is shown in Appendix A.2.

Note that the solution for the estimator ($\hat{J}$) and its variance ($Var[\hat{J}]$) has the same format as in the study [59], which explored database join on the samples. This correspondence shows the validity of FreeSam in non-stream conditions and prompts further investigation into sampling awareness in streams.

Maximum probe utilization. The following case considers the maximum impact of utilizing the sampling scheme in probing. With σ' set to Id, σ' samples arbitrary tuples, and makes $\lambda = 1$. In this extreme case, to simplify the problem, we assume that the sequence in which tuples arrive is random. Thus, for two tuples r and s, the probability of r coming before s is equal to the probability of s coming before r, both being $\frac{1}{2}$. We will address the problem of arrival order in Section 5.3.

Lemma 3. *Assuming the arrival order of tuples random, let ϵ_R, ϵ_S, and p determine σ_R and σ_S, and set σ'_R and σ'_S by Id, making $\lambda_R = \lambda_S = 1$. Then, the expectation of $|\Psi|$ is given by:*

$$E[|\Psi|] = \frac{\varepsilon_R + \varepsilon_S}{2}\gamma_{1,1} = \frac{\varepsilon_R + \varepsilon_S}{2} J.$$

The detailed derivation is shown in Appendix A.3. The unbiased estimation is $\hat{J} = 2|\Psi|/(\varepsilon_R + \varepsilon_S)$ and its variance is given by Lemma 4.

Lemma 4. *Assuming the arrival order of tuples random, let ϵ_R, ϵ_S, and p determine σ_R and σ_S, and set σ'_R and σ'_S by Id, making $\lambda_R = \lambda_S = 1$. Then, the variance of $\hat{J}$ is given by:*

$$\begin{aligned} Var[\hat{J}] \quad = \quad & \frac{1-p}{p}\gamma_{2,2} + \frac{(\varepsilon_S - 3\varepsilon_R)(\varepsilon_S + \varepsilon_R) + 4p\varepsilon_R}{3p(\varepsilon_R + \varepsilon_S)^2}\gamma_{2,1} \\ & + \frac{(\varepsilon_R - 3\varepsilon_S)(\varepsilon_R + \varepsilon_S) + 4p\varepsilon_S}{3p(\varepsilon_R + \varepsilon_S)^2}\gamma_{1,2} + \frac{-\varepsilon_R - \varepsilon_S - 2p}{3p(\varepsilon_R + \varepsilon_S)}\gamma_{1,1}. \end{aligned}$$

The proof of variance is shown in Appendix A.4.

Evolution of output size. We compare Lemma 1 with Lemma 3. The coefficient changes from $\epsilon_R\epsilon_S/p$ to $(\epsilon_R + \epsilon_S)/2$. When recognizing p as a fixed parameter, the utilization of λ changes the relation of the two sampling rates from multiplication to addition. Since the variables do not exceed 1, this change enhances the expected output size, resulting in an order-of-magnitude increase. A practical application of setting $\lambda = 1$ in the context of traffic monitoring mentioned in Section 1 is to join data across streets to gather information about intersections. With maximal probe utilization, for any sampled street, we obtain samples of all its intersections, regardless of whether the other streets that form those intersections are sampled or not.

Stabilization of variance. When investigating Lemma 2 and Lemma 4, we find that they hold the same format of $\frac{1-p}{p}\gamma_{2,2} + o(\gamma_{2,2})$, which means that the variance remains stable and makes the output size improvement obtained from λ almost costless.

5.3 General Cases

In this subsection, we focus on presenting an assumption-free solution, without presupposing any specific data distribution of arrival order in the stream.

5.3.1 Impact on Output Size. When applying the sampling scheme in probing, the order in which tuples arrive can impact the output. Using the traffic monitoring scenario described in Section 1 as an example, if the latency of the street detector's stream is less than that of the vehicle GPS streams, then σ plays a more significant role and determines the street-wise detection. Conversely, if the latency of the street detector's stream exceeds that of the vehicle GPS streams, σ' primarily attaches street information to vehicles and provides more about the mapping from location to vehicle. Therefore, we introduce an indicator variable $\xi_{r,s}$ to describe whether r arrives at the system before s, and thus $\xi_{s,r} = 1 - \xi_{r,s}$. Defining $\Xi_{r,s} = \sum_{r,s} I[r.k = s.k] \cdot \xi_{r,s}$, $\Xi_{s,r} = \sum_{r,s} I[r.k = s.k] \cdot \xi_{s,r}$, we proceed as follows:

Theorem 1. *Given ϵ_R, ϵ_S, and p, which determine σ_R and σ_S, let σ'_R and σ'_S be set with arbitrary λ_R and λ_S. The expectation of $|\Psi|$ is given by:*

$$E[|\Psi|] = \Xi_{s,r}(\epsilon_S - \frac{\epsilon_R\epsilon_S}{p})\lambda_R + \Xi_{r,s}(\epsilon_R - \frac{\epsilon_R\epsilon_S}{p})\lambda_S + \frac{\epsilon_R\epsilon_S}{p}J.$$

With $\xi_{r,s}$, the expected contribution of r and s to $E[|\Psi|]$ is given by $p(\xi_{r,s} \cdot q_R(q_S + (1-q_S)\lambda_S) + \xi_{s,r} \cdot q_S(q_R + (1-q_R)\lambda_R))$. For any given r and s, all coefficients in the aforementioned formula remain consistent except for $\xi_{r,s}$ and $\xi_{s,r}$. Given that $\Xi_{r,s} = \sum_{r,s} I[r.k = s.k] \cdot \xi_{r,s}$, $\Xi_{s,r} = \sum_{r,s} I[r.k = s.k] \cdot \xi_{s,r}$, and considering $J = \sum_{r,s} I[r.k = s.k]$ with $\xi_{s,r} = 1 - \xi_{r,s}$, we have $J = \Xi_{r,s} + \Xi_{s,r}$. Theorem 1 is subsequently derived by rearranging the formula to incorporate $\Xi_{r,s}$ and $\Xi_{s,r}$.

The last term in Theorem 1 implies that λ_R and λ_S are making positive contributions on the base of Lemma 1. Setting $\lambda_R = \lambda_S = 1$ in Theorem 1, we get $E[|\Psi|] = \epsilon_R\Xi_{r,s} + \epsilon_S\Xi_{s,r}$. Comparing it with Lemma 3, we find that the arrival order determines only the ratio of original join output size assigned to the two sampling rates, and the random assumption just makes them both to $\frac{1}{2}$. Finally, we can see that λ_R and λ_S linearly control $\Xi_{s,r}/J$ and $\Xi_{r,s}/J$ portion of the additional output size, respectively.

5.3.2 Impact on Variance. Variance analysis in general cases requires a general estimator first. In Theorem 1, $\Xi_{r,s} + \Xi_{s,r} = J$ implies that we need to take $\Xi_{r,s}$ and $\Xi_{s,r}$ into consideration when constructing the estimator. Abbreviating $\Xi_{s,r}/J$ and $\Xi_{r,s}/J$ as $\mu_{R,S}$ and $\mu_{S,R}$ respectively, the estimator can be given by:

$$\hat{J} = |\Psi|/(\mu_{S,R}(\epsilon_S - \frac{\epsilon_R\epsilon_S}{p})\lambda_R + \mu_{R,S}(\epsilon_R - \frac{\epsilon_R\epsilon_S}{p})\lambda_S + \frac{\epsilon_R\epsilon_S}{p}). \tag{1}$$

Afterward, $Var[\hat{J}] = (\hat{J}/|\Psi|)^2 Var[|\Psi|]$. The detailed format of the estimator's variance is given in Appendix A.5. Extracting its leading-order term, we have:

Theorem 2. *Given ϵ_R, ϵ_S, and p, which determine σ_R and σ_S, let σ'_R and σ'_S be set with arbitrary λ_R and λ_S. Without any assumption about the data, the leading-order term of the join size estimator's variance is given by:*

$$\frac{1-p}{p} \sum_{r,s,r',s'} I[r.k = s.k = r'.k = s'.k] \frac{f(\xi_{s,r}, \xi_{r,s})}{f(\mu_{S,R}, \mu_{R,S})} \frac{f(\xi_{s',r'}, \xi_{r',s'})}{f(\mu_{S,R}, \mu_{R,S})}, \tag{2}$$

where $f(x_1, x_2) = x_1 a_1 + x_2 a_2 + a_3$, and $a_1 = (\epsilon_S - \frac{\epsilon_R \epsilon_S}{p})\lambda_R$, $a_2 = (\epsilon_R - \frac{\epsilon_R \epsilon_S}{p})\lambda_S$, $a_3 = \frac{\epsilon_R \epsilon_S}{p}$.

Generally, we employ $X_{r,s}$ as the indicator variable to signify whether the match of r and s is in Ψ. Consequently, the corresponding term in $Var[|\Psi|]$ is $\sum_{r,s,r',s'} E[X_{r,s}X_{r',s'}] - E[X_{r,s}]E[X_{r',s'}]$. By expanding this and integrating it back into $Var[\hat{J}]$, we derive Theorem 2.

Considering the worst case, we give two unreachable upper bounds. The first bound is derived as follows: The relationship $\xi_{s,r} = 1 - \xi_{r,s}$ leads to the linearity of f, implying that $\frac{\epsilon_R \epsilon_S}{p} \le f(\xi_{s,r}, \xi_{r,s}) \le \max(\epsilon_R, \epsilon_S)$. Without loss of generality, let us assume that $\epsilon_R > \epsilon_S$. This allows us to conclude that $f(\xi_{s,r}, \xi_{r,s})/f(\mu_{S,R}, \mu_{R,S}) \le p/\epsilon_S$. Considering that μ represents the average of ξ, we can deduce that the leading-order term (2) is strictly less than $\frac{1-p}{p}\frac{p^2}{\epsilon_S^2}\gamma_{2,2}$, which forms the first bound.

The second bound is derived by applying the Chebyshev's sum inequality to (2), and we have:

$$(2) \le \frac{1-p}{p} \sum_{r,s} I[r.k = s.k] R[k] S[k] \left(\frac{f(\xi_{s,r}, \xi_{r,s})}{f(\mu_{S,R}, \mu_{R,S})} \right)^2. \tag{3}$$

By the positivity of $f(x_1, x_2)$, we have:

$$\sum_{r,s} I[r.k = s.k] \left(\frac{f(\xi_{s,r}, \xi_{r,s})}{f(\mu_{S,R}, \mu_{R,S})} \right)^2 \le \left(\frac{\sum_{r,s} I[r.k = s.k] f(\xi_{s,r}, \xi_{r,s})}{\sum_{r,s} I[r.k = s.k] f(\mu_{S,R}, \mu_{R,S})} \right)^2 = \gamma_{1,1}^2. \tag{4}$$

Refilling (4) back and considering the former bound together, we have $(2) < \frac{1-p}{p} \min(\gamma_{1,1}^2, \frac{p^2}{\epsilon_S^2}\gamma_{2,2})$. Meanwhile, experiments on real-world datasets indicate that the variance is relatively close to the random situation.

5.4 Aggregation

In this subsection, we expand FreeSam's approach to include aggregations. The simplified query holds a form of `SELECT AGG(R.v) FROM R join S on k`, and we could replace `AGG` with the aggregations we concern, including `COUNT`, `SUM`, and `AVG`. To avoid confusion, we use J_{AGG} and $\hat{J}_{\mathsf{AGG}}$ to denote the true value of the aggregation `AGG` and its corresponding estimator, respectively.

`COUNT`. The approximation for `COUNT` reflects the cardinality of the original join output, which aligns with our objective of estimating output size. Consequently, $\hat{J}_{\mathsf{COUNT}}$ is equivalent to $\hat{J}$ in (1), and it can be directly applied to estimate `COUNT`. Additionally, the variance analysis for this estimation aligns with those described in Theorem 2.

`SUM`. The aggregation for `SUM` can be theoretically regarded as a weighted version of `COUNT`, and thus its estimator and variance share a similar form to that of `COUNT`. We give the form of them in Appendix A.6, and a sketch is as follows:

Let Ψ_{SUM} be the direct `SUM` of FreeSam's output data, and $\Xi_{r,s}^{(s)} = \sum_{r,s} I[r.k = s.k] \cdot \xi_{r,s} \cdot r.v$, $\Xi_{s,r}^{(s)} = \sum_{r,s} I[r.k = s.k] \cdot \xi_{s,r} \cdot r.v$. Thus, from the property $J_{\mathsf{SUM}} = \Xi_{r,s}^{(s)} + \Xi_{s,r}^{(s)}$, by defining $\Xi_{s,r}^{(s)}/J_{\mathsf{SUM}}$ and $\Xi_{r,s}^{(s)}/J_{\mathsf{SUM}}$ as $\mu_{R,S}^{(s)}$ and $\mu_{S,R}^{(s)}$ respectively, the unbiased estimator can be given by $\hat{J}_{\mathsf{SUM}} = \Psi_{\mathsf{SUM}}/(\mu_{S,R}^{(s)}(\epsilon_S - \frac{\epsilon_R \epsilon_S}{p})\lambda_R + \mu_{R,S}^{(s)}(\epsilon_R - \frac{\epsilon_R \epsilon_S}{p})\lambda_S + \frac{\epsilon_R \epsilon_S}{p})$, and similarly, the leading-order term of its variance is $\frac{1-p}{p} \sum_{r,s,r',s'} I[r.k = s.k = r'.k = s'.k] r.v f(\xi_{s,r}, \xi_{r,s}) \cdot r'.v f(\xi_{s',r'}, \xi_{r',s'}) / f(\mu_{S,R}^{(s)}, \mu_{R,S}^{(s)})^2$.

AVG. Unlike COUNT and SUM, the complexity of handling reciprocals makes designing a completely unbiased estimator for AVG challenging, since the expectation of a random variable's reciprocal and the reciprocal of its expectation are not the same. Fortunately, the estimator formulated as $\hat{J}_{\mathsf{AVG}} = \hat{J}_{\mathsf{SUM}}/\hat{J}_{\mathsf{COUNT}}$ generally performs well and is almost unbiased [24]. Additionally, as SUM and COUNT are highly related, the variance $Var[\hat{J}_{\mathsf{AVG}}] = Var[\hat{J}_{\mathsf{SUM}}/\hat{J}_{\mathsf{COUNT}}]$ may still maintain high quality.

5.5 Inferring the Latency Cost

Different configurations of sampling rate and probe utilization not only affect the quantity and quality of outputs but also introduce varying levels of latency. In order to return timely results, it is necessary for an approximate system to know the expected latency for a specific parameter setting. In Section 5.5.1, we introduce an analytical cost model for sampling-aware *IaWJ*. In Section 5.5.2, we discuss how to build a tiny but useful neural network based on the analytical method.

5.5.1 Sampling-Aware Cost Model. By employing the symmetric algorithmic procedure, we can construct the cost model consisting of two parts, denoted as $C_{R\bowtie S} = C_{R\rtimes S} + C_{R\ltimes S}$. Upon resolving $C_{R\rtimes S}$, we can subsequently deduce $C_{R\ltimes S}$. For uniformity, the base unit of all kinds of cost we mention is time.

Based on the hash join structure, we denote the cost of completing a single probing and building by c_{probe} and c_{build} in an average sense. Thus, the problem is converted to determine the number of probing and building operations. The building phase is conducted by every tuple stored and controlled by σ_S. Thereby, the number is $\epsilon_S|S|$. Making $P_R = \epsilon_R + (p - \epsilon_R)\lambda_R$ to represent the effective probing ratio, $P_R|R|$ will be the number of tuples to probe. Moreover, the probing phase also requires checking the tuples in the retrieving bucket, and for the given bucket number B, the average number of tuples in each bucket is $\epsilon_S|S|/B$. Finally, the cost of $C_{R\rtimes S}$ is given by:

$$C_{R\rtimes S} = P_R|R|\cdot\frac{\epsilon_S|S|}{B}\cdot c_{probe} + |\epsilon_S S|\cdot c_{build}. \tag{5}$$

However, (5) solely represents the ideal scenario where the machine processes tuples continuously, disregarding any idling periods. For example, intermittent arrivals of IoT data can cause frequent switching between active and idle states. Therefore, it becomes necessary to detect the flux of tuples, i.e. the input amount, within the progressive procedure. To accomplish this, we employ the finite difference to measure the flux within small subintervals of the window. Specifically, the window w is logically divided into n consecutive pieces, $\Delta_1, \Delta_2, \ldots, \Delta_n$. When entering Δ_i, there are two parts of workload: the remaining workload inherited from Δ_{i-1} denoted by Z_{i-1}, and the newly arrived workload denoted by C_{Δ_i}. Assuming a constant processing velocity $v_{process}$, the workload resolved within Δ_i is given by $\Delta_i \cdot v_{process}$. Consequently, the remaining workload after Δ_i is calculated as $Z_i = \max(0, Z_{i-1} + C_{\Delta_i} - \Delta_i \cdot v_{process})$. By converting the concept of Z_i and C_{Δ_i} into time cost, (6) provides a recursive approach for determining Z_i.

$$Z_i = \max(0, Z_{i-1} + C_{\Delta_i} - \Delta_i). \tag{6}$$

Accordingly, Z_n represents the cost that remains after the arrival of all tuples, i.e., the latency. In the expression, C_{Δ_i} is inferable with (5) by applying the number of tuples arrived within Δ_i to it. By setting checkpoints along the stream, and recording the number of tuples cumulatively at these points, we can simulate the finite difference method. Theoretically, the smaller Δ_i is, the more accurate the cost inference is. However, for faster inference against the load of recording, Δ_i is better to be relatively large. For example, we use $|\Delta|$=100ms for practice.

5.5.2 Inference via Neural Network. Informed by (5), we aim to model both the multiplicative and additive items. Furthermore, the structure outlined in (6) resembles a piecewise function, suggesting

a segmented approach to modeling. To this end, we employ a two-layer Multi-Layer Perceptron (MLP) to encapsulate the items described in (5) and integrate the flux information at checkpoints as features, in line with (6). Specifically, we prepare the feature with both the flux amount values and their logarithms. This dual representation enhances the model's ability to capture the multiplicative items and address sublinear variations. The MLP's hidden layer adopts the *ReLU* activation function to effectively mirror the piecewise characteristic. With a lean architecture comprising 200 neurons, the model guarantees a swift response. Finally, to measure the accuracy of the inference, we use MSE as the loss function.

5.6 Parameter Setting

We provide a detailed procedure for appropriately setting parameters based on user requests. Users often prioritize certain aspects among accuracy, maximum data volume, and latency budget. Therefore, the basic idea of our solution is to first satisfy the accuracy requirement and then balance the achievable data volume within the constraints of latency budget. When a user is concerned with only specific metrics, we can simplify the parameter setting process by omitting steps related to less relevant metrics.

Users typically can directly specify the latency budget, the sampling rate (ϵ) or desired accuracy, and whether they prefer raw data output, which generally implies a preference for obtaining as much data as possible over mere aggregation estimates. If ϵ is specified, the accuracy is directly bounded. In cases where ϵ is not specified, we replace p with ϵ in Theorem 2 to construct an inequality and derive the appropriate ϵ from it. Given that $p \leq \epsilon$, the accuracy inequality remains valid after establishing p. With ϵ set, we then adjust λ to utilize any remaining latency budget to produce additional tuples. To determine the maximum λ, we consult (5) and (6) in Section 5.5. In situations where flux information is unavailable, we can rely on user-provided prior knowledge or assume a worst-case scenario in which all tuples arrive towards the end of the window. Moreover, to further reduce variance, we configure p with a presample at the beginning of the window.

Configuration of p. The fluctuation of the variance caused by the arrival order has been demonstrated in Theorem 2. However, in the absence of arrival information, setting the value of p still requires a random assumption. To simplify the problem, we can leverage Lemma 2 and consider the impact of p on the coefficient magnitudes. The derivative of p can then be calculated as: $\sqrt{\epsilon_R \epsilon_S(\gamma_{2,2} - \gamma_{1,2} - \gamma_{2,1} + \gamma_{1,1})}/\gamma_{1,1}$. It is important to note that p should not exceed 1 and should be greater than or equal to the larger one of ϵ_R and ϵ_S. A more detailed analysis is provided in Appendix A.6. Another challenge arises from unknowing constants such as $\gamma_{2,2}$ at the start of the stream. To address this, we employ a count-based presampling technique to capture the characteristics of the initial stream and use it as an indicator for the subsequent data. We propose two variants of FreeSam based on two heuristic options for interpreting the presample:

1) FreeSam^O. A natural heuristic is to assume that a segment of the partial stream shares the same properties as the entire stream. With the "representativeness heuristic", FreeSam^O operates under the assumption that the ratio between the norms of keys remains relatively stable. It computes the γ values of the presample and directly applies them to the derivative to deduce the extreme point and set p. Specifically, the system captures the first k tuples that arrive and calculates the γ values using these tuples to immediately set p for later processing. Additionally, these k presample tuples are resampled with the newly set parameters, which is almost costless since $k \ll |R|+|S|$.

2) FreeSam^B. An alternative approach is to treat the first k presample as a Bernoulli sampling and use it to estimate the values of γ. This approach is statistically and theoretically sound under the random arrival assumption. However, the random arrival assumption seldom holds true in real-world situations, which is why we still categorize it as a heuristic. As discussed in Section 2.1,

a Bernoulli sampling rate of q tends to result in an output join size of q^2 compared to the original data, whose effect is equivalent to $\gamma_{1,1}$. By applying the same technique, we can estimate the values of $\gamma_{i,j}$ using q^{i+j} as the denominator. Consequently, $\gamma_{2,2}$ will have a larger scaling factor than $\gamma_{1,1}$, $\gamma_{1,2}$, and $\gamma_{2,1}$. This leads to a higher derivative value in the calculation of p compared to the strategy used in FreeSam^O. As a result, with a higher p, FreeSam^B behaves more similarly to Bernoulli sampling, resulting in lower variance. On the other hand, FreeSam^O is closer to universe sampling. We evaluate both variants of FreeSam in our experiments.

Special cases of λ configuration. There is a particularly useful technique to avoid the fluctuation brought about by the arrival order. Making $(\epsilon_S - \frac{\epsilon_R\epsilon_S}{p})\lambda_R = (\epsilon_R - \frac{\epsilon_R\epsilon_S}{p})\lambda_S$, then $f(\xi_{r,s}, \xi_{s,r})$ stays unchanged. The simplest setting to achieve this is quite common, i.e., making $\epsilon_R = \epsilon_S$ and $\lambda_R = \lambda_S$.

6 Implementation

We achieve practical usability and parallelism by performing FreeSam on multicores. Using the Join-Matrix (JM) partition strategy, we assign sub-matrices to cores to strengthen spatial locality. Moreover, to optimize SIMD utilization, we introduce an AVX-accelerated pseudo-random number generator (PRNG). For more detailed information on the implementation, due to space limitations, please refer to Appendices A.8 and A.9.

7 Evaluation

7.1 Experimental Setup

Evaluated methods. To verify the advantages of FreeSam, we compare FreeSam with three baselines. **Universe** [122] sampler is practically applied in previous works like Quickr [68] for larger output size. **Bernoulli** [132] sampling promises the uniformity of sampling and is still the prime sampling method in Flink [39] and Spark [110]. **UBS** [59] is a hybrid method aimed at optimal variance, similar to the minimum probe utilization case mentioned in Section 5.2. Furthermore, it is allowed to use the overall statistics, including $\gamma_{2,2}$, $\gamma_{2,1}$, etc. For a fair comparison, we conduct a fast parallel implementation of all methods above based on the study [140]. In evaluating FreeSam, we assess its two variants: **FreeSam^O** and **FreeSam^B**. These variants are derived from different approaches of perceiving the presampled data used for parameter setting, as discussed in Section 5.6. By default, $\epsilon_R = \epsilon_S$ and $\lambda_R = \lambda_S$, and we compare two algorithms using the average ratio of their results at the same sampling rate and probe utilization. Without causing ambiguity, we use the term variance to abbreviate the variance of the estimator's relative error.

Platform. We conduct experiments on an Intel Core i9-10900X platform. The CPU has ten physical cores supporting 20 threads, along with the SSE4.2, AVX2, and AVX-512 instruction set extension. The platform is equipped with four 32GB DDR4 memories at 2.4GHz.

Datasets. We use three real-world datasets for evaluation. The first dataset is *Rovio* [70], which joins advertisement streams and purchase streams to determine advertisement revenue. The second dataset is *DEBS* [50], which focuses on joining posts and comments from users on social networks to analyze user behavior. Notably, the timestamps of DEBS data range from 2010 to 2012, prompting us to consider it a static dataset. Consequently, we process it while preserving the logical order of arrival. The third dataset is *EECR* [54], which joins sensor data from the same location to analyze weather conditions. All datasets have been widely used in previous studies [28, 120, 140]. Table 3 summarizes the characteristics of the three real-world datasets, including key skewness ($skew_{key}$), timestamp skewness ($skew_{ts}$), arrival rate of the input (v), and average number of duplicates per key ($dupe$).

Table 3. Statistics of the three real-world workloads (w=1000ms).

	Arrival rate (tuples/ms)	Key duplication	Key skewness (Zipf)	Number of tuples
Rovio	$v_R = v_S \approx 2873$	$dupe(R) = dupe(S) \approx 17960$	$skew_{key}(R) = skew_{key}(S) \approx 0.042$	$\|R\|(\|S\|) \approx v_{R(S)} \cdot \|w\|$
DEBS	$v_R = v_S = \infty$	$dupe(R) \approx 172, dupe(S) \approx 111$	$skew_{key}(R) \approx 0.003, skew_{key}(S) \approx 0.011$	$\|R\| = \|S\| = 10^6$
EECR	$v_R \approx 1013, v_S = \infty$	$dupe(R) \approx 39.6, dupe(S) \approx 41.1$	$skew_{key}(R) \approx 0.073, skew_{key}(S) \approx 0.072$	$\|R\|(\|S\|) \approx 10^6$

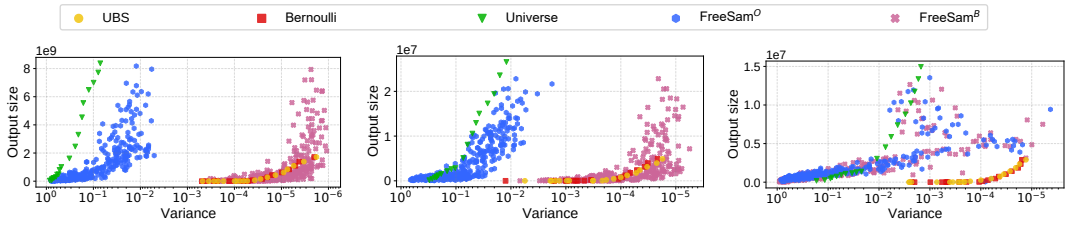
7.2 Quantity and Quality

As discussed in Section 4.3, we use the output size and variance of the estimator of original join output size to measure the quantity and quality of the result. For the baselines, the values of the two metrics are bounded together, and thus the algorithms are likely to perform well on only one of the metrics. FreeSam shows that it is possible to decouple the values of the metrics and achieve a two-dimensional value range, which can make the algorithm adaptive to applications concerning any or both of the metrics. Figure 4 shows how the output size and the variance change when varying the effective sampling rate from 0.001 to 0.1 and varying the probe utilization from 0.01 to 0.9. The x-axis represents the variance of the percentage estimation error (relative error). The y-axis represents the generated output size. A result is better if it is closer to the top right of the diagram. Sampling rate and probe utilization are omitted in Figure 4 since the major concern is how the methods perform on the two metrics.

Result. We can obtain the following results. First, beyond the baseline methods, the two FreeSam methods are capable of spanning into a two-dimensional space on the two metrics. For example, when fixing a 10^{-3} magnitude variance, FreeSam's output size is able to vary from 9.49×10^5 to 1.35×10^7, while the baselines can produce only one value. It is λ (probe utilization) that empowers this spanning ability. Second, the outputs of FreeSam are roughly in the range outlined by the other methods and are closer to the top right of the diagram. More precisely, Universe forms a boundary closer to the output size axis, while UniSample and UBS form a boundary closer to the variance axis. When linearly linking the points, 58.15% of the outputs of FreeSam lie in the range. Third, FreeSam^O is more likely to have a larger output size while FreeSam^B achieves a smaller variance, which is conspicuous in Figure 4 (a) and Figure 4 (b) where they divide FreeSam to two parts. For example, comparing FreeSam^O with FreeSam^B on Rovio, FreeSam^O generates 3.51× the output size while making the variance 4134.43× larger.

Analysis. To better show the impact of introducing probe utilization (λ), we compare the average output size generated at the same level of variance in terms of powers of 10. In terms of datasets, FreeSam achieves 8.01×, 1620.47×, and 1057.02× output size compared to the baselines on Rovio, DEBS, and EECR, respectively. In terms of baselines, FreeSam achieves 1.02×, 2121.20×, and 554.51× output size compared with Universe, Bernoulli, and UBS, respectively. It is worth noting that the universe sampler in Universe is the theoretical optimal method without probe utilization for the output size metric [59], and FreeSam gains a similar output size with it while shrinking the geometric average of variance better from 7.54×10^{-2} to 5.80×10^{-3}. The geometric average of variances of Bernoulli and UBS is 1.22×10^{-3} and 1.18×10^{-3}, respectively, which is at the same level with FreeSam, corresponding to Section 5.3.2. As to comparison between the two FreeSam methods, Figure 4 shows that the value of FreeSam^O is closer to Universe and the value of FreeSam^B is closer to Bernoulli and UBS. The cause is the different settings of p, which is the only difference between them. For example, on Rovio in Figure 4 (a), the average p of FreeSam^O and FreeSam^B is 0.126 and 1.0, respectively. A smaller p is prone to produce a larger output size and higher variance. Despite the parameter setting scheme, the essential cause of a different p lies in the data distribution. The $dupe(R)$ of Rovio, DEBS, and EECR is 17960, 171, and 39.6, which are in a descending relationship. The smaller $dupe(R)$ can make $\gamma_{1,1}$, $\gamma_{1,2}$, $\gamma_{2,1}$, and $\gamma_{2,2}$ smaller, implying that the p for the lowest variance is smaller according to the formula in Section 5.6. Then, it indicates that the variance of the universe sampler becomes closer to UBS (the hybrid method), thus making the two boundaries

closer on the x-axis in the three subgraphs of Figure 4 from left to right. Meanwhile, the inevitable estimation biases of p caused by the unpredictable streaming data also account for the presence of the minor fraction of results inferior to the baselines. Another noticeable phenomenon is that FreeSam^O and FreeSam^B perform similar in Figure 4 (c). More precisely, when we compare FreeSam^O with FreeSam^B on EECR, FreeSam^O generates just 1.11× the output size while making the variance only 2.49× larger. The reason for this is that the velocity of the two streams ($v_R \approx 1013, v_S = \infty$) is extremely uncoordinated, which causes the presampling to underestimate γs, leading to a similar parameter setting for both algorithms. In general, FreeSam^O is better when a larger output size is more desirable, while FreeSam^B is closer to achieving a lower variance.



(a) Rovio (b) DEBS (c) EECR

Fig. 4. Value ranges of two metrics (output size and the variance of the estimator of original join output size) on real-world datasets.

7.3 Latency

The three baselines use separate sampler and joiner while FreeSam fuses the operators. The efficiency of coupling sampler and joiner in stream data processing needs to be verified. Thus, in this part, we first show the effect of coupling sampler and joiner in FreeSam by comparing the 95th latency with the baselines. We compare the latency of the algorithms on the three datasets in Figure 5. The x-axis varies the sampling rate from 0.01 to 0.8 without changing the probe utilization ($\lambda = 0$). The y-axis is the 95th latency in a logarithmic scale. FreeSam^O and FreeSam^B have similar latency results, so we denote their average as FreeSam to save space.

Compared to the baselines, FreeSam reduces the latency by an average of 61.71% and 84.58% on Rovio and EECR, respectively. The significant retrenchment of processing time proves that coupling the sampler and joiner does improve the efficiency of sampling-based *IaWJ*. However, FreeSam is not the optimal algorithm on the DEBS dataset. The reason is that the window size of DEBS is 0 ($|w| = 0$ms). It implies that DEBS is a stale dataset and is actually more like a database scenario. We still use DEBS as a test case, since it can be called a stream with infinite tuple arriving velocity ($v = \infty$) in an extended sense. Therefore, in the case of stale data, more intense processing brought by separating the operators makes Bernoulli and UBS the better choice. Notably, at low sampling rates of 0.01 and 0.04 on Rovio and EECR, the latency of FreeSam is less than 1 ms. The reason is that the coupled implementation allows the entire procedure to be done within a window, which is especially useful in applications with small sampling rates.

7.4 Impact of λ and ϵ

In Section 7.4.1, we use DEBS, whose window size is 0, to analyze the net latency cost of using the probe utilization and its co-effect with sampling rate when setting the same parameters between R and S. In Section 7.4.2, we use EECR to evaluate the impact on result quantity and quality when varying the sampling rates between R and S, which further explains the co-effect between λ and ϵ, and verifies the correctness of the theorems in Section 5.3.

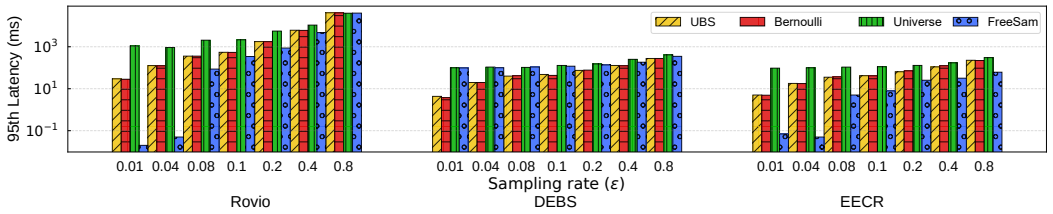


Fig. 5. Latency comparison of the algorithms with varying sampling rates on the three real-world datasets.

7.4.1 Costs and Benefits of λ. As mentioned in Section 7.3, DEBS, as stale data, expels the fluctuation of stream data and exposes the worst latency cost of using λ. In Figure 6, we show how the latency changes when varying the probe utilization (λ). We also vary the sampling rate (ϵ) to determine the co-effect between ϵ and λ. The corresponding output size is shown in Figure 7. We use FreeSam^O since it is proved to be apt to generate a larger output size, which makes the time consumption more conspicuous.

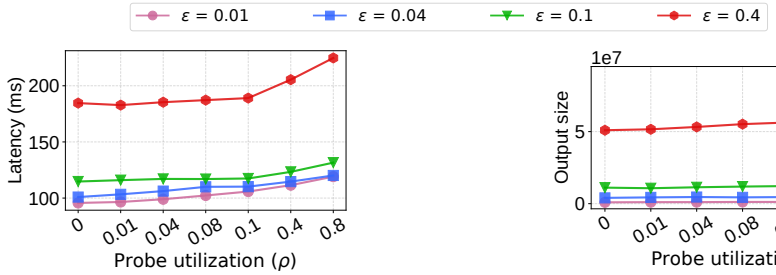
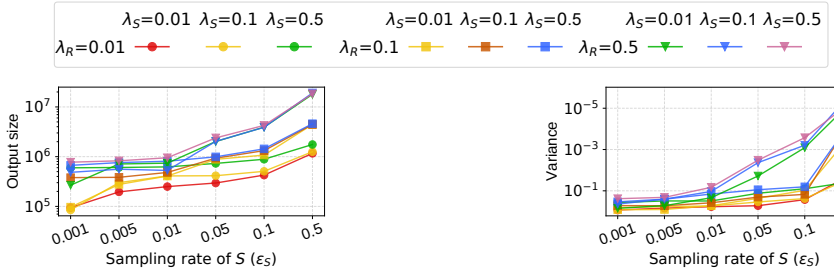


Fig. 6. Latency of FreeSam^O with varying probe utilization on DEBS.

Fig. 7. Output size of FreeSam^O with varying probe utilization on DEBS.

We use Δ_{Lat} to refer to the ratio of increased latency compared with the $\lambda = 0$ condition. Similarly, $\Delta_{|\Psi|}$ refers to the ratio of increased output size compared with the $\lambda = 0$ condition. The overall average of Δ_{Lat} and $\Delta_{|\Psi|}$ is 7.81% and 33.45%, respectively, from which we can observe the advantage of introducing probe utilization in achieving a large output size with little latency cost. Moreover, we can analyze the effect per unit of λ by the average of Δ_{Lat}/λ and $\Delta_{|\Psi|}/\lambda$. Thereby, the result $\text{avg}(\Delta_{Lat}/\lambda) = 0.57$ and $\text{avg}(\Delta_{|\Psi|}/\lambda) = 2.19$ means that for every 1% we add to λ, it brings 2.19% increase in output size while costing only 0.57% more latency. We can also say that FreeSam trades off 3.83% of the output size for every 1% increase in latency. To understand the co-effect of the sampling rate (ϵ) and probe utilization (λ), we compare the effect of λ with varying ϵ. In Figure 7, when varying ϵ in 0.01, 0.04, 0.1, and 0.4, the corresponding average $\Delta_{|\Psi|}$ brought by λ is 44.03%, 38.48%, 27.46%, and 23.82%, which is in a descending relationship. We can use Theorem 1 in Section 5.3.1 along with the parameter setting scheme in Section 5.6 to elucidate this phenomenon. By Theorem 1, when we set $\epsilon_R = \epsilon_S = \epsilon$ and $\lambda_R = \lambda_S = \lambda$, the expectation of $\Delta_{|\Psi|}$ is $E[\Delta_{|\Psi|}] = (\frac{p}{\epsilon} - 1)\lambda$. Regardless of p, the coefficient of λ, which is $(\frac{p}{\epsilon} - 1)$, descends with an ascending ϵ. When we take p into consideration, by the parameter setting scheme, p either scales larger with ϵ or touches the ceiling of 1. Thus, $(\frac{p}{\epsilon} - 1)\lambda$ monotonically decreases as ϵ increases. Besides, the overall decreasing speed is also determined by the sampling stratums, since the point that p touches the ceiling may be different in the stratums. Therefore, the varying p in stratums and the offset 1 in $(\frac{p}{\epsilon} - 1)\lambda$ explain why the output size is not totally decreasing proportionally to ϵ. When we exactly calculate the ratio of $\Delta_{|\Psi|}$ to $(\frac{p}{\epsilon} - 1)\lambda$, it achieves 0.82 at $\epsilon = 0.4$, which is close to 1 and guarantees an accurate estimator. Meanwhile, it is notable that the ratio has achieved this level of precision even with only one random set of data. This experiment provides a strong corroboration of the foregoing theoretical derivation.

7.4.2 Impact of Varying Sampling Rate ϵ. By varying the sampling rate, we change the sample size that resides in memory, such as the size $\epsilon_R|R|$. In this experiment, we make $\epsilon_R|R|$ and $\epsilon_S|S|$ different by fixing ϵ_R to 0.01 and varying ϵ_S from 0.001 to 0.5 as well as varying λ_R and λ_S from 0.01 to 0.5. We conduct this experiment on EECR, and show the quantity and quality change in Figure 8. Since $|R|$ and $|S|$ are similar in EECR, we can approximately replace the size relationship, $\epsilon_R|R|$: $\epsilon_S|S|$, with $\epsilon_R : \epsilon_S$. The varying sampling rate of S (ϵ_S) forms the x-axis, and the metrics, output size and variance, form the y-axis in Figure 8 (a) and Figure 8 (b), respectively. Each line has a different setting of λ_R and λ_S, and we show their legends above Figure 8. The lines with the same $\lambda_R \cdot \lambda_S$ share the same color. The lines with the same λ_R share the same marker.



(a) Output size with varying effective sample size and λ. (b) Variance with varying effective sample size and λ.

Fig. 8. Quality changing with varying effective sample size and λ on EECR.

When ϵ_S is fixed to 0.5, it is noticeable that the lines stratify by different λ_R (different markers) in both Figure 8 (a) and Figure 8 (b). More precisely, when we vary λ_R in 0.01, 0.1, and 0.5, the average output size varies in 1.39×10^6, 4.49×10^6, and 1.85×10^7, while the geometric average variance varies in 2.36×10^{-2}, 6.20×10^{-5}, and 3.51×10^{-6}, respectively. It is conspicuous that the probe utilization helps in both of the metrics in join between large and small streams. The aforementioned theorems can explain this phenomenon. When λ is 10× greater than ϵ_R ($\lambda_R \gg \epsilon_R$), $\epsilon_S \cdot \lambda_R$ begins to occupy the vast majority of the join results according to Theorem 1 and thus significantly increases the output size. Meanwhile, the large $\epsilon_S \cdot \lambda_R$ plays a dominant role in $f(\mu_{r,s}, \mu_{s,r})$ and $f(\xi_{r,s}, \xi_{s,r})$, and makes their ratio more stable inside the variance. According to Theorem 2, a more stable ratio reduces the fluctuation caused by the arrival order and randomness of sampling, thus making the variance better. When there is not a high λ_R, the dominant factor falls back to $\epsilon_R = 0.01$ and brings a smaller output size with a larger variance, which is explainable as well by inversely applying the theorems.

7.5 Latency Inference

In Figure 9, we show the inference made by the analytical method in Section 5.5.1, and the tiny neural network in Section 5.5.2 along with the real latency value on the Rovio dataset. The three subfigures compare the inference accuracy with varying probe utilization. We present the result only on Rovio because it is a heavier workload and shows a more visible demonstration of the discrepancy in accuracy between the inference methods.

We use the average of relative error to judge the accuracy. The average accuracy of the analytical cost model on the three datasets is 88.05%, and the tiny neural network improves it to 96.75%. On Rovio, the accuracy of the analytical method and the neural network is 78.51% and 92.95%, respectively. Accordingly, we claim the analytical cost model usable and verifiable, and that the neural network is an even better solution. For a more detailed analysis, we find that the accuracy varies on smaller and larger sampling rates. For sampling rates smaller than 0.1, the accuracy of the two methods on the selected points in Figure 9 is only 27.56% and 39.81%. The reason is that when

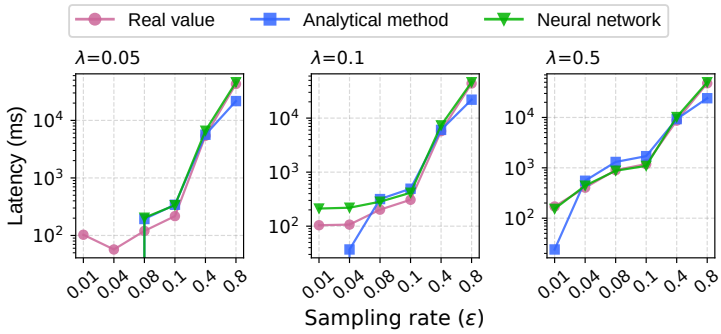


Fig. 9. Comparision of the inference with the real latency on Rovio.

the latency is small, even a small fluctuation can bring a large relative error, and their absolute errors are only 136.35 ms and 69.68 ms, which is totally acceptable for the window $|w| \geq 1000$ ms. Moreover, the latency of setting $\lambda = 0.05$ and $\epsilon = 0.01$ is 45.86 ms larger than that of $\lambda = 0.05$ and $\epsilon = 0.04$, which shows that the latency is prone to fluctuate at a relatively low processing burden.

7.6 Aggregation

We conduct aggregations, including `COUNT`, `SUM`, and `AVG`, on the cloud count field of EECR dataset to showcase the ability of FreeSam to scale on aggregation queries.

Figure 10 shows the direct comparison between one-shot approximations and ground-truth values, along with the variance of relative error. For all the queries analyzed, it is evident that the unbiased estimations closely approximate the actual values, and the variance consistently decreases as the sampling rate increases. Specifically, at a sampling rate of 0.01, the average accuracy for `COUNT`, `SUM`, and `AVG` reaches 96.09%, 92.18%, and 93.85%, respectively.

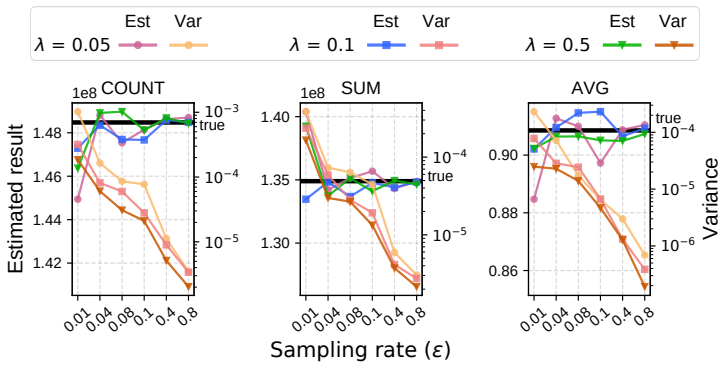


Fig. 10. Comparison of estimated and true aggregation values, along with the variance of relative error, on the EECR dataset.

For `GROUP BY` clauses, Figures 11 and 12 respectively show the percentage of groups identified from the ground truth and the associated query error. The term "group percent" refers to the proportion of groups correctly retrieved in comparison to the ground-truth result [11]. The error for an individual group is calculated as $1\text{-}e^{(\hat{x}-x)/x}$, while the overall query error is determined by the mean of these individual errors [47]. The adjustment of λ significantly influences the retrieval rate of groups. For instance, at a sampling rate of 0.01, adjusting λ to 0.5 results in obtaining 33.68% more groups compared to setting λ to 0.05. Nonetheless, without specific optimizations for `GROUP BY`, even at a sampling rate of 0.8, FreeSam still averages over a 10% error rate for each group.

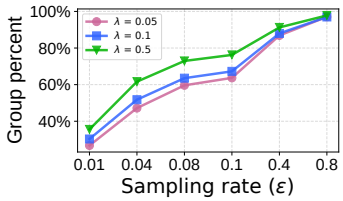


Fig. 11. Group percentage across different sampling rates and probe utilization on the EECR dataset.

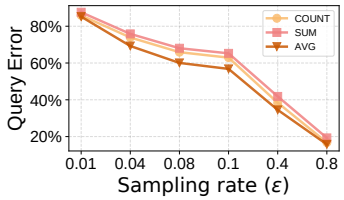


Fig. 12. Query error analysis across varying sampling rates and probe utilization on the EECR dataset.

7.7 Impact of Stream Data Features

We examine the effects of specific stream data characteristics on FreeSam, focusing on data drifting and windowing. These aspects are detailed in Sections 7.7.1 and 7.7.2, respectively. The volume of synthetic data generated for this study is on the order of 10^6.

7.7.1 Impact of Data Drifting. To examine the impact of varying data drifts, we incorporate the patterns of sudden, gradual, and incremental drift [27, 61]. We simulate data drifting by combining two synthetic datasets according to the three drift patterns. In Figure 13, we compare the accuracy of latency inference with and without data drifting, and the feature of the synthetic data chronically changes from $dupe(R)$=$dupe(S)$=10^4 and $skew_{key}(R)$=$skew_{key}(S)$=0.05 into $dupe(R)$=$dupe(S)$=10^3 and $skew_{key}(R)$=$skew_{key}(S)$=0.5. Meanwhile, the stability of the inference models is enhanced by incorporating output flux information within the window. In Table 4, we show the impact of data drifting on parameter setting, output size, and variance when fixing ϵ=0.01, λ = 0.1 and chronically changing the data feature from $dupe(R)$=$dupe(S)$=10 into $dupe(R)$=$dupe(S)$=10^3.

From analyzing the comparison between real latency and the predictions made by the analytical and neural network models across different drift scenarios in Figure 13, it is observed that data drifting does impact the performance of the inference models, but the accuracy remains within an acceptable range. The lowest accuracy observed across all drift patterns is 69.20% for the analytical model and 87.69% for the neural network model. Specifically, the accuracy of the analytical model decreases from 81.07% to 78.36%, 77.94%, and 69.20% for sudden, gradual, and incremental drifts, respectively. Meanwhile, the neural network model demonstrates a stable performance, with its accuracy declining from 95.43% to 89.27%, 87.69%, and 94.45% for sudden, gradual, and incremental drifts, respectively.

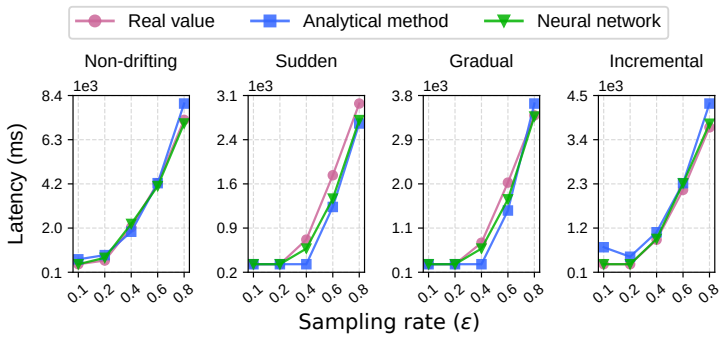


Fig. 13. Comparison of inference accuracy against real latency across different types of data drifting.

Table 4 shows the average settings of p under different drift scenarios, along with their corresponding output sizes and variances. The configurations under sudden and gradual drifts remain

similar to the non-drifting condition, as they exert minor impact on the early part of the stream, thus barely influencing the presample used for parameter setting. On the other hand, incremental drift exerts a more pronounced influence due to its more discrete and random distribution. The findings indicate that data drifting indeed affects the algorithm's behavior; however, the assumption-free approach enhances the stability and keeps the variance at a sampling rate of 0.01 within a level of 10^{-4}. Even more encouragingly, it achieves a lower level of 10^{-5} when the setting procedure is conscious of the incremental drift.

Table 4. Average p values, output size, and variance across varying patterns of data drifting.

Data	Non-drifting	Sudden	Gradual	Incremental
Average of p	0.722	0.729	0.727	1.0
Output size	1.11×10^4	5.52×10^5	5.66×10^5	5.53×10^5
Variance	6.60×10^{-6}	5.39×10^{-4}	2.93×10^{-4}	3.05×10^{-5}

7.7.2 Impact of Windowing. In the absence of a fixed window size in real-world datasets, we explore the impact of varying windowing conditions across the streams. Specifically, by setting ϵ and λ both to 0.1 and the window length of $|w_R|$ at 1000ms, while varying the ratio of window lengths between the two streams ($|w_R|$:$|w_S|$) to 1:1, 2:1, 4:1, 8:1, and ∞:1 (with $|w_S|$=0), we generate synthetic data characterized by $dupe(R)$=$dupe(S)$=10^3 and $skew_{key}(R)$=0.05, $skew_{key}(S)$=0.5. In Figure 14, we show the CPU cycles per tuple incurred during the build and probe procedures, as measured using Intel® PCM. Table 5 outlines the parameter settings, output sizes, and variance for the respective window ratios.

In Figure 14, there is a noticeable variation in the cycles consumed in the build and probe procedures as the window ratio changes. One notable trend is the continuous decrease in the cycles required for build per tuple, dropping from 64.61 to 41.75. This reduction is attributed to the shorter window, which facilitates more intensive processing and mitigates inefficiencies related to transitioning from idle states caused by sporadic tuple arrivals. Another observation is the two abrupt increases in the cycles for probe per tuple: from 22.04 to 41.93 and from 38.39 to 46.73, corresponding to window ratio changes from 1:1 to 2:1 and from 8:1 to ∞:1, respectively. However, the cycles needed at window ratios of 2:1, 4:1, and 8:1 remain relatively stable, recorded at 41.93, 38.32, and 38.39, respectively. This phenomenon is the result of a combination of three factors: First, as indicated in Table 5 and Section 5.5, an increase in the p value escalates the proportion of tuples that require probing. Second, at the two major transitions, tuples with a higher $skew_{key}$ from stream S tend to arrive much earlier, leading to a more populated hash table and, consequently, an increased average number of buckets that need to be probed. Third, the intensive processing associated with short windows helps counterbalance partial increases brought on by the aforementioned two factors.

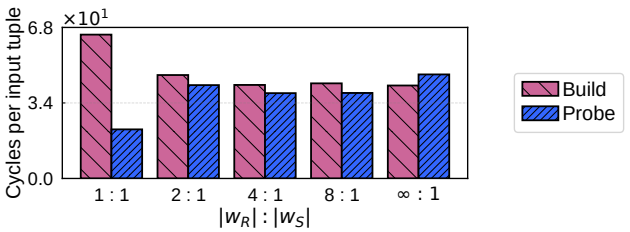


Fig. 14. Cypcles per input tuple when changing the ratio between the length of two windows.

Table 5 also sheds light on the impact of windowing on variance, output size, and, more essentially, parameter settings. A noteworthy observation is the emergence of a unimodal pattern in the p

values as the window ratio changes, with the peak of this pattern occurring near a window ratio of 8:1. This pattern arises because the γ values, which are used to set p, adjust in response to an increased influx of tuples from stream S relative to R, as the widening $|w_R|$:$|w_S|$ ratio accelerates the stream speed of S. The underlying principle here is the mean inequality chain, which elucidates the unimodal nature of the product of two variables when their sum is constant. In this context, the two variables are the counts of tuples from streams R and S in the presample, while the instances of the multiplicative factors are represented by the γ values. Consequently, both the output size and variance exhibit similar unimodal patterns, illustrating how the dynamics of windowing intricately influence these aspects of the system's performance.

Table 5. Average p values, output size, and variance across varying ratios between the window length of R and S.

$\lvert w_R\rvert$: $\lvert w_S\rvert$	1 : 1	2 : 1	4 : 1	8 : 1	∞ : 1
Average of p	0.682	0.841	0.967	1.0	0.931
Output size	2.32×10^7	2.04×10^7	1.92×10^7	1.89×10^7	1.95×10^7
Variance	2.25×10^{-4}	3.23×10^{-5}	2.32×10^{-5}	1.30×10^{-5}	1.57×10^{-5}

8 Conclusion

This paper presents FreeSam, a solution for adaptive sampling for intra-window join. To the best of our knowledge, it is the first method to permeate sampling schemes in the stream join procedure, thus enabling the expansion of the resultant value range on two metrics. Our basic idea is to utilize the symmetry of hash join to add extra sampling schemes to control the probe behavior. In detail, we first give a definition of the sampling-aware intra-window join problem, which reveals the possibility of introducing symmetry in the sampling design. Then, we bring out FreeSam as an instance of the problem, which applies a hybrid sampling approach and is able to control the probe behavior with a new concept, probe utilization. Based on the experiments, we conclude that FreeSam is an efficacious solution for adaptive sampling-aware intra-window join, and can handle a variety of application scenarios. Our work fills the gap in sampling design between traditional database and stream processing, and sheds light on future work for better combinations of sampling and join.

Supplemental Materials: The datasets, code, and an appendix with more discussions of this work are available at https://github.com/intellistream/AllianceDB/tree/FreeSam.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 62322213 and 62172419) and Beijing Nova Program (No. 20230484397 and 20220484137). Shuhao's research is in part supported by the Ministry of Education AcRF Tier 2 grant (No. MOE-T2EP20122-0010) in Singapore. Bingsheng's research is in part supported by the Ministry of Education AcRF Tier 2 grant (No. MOE-T2EP20121-0016) in Singapore. X. Tang, F. Zhang, Y. Liu, and X. Du are with the Key Laboratory of Data Engineering and Knowledge Engineering (MOE), and the School of Information, Renmin University of China. Xilin Tang and Feng Zhang have equal contributions and are co-first authors. Xiaoyong Du is the corresponding author.

References

[1] 2000. Sampling: Design and Analysis. *Technometrics* (2000).

[2] 2018. Interval Join in Apache Flink. Retrieved March 19, 2022 from https://nightlies.apache.org/flink/flink-docs-release-1.14/docs/dev/datastream/operators/joining/

[3] Daniel J. Abadi, Donald Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stanley B. Zdonik. 2003. Aurora: a new model and architecture for data stream management. *The VLDB Journal* 12 (2003), 120–139.

[4] Fatima Abdullah, Limei Peng, and Byungchul Tak. 2021. A Survey of IoT Stream Query Execution Latency Optimization within Edge and Cloud. *Wireless Communications and Mobile Computing* 2021 (2021), 1–16.

[5] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. 1999. Join Synopses for Approximate Query Answering. *SIGMOD Rec.* 28, 2 (jun 1999), 275–286. https://doi.org/10.1145/304181.304207

[6] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *Proceedings of the 8th ACM European Conference on Computer Systems* (Prague, Czech Republic) *(EuroSys '13)*. Association for Computing Machinery, New York, NY, USA, 29–42. https://doi.org/10.1145/2465351.2465355

[7] Charu C. Aggarwal. 2006. On Biased Reservoir Sampling in the Presence of Stream Evolution. In *Proceedings of the 32nd International Conference on Very Large Data Bases* (Seoul, Korea) *(VLDB '06)*. VLDB Endowment, 607–618.

[8] Asaad Althoubi, Reem Alshahrani, and Hassan Peyravi. 2021. Delay Analysis in IoT Sensor Networks. *Sensors* 21, 11 (2021). https://doi.org/10.3390/s21113876

[9] Jieliang Ang, Tianyuan Fu, Johns Paul, Shuhao Zhang, Bingsheng He, Teddy Wenceslao, and Sien Tan. 2019. TraV: An Interactive Exploration System for Massive Trajectory Data. 309–313. https://doi.org/10.1109/BigMM.2019.000-4

[10] Albert Atserias, Martin Grohe, and Dániel Marx. 2013. Size Bounds and Query Plans for Relational Joins. *SIAM J. Comput.* 42, 4 (2013), 1737–1767. https://doi.org/10.1137/110859440 arXiv:https://doi.org/10.1137/110859440

[11] Brian Babcock, Surajit Chaudhuri, and Gautam Das. 2003. Dynamic sample selection for approximate query processing. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data* (San Diego, California) *(SIGMOD '03)*. Association for Computing Machinery, New York, NY, USA, 539–550. https://doi.org/10.1145/872757.872822

[12] Cagri Balkesen, Gustavo Alonso, Jens Teubner, and M. Tamer Özsu. 2013. Multi-Core, Main-Memory Joins: Sort svs. Hash Revisited. *Proc. VLDB Endow.* 7, 1 (sep 2013), 85–96. https://doi.org/10.14778/2732219.2732227

[13] Paul Beame, Paraschos Koutris, and Dan Suciu. 2014. Skew in Parallel Query Processing. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (Snowbird, Utah, USA) *(PODS '14)*. Association for Computing Machinery, New York, NY, USA, 212–223. https://doi.org/10.1145/2594538.2594558

[14] Spyros Blanas, Yinan Li, and Jignesh M. Patel. 2011. Design and Evaluation of Main Memory Hash Join Algorithms for Multi-Core CPUs. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data* (Athens, Greece) *(SIGMOD '11)*. Association for Computing Machinery, New York, NY, USA, 37–48. https://doi.org/10.1145/1989323.1989328

[15] 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* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 18–35. https://doi.org/10.1145/3299869.3319894

[16] Moses Charikar, Kevin Chen, and Martin Farach-Colton. 2002. Finding Frequent Items in Data Streams. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP '02)*. Springer-Verlag, Berlin, Heidelberg, 693–703.

[17] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. 1999. On Random Sampling over Joins. *SIGMOD Record (ACM Special Interest Group on Management of Data)* 28, 2 (1999), 263–273. https://doi.org/10.1145/304181.304206

[18] Yu Chen and Ke Yi. 2017. Two-Level Sampling for Join Size Estimation. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 759–774. https://doi.org/10.1145/3035918.3035921

[19] Y. Chen and Ke Yi. 2020. Random Sampling and Size Estimation Over Cyclic Joins. In *International Conference on Database Theory*.

[20] Yanqiu Chen, Linjiang Zheng, and Weining Liu. 2020. Performance-Sensitive Data Distribution Method for Distributed Stream Processing Systems. In *Proceedings of the 2020 4th High Performance Computing and Cluster Technologies Conference & 2020 3rd International Conference on Big Data and Artificial Intelligence* (Qingdao, China) *(HPCCT & BDAI '20)*. Association for Computing Machinery, New York, NY, USA, 212–217. https://doi.org/10.1145/3409501.3409536

[21] Yu Cheng, Weijie Zhao, and Florin Rusu. 2017. Bi-level online aggregation on raw data. *ACM International Conference Proceeding Series* Part F1286 (2017). https://doi.org/10.1145/3085504.3085514

[22] Jatin Chhugani, Anthony D. Nguyen, Victor W. Lee, William Macy, Mostafa Hagog, Yen-Kuang Chen, Akram Baransi, Sanjeev Kumar, and Pradeep Dubey. 2008. Efficient Implementation of Sorting on Multi-Core SIMD CPU Architecture. *Proc. VLDB Endow.* 1, 2 (aug 2008), 1313–1324. https://doi.org/10.14778/1454159.1454171

[23] R. Cirstea, B. Yang, C. Guo, T. Kieu, and S. Pan. 2022. Towards Spatio- Temporal Aware Traffic Time Series Forecasting. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 2900–2913. https://doi.org/10.1109/ICDE53745.2022.00262

[24] Graham Cormode, Minos Garofalakis, Peter J. Haas, and Chris Jermaine. 2011. *Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches*.

[25] Graham Cormode, Minos Garofalakis, and Dimitris Sacharidis. 2006. Fast Approximate Wavelet Tracking on Streams. In *Advances in Database Technology - EDBT 2006*, Yannis Ioannidis, Marc H. Scholl, Joachim W. Schmidt, Florian Matthes, Mike Hatzopoulos, Klemens Boehm, Alfons Kemper, Torsten Grust, and Christian Boehm (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 4–22.

[26] Graham Cormode and S. Muthukrishnan. 2004. An improved data stream summary: The Count-Min sketch and its applications. *J. Algorithms* 55 (2004), 29–38.

[27] James Croft. 2024. Identifying drift in ML models: Best practices for generating consistent, reliable responses. Retrieved April, 2024 from https://techcommunity.microsoft.com/t5/fasttrack-for-azure/identifying-drift-in-ml-models-best-practices-for-generating/ba-p/4040531

[28] Abhinandan Das, Johannes Gehrke, and Mirek Riedewald. 2003. Approximate join processing over data streams. (2003), 40. https://doi.org/10.1145/872763.872765

[29] A. Das, J. Gehrke, and M. Riedewald. 2005. Semantic approximation of data stream joins. *IEEE Transactions on Knowledge and Data Engineering* 17, 1 (2005), 44–59. https://doi.org/10.1109/TKDE.2005.17

[30] Shiyuan Deng, Shangqi Lu, and Yufei Tao. 2023. On Join Sampling and the Hardness of Combinatorial Output-Sensitive Join Algorithms. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (Seattle, WA, USA) *(PODS '23)*. Association for Computing Machinery, New York, NY, USA, 99–111. https://doi.org/10.1145/3584372.3588666

[31] Jens-Peter Dittrich, Bernhard Seeger, David Scot Taylor, and Peter Widmayer. 2002. Progressive Merge Join: A Generic and Non-Blocking Sort-Based Join Algorithm. In *Proceedings of the 28th International Conference on Very Large Data Bases* (Hong Kong, China) *(VLDB '02)*. VLDB Endowment, 299–310.

[32] Marina Drosou and Evaggelia Pitoura. 2010. Search Result Diversification. *SIGMOD Rec.* 39, 1 (sep 2010), 41–47. https://doi.org/10.1145/1860702.1860709

[33] Pavlos Efraimidis. 2010. Weighted Random Sampling over Data Streams. (12 2010). https://doi.org/10.1007/978-3-319-24024-4_12

[34] Pavlos S. Efraimidis and Paul G. Spirakis. 2006. Weighted random sampling with a reservoir. *Inform. Process. Lett.* 97, 5 (2006), 181–185. https://doi.org/10.1016/j.ipl.2005.11.003

[35] Mohammed Elseidy, Abdallah Elguindy, Aleksandar Vitorovic, and Christoph Koch. 2014. Scalable and Adaptive Online Joins. *Proc. VLDB Endow.* 7, 6 (feb 2014), 441–452. https://doi.org/10.14778/2732279.2732281

[36] C. Estan and J.F. Naughton. 2006. End-biased Samples for Join Cardinality Estimation. In *22nd International Conference on Data Engineering (ICDE'06)*. 20–20. https://doi.org/10.1109/ICDE.2006.61

[37] Wenfei Fan, Ziyan Han, Yaoshu Wang, and Min Xie. 2022. Parallel Rule Discovery from Large Datasets by Sampling. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 384–398. https://doi.org/10.1145/3514221.3526165

[38] Raphaël Féraud, Fabrice Clérot, and Pascal Gouzien. 2010. Sampling the Join of Streams. In *Classification as a Tool for Research*, Hermann Locarek-Junge and Claus Weihs (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 307–314.

[39] Flink. 2022. "Package org.apache.flink.api.java.sampling". https://nightlies.apache.org/flink/flink-docs-master/api/java/org/apache/flink/api/java/sampling/package-summary.html

[40] Sumit Ganguly, Phillip B. Gibbons, Yossi Matias, and Avi Silberschatz. 1996. Bifocal Sampling for Skew-Resistant Join Size Estimation. *SIGMOD Rec.* 25, 2 (jun 1996), 271–281. https://doi.org/10.1145/235968.233340

[41] Buğra Gedik, Rajesh R. Bordawekar, and Philip S. Yu. 2009. CellJoin: A Parallel Stream Join Operator for the Cell Processor. *The VLDB Journal* 18, 2 (apr 2009), 501–519. https://doi.org/10.1007/s00778-008-0116-z

[42] Bugra Gedik, Kun-Lung Wu, Philip S. Yu, and Ling Liu. 2007. GrubJoin: An Adaptive, Multi-Way, Windowed Stream Join with Time Correlation-Aware CPU Load Shedding. *IEEE Transactions on Knowledge and Data Engineering* 19, 10 (2007), 1363–1380. https://doi.org/10.1109/TKDE.2007.190630

[43] Bugra Gedik, Kun-Lung Wu, Philip S. Yu, and Ling Liu. 2007. A Load Shedding Framework and Optimizations for M-way Windowed Stream Joins. In *2007 IEEE 23rd International Conference on Data Engineering*. 536–545. https://doi.org/10.1109/ICDE.2007.367899

[44] Thanasis Georgiadis and Nikos Mamoulis. 2023. Raster Intervals: An Approximation Technique for Polygon Intersection Joins. *Proc. ACM Manag. Data* 1, 1, Article 36 (may 2023), 18 pages. https://doi.org/10.1145/3588716

[45] Lukasz Golab, Shaveen Garg, and M Tamer Özsu. 2004. On indexing sliding windows over online data streams. In *International Conference on Extending Database Technology*. Springer, 712–729.

[46] Xiangyang Gou, Long He, Yinda Zhang, Ke Wang, Xilai Liu, Tong Yang, Yi Wang, and Bin Cui. 2020. Sliding Sketches: A Framework Using Time Zones for Data Stream Processing in Sliding Windows. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Virtual Event, CA, USA) *(KDD '20)*. Association for Computing Machinery, New York, NY, USA, 1015–1025. https://doi.org/10.1145/3394486.3403144

[47] Rong Gu, Han Li, Haipeng Dai, Wenjie Huang, Jie Xue, Meng Li, Jiaqi Zheng, Haoran Cai, Yihua Huang, and Guihai Chen. 2023. ShadowAQP: Efficient Approximate Group-by and Join Query via Attribute-Oriented Sample Size

Allocation and Data Generation. *Proc. VLDB Endow.* 16, 13 (sep 2023), 4216–4229. https://doi.org/10.14778/3625054.3625059

[48] Sudipto Guha and Boulos Harb. 2005. Wavelet Synopsis for Data Streams: Minimizing Non-Euclidean Error. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining* (Chicago, Illinois, USA) *(KDD '05)*. Association for Computing Machinery, New York, NY, USA, 88–97. https://doi.org/10.1145/1081870.1081884

[49] Sudipto Guha, Nick Koudas, and Kyuseok Shim. 2006. Approximation and Streaming Algorithms for Histogram Construction Problems. *ACM Trans. Database Syst.* 31, 1 (mar 2006), 396–438. https://doi.org/10.1145/1132863.1132873

[50] Vincenzo Gulisano, Zbigniew Jerzak, Spyros Voulgaris, and Holger Ziekow. 2016. The DEBS 2016 Grand Challenge. In *Proceedings of the 10th ACM International Conference on Distributed and Event-Based Systems* (Irvine, California) *(DEBS '16)*. Association for Computing Machinery, New York, NY, USA, 289–292. https://doi.org/10.1145/2933267.2933519

[51] Peter J. Haas. 2016. *Data-Stream Sampling: Basic Techniques and Results*. Springer Berlin Heidelberg, Berlin, Heidelberg, 13–44. https://doi.org/10.1007/978-3-540-28608-0_2

[52] Peter J. Haas and Joseph M. Hellerstein. 1999. Ripple Joins for Online Aggregation. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data* (Philadelphia, Pennsylvania, USA) *(SIGMOD '99)*. Association for Computing Machinery, New York, NY, USA, 287–298. https://doi.org/10.1145/304182.304208

[53] Peter J. Haas and Christian König. 2004. A Bi-Level Bernoulli Scheme for Database Sampling. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data* (Paris, France) *(SIGMOD '04)*. Association for Computing Machinery, New York, NY, USA, 275–286. https://doi.org/10.1145/1007568.1007601

[54] Carole J. Hahn, Stephen G. Warren, and Julius London. 1996. Edited Synoptic Cloud Reports from Ships and Land Stations Over the Globe, 1982-1991 (NDP-026B). (1 1996). https://doi.org/10.3334/CDIAC/cli.ndp026b

[55] 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

[56] Gilseok HONG, Seonghyeon KANG, Chang soo KIM, and Jun-Ki MIN. 2018. Efficient Parallel Join Processing Exploiting SIMD in Multi-Thread Environments. *IEICE Transactions on Information and Systems* E101.D, 3 (2018), 659–667. https://doi.org/10.1587/transinf.2017EDP7300

[57] D. G. Horvitz and D. J. Thompson. 1952. A Generalization of Sampling Without Replacement from a Finite Universe. *J. Amer. Statist. Assoc.* 47, 260 (1952), 663–685. https://doi.org/10.1080/01621459.1952.10483446

[58] Xiao Hu, Yufei Tao, and Ke Yi. 2017. Output-Optimal Parallel Algorithms for Similarity Joins. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (Chicago, Illinois, USA) *(PODS '17)*. Association for Computing Machinery, New York, NY, USA, 79–90. https://doi.org/10.1145/3034786.3056110

[59] Dawei Huang, Dong Young Yoon, Seth Pettie, and Barzan Mozafari. 2019. Joins on Samples: A Theoretical Guide for Practitioners. *Proc. VLDB Endow.* 13, 4 (dec 2019), 547–560. https://doi.org/10.14778/3372716.3372726

[60] Xinmei Huang, Haoyang Li, Jing Zhang, Xinxin Zhao, Zhiming Yao, Yiyan Li, Zhuohao Yu, Tieying Zhang, Hong Chen, and Cuiping Li. 2024. LLMTune: Accelerate Database Knob Tuning with Large Language Models. *arXiv preprint arXiv:2404.11581* (2024).

[61] Janardan and Shikha Mehta. 2017. Concept drift in Streaming Data Classification: Algorithms, Platforms and Issues. *Procedia Computer Science* 122 (2017), 804–811. https://doi.org/10.1016/j.procs.2017.11.440 5th International Conference on Information Technology and Quantitative Management, ITQM 2017.

[62] Christopher Jermaine, Subramanian Arumugam, Abhijit Pol, and Alin Dobra. 2007. Scalable approximate query processing with the DBO engine. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (2007), 725–736. https://doi.org/10.1145/1247480.1247560

[63] Yuanzhen Ji, Jun Sun, Anisoara Nica, Zbigniew Jerzak, Gregor Hackenbroich, and Christof Fetzer. 2016. Quality-driven disorder handling for m-way sliding window stream joins. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. 493–504. https://doi.org/10.1109/ICDE.2016.7498265

[64] Wei Jiang, Liu-Gen Xu, Hai-Bo Hu, and Yue Ma. 2019. Improvement design for distributed real-time stream processing systems. *Journal of Electronic Science and Technology* 17, 1 (2019), 3–12.

[65] Ming Jin, Yu Zheng, Yuan-Fang Li, Siheng Chen, Bin Yang, and Shirui Pan. 2023. Multivariate Time Series Forecasting With Dynamic Graph Neural ODEs. *IEEE Transactions on Knowledge and Data Engineering* 35, 9 (2023), 9168–9180. https://doi.org/10.1109/TKDE.2022.3221989

[66] Theodore Johnson, Shanmugavelayutham Muthukrishnan, and Irina Rozenbaum. 2005. Sampling algorithms in a stream operator. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. 1–12.

[67] Evangelia Kalyvianaki, Wolfram Wiesemann, Quang Hieu Vu, Daniel Kuhn, and Peter Pietzuch. 2011. SQPR: Stream query planning with reuse. In *2011 IEEE 27th International Conference on Data Engineering*. 840–851. https://doi.org/10.1109/ICDE.2011.5767851

[68] Srikanth Kandula, Anil Shanbhag, Aleksandar Vitorovic, Matthaios Olma, Robert Grandl, Surajit Chaudhuri, and Bolin Ding. 2016. Quickr: Lazily Approximating Complex AdHoc Queries in BigData Clusters. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 631–646. https://doi.org/10.1145/2882903.2882940

[69] Jaewoo Kang, Jeffrey F Naughton, and Stratis D Viglas. 2003. Evaluating window joins over unbounded streams. In *Proceedings 19th International Conference on Data Engineering (Cat. No. 03CH37405)*. IEEE, 341–352.

[70] Jeyhun Karimov, Tilmann Rabl, Asterios Katsifodimos, Roman Samarev, Henri Heiskanen, and Volker Markl. 2018. Benchmarking Distributed Stream Data Processing Systems. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. 1507–1518. https://doi.org/10.1109/ICDE.2018.00169

[71] Martin Kiefer, Max Heimel, Sebastian Breß, and Volker Markl. 2017. Estimating Join Selectivities Using Bandwidth-Optimized Kernel Density Models. *Proc. VLDB Endow.* 10, 13 (sep 2017), 2085–2096. https://doi.org/10.14778/3151106.3151112

[72] Changkyu Kim, Tim Kaldewey, Victor W. Lee, Eric Sedlar, Anthony D. Nguyen, Nadathur Satish, Jatin Chhugani, Andrea Di Blas, and Pradeep Dubey. 2009. Sort vs. Hash Revisited: Fast Join Implementation on Modern Multi-Core CPUs. *Proc. VLDB Endow.* 2, 2 (aug 2009), 1378–1389. https://doi.org/10.14778/1687553.1687564

[73] Ilya Kolchinsky and Assaf Schuster. 2019. Real-Time Multi-Pattern Detection over Event Streams. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 589–606. https://doi.org/10.1145/3299869.3319869

[74] 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.

[75] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. 2016. Wander Join: Online Aggregation via Random Walks. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 615–629. https://doi.org/10.1145/2882903.2915235

[76] F. Li, B. Wu, K. Yi, and Z. Zhao. 2017. Wander Join and XDB: Online Aggregation via Random Walks. *Acm Sigmod Record* 46, 1 (2017), 33–40.

[77] Yanying Li, Haipei Sun, Boxiang Dong, and Hui (Wendy) Wang. 2018. Cost-Efficient Data Acquisition on Online Data Marketplaces for Correlation Analysis. *Proc. VLDB Endow.* 12, 4 (dec 2018), 362–375. https://doi.org/10.14778/3297753.3297757

[78] Qian Lin, Beng Chin Ooi, Zhengkui Wang, and Cui Yu. 2015. Scalable Distributed Stream Join Processing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (Melbourne, Victoria, Australia) *(SIGMOD '15)*. Association for Computing Machinery, New York, NY, USA, 811–825. https://doi.org/10.1145/2723372.2746485

[79] R. J. Lipton and J. F. Naughton. 1990. Query size estimation by adaptive sampling (extended abstract). *Journal of Computer & System ences* 51, 1 (1990), 18–25.

[80] Jiesong Liu, Feng Zhang, Lv Lu, Chang Qi, Xiaoguang Guo, Dong Deng, Guoliang Li, Huanchen Zhang, Jidong Zhai, Hechen Zhang, Yuxing Chen, Anqun Pan, and Xiaoyong Du. 2024. G-Learned Index: Enabling Efficient Learned Index on GPU. *IEEE Transactions on Parallel and Distributed Systems* 35, 6 (2024), 950–967. https://doi.org/10.1109/TPDS.2024.3381214

[81] Tianyu Liu and Chi Wang. 2020. Understanding the hardness of approximate query processing with joins. arXiv:2010.00307 [cs.DB]

[82] Clemens Lutz, Sebastian Breß, Steffen Zeuch, Tilmann Rabl, and Volker Markl. 2022. Triton Join: Efficiently Scaling to a Large Join State on GPUs with Fast Interconnects. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1017–1032. https://doi.org/10.1145/3514221.3517911

[83] Riccardo Mancini, Srinivas Karthik, Bikash Chandra, Vasilis Mageirakos, and Anastasia Ailamaki. 2022. Efficient Massively Parallel Join Optimization for Large Queries. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 122–135. https://doi.org/10.1145/3514221.3517871

[84] Wanli Min and Laura Wynter. 2011. Real-time road traffic prediction with spatio-temporal correlations. *Transportation Research Part C: Emerging Technologies* 19, 4 (2011), 606–616. https://doi.org/10.1016/j.trc.2010.10.002

[85] J. Misra and David Gries. 1982. Finding repeated elements. *Science of Computer Programming* 2, 2 (1982), 143–152. https://doi.org/10.1016/0167-6423(82)90012-0

[86] Aloysius K. Mok, Honguk Woo, and Chan-Gun Lee. 2006. Probabilistic Timing Join over Uncertain Event Streams. In *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)*. 17–26. https://doi.org/10.1109/RTCSA.2006.52

[87] Barzan Mozafari. 2017. Approximate Query Engines: Commercial Challenges and Research Opportunities. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 521–524. https://doi.org/10.1145/3035918.3056098

[88] Mohammadreza Najafi, Mohammad Sadoghi, and Hans-Arno Jacobsen. 2016. SplitJoin: A Scalable, Low-Latency Stream Join Architecture with Adjustable Ordering Precision. In *Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference* (Denver, CO, USA) *(USENIX ATC '16)*. USENIX Association, USA, 493–505.

[89] Creator of the Angry Birds game. 2022. Flink. Retrieved March 19, 2022 from https://nightlies.apache.org/flink/flink-docs-release-1.14/

[90] Adegoke Ojewole, Qiang Zhu, and Wen-Chi Hou. 2006. Window Join Approximation over Data Streams with Importance Semantics. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management* (Arlington, Virginia, USA) *(CIKM '06)*. Association for Computing Machinery, New York, NY, USA, 112–121. https://doi.org/10.1145/1183614.1183635

[91] Frank Olken. 1993. Random sampling from databases. *thesis UC Berkeley* (1993), 172. http://www.cs.washington.edu/education/courses/cse590q/05au/papers/Olken-Sampling.pdf

[92] Margaret A. Palmer, Christine C. Hakenkamp, and Kären Nelson-Baker. 1997. Ecological Heterogeneity in Streams: Why Variance Matters. *Journal of the North American Benthological Society* 16, 1 (1997), 189–202. http://www.jstor.org/stable/1468251

[93] Niketan Pansare, Vinayak Borkar, Chris Jermaine, and Tyson Condie. 2011. Online Aggregation for Large MapReduce Jobs. *Proc. VLDB Endow.* 4, 11 (aug 2011), 1135–1145. https://doi.org/10.14778/3402707.3402748

[94] Jinglin Peng, Bolin Ding, Jiannan Wang, Kai Zeng, and Jingren Zhou. 2022. One Size Does Not Fit All: A Bandit-Based Sampler Combination Framework with Theoretical Guarantees. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 531–544. https://doi.org/10.1145/3514221.3517900

[95] Vibhor Porwal, Subrata Mitra, Fan Du, John Anderson, Nikhil Sheoran, Anup Rao, Tung Mai, Gautam Kowshik, Sapthotharan Nair, Sameeksha Arora, and Saurabh Mahapatra. 2022. Efficient Insights Discovery through Conditional Generative Model Based Query Approximation. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 2397–2400. https://doi.org/10.1145/3514221.3520161

[96] Navneet Potti and Jignesh M. Patel. 2015. DAQ: A New Paradigm for Approximate Query Processing. *Proc. VLDB Endow.* 8, 9 (may 2015), 898–909. https://doi.org/10.14778/2777598.2777599

[97] Xu Qian, Yang Juan, Zhang Feng, Chen Zheng, Guan Jiawei, Chen Kang, Fan Ju, Shen Youren, Yang Ke, Zhang Yu, and Du Xiaoyong. 2024. Improving Graph Compression for Efficient Resource-Constrained Graph Analytics. *PVLDB* (2024). https://doi.org/10.14778/3665844.3665852

[98] Yuan Qiu, Serafeim Papadias, and Ke Yi. 2019. Streaming HyperCube: A Massively Parallel Stream Join Algorithm. In *Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, March 26-29, 2019*, Melanie Herschel, Helena Galhardas, Berthold Reinwald, Irini Fundulaki, Carsten Binnig, and Zoi Kaoudi (Eds.). OpenProceedings.org, 642–645. https://doi.org/10.5441/002/edbt.2019.76

[99] Do Le Quoc, Ruichuan Chen, Pramod Bhatotia, Christof Fetzer, Volker Hilt, and Thorsten Strufe. 2017. StreamApprox: Approximate Computing for Stream Analytics. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference* (Las Vegas, Nevada) *(Middleware '17)*. Association for Computing Machinery, New York, NY, USA, 185–197. https://doi.org/10.1145/3135974.3135989

[100] Chuitian Rong, Chunbin Lin, Yasin N. Silva, Jianguo Wang, Wei Lu, and Xiaoyong Du. 2017. Fast and Scalable Distributed Set Similarity Joins for Big Data Analytics. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. 1059–1070. https://doi.org/10.1109/ICDE.2017.151

[101] 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* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 1449–1463. https://doi.org/10.1145/2882903.2882948

[102] Pratanu Roy, Jens Teubner, and Rainer Gemulla. 2014. Low-Latency Handshake Join. *Proc. VLDB Endow.* 7, 9 (may 2014), 709–720. https://doi.org/10.14778/2732939.2732944

[103] Florin Rusu and Alin Dobra. 2008. Sketches for Size of Join Estimation. *ACM Trans. Database Syst.* 33, 3, Article 15 (sep 2008), 46 pages. https://doi.org/10.1145/1386118.1386121

[104] Viktor Sanca and Anastasia Ailamaki. 2022. Sampling-Based AQP in Modern Analytical Engines. In *Data Management on New Hardware* (Philadelphia, PA, USA) *(DaMoN'22)*. Association for Computing Machinery, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.1145/3533737.3535095

[105] Viktor Sanca, Periklis Chrysogelos, and Anastasia Ailamaki. 2023. LAQy: Efficient and Reusable Query Approximations via Lazy Sampling. *Proc. ACM Manag. Data* 1, 2, Article 174 (jun 2023), 26 pages. https://doi.org/10.1145/3589319

[106] Majid Rostami Shahrbabaki, Ali Akbar Safavi, Markos Papageorgiou, and Ioannis Papamichail. 2018. A data fusion approach for real-time traffic state estimation in urban signalized links. *Transportation Research Part C: Emerging Technologies* 92 (2018), 525–548. https://doi.org/10.1016/j.trc.2018.05.020

[107] Ali Mohammadi Shanghooshabad, Meghdad Kurmanji, Qingzhi Ma, Michael Shekelyan, Mehrdad Almasi, and Peter Triantafillou. 2021. PGMJoins: Random Join Sampling with Graphical Models. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, 1610–1622. https://doi.org/10.1145/3448016.3457302

[108] Michael Shekelyan, Graham Cormode, Peter Triantafillou, Ali Mohammadi Shanghooshabad, and Qingzhi Ma. 2022. Weighted Random Sampling over Joins. *CoRR* abs/2201.02670 (2022). arXiv:2201.02670 https://arxiv.org/abs/2201.02670

[109] Nikhil Sheoran, Subrata Mitra, Vibhor Porwal, Siddharth Ghetia, Jatin Varshney, Tung Mai, Anup Rao, and Vikas Maddukuri. 2022. Conditional Generative Model Based Predicate-Aware Query Approximation. *Proceedings of the AAAI Conference on Artificial Intelligence* 36, 8 (Jun. 2022), 8259–8266. https://doi.org/10.1609/aaai.v36i8.20800

[110] Spark. 2022. "Java API for random utilities in Spark ". https://spark.apache.org/docs/3.4.0/api/java/org/apache/spark/util/random/

[111] Utkarsh Srivastava, Kamesh Munagala, and Jennifer Widom. 2005. Operator Placement for In-Network Stream Query Processing. In *Proceedings of the Twenty-Fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (Baltimore, Maryland) *(PODS '05)*. Association for Computing Machinery, New York, NY, USA, 250–258. https://doi.org/10.1145/1065167.1065199

[112] Utkarsh Srivastava and Jennifer Widom. 2004. Memory-Limited Execution of Windowed Stream Joins. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30* (Toronto, Canada) *(VLDB '04)*. VLDB Endowment, 324–335.

[113] Michał Startek. 2016. An asymptotically optimal, online algorithm for weighted random sampling with replacement. *ArXiv* abs/1611.00532 (2016).

[114] GYÖRGY STEINBRECHER and WILLIAM T. SHAW. 2008. Quantile mechanics. *European Journal of Applied Mathematics* 19, 2 (2008), 87–112. https://doi.org/10.1017/S0956792508007341

[115] Yufei Tao, Xiang Lian, Dimitris Papadias, and Marios Hadjieleftheriou. 2007. Random Sampling for Continuous Streams with Arbitrary Updates. *IEEE Transactions on Knowledge and Data Engineering* 19, 1 (2007), 96–110. https://doi.org/10.1109/TKDE.2007.250588

[116] Jens Teubner and Rene Mueller. 2011. How Soccer Players Would Do Stream Joins. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data* (Athens, Greece) *(SIGMOD '11)*. Association for Computing Machinery, New York, NY, USA, 625–636. https://doi.org/10.1145/1989323.1989389

[117] Lasse Thostrup, Gloria Doci, Nils Boeschen, Manisha Luthra, and Carsten Binnig. 2023. Distributed GPU Joins on Fast RDMA-Capable Networks. *Proc. ACM Manag. Data* 1, 1, Article 29 (may 2023), 26 pages. https://doi.org/10.1145/3588709

[118] Daniel Ting. 2022. Adaptive Threshold Sampling. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1612–1625. https://doi.org/10.1145/3514221.3526122

[119] Daniel Ting and Rick Cole. 2021. *Conditional Cuckoo Filters*. Association for Computing Machinery, New York, NY, USA, 1838–1850. https://doi.org/10.1145/3448016.3452811

[120] Wee Hyong Tok, Stéphane Bressan, and Mong-Li Lee. 2008. A Stratified Approach to Progressive Approximate Joins. In *Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology* (Nantes, France) *(EDBT '08)*. Association for Computing Machinery, New York, NY, USA, 582–593. https://doi.org/10.1145/1353343.1353414

[121] Jonas Traub, Philipp Marian Grulich, Alejandro Rodriguez Cuellar, Sebastian Breß, Asterios Katsifodimos, Tilmann Rabl, and Volker Markl. 2019. Efficient Window Aggregation with General Stream Slicing. In *International Conference on Extending Database Technology*.

[122] 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

[123] Jeffrey S. Vitter. 1985. Random Sampling with a Reservoir. *ACM Trans. Math. Softw.* 11, 1 (mar 1985), 37–57. https://doi.org/10.1145/3147.3165

[124] Martha Vlachou-Konchylaki. 2016. Efficient Data Stream Sampling on Apache Flink.

[125] Feiyu Wang, Qizhi Chen, Yuanpeng Li, Tong Yang, Yaofeng Tu, Lian Yu, and Bin Cui. 2023. JoinSketch: A Sketch Algorithm for Accurate and Unbiased Inner-Product Estimation. *Proc. ACM Manag. Data* 1, 1, Article 81 (may 2023), 26 pages. https://doi.org/10.1145/3588935

[126] Y. Wang, A. Khan, X. Xu, J. Jin, Q. Hong, and T. Fu. 2022. Aggregate Queries on Knowledge Graphs: Fast Approximation with Semantic-aware Sampling. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 2914–2927. https://doi.org/10.1109/ICDE53745.2022.00263

[127] Xiaohui Wei, Yuanyuan Liu, Xingwang Wang, Bingyi Sun, Shang Gao, and Jon Rokne. 2019. A survey on quality-assurance approximate stream processing and applications. *Future Generation Computer Systems* 101 (2019), 1062–1080. https://doi.org/10.1016/j.future.2019.07.047

[128] A.N. Wilschut and P.M.G. Apers. 1991. Dataflow query execution in a parallel main-memory environment. In *[1991] Proceedings of the First International Conference on Parallel and Distributed Information Systems*. 68–77. https://doi.org/10.1109/PDIS.1991.183069

[129] Xinle Wu, Dalin Zhang, Miao Zhang, Chenjuan Guo, Bin Yang, and Christian S. Jensen. 2023. AutoCTS+: Joint Neural Architecture and Hyperparameter Search for Correlated Time Series Forecasting. *Proc. ACM Manag. Data* 1, 1, Article 97 (may 2023), 26 pages. https://doi.org/10.1145/3588951

[130] Le Xu, Shivaram Venkataraman, Indranil Gupta, Luo Mai, and Rahul Potharaju. 2021. Move fast and meet deadlines: Fine-grained real-time stream processing with cameo. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 389–405.

[131] Yu Ya-xin, Yang Xing-hua, Yu Ge, and Wu Shan-shan. 2006. An indexed non-equijoin algorithm based on sliding windows over data streams. *Wuhan University Journal of Natural Sciences* 11, 1 (2006), 294–298.

[132] Zongheng Yang and Chenggang Wu. "2019". "Github repository: naru project". https://github.com/naru-project/naru

[133] Alpaslan Yarar. 2014. A Hybrid Wavelet and Neuro-Fuzzy Model for Forecasting the Monthly Streamflow Data. *Water Resources Management* 28 (2014), 553–565.

[134] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Zachary Zimmerman, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh. 2018. Time Series Joins, Motifs, Discords and Shapelets: A Unifying View That Exploits the Matrix Profile. *Data Min. Knowl. Discov.* 32, 1 (jan 2018), 83–123. https://doi.org/10.1007/s10618-017-0519-9

[135] Kai Zeng, Shi Gao, Barzan Mozafari, and Carlo Zaniolo. 2014. The Analytical Bootstrap: A New Method for Fast Error Estimation in Approximate Query Processing. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (Snowbird, Utah, USA) *(SIGMOD '14)*. Association for Computing Machinery, New York, NY, USA, 277–288. https://doi.org/10.1145/2588555.2588579

[136] Feng Zhang, Jidong Zhai, Xipeng Shen, Onur Mutlu, and Xiaoyong Du. 2022. POCLib: A High-Performance Framework for Enabling Near Orthogonal Processing on Compression. *IEEE Transactions on Parallel and Distributed Systems* 33, 2 (2022), 459–475. https://doi.org/10.1109/TPDS.2021.3093234

[137] Huanchen Zhang, Hyeontaek Lim, Viktor Leis, David G. Andersen, Michael Kaminsky, Kimberly Keeton, and Andrew Pavlo. 2018. SuRF: Practical Range Query Filtering with Fast Succinct Tries. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 323–336. https://doi.org/10.1145/3183713.3196931

[138] Jiaoyi Zhang, Kai Su, and Huanchen Zhang. 2024. Making In-Memory Learned Indexes Efficient on Disk. *Proc. ACM Manag. Data* 2, 3, Article 151 (may 2024), 26 pages. https://doi.org/10.1145/3654954

[139] Jiayao Zhang, Qiheng Sun, Jinfei Liu, Li Xiong, Jian Pei, and Kui Ren. 2023. Efficient Sampling Approaches to Shapley Value Approximation. *Proc. ACM Manag. Data* 1, 1, Article 48 (may 2023), 24 pages. https://doi.org/10.1145/3588728

[140] Shuhao Zhang, Yancan Mao, Jiong He, Philipp M. Grulich, Steffen Zeuch, Bingsheng He, Richard T. B. Ma, and Volker Markl. 2021. *Parallelizing Intra-Window Join on Multicores: An Experimental Study*. Association for Computing Machinery, New York, NY, USA, 2089–2101. https://doi.org/10.1145/3448016.3452793

[141] Shuhao Zhang, Feng Zhang, Yingjun Wu, Bingsheng He, and Paul Johns. 2020. Hardware-Conscious Stream Processing: A Survey. *SIGMOD Rec.* 48, 4 (feb 2020), 18–29. https://doi.org/10.1145/3385658.3385662

[142] Xiaojian Zhang, Wanchang Jiang, Yadong Zhang, and Cong Huo. 2007. Clustering-Variable-Width Histogram Based Window Semi-hash Multi-join over Streams. In *2007 International Conference on Convergence Information Technology (ICCIT 2007)*. 850–853. https://doi.org/10.1109/ICCIT.2007.246

[143] Bo Zhao, Han van der Aa, Thanh Tam Nguyen, Quoc Viet Hung Nguyen, and Matthias Weidlich. 2021. EIRES: Efficient Integration of Remote Data in Event Stream Processing. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 2128–2141. https://doi.org/10.1145/3448016.3457304

[144] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. 2018. Random Sampling over Joins Revisited. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 1525–1539. https://doi.org/10.1145/3183713.3183739

[145] Zhuoyue Zhao, Feifei Li, and Yuxi Liu. 2020. Efficient Join Synopsis Maintenance for Data Warehouse. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 2027–2042. https://doi.org/10.1145/3318464.3389717

Received January 2024; revised April 2024; accepted May 2024