PECJ: Stream Window Join on Disorder Data Streams with Proactive Error Compensation

Xianzhi Zeng
Singapore University of Technology and Design
xianzhi_xianzhi@mymail.sutd.edu.sg

Shuhao Zhang*
Nanyang Technological University
shuhao.zhang@ntu.edu.sg

Hongbin Zhong
4paradigm Inc.
zhonghongbin@4paradigm.com

Hao Zhang
4paradigm Inc.
zhanghao@4paradigm.com

Mian Lu
4paradigm Inc.
lumian@4paradigm.com

Zhao Zheng
4paradigm Inc.
zhengzhao@4paradigm.com

Yuqiang Chen
4paradigm Inc.
chenyuqiang@4paradigm.com

ABSTRACT

Stream Window Join (SWJ), a vital operation in stream analytics, struggles with achieving a balance between accuracy and latency due to common out-of-order data arrivals. Existing methods predominantly rely on adaptive buffering, but often fall short in performance, thereby constraining practical applications. We introduce PECJ, a solution that *proactively* incorporates unobserved data to enhance accuracy while reducing latency, thus requiring robust predictive modeling of evolving data streams. At the heart of PECJ lies a mathematical formulation of the *posterior distribution approximation* (PDA) problem using *variational inference* (VI). This approach circumvents error propagation while meeting the low-latency demands of SWJ. We detail the implementation of PECJ, striking a balance between complexity and generality, and discuss both *analytical* and *learning-based* approaches. Experimental evaluations reveal PECJ's superior performance. The successful integration of PECJ into a multi-threaded SWJ benchmark testbed further establishes its practical value, demonstrating promising advancements in enhancing data stream processing capabilities amidst out-of-order data.

ACM Reference Format:
Xianzhi Zeng, Shuhao Zhang, Hongbin Zhong, Hao Zhang, Mian Lu, Zhao Zheng, and Yuqiang Chen . 2023. PECJ: Stream Window Join on Disorder Data Streams with Proactive Error Compensation. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

*Shuhao'work is majorly done while working at SUTD.

1 INTRODUCTION

Stream Window Join (SWJ) is an operation for merging two input streams within distinct, finite subsets, or 'windows', of infinite streams. SWJ, a crucial component of data stream analytics [43], departs from traditional relational join operations. Rather than waiting for the full input data to become available, SWJ is tasked with generating join results in real-time. This requirement arises from its essential role across various sectors, such as financial markets [12], fraud detection systems [2], and sensor networks [29].

The emergence of machine learning applications like online decision augmentation (OLDA) [1, 42] has amplified the importance of SWJ. These applications leverage SWJ to combine dynamic features, such as short-term user behavior, within time-bounded windows. This enables rapid downstream feature computations and model updates. Especially in OLDA, certain banking applications impose strict end-to-end latency as low as $20ms$ [42], pushing the limits of latency requirements. Such stringent constraints underscore the insufficiency of traditional batch-based approaches to handle SWJ, as they require the complete input data to be present before commencing processing, resulting in increased latency.

SWJ is complicated by the disorderly arrival of input tuples from streams, primarily due to factors like network delays and parallel processing [5, 6, 8]. The management of these disordered data streams typically involves buffering input data [18, 19], providing a more comprehensive view of *in-window* data, thereby facilitating higher accuracy results from running SWJ directly on potentially disordered data streams. However, the additional buffering time needed to gain this comprehensive view often leads to substantial latency costs. These costs become particularly pronounced when waiting for straggling tuples, a situation exacerbated by the non-linear nature of SWJ [18, 43].

To address these issues, we propose a novel solution: PECJ[1] algorithm, designed to proactively manage disordered data streams. Unlike existing methods, which rely exclusively on already-arrived data (i.e., in-window data), PECJ actively takes into account the contributions of future, disordered data to enhance join accuracy. This innovative approach to disorder management introduces a

[1]PECJ: Proactive Error Compensation-Join

promising avenue for achieving significant accuracy enhancements without corresponding increases in latency. Notably, while subjects such as disorder handling parallelization [21, 23, 28] and efficient buffer structures [10] have been thoroughly explored in prior studies, these aspects are orthogonal to our work.

Aware of the inherent challenge in PECJ's approach—predicting the evolution of data streams—we designed PECJ to follow three conceptual steps of mathematical formulation and implementation. In the *first step*, we abstract disorder SWJ handling into the *posterior distribution approximation* (PDA) problem, formulating its probability model under data stream scenarios. This method, in contrast with individual prediction of each unseen data point as in traditional time series [36, 39], estimates the overall contributions of all unobserved data, bypassing error propagation inherent in individual predictions.

In the *second step*, we optimize the parameterization process of the probabilistic model. Instead of applying brute-force parameterization—which is not only infeasible but also compromises SWJ's low latency requirements—we employ the *variational inference* (VI) approach [17, 36] to minimize overhead. VI enables efficient and continual model parameterization throughout PDA. For the *third step*, we instantiate the previously mentioned steps of mathematical formulation into both *analytical* and *learning-based* implementations. The *analytical* implementation is suited to simpler stream dynamics scenarios, offering a low-overhead solution of linear equations. The *learning-based* implementation, conversely, addresses more complex situations, using neural networks to approximate the posterior distribution.

We assess the viability of the PECJ algorithm by executing a series of experiments that underline its superior performance compared to several existing solutions [8, 18]. Additionally, we validate its effectiveness in AllianceDB, a multi-threaded SWJ benchmark testbed [43], showcasing an enhancement in managing out-of-order processing errors without compromising scalability. In summary, this paper provides the following contributions:

- Section 3 introduces the PECJ algorithm, tailored to balance both accuracy and latency in SWJ operations amid disordered data. The distinct advantage of PECJ lies in its proactive approach of incorporating the impact of yet-to-be-seen data for join error compensation.
- In Section 4, we delve into the mathematical formulation of how PECJ addresses the challenge of forecasting the future of evolving data streams. The disorder SWJ handling is initially abstracted into a *posterior distribution approximation* (PDA) problem, which is followed by optimizing the parameterization of its probability model via *variational inference* (VI).
- Section 5 presents two practical implementations of PECJ, demonstrating its adaptability. We begin with a straightforward, *analytical* implementation suitable for less dynamic streams and gradually progress to a more generalized form (*learning-based*) that employs machine learning for handling complex stream dynamics.
- Our experimental results, highlighted in Section 6, offer a comprehensive comparison between PECJ and the existing state-of-the-art methods. We provide data from both standalone tests and system integration tests, underscoring the superior performance of PECJ.

Table 1: Notations used in this paper

Type	Notations	Description
Tuple property	κ	Key of a tuple
	v	Payload of a tuple
	τ_{event}	The time of event occurrence of an input tuple
	$\tau_{arrival}$	The input tuple arrival time
	τ_{emit}	The time to emit an output tuple
	δ	The delay from event occurrence (τ_{event}) to event arrival ($\tau_{arrival}$) of an input tuple
Stream property	R, S	Two input streams to join
	$\mathbb{W}$	A bounded subset of data stream to join
	O	The aggregated results of $R \bowtie_{\mathbb{W}} S$
	ϵ	The relative error of output
	l	The processing latency
	ω	The assumed time point of window completeness
	n	The number of tuples
	σ	The join selectivity, as defined by [18]
	α	The average payload of joined tuples
	$\bar{r}_n$	Window-averaged tuple rate corresponding to n
	Δ	Maximum delay among all events from the time of occurrence (τ_{event}) to the time of arrival ($\tau_{arrival}$). $\Delta = \max_{\forall i}(\tau_{arrival} - \tau_{event})$
PDA abstraction	μ_w	A global variable for describing window-averaged contribution
	φ_w	A variable for describing other global information of a window
	U	The set of global variables, including the interested μ_w and φ_w
	X	The set of observations made on acquired tuples
	$p()$	The probability distribution
	$\mathbb{E}(k)$	The expectation of k
	Z	The set of latent variables
VI optimization	$q()$	The approximation function in variational family [17]
	$\mathbb{E}_j(k)$	The expectation of k, regarding on j (i.e., replace j by $\mathbb{E}(j)$ during estimating $\mathbb{E}(k)$)
	$ELBO_q$	The evidence lower bound
	H	The set of remapped parameters in U, Z

2 PRELIMINARY

This section provides a detailed introduction to Stream Window Join (SWJ), including the buffering mechanisms for handling disorder prevalent in existing research. We underline the fundamental distinction that PECJ introduces and briefly touch upon the technical challenges posed by our approach.

2.1 Stream Window Join and Key Definitions

Table 1 summarizes the notations used in this paper. For the purposes of this paper, we define a *tuple* y as $y = \tau_{event}, \kappa, v, \tau_{arrival}, \tau_{emit}$, where τ_{event}, κ, and v represent the event timestamp, key, and payload of the tuple, respectively. The tuple's arrival time at a system is denoted by $\tau_{arrival}$, while τ_{emit} signifies the moment the final result incorporating y is released to the user. An input stream, referred to as R or S, is a sequence of tuples arriving at the system (e.g., a query processor), which may arrive out-of-order with respect to their event timestamp.

We adopt the *windows* concept from [43] to perform infinite stream joins over limited subsets. A ***window*** is an arbitrary time range ($t1 \sim t2$), represented as $\mathbb{W} = [t1, t2]$. A tuple y belongs to $\mathbb{W}$ if its t_e falls within the $\mathbb{W}$ range. To denote the length of the window, we use $|\mathbb{W}|$. There are various types of SWJ operations, such as intra-window join [43], online interval-join [42], and sliding window join [31, 34] (e.g., sliding, tumbling, interval, etc.). In this paper, we use the intra-window join operation as an example for evaluating PECJ. However, PECJ can be readily adapted for other types of SWJ.

For given input streams R and S and a window $\mathbb{W}$, the intra-window join, hereafter referred to simply as SWJ, is represented as $R \bowtie_{\mathbb{W}} S = (r \cup s) | r \in R, s \in S, r \in \mathbb{W}, s \in \mathbb{W}$. The outcome of $R \bowtie_{\mathbb{W}} S$ is condensed into a scalar value O, either by counting the number of joined $(r \cup s)$ tuples or by executing a summation or average computation over $R.v$ and $S.v$. When O is dispatched to the user at the time point τ_{emit}, we consider the following two performance metrics for stream window join:

- **Accuracy**: This metric assesses the precision of O and is quantified by its relative error ϵ. Specifically, $\epsilon = \frac{|O^{opr} - O^{exp}|}{O^{exp}}$, where O^{opr} represents the aggregated value produced by an algorithm and O^{exp} is the expected value.
- **Latency**: For all tuples contributing to the generation of O, their τ_{emit} is defined as the moment when O is produced, and the latency l for each tuple is calculated as $l = \tau_{emit} - \tau_{arrival}$. In this study, we report the 95th percentile of the worst-case latency, a commonly used measure, referring to it as 95% l.

2.2 Limitations of Current Approaches

The optimal condition for SWJ is when data arrives *in sequence*—meaning, the ordering defined by τ_{event} perfectly aligns with the one determined by $\tau_{arrival}$ as depicted in Figure 1(a). In this situation, all data is fully accessible to the system, enabling the completion of the calculated window.

Figures 1(b)-(d) illustrate a scenario where the $\tau_{arrival}$ sequence diverges from that defined by τ_{event}, resulting in a *disordered* arrival. Under these circumstances, ensuring window completeness becomes challenging, with some data often remaining unseen (e.g., the late-arriving tuples $R1$ and $S2$, highlighted in red). Ignoring such unobserved data compromises accuracy. Conversely, waiting for this late data to arrive induces an indeterminate rise in processing latency, given the unpredictable arrival times of these tuples.

Existing methodologies attempt to combat disordered arrivals using a *buffering mechanism*, where observed data is retained in buffers while the system awaits a more complete set of window data. The longer the system waits, the fewer unobserved data points there are. To prevent infinite waiting, these systems often designate a certain point in time, ω, at which they assume the window is complete and all data has been observed, marking the end of data buffering. O is then emitted at τ_{emit}, where τ_{emit} equals ω plus the *processing time*. Given that ω tends to be smaller than the $\tau_{arrival}$ of late tuples, it effectively decreases the overall processing latency. Previous studies [8, 18, 19, 23] have proposed both explicit and implicit methodologies for determining ω.

Despite providing potentially autonomous and adaptable ω decisions, these approaches still frequently neglect the impact of *unobserved data*—data arriving post-ω —on the results. For example, in Figure 1(b), a ω of $10ms$ causes $R1$ and $S2$ to be missed, leading to an inaccurate output. To rectify this, ω can be extended to ensure $R1$ and $S2$ are included, as shown by the $50ms$ ω in Figure 1(c). However, increasing ω from $10ms$ to $50ms$ significantly raises latency, creating an inescapable sub-optimal trade-off between accuracy and latency.

In addressing the challenges, we present PECJ, a solution designed to elevate the performance of SWJ by integrating unobserved data into the processing workflow. Instead of merely waiting, PECJ proactively factors in the unobserved tuples (i.e., $R1$ and $S2$) for error correction before their arrival, as showcased in Figure 1(d). This strategy allows PECJ to achieve improved accuracy under the same ω compared to Figure 1(b), without needing to increase ω as in Figure 1(c), thus avoiding exacerbation of latency issues. This active inclusion of unobserved data is a cornerstone of PECJ's drive for enhanced results. However, this integration presents distinct challenges, the foremost of which is the need to *anticipate the impact of future data streams*. These challenges are further explored in the following section.

2.3 Challenges in Implementing PECJ

The realization of PECJ, while promising, poses a set of intricate challenges. The fundamental hurdle stems from the requirement to incorporate unseen data - a task that treads into the territory of predictions and probabilistic estimations.

The most straightforward approach might suggest leveraging time series prediction techniques to anticipate the contributions from each unseen tuple [39]. However, this approach can potentially lead to inconsistent accuracy levels. The prediction of individual tuple contributions is contingent on the estimation of the tuple volume, which further amplifies the risk of error propagation.

Further compounding the problem is the escalating complexity associated with time series predictions. As the length of the data increases, the complexity of predicting attributes of a *specific and predetermined number* of future data points can scale super-linearly [39]. This brings about substantial predictive overhead, which becomes increasingly pronounced when a large number of tuples remain unobserved.

Furthermore, the challenges are not solely limited to predictive accuracy and computational overhead. The need to keep latency within permissible thresholds adds another layer of complexity. The interplay between accuracy, computational efficiency, and latency management needs to be carefully navigated, requiring a more innovative and sophisticated approach than traditional methods can offer. Those challenges motivate our proposal of PECJ.

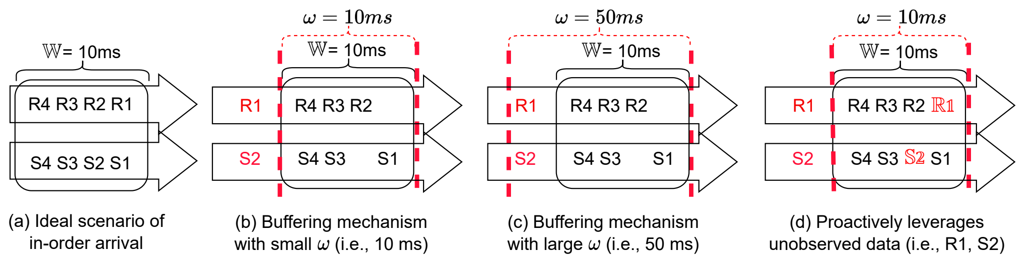
3 OVERVIEW OF PECJ

This section presents an overview of PECJ by first introducing its conceptual framework, followed by running examples.

3.1 Conceptual Framework of PECJ

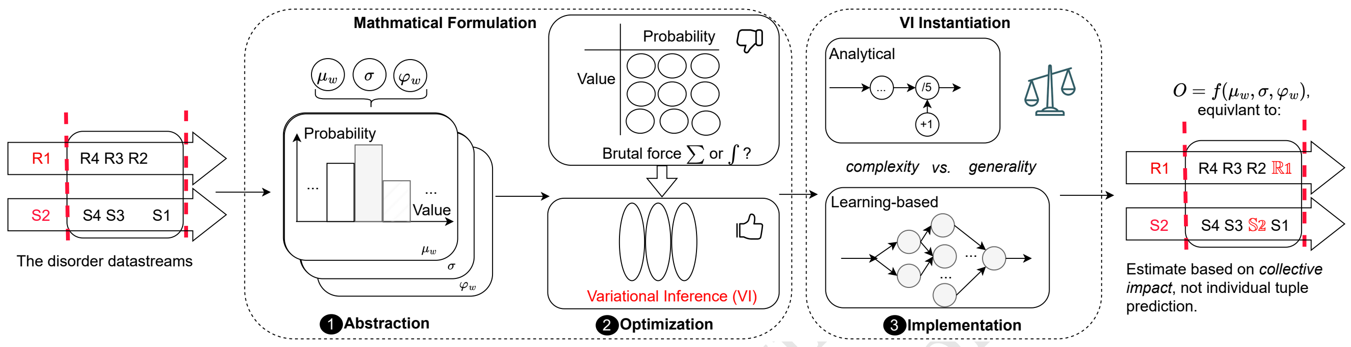
Designed to actively incorporate unobserved data, PECJ compensates for errors that arise in SWJ when dealing with disordered data streams. This subsection outlines the conceptual framework of PECJ, as depicted in Figure 2.

Abstraction: The first step involves directly abstracting the accurate SWJ result by extracting essential information from the disordered data streams to avoid the error propagation caused by per-tuple estimation (discussed in detail in Section 4.1). This phase essentially constitutes a *posterior distribution approximation* (PDA) problem, requiring the development of a probability model that is conscious of the data streams.



$\mathbb{W}$= 10ms
R4 R3 R2 R1
S4 S3 S2 S1
(a) Ideal scenario of in-order arrival
$\omega = 10ms$
R1 R4 R3 R2
S2 S4 S3 S1
(b) Buffering mechanism with small ω (i.e., 10 ms)
$\omega = 50ms$
(c) Buffering mechanism with large ω (i.e., 50 ms)
R4 R3 R2 R1
S4 S3 S2 S1
(d) Proactively leverages unobserved data (i.e., R1, S2)

Figure 1: Disorder handling of SWJ ($R \bowtie_{|\mathbb{W}|=10ms} S$)



The disorder datastreams
Mathmatical Formulation
μ_w σ φ_w
Probability
Value
1 Abstraction
Brutal force $\sum$ or $\int$?
Variational Inference (VI)
2 Optimization
VI Instantiation
Analytical
/5
+1
complexity vs. generality
Learning-based
3 Implementation
$O = f(\mu_w, \sigma, \varphi_w)$, equivlant to:
Estimate based on collective impact, not individual tuple prediction.

Figure 2: Conceptual framework of PECJ.

Optimization: Given the inherent challenges of efficient PDA parameterization, we turn to the *variational inference* (VI) approach for theoretical optimization (explained in Section 4.2). This approach drastically reduces the overhead of PDA parameterization, compared to brute force methods that involve possibly unmanageable summations and integrations, and inherently facilitates the evolution of the probability model in parallel with the data streams.

Implementation: Bridging the gap between the mathematical formulations of the previous stages and practical application, we provide both *analytical* and *learning-based* approaches of VI instantiations in Section 5. The *analytical* approach (Section 5.1) offers ultra-low overhead while accommodating relatively straightforward stream dynamics. We realize it using both Stochastic Variational Inference [17] (SVI) iterations and an Adaptive Exponential Moving Average Filter [14, 30] (AEMA) in PECJ. SVI offers a general way of conducting *analytical* approach by utilizing gradient descent, while AEMA involves much lower complexity. The *learning-based* approach (Section 5.2) seeks to depict various stream dynamics in a more generalized manner. We accomplish this by incorporating VI principles with PECJ's parameters of interest to formulate a loss function and use a simple Multilayer Perceptron (MLP) to demonstrate the core ideas.

3.2 Running Examples of PECJ

To further elucidate the application of PECJ, we present a running example. The tuples to be joined are outlined in Figure 3(a), with a window length of $6ms$. These consist of 6 tuples from streams R and S, formatted as 'Key (κ), Payload (v), Event Time (τ_{event}, in ms)'. Intriguingly, tuples $R4$ and $S1$ have not been observed at a certain ω (e.g., $5.1ms$).

Applying PECJ to the observed data enables us to enumerate the tuples in R, S. This yields $n_S = 5$ and $n_R = 5$ respectively (as displayed in Figure 3(b)). Additionally, PECJ detects 4 matches, of which two are under $\kappa = A$ and the other two fall under $\kappa = B$. This leads to a join selectivity [18] σ computed as $4/25$. In the case of a $JOIN - COUNT()$ query where the payload v doesn't affect results, O aligns with the number of matches, resulting in a count of 4. For a $JOIN - SUM(R.v)$ query where the v of the joined R is accumulated, we get $O = 20$. Moreover, the mean v of the joined R results in $\alpha_R = 20/4 = 5$. Nonetheless, these results do not reflect the true outcome as they exclude contributions from $R4$ and $S1$ who have not arrived by the ω.

To address the discrepancy of unobserved data, PECJ proposes to answer the question, 'what would O appear like if the contributions from unobserved data were factored in?' To do this, PECJ tackles a PDA problem using a VI approach, as shown in Figure 3(c). In this context, the PDA problem involves using patterns and hidden tendencies within data streams as evidence to estimate n_R, n_S, σ, and α_R. This represents an effort to account for the effects of stream dynamics on the observed data, which often distorts the true picture.

Unfortunately, calculating all possible configurations related to stream dynamics through brute force is intractable. For this reason, PECJ uses the VI approach, which transforms the cumbersome tasks of summation and integration into maximizing the evidence lower bound ($ELBO_q$) of VI with the gathered pieces of evidence, leading to significantly improved computation efficiency. This theoretical optimization is practically implemented under the *analytical* or *learning-based* approaches, effectively tailoring the posterior distributions of the estimated values.

As an example, PECJ might detect a high probability of a distortion of approximately -1 for n_S, n_R. This would suggest that the estimated n_S, n_R should conform to a Gaussian Distribution of $\mathcal{N}(6, 0.2)$, allowing us to use the expected value of 6 to estimate n_S, n_R. Upon amalgamating these estimated values of n_R, n_S, σ,

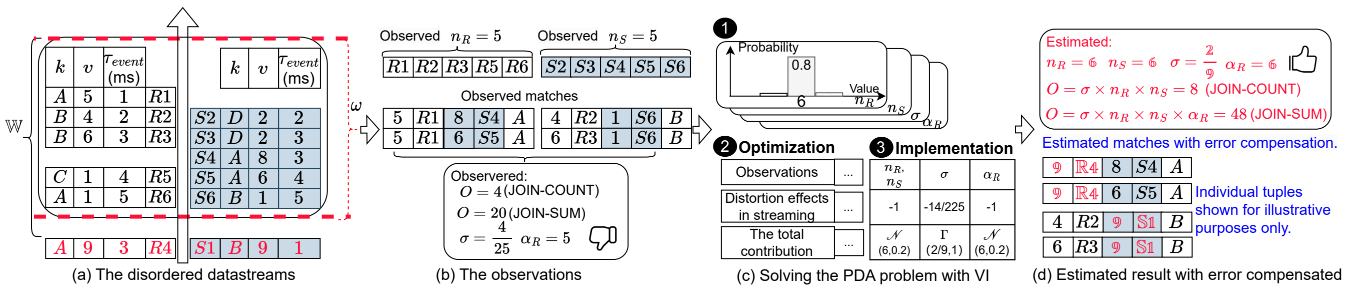

(a) The disordered datastreams
(b) The observations
Observed $n_R = 5$
Observed $n_S = 5$
Observed matches
Observered: $O = 4$ (JOIN-COUNT) $O = 20$ (JOIN-SUM) $\sigma = \frac{4}{25}$ $\alpha_R = 5$
(c) Solving the PDA problem with VI
Probability
Value
Optimization
Implementation
Observations
Distortion effects in streaming
The total contribution
(d) Estimated result with error compensated
Estimated: $n_R = 6$ $n_S = 6$ $\sigma = \frac{2}{9}$ $\alpha_R = 6$
$O = \sigma \times n_R \times n_S = 8$ (JOIN-COUNT)
$O = \sigma \times n_R \times n_S \times \alpha_R = 48$ (JOIN-SUM)
Estimated matches with error compensation.
Individual tuples shown for illustrative purposes only.

Figure 3: Running Example of PECJ.

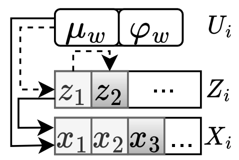


Figure 4: Probability model.

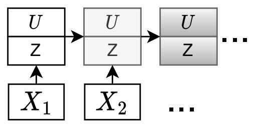


Figure 5: Parameterization as continual learning.

and α_R, PECJ can compute the rectified O. The calculation for the $JOIN-COUNT()$ query would result in

$$O = \sigma \times n_S \times n_R,$$

and for the $JOIN-SUM(R.v)$ query it would be

$$O = \sigma \times n_S \times n_R \times \alpha_R.$$

These computations integrate the contributions *as if* $R4$ and $S1$ had been present at the time of computation, as illustrated in Figure 3(d).

4 MATHEMATICAL FORMULATION

PECJ solves a *posterior distributions approximation* (PDA) problem, optimized via *variational inference* (VI), to address the challenges discussed in Section 2.3. We begin by extracting critical information from the data streams and formulating a streaming-aware probability model to minimize error propagation (Section 4.1). We then employ VI for efficient model parameterization, ensuring our solution caters to the low-latency demands of SWJ (Section 4.2).

4.1 Formulating the Probability Model

PECJ approximates the posterior distribution of the total contribution from all tuples within a window, encompassing both observed and unobserved data. This strategy diverges from the approach of predicting individual tuples via time-series predictions [39]. Our solution eliminates the need for per-tuple approximation or compensation, thereby reducing potential error propagation. This propagation originates from the interdependent prediction of tuple number (n_S, n_R) and the contribution of each tuple to α_R, σ (as discussed in Section 2.3). Specifically, PECJ estimates the parameters of the *window-averaged total contribution* (μ_w) directly, perpetually learning from the data stream observations. μ_w is defined as $\mu_w = \frac{1}{|\mathbb{W}|}\sum_{y\in\mathbb{W}}^{\mathbb{W}} \mu(y)$. Here, $\mu(y)$ represents a tuple's additive contribution factor, which is summed across all tuples in a window and then normalized by the window length $|\mathbb{W}|$ to define a μ_w. Each μ_w encapsulates a certain type of *averaged* global information within a window, such as join selectivity (σ) or average payload (α) in Section 3.2. For the *accumulated* effects, represented by the n notation, we convert it by the corresponding window average, e.g., $n = \bar{r}_n \times |\mathbb{W}|$, where $\bar{r}_n$ refers to the averaged tuple rate and can also be viewed as a parameter of the window-averaged total contribution. It is crucial to note that σ, α_R, and $\bar{r}_n$ are abstracted in a manner similar to the μ_w notation as each of them describes a certain type of *window-averaged total contribution*. Furthermore, they can be estimated *independently*, avoiding the prediction dependency mentioned in Section 2.3.

PECJ employs specific μ_w variables such as σ to calculate the join aggregation output O (as defined in Section 3.2), thereby facilitating proactive compensation for disorder handling errors. The remaining challenge involves approximating the posterior distribution of μ_w given the corresponding observations $X = \{x_1, x_2, ...\}$ from the data streams. Essentially, we aim to determine the posterior distribution $p(\mu_w|X)$, with X evolving as the data stream progresses.

We might also desire additional parameters φ_w, such as the inverse variance of μ_w estimation, which is connected to the credible interval. Both μ_w and φ_w form part of a window's global information U, i.e., $\mu_w, \varphi_w \in U$. For a general illustration, we utilize the $p(U|X)$ notation, as it encompasses both $p(\mu_w|X)$ and $p(\varphi_w|X)$. Our approximation objective can be summarized as follows:

Objective 1. *Approximate the $p(U|X)$, estimating the U by utilizing its expectation given X, i.e., $\hat{U} = \mathbb{E}(U|X)$.*

Data streams' inherent dynamics and randomness [7] can cause significant deviations in the observations X from the global U. Therefore, unlike approximating a static dataset [22, 40], the dynamic nature of streaming data requires special consideration. Hence, we employ *latent variables* in our model. As illustrated in Figure 4, our probabilistic model incorporates both global information U and latent variables $Z = \{z_1, z_2, ...\}$.

In our model, directed arrows denote probabilistic dependencies. Specifically, our observations X depend on both the global variables U and the latent variables Z, while the latent variables Z may also be influenced by the global variables U. The task of inferring the posterior distribution involves utilizing our observations to update our understanding of these dependencies, which is essential for probabilistic data association tasks in SWJ. Notably, the latent variables Z serve as mediators between the global variables U and the observations X.

Each variable z_i in Z directly influences specific observations in X, embodying temporal or local dynamics. For instance, in Figure 4, z_1 impacts both x_1 and x_2, while z_2 only affects x_3. To encapsulate a wide spectrum of streaming dynamics, we emphasize that Z: 1) does not necessarily have to correspond to X in length, 2) can contain variables (z_i) of any dimension, and 3) might include variables that are influenced by U or other latent variables.

The incorporation of Z into the model helps unravel complex relationships and dependencies between X and U. In particular, Z can expose patterns and trends in the data streams that might not be immediately noticeable when examining X alone, providing valuable insights into the underlying streaming processes for more accurate U estimation. However, this introduces an additional level of complexity, as the latent variables can cause our observations to deviate from the underlying state represented by U.

4.2 Optimizing Model Parameterization with `VI`

The inherent complexity of streaming data significantly challenges the parameterization of a probability model, especially when latent variables are involved. In particular, calculating $p(U|X)$ necessitates treating Z as constants within the joint distribution $p(U, Z, X)$. This step entails exponential computational complexity growth due to the integration or summation over all potential Z configurations. In SWJ applications where low latency is essential, such computational overhead is undesirable. Moreover, we continuously need to update the model parameters to handle new incoming data and promptly make inferences.

Considering these challenges, we employ *variational inference* (`VI`) [7, 17, 36] for model parameterization. `VI`'s central idea is to approximate the true posterior (e.g., $p(U|X)$ in Objective 1) with a simpler, tractable distribution family. `VI` strikes a balance between the efficiency and accuracy of a complex probability model's parameterization, allowing U and Z to be approximated without resorting to brute-force integration or summation. Moreover, it inherently supports continual learning on data streams by integrating new observations into the existing distributions [11].

`VI` is advantageous for `PECJ`, outperforming traditional approaches in three major aspects. Firstly, `VI` is much less prone to overfitting compared to Maximum Likelihood Estimation (MLE) [9]. Although straightforward, MLE will often fail to accurately estimate the latent variables set Z due to overfitting, despite Z being crucial for reflecting stream dynamics. Secondly, `VI` incurs less overhead than Markov Chain Monte Carlo (MCMC) [9], making it easier to meet the SWJ's low latency requirements. While MCMC ensures high accuracy by directly sampling from the posterior distribution $p(U|X)$, the computational cost is prohibitive, particularly when dealing with a large Z and a high z_i dimension to reflect complex stream dynamics. Lastly, compared to regularization methods like L1 and L2 [15], `VI` enables a robust evolution with the data streams. While L1 and L2 can mitigate MLE's overfitting to some extent, they introduce additional hyperparameters that need careful tuning. This tuning becomes increasingly challenging as the data streams observed by `PECJ` evolve.

Although the successful use of `VI` in other problems such as latent dirichlet allocation [16], autoencoder construction [38], and concept drift detection [7] is acknowledged, these existing works aren't designed for the `PDA` process involved in SWJ. These works are meant for different probability models where estimating the global information U from data streams isn't required. In the following sections, we delve deeper into our `VI` approach's mechanics.

Approximation of $p(U|X)$. We select a manageable family of $q()$ functions, referred to as the *variational family* [17], to approximate the $p()$ distributions. The variational family liberates us from intractable and costly integration computations. A popular choice is the mean-field variational family. Specifically, our $q()$ functions hold the following relationships and constraints. The $\approx$ symbol denotes approximation.

Equation 1 requires that each component in U is considered independent in the approximation function $q()$, and it further designates $q(U)$ as an approximation to our target distribution $p(U|X)$. Equations 2 and 3 are similar, approximating the conditional prior distribution of $Z|U$ and the joint distribution of U, Z, respectively. When U and Z are independent, the $Z|U$ notation inside brackets can be further simplified to Z. The core idea of Equations 1 to 3 is to decompose each variable into separate distributions during the approximation, e.g., the $q(\mu_w)$, $q(\varphi_w)$, and $q(z_i)$ components. This way, we can apply divide and conquer to each variable, avoiding brute force summation or integration.

$$q(U) = \prod_{\mu_w \in U} q(\mu_w) \times \prod_{\varphi_w \in U} q(\varphi_w) \approx p(U|X) \tag{1}$$

$$q(Z|U) = \prod_{z_i \in Z} q(z_i) \approx p((Z|U)|X) \tag{2}$$

$$q(U, Z) = q(U) \times q(Z|U) \approx p(U, Z|X) \tag{3}$$

Rather than brute force computation, the primary mathematical task of `VI` is to bring $q()$ close to $p()$, which is a simpler *optimization* problem. Specifically, the optimization goal is to maximize the *evidence lower bound* ($ELBO_q$), defined in Equation 4. The $\mathbb{E}_q()$ notation refers to the expectation regarding our approximation functions $q()$. The key insight of Equation 4 is to optimize the utilization on the X (i.e., used as the *evidence*) by finding the balance between explaining the observations and retaining uncertainty. The first term, $\mathbb{E}_q(log((p(U,Z,X)))$, represents the expected log-likelihood of our observations given the model. It encourages the model to explain the X well. The second term, $\mathbb{E}_q(log((q(U,Z))))$, is the entropy of the approximation function $q()$. This term encourages the model to remain uncertain and not commit to a single explanation prematurely. In this way, we can find a good approximation of the posterior $p(U|X)$.

Objective 2. *maximize $ELBO_q$*

$$s.t., ELBO_q = \mathbb{E}_q(log((p(U, Z, X))) - \mathbb{E}_q(log((q(U, Z)))) \tag{4}$$

Continual Learning from Observations. Crucial to any real-world model is continual learning—the capacity to assimilate new information progressively, while retaining previously learned knowledge. This feature is particularly important for models dealing with infinite data streams, such as those seen in finance, health informatics, and social media analytics, where distributions are in constant flux.

In contrast to traditional machine learning models that undergo batch training and risk catastrophic forgetting when exposed to new data, our model operates within the continual learning paradigm. This design equips our model with a 'rolling memory', which recalls past information, prevents catastrophic forgetting, and adapts seamlessly to evolving environments and incoming data. However, effectively implementing continual learning in the face of endless data streams poses its challenges. Specifically, it's impractical to store the complete history of X and execute VI for every new addition to X. Thus, we treat model parameterization as a continual learning process as illustrated in Figure 5.

Assume that we have drawn insights from a previous observation X_1 and have established approximations for U and Z. These approximations can then be updated with the new observation X_2, eliminating the need to recompute using the entire $X = \{X_1, X_2\}$. In line with the method proposed in [11], we employ the *prior distribution* of U (i.e., $p(U)$) as the initial conditions, with "starting" not indicating a clean slate. The following equation, Equation 5, illustrates this process. The $q(U_1)$, derived from old observation X_1, can act as the new prior distribution. This prior can then be integrated with the impacts from the new observation (i.e., $p(X_2|U)$) to update our approximation. We acknowledge the complexity of continual learning optimization methodologies such as coreset selection [26] and designate them as subjects for future research.

$$p(U|X) = p(U|X_1, X_2) \propto p(X_2|U)p(U|X_1) \approx p(X_2|U)q(U_1) \quad (5)$$

5 VI INSTANTIATION

Implementing PECJ necessitates the instantiation of the VI equations as delineated in Section 4.2. However, the precise organization and interrelationships between U and Z have substantial implications for PECJ's overhead and versatility, requiring a judicious design approach. This section explores two pragmatic instantiations, initially focusing on the *analytical* method [17], and subsequently examining the *learning-based* approach [7, 36].

5.1 Analytical Instantiation

This instantiation extends the central limit theorem by incorporating a basic awareness of streaming dynamics. Specifically, Z is positioned between the global (i.e., U) of central limit theorem and its sample (i.e., X). There is no local latent variable z_i dependent on the global variable $U = \{\mu_w, \varphi_w\}$. Furthermore, μ_w is independent of φ_w here. As a result, we can simplify the $U|Z$ notations in Equation 2 (Section 4.2), given that we have independent U, z_i.

Equation 6 asserts that each observation $x_i \sim \mathcal{N}(\mu_w/z_i, 1/z_i\varphi_w)$. It implies that x_i is influenced not solely by the global mean μ_w and global variance $1/\varphi_w$ of a Gaussian Distribution, but also by the reverse linear distortions represented by the transient dynamics z_i. When we couple Equation 6 with the prior distribution of μ_w, φ_w, z_i, denoted as $p(\mu_w), p(\varphi_w), p(z_i)$ respectively, we can derive the joint distribution function $p(U, Z, X)$ for all variables in Equation 7, where $Z = \{z_1, z_2, ...z_n\}$ and $X = \{x_1, x_2, ..., x_n\}$.

$$f(x_i|\mu_w, \varphi_w, z_i) = e^{-(z_i \times x_i - \mu_w)^2 * \varphi_w/2} \times \sqrt{\varphi_w} \times const \quad (6)$$

$$p(U, Z, X) = const \times \varphi_w^{n/2} \times e^{\sum_{i=1}^{n}(z_i \times x_i - \mu_w)^2 \times \varphi_w/2} \times p(\mu_w)p(\varphi_w)\prod_{i=1}^{n} p(z_i) \quad (7)$$

When VI converges to a mean-field family of $q()$ and Equation 4 is achieved, an analytical solution [9, 17] exists for $q(\mu_w)$, as shown in Equation 8. The notation $\mathbb{E}_{\varphi_w,Z}$ indicates that the approximations of μ_w can be facilitated by the expectations of other variables, specifically φ_w and Z, rather than performing exhaustive computation of their integration or summation. This is a consequence of the decoupling property inherent to the mean-field family. Moreover, if the prior distribution of μ_w is a Gaussian $\mathcal{N}(\mu_0, 1/\tau_0)$, $q(\mu_w)$ culminates in a Gaussian posterior distribution of μ_w expressed as $\mu_w \sim \mathcal{N}(\frac{\tau_0\mu_0 + ng(X)}{\tau_0 + n}, \frac{1}{(\tau_0+n)\mathbb{E}(\varphi_w)})$. From this Gaussian posterior distribution, we can deduce two crucial insights:

(1) The **estimated value** of μ_w (denoted as $\bar{\mu_w}$) behaves like a *linear function* of X as shown in Equation 9. Notably, the coefficient vector K correlates with the expectations of each latent variable, represented as $\mathbb{E}(z_i)$.
(2) The **credible interval** for estimating μ_w is related to $\mathbb{E}(\varphi_w)$, as depicted in Equation 10. For example, the 95% credible interval is calculated as $\bar{\mu_w} \pm 1.96\frac{1}{\sqrt{(\tau_0+n)\mathbb{E}(\varphi_w)}}$.

$$\begin{aligned} q(\mu_w) &= \mathbb{E}_{\varphi_w,Z}(f(U, Z, X)) \\ &= const \times p(\mu_w)e^{-(\mu_w - g(X,Z))^2 \times (n\mathbb{E}(\varphi_w)/2)} \end{aligned} \quad (8)$$
$$\text{where } g(X, Z) = \sum_{i=1}^{n} \frac{\mathbb{E}(z_i) * x_i}{n}$$

$$\exists \text{vector } K \text{ and scalar } b, s.t., \bar{\mu_w} = \mathbb{E}(\mu_w) = KX + b$$
$$\text{where } KX = \frac{ng(X, Z)}{\tau_0 + n}, b = \frac{\tau_0\mu_0}{\tau_0 + n} \quad (9)$$

$$\forall \text{credible interval } \delta \in (0, 1),$$
$$\bar{\mu_w} - i(\delta)\frac{1}{\sqrt{(\tau_0 + n)\mathbb{E}(\varphi_w)}} \leq \mu_w \leq \bar{\mu_w} + i(\delta)\frac{1}{\sqrt{(\tau_0 + n)\mathbb{E}(\varphi_w)}}$$
$$\text{where } i(\delta) \text{ is the } \delta \text{ interval quantile of a standard Gaussian.} \quad (10)$$

We can use Stochastic Variational Inference (SVI) [17] to conduct the *analytical* instantiation by extending Equation 8 to calculate φ_w and z_i. We then employ gradient descent to maximize $ELBO_q$. Technically, gradient descent minimizes functions, but by applying it to the negative of $ELBO_q$, we can effectively maximize $ELBO_q$. Alternatively, given the straightforward linear form, techniques such as the Exponential Moving Average (EMA) or the ARIMA model [14, 30] can also be applied. However, a distinguishing aspect of our scenario is that the parameters of the filter should dynamically evolve with the data streams, rather than being preset. This dynamic adaptability ensures accurate on-the-fly approximation of $\mathbb{E}(z_i)$.

By default, PECJ employs a variant of the EMA, which we term as an *Adaptive EMA* (AEMA). In AEMA, the decay parameter of the EMA is not fixed but continuously updated based on rule-based learning from the data streams. This choice is motivated by the expectation that an adaptive approach will incur significantly less overhead compared to SVI, while also being simpler to design and adjust.

5.2 Learning-based Instantiation

Although more intricate U, Z relationships can be defined to enhance the representative capacity of the *analytical* instantiation, this approach demands significant manual effort. Moreover, implementing a more complex *analytical* approach may be impractical due to the intricate mathematical relationships involved (see Appendix A for details).

To overcome the challenges of capturing complex stream dynamics, we refer back to the abstract ELBO definition in Equation 4 for a more universal solution. This approach doesn't require knowledge or assumptions about specific relationships between U and Z, nor does it require familiarity with the length of Z or the dimensions of z_i. In particular, just awarding the existence of U, Z dependency is sufficient.

First, we remap the entire parameter space of U and Z into another space, $H = \{h_1, h_2, ..., h_m\}$, i.e., $U, Z \rightarrow H$. Hence, Equation 4 can be rewritten as Equation 11. **Second**, we further constrain H by ensuring 1) the independent μ_w and φ_w presented in Equation 6 and Equation 7 are assigned to h_1 and h_2, respectively, and 2) the remaining factors $h_3, h_4, ...h_m$ form an Orthogonal Basis (i.e., they are independent of each other) given h_1, h_2. As a result, the $log((p(H, X))$ term can be decomposed as shown in Equations 12∼ 13. Note that $log((p(X|H)))$ is the *log-likelihood* of X in the H space, and $log((p(h_i)))$ is the *log-prior-distribution* of h_i. As both are irrelevant to q, we can conveniently remove the $\mathbb{E}_q$ notations. **Third**, based on the mean-field property [9, 17], $\mathbb{E}_q(log(q(H)))$ can be further decomposed as per Equation 14. **Finally**, by separating $q(\mu_w)$ and $q(\varphi_w)$ from the other $q(h_i)$, we can derive Equation 15. It should be noted that the resulting $\mathbb{E}(\mu_w|X)$ and $\mathbb{E}(\varphi_w|X)$ are the expectations of μ_w and φ_w given X, respectively. They can be directly utilized for the estimated value in PECJ's error compensation, as discussed in Section 4.1.

$$
\begin{aligned}
ELBO_q &= \mathbb{E}_q(log((p(H,X))) - \mathbb{E}_q(log((q(H))) && (11)\\
&= \mathbb{E}_q(log((p(X|H)p(H)))) - \mathbb{E}_q(log((q(H))) && (12)\\
&= log(p(X|H)) + log(p(\mu_w)) + log(p(\varphi_w)) && \\
&+ \sum_{i=3}^{m} log(p(h_i|\mu_w,\varphi_w)) - \mathbb{E}_q(log(q(H))) && (13)\\
&= log(p(X|H)) + log(p(\mu_w)) + log(p(\varphi_w)) && \\
&+ \sum_{i=3}^{m} log(p(h_i|\mu_w,\varphi_w)) - (\sum_{i} \mathbb{E}_q(log(q(h_i)))) && (14)\\
&= log(p(X|H)) + log(p(\mu_w)) + log(p(\varphi_w)) && \\
&+ \sum_{i=3}^{m} log(p(h_i|\mu_w,\varphi_w)) - (\sum_{i=3}^{m} \mathbb{E}_q(log(q(h_i))) && \\
&+ log(\underline{\mathbb{E}(\mu_w|X)}) + log(\underline{\mathbb{E}(\varphi_w|X)}) && (15)
\end{aligned}
$$

Equation 15 can further be leveraged to regulate the behavior of neural networks (NNs), enabling them to conform to the PDA process without requiring knowledge of the relationships between U and Z. Here's a detailed three-step, ELBO-driven solution:

(1) Construct an NN for function fitting, ensuring that the final output is at least seven-dimensional to correspond with the seven scalars depicted in Equation 15.

(2) Conduct supervised pre-training over the entire NN so that each dimension accurately estimates the target scalar, such as $log(\mathbb{E}(\mu_w|X))$. Given that pre-training is fundamentally a function-fitting process, loss functions that have been originally designed for fitting, such as the mean square error, are appropriately suitable for this task.

(3) During continual learning in a streaming environment, Equation 15 can be employed to optimize NN loss. For example, if gradient descent is implemented via ADAM or SGD [3], the loss function can be designed to decrease monotonically with $ELBO_q$. Note that, if the NN is overly 'confident,' the numerical evaluation of $ELBO_q$ could potentially be ∞. In such instances, we use bounded functions such as $-sigmoid(ELBO_q)$ as the loss function.

In PECJ, we implemented a straightforward multilayer perceptron (MLP) to briefly illustrate this concept, leaving more powerful structures like LSTM [7] or transformer [36] for future exploration. Furthermore, given the necessity for NNs to meet low latency requirements, it's critical to efficiently perform their inference and learning processes. As a result, an effective solution for deploying PECJ across various dynamic situations is to integrate a well-structured NN with high-performance computing. Pursuing this combination represents an important area of ongoing work.

6 EVALUATION

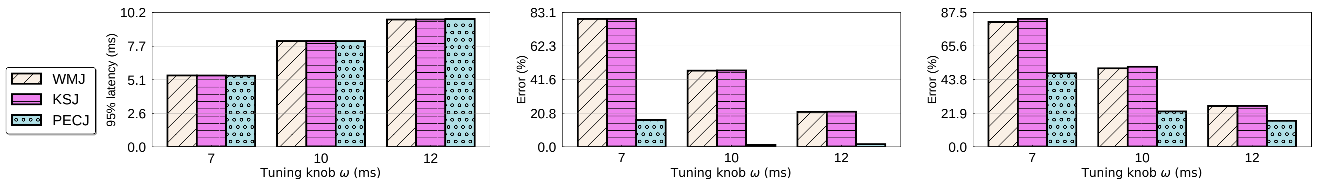
In this section, we present a comprehensive evaluation of PECJ in comparison with other state-of-the-art techniques. In summary, across various aspects of our investigation, we have made the following key observations.

- PECJ has consistently proven superior in managing disordered data. From an end-to-end comparison with *WMJ* and *KSJ* (Section 6.3), PECJ emerged as more effective, maintaining lower error rates even under intricate disorder arrival patterns and lenient real-time requirements.
- The efficiency of PECJ was further validated under different workload conditions, as it successfully handled complex scenarios with varying numbers of join keys and high event rates (Section 6.4).
- By comparing $PECJ_{\text{analytical}}$ and $PECJ_{\text{learning}}$ (Section 6.5), we observed that while both models outperform the baseline, $PECJ_{\text{learning}}$ exhibits a higher degree of resilience in complex situations and under substantial observation distortion.
- Lastly, the integration of PECJ into PRJ and SHJ demonstrated substantial error rate reductions without significantly impacting latency or scalability (Section 6.6).

6.1 Experimental Setup

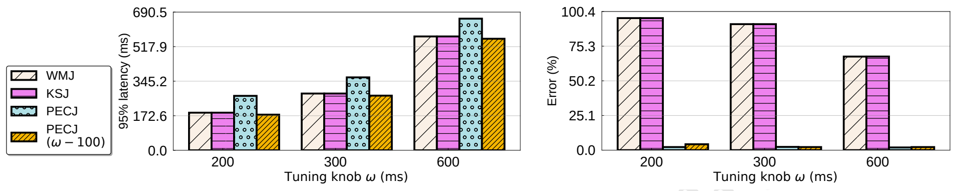
We established a robust experimental setup to thoroughly evaluate the performance of PECJ. The various components of this setup are detailed below.

Server: The experiments were conducted on a state-of-the-art multicore server powered by Intel Xeon Gold 6252 processors, which feature 24 cores and support 2 threads per core through HyperThreading. The server has a considerable L3 cache size of 35.75MB and a massive memory capacity of 384GB. It operates on



(a) 95% latency of Q1 (b) Relative error of Q1 (c) Relative error of Q2

Figure 6: End-to-end comparison.



(a) 95% latency (b) Relative error

Figure 7: End-to-end comparison of Q3. PECJ (ω-100) notation refers to subtracting the ω of PECJ by $100ms$.

the Ubuntu 22.04 system and uses the g++ 11.3.0 compiler for the compilation of the source codes.

Datasets: The evaluation was carried out using a diverse collection of four widely-used real-world datasets - **Stock**, **Rovio**, **Logistics**, **Retail**, and a synthetic dataset known as **Micro**. The **Stock**, **Rovio**, and **Micro** datasets were adopted from AllianceDB [43], while the **Logistics** and **Retail** datasets were obtained from a recent open source project [42]. To simulate a realistic scenario, we introduced disorder in the data arrival by reordering the arrival timestamps $\tau_{arrival}$ differently from the event timestamps τ_{emit} (as mentioned in Section 2). The difference between $\tau_{arrival}$ and τ_{emit}, i.e., δ, was set randomly for all tuples. We kept the event rate (controlled by event timestamp τ_{emit}) of both R and S streams consistent at $100Ktuples/s$ unless stated otherwise. We use **Stock** datasets in Section 6.3 and 6.5, and vary the usage of datasets in Section 6.4 and 6.6.

Queries: Three different queries were employed in our evaluation. **Q1**: This query entails a SWJ aggregated by COUNT (Section 3.2), with a $|\mathbb{W}|$ of $10ms$, and a maximum value of δ among all tuples, i.e., Δ, set as $5ms$. The small Δ is representative of a scenario where the stream processing is geographically close to the data source, such as on the edge of a cloud network [41]. **Q2**: This query modifies **Q1** by changing the aggregation function to SUM (Section 3.2), with all other settings retained as per **Q1**. **Q3**: This query extends **Q1** by altering the disordered arrival pattern of data and setting the Δ to $1000ms$. The significant Δ simulates situations where the stream analytic is situated far from the data source, such as during multiple intercontinental communications within a TOR network [13].

While **Q1** and **Q2** are tailored to require ultra-low latency processing, typically tens of milliseconds or less, **Q3** cannot expect such low latency due to the large arrival delay. Nonetheless, the goal is to achieve a latency below $500ms$, which is half of its Δ.

6.2 Implementation Details

In our evaluation, we scrutinize the performance of PECJ using two distinct setups: standalone and integrated implementations. Each setup facilitates a comprehensive comparison with different existing approaches. Note that, while the automatic determination of suitable ω is orthogonal to this work, it serves as a tuning knob for all mechanisms during the experiments. Specifically, we set ω to $|\mathbb{W}|$ of three queries, i.e., $10ms$ by default and manually tune it in the experiments.

A) Standalone Implementation: In the standalone implementation setup, we're aiming for an algorithmic comparison between PECJ and two existing methodologies, namely **K-Slack-Join (*KSJ*)** [18] and **Watermark-Join (*WMJ*)** [8]. For these standalone implementations, we employed the same C++ codebase for *KSJ*, *WMJ*, and PECJ.

Our implementation of PECJ included three separate approaches for the *analytical* and *learning-based* approaches. For the former (discussed in Section 5.1), we utilized both the Adaptive Exponential Moving Average (AEMA) and Stochastic Variational Inference (SVI) instantiations. For *learning-based* (Section 5.2), we opted for a simple learning approach of Multi-Layer Perceptron (MLP). The AEMA instantiation served as the default configuration for PECJ's *analytical* approach.

KSJ uses a k-slack buffer approach to manage the disorder in data streams. After data streams are preprocessed through the k-slack buffer, *KSJ* conducts a standard hash-join operation, treating the data as ordered. Importantly, our tuning parameter, ω, is tied to the k-slack buffer's control conditions, as discussed in Section 2. On the other hand, *WMJ* applies the watermark mechanism [8] for data preprocessing, eliminating the need for a k-slack buffer. Each watermark indicates the arrival of tuples with $\tau_{event} < T$, enabling the computation to commence early upon watermarks' arrival. However, the emission of O waits until the ω is reached.

B) Integrated Implementations: This setup is designed to assess PECJ's performance when incorporated into an existing multi-threaded stream processing system. AllianceDB [43], which is a recent multi-threaded SWJ testbed and serves as our integration platform. In this environment, we selected two representative parallel SWJ algorithms, Parallel Radix Join (PRJ) and Symmetric Hash Join (SHJ), to perform our assessment.

PRJ adopts a 'lazy' approach, delaying the join operation until all tuples have arrived. Conversely, SHJ pursues an 'eager' strategy, initiating the join process as soon as a portion of tuples arrives. Both PRJ and SHJ operate under the assumption of in-order arrival, and consider a window complete when the first tuple's arrival timestamp ($\tau_{arrival}$) surpasses the window's boundary.

6.3 End-to-End Comparison

We initiate our analysis by juxtaposing PECJ, *KSJ*, and *WMJ* under the conditions stipulated by **Q1** using the Stock dataset. The assumed time point of window completeness ω is fine-tuned to $7ms$, $10ms$, and $12ms$ for each methodology. Subsequently, we elucidate the ensuing 95% processing latency (95% l) and relative error (ϵ) in Figures 6(a) and 6(b).

Three critical insights emerge from this comparative analysis. Initially, it is observed that for the same ω, each strategy incurs a similar latency, as depicted in Figure 6(a). This congruity arises mainly due to the similar overhead incurred from waiting for a more comprehensive window of data. Relative to this waiting overhead, the specific overheads engendered by *WMJ*, *KSJ*, and PECJ are marginal. Secondly, as anticipated, the error generated by *WMJ* and *KSJ* exhibits similarity and consistently decreases with larger ω values. Despite their distinct mechanisms for handling disordered data, they have an identical level of data completeness within a given window under the same ω. Consequently, their ignorance extent towards unobserved data also aligns.

Most notably, PECJ manifests its superior performance in significantly lower errors compared to *WMJ* and *KSJ*. For instance, when ω is set to $7ms$, PECJ can maintain an error as low as $\leq 16\%$ with a 95% l of $\leq 5.5ms$. In contrast, *WMJ* and *KSJ* register an error in excess of 20%, even when the 95% l escalates above $9.5ms$ by setting ω to $12ms$. As expounded earlier, this improved performance is attributed to PECJ's proactive strategy of incorporating the contributions of unobserved data, unlike the passive waiting approach of *WMJ* and *KSJ* (Section 3).

Shifting our focus to **Q2**, we maintain identical settings as in the previous experiment. Given the similar 95% l patterns across these strategies, we primarily present the resulting relative error (ϵ) in Figure 6(c). Despite **Q2** demanding a more intricate syntax and involving additional parameters compared to **Q1** (Section 3.2), PECJ retains its superior performance, evident through its significantly reduced error. For instance, when the ω is adjusted to $10ms$, the error incurred by PECJ is as low as 25.0%, compared to a substantial 52% for *WMJ* and 51.5% for *KSJ*. The minor 0.5% ϵ reduction of *KSJ* compared with *WMJ* is due to the partial re-ordering inherent in the k-slack methodology.

Lastly, we examine **Q3**, which has a more intricate disorder arrival pattern and less stringent real-time requirements (Section 6.1), we adjust PECJ from *analytical* to *learning-based*. We set ω to $200ms$, $300ms$, and $600ms$ and present the corresponding results in Figures 7(a) and 7(b). Our findings show that *WMJ* and *KSJ* fall short in adapting to this scenario, where data disordering manifests in an extreme fashion. Notably, even with ω set to a lenient $600ms$, allowing for a latency of around $530ms$, they still yield an unacceptably high error over 70%.

Contrarily, PECJ consistently maintains the error within 3%, leveraging the *learning-based* PDA to compensate for the error (Section 5.2). It's important to acknowledge that the *learning-based* approach of PECJ introduces an additional latency of around $90ms$ (Figure 7(a)). However, as this extra latency is a by-product of a constant inference process, it can be circumvented by reducing ω by $100ms$, i.e., the PECJ (ω-100) configuration. Consequently, the PECJ (ω-100) still manages to maintain the error within 5%.

6.4 Workload Sensitivity Study

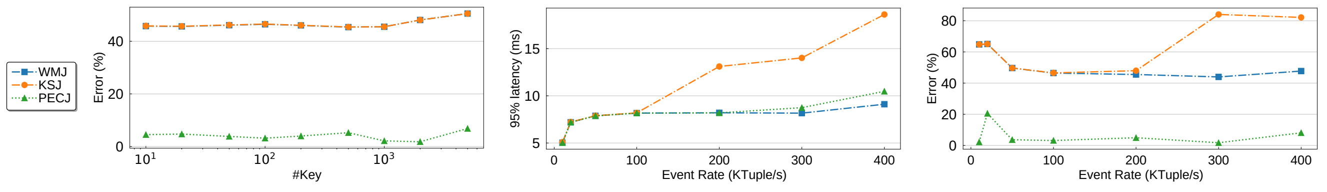
This subsection of the sensitivity study aims to contrast PECJ with the baseline models, *WMJ* and *KSJ*, under a range of workload characteristics. These include the number of join keys and the event rate. For the purposes of this study, we fix ω to $10ms$ and operate under a SWJ with a window length of $10ms$, followed by SUM. To adjust the workload characteristics, we utilize the synthetic dataset **Micro** [43] and set the Δ as $5ms$.

To assess the impacts of join keys, we distribute the keys of both R and S randomly and vary the number of keys from 10 to 5000, while maintaining the event rate at our default setting of $100Ktuple/s$. Since the number of join keys has virtually no impact on the latency of PECJ, *WMJ*, and *KSJ* (with a fluctuation of approximately $\pm 0.6\%$ around $8.25ms$ at most), we present the relative error in Figure 8(a). In general, PECJ outperforms the baseline models across a wide range of the number of keys. However, when the number of keys increases to as high as 5000, the likelihood of encountering a join match diminishes, which leads to fewer observations on join selectivity σ and slightly elevates its error.

Next, we hold the number of join keys at 10, and adjust the event rate from $10KTuple/s$ to $400KTuple/s$. The resulting 95% l and ϵ are displayed in Figure 8. Our findings show that *KSJ* experiences a latency 50% higher than either *WMJ* or PECJ when the event rate reaches $200KTuple/s$, and its ϵ also begins to escalate under such high event rate. This phenomenon occurs because 1) the k-slack overhead swells with a larger number of tuples processed per unit of time (i.e., the higher event rate), causing *KSJ* to overload much more readily than *WMJ* or PECJ, and 2) when an overload transpires, the partial reorder in *KSJ* becomes asynchronous, further increasing its error. Compared to *WMJ*, PECJ is slightly more prone to overload, particularly at event rates as high as $400Ktuple/s$ due to the extra overhead involved in making observations and executing compensations. Nonetheless, PECJ consistently achieves the smallest error under a non-overload rate, and even under a mild overload.

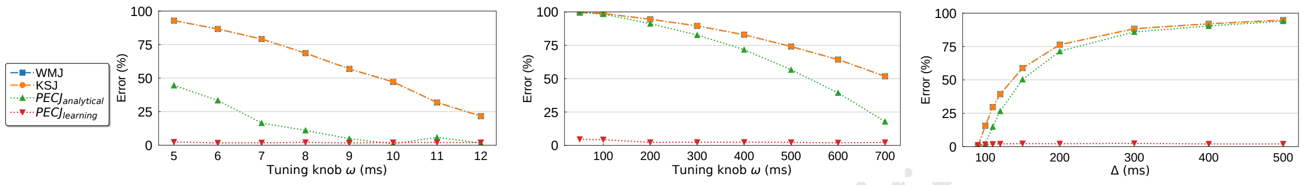
6.5 Algorithm Sensitivity Study

This section delves into a sensitivity analysis aimed at evaluating the accuracy of PECJ when implemented using varying strategies, specifically the *analytical* (referred to as $PECJ_{\text{analytical}}$ henceforth, which demonstrates $PECJ_{\text{analytical}}$ via the minimum error of SVI-based and AEMA-based methodologies) that leans on the



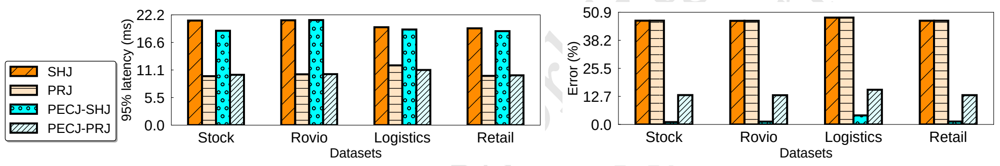
(a) Relative error under a varying number of keys (b) 95% latency under varying event rate (c) Relative error under varying event rate

Figure 8: Workload sensitivity study.



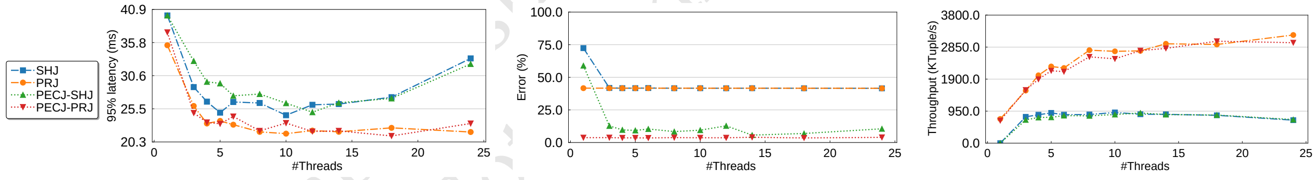
(a) Q1 (b) Q3 (c) Variant of Q1 under varying Δ, fix ω at $90ms$

Figure 9: Algorithm Sensitivity study.



(a) 95% latency (b) Relative error

Figure 10: Integrated implementation evaluation using Q1.



(a) 95% latency (b) Relative error (c) Throughput

Figure 11: Scaling-up integrated implementation using Stock dataset with an event rate of $1600Ktuples/s$.

central limit theorem as detailed in Section 5.1, and the *learning-based* (referred to as $PECJ_{\text{learning}}$ henceforth), which prioritizes generalization and the capture of unobserved data as elaborated in Section 5.2. Initially, we examine the **Q1** scenario, characterized by relatively straightforward stream dynamics and observation distortion. As illustrated in Figure 9(a), we perform a comparative analysis of the relative error (ϵ) between $PECJ_{\text{analytical}}$, $PECJ_{\text{learning}}$, and two baseline methods, *WMJ* and *KSJ*, while adjusting the ω within the range of $5ms$ to $12ms$.

Our analysis yields several key insights. First, as anticipated in Section 2, both *WMJ* and *KSJ* display similar error profiles across different ω values and consistently record higher errors compared to $PECJ_{\text{analytical}}$ or $PECJ_{\text{learning}}$. Second, while $PECJ_{\text{analytical}}$ adeptly corrects errors and mirrors the arrival pattern in **Q1**, its accuracy is enhanced with a larger ω, reflecting its reliance on the central limit theorem (refer to Section 5.1). In essence, a larger ω provides a more significant pool of observational data, hence boosting $PECJ_{\text{analytical}}$'s accuracy. Finally, $PECJ_{\text{learning}}$, engineered for broad applicability, extracts latent information from the data streams and rectifies errors more effectively than $PECJ_{\text{analytical}}$. Notably, this robustness persists even when the pool of observational data is curtailed by a smaller ω.

We then proceed to evaluate the **Q3** scenario, which introduces more complexity to the stream dynamics and observation distortion due to a larger Δ. The ω is tuned from $50ms$ to $700ms$, and the relative errors (ϵ) of all methods are illustrated in Figure 9(b). Generally, $PECJ_{\text{analytical}}$ struggles to accurately reflect **Q3**'s arrival pattern and provides sub-optimal error compensation. Each observation on join selectivity or event rate is heavily biased, violating the preconditions for applying the central limit theorem

(Section 5.1). While this bias can be reduced with a larger volume of observations, it necessitates a larger ω. Contrarily, $PECJ_{\text{learning}}$ is equipped to recognize these biases, overcoming the constraints of the central limit theorem, and thus delivers superior error compensations as a general instantiation method.

Lastly, we delve into the scenarios where $PECJ_{\text{analytical}}$ might fail. Specifically, we maintain the SUM aggregation function of **Q1**, fix the ω to $100ms$, and increment the Δ from $90ms$ to $500ms$. The resultant error is presented in Figure 9(c). It is observed that the error of $PECJ_{\text{analytical}}$ gradually escalates with Δ, exceeding 50% when Δ reaches 150ms or higher. It eventually matches the high error levels of *WMJ* or *KSJ* when Δ becomes sufficiently large. This confirms that the range of Δ is a key contributor to observation distortion, resulting in the unsuitability of the central limit theorem and, hence, the sub-optimal performance of $PECJ_{\text{analytical}}$.

6.6 Integrated Implementation Evaluation

In this evaluation, we contrast the original parallel SHJ and PRJ in AllianceDB with their corresponding modifications under PECJ, namely, PECJ-SHJ and PECJ-PRJ. It is important to note that the assumed time point of window completeness ω doesn't impact SHJ and PRJ as they do not handle disordered data streams. For both PECJ-SHJ and PECJ-PRJ, we set it to $10ms$.

In an assessment of four real-world datasets, we observe the 95% l and ϵ under **Q1**, as illustrated in Figure 10. Three key observations stand out. Firstly, both PRJ and SHJ produce high error rates, for instance, a substantial 47% on the Stock dataset when faced with disordered arrivals. Secondly, PECJ-PRJ and PECJ-SHJ notably decrease these errors while managing to maintain similar latency to their counterparts, PRJ and SHJ. This outcome attests to the robust efficiency in the optimization and implementation of PECJ. Lastly, PECJ-SHJ showcases a lower ϵ than PECJ-PRJ, specifically, 1% versus 13% in the Stock dataset. This improvement is a consequence of PECJ-SHJ's real-time data stream analysis approach. In contrast to PECJ-PRJ which waits for a window of tuples before starting the processing, PECJ-SHJ promptly processes each input tuple upon arrival. This strategy enables PECJ-SHJ to rapidly detect and adapt to immediate and ongoing changes in the data streams.

In the scalability evaluation, we gradually increase the number of **Stock** tuples in each window and ensure that the event rate of both R and S surpasses $1600KTuples/s$. By varying the number of threads from 1 to 24, we depict the 95% l, ϵ, and system throughput of each mechanism in Figure 11. It becomes clear that the lazy approaches, namely PRJ and PECJ-PRJ, consistently outshine their eager counterparts (SHJ and PECJ-SHJ), in terms of latency reduction and throughput improvement. This result aligns with previous studies [43] conducted under in-order arrival scenarios, reaffirming the enduring challenges faced by eager approaches such as cache thrashing, particularly when scaling up.

Moreover, PECJ-PRJ matches PRJ in terms of efficient scalability, largely thanks to its reduced overhead in managing disorder. This reaffirms the efficacy of our theoretical optimization for the PDA problem, using VI as outlined in Section 4. The integration of low-overhead AEMA VI instationation further contributes to an enhanced execution efficiency (Section 5.1). On the other hand, despite its earlier successes, PECJ-SHJ incurs higher errors than PECJ-PRJ under a heavy input workload, as illustrated in Figure 10(b). This can be attributed to distortions resulting from eager disorder handling, which can potentially mislead PECJ by providing inaccurate information for error compensation. Nonetheless, these findings collectively underscore PECJ's practicality in scaling up SWJ algorithms under challenging conditions of disordered data arrival.

7 RELATED WORK

This section discusses related research in *Stream Window Join*, *Buffer-based Disorder Handling*, and *Approximate Query Processing*.

Stream Window Join (SWJ). The predominant aim in optimizing stream window join operations has traditionally centred around enhancing efficiency and facilitating incremental processing. For example, both the Handshake Join [37] and the Split Join [32] use a dataflow model to achieve scalability on modern multicore architectures, whereas the *IBWJ* [34] utilizes a shared index structure to expedite tuple matching. An exhaustive experimental study conducted by Zhang et al. [43] contrasts these techniques across a wide spectrum of workload characteristics, application necessities, and hardware designs. This study also underscores the successful adaptation of relational join algorithms to hasten SWJ. Typically, these methodologies presume that data arrives in an ordered manner and is fully accessible. Our work, however, ventures into investigating ways to offset errors induced by incomplete data in the face of disorderly conditions.

Buffer-based Disorder Handling. A number of studies have delved into the accuracy-latency tradeoff utilizing buffers. To prevent potential infinite buffering, existing research employs different mechanisms for controlling buffer flushing and for making assumptions about the temporary completeness of incoming data. These mechanisms include k-slack [19, 25], watermarks [5, 8, 35], and punctuations [23]. For example, Ji et al. [18] introduced a k-slack-based disordered SWJ, which regards the tradeoff between accuracy and latency as a crucial factor. They highlight that joins inherently possess more complexity than single-stream linear operators, such as summation or average, when handling disordered data. This complexity stems from the mutual and non-linear relationships existing among multiple streams. Despite the variations in specific tradeoff rules and methodologies, these approaches rely on data that has already arrived to generate results, thus overlooking the contributions of future data. PECJ stands out by proactively compensating for this yet-to-be-received data.

Approximate Query Processing (AQP). The goal of AQP is to reduce computational overhead by selecting a data subset to approximate the result of the whole dataset [20, 24]. As data selection is system-controlled, error compensation can be predefined and is relatively stable in AQP. Compensation can use either linear [33] or non-linear formulas [4], depending on the algorithm's subset selection. More advanced AQP approaches employ machine learning [27] and bootstrap methods [40] to tackle ubiquitous queries under static data, albeit with higher computational costs. To address this issue, the *Wander Join* algorithm [22] applies stochastic and graph optimizations to reduce overhead and optimize online aggregation for joins. Our work addresses a different and more challenging problem—handling of disordered SWJ where observation distortion cannot be system-controlled. Therefore, we propose to solve a PDA problem by VI and discuss its implementations for disordered SWJ (Sections 4 and 5).

8 CONCLUSION

In this paper, we have introduced PECJ, a novel solution for executing SWJ, a critical operation in stream analytics, amidst the challenges posed by disordered data. What sets PECJ apart is its unique ability to proactively incorporate unobserved data, thereby enhancing the accuracy-latency tradeoff. This feat is achieved by leveraging a sophisticated approach to PDA using efficient VI instantiations. As evidenced by the successful implementation of PECJ in the multi-threaded SWJ benchmark testbed, this method presents a promising advancement for enhancing data stream processing capabilities under disordered data arrival conditions. Particularly, it has successfully reduced the relative error from 47% to a remarkable 1%, while maintaining constant latency. Looking ahead, an exciting prospect lies in expanding the applicability of PECJ and exploring how its principles can integrate with approximate computing methodologies. This includes techniques such as sampling and compression, which deliberately introduce data distortion to strike a balance between accuracy and latency. The integration of these approaches would certainly open up new avenues for future research.

REFERENCES

[1] [n.d.]. A Benchmark for Real-Time Relational Data Feature Extraction. https://github.com/decis-bench/febench. Last Accessed: 2023-01-03.
[2] [n.d.]. OpenMLDB Use Cases. https://openmldb.ai/docs/en/main/use_case/index.html. Last Accessed: 2022-09-23.
[3] 2023. *Pytorch homepage, https://pytorch.org/.*
[4] 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*. 29–42.
[5] Tyler Akidau, Edmon Begoli, Slava Chernyak, Fabian Hueske, Kathryn Knight, Kenneth Knowles, Daniel Mills, and Dan Sotolongo. 2021. *Watermarks in Stream Processing Systems: Semantics and Comparative Analysis ofApache Flink and Google Cloud Dataflow.* Technical Report. Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States).
[6] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, et al. 2015. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. (2015).
[7] Abdullah Alsaedi, Nasrin Sohrabi, Redowan Mahmud, and Zahir Tari. 2023. RADAR: Reactive Concept Drift Management with Robust Variational Inference for Evolving IoT Data Streams. In *Proceedings of the 39th IEEE International Conference on Data Engineering (ICDE2023)*. IEEE.
[8] Ahmed Awad, Jonas Traub, and Sherif Sakr. 2019. Adaptive Watermarks: A Concept Drift-based Approach for Predicting Event-Time Progress in Data Streams.. In *EDBT*. 622–625.
[9] Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern recognition and machine learning*. Vol. 4. Springer.
[10] Savong Bou, Hiroyuki Kitagawa, and Toshiyuki Amagasa. 2021. Cpix: real-time analytics over out-of-order data streams by incremental sliding-window aggregation. *IEEE Transactions on Knowledge and Data Engineering* 34, 11 (2021), 5239–5250.
[11] Tamara Broderick, Nicholas Boyd, Andre Wibisono, Ashia C Wilson, and Michael I Jordan. 2013. Streaming variational bayes. *Advances in neural information processing systems* 26 (2013).
[12] Badrish Chandramouli, Mohamed Ali, Jonathan Goldstein, Beysim Sezgin, and Balan Sethu Raman. 2010. Data stream management systems for computational finance. *Computer* 43, 12 (2010), 45–52.
[13] Roger Dingledine, Nick Mathewson, Paul F Syverson, et al. 2004. Tor: The second-generation onion router.. In *USENIX security symposium*, Vol. 4. 303–320.
[14] Behrouz A Forouzan. 2002. *TCP/IP protocol suite*. McGraw-Hill Higher Education.
[15] Tom Goldstein and Stanley Osher. 2009. The split Bregman method for L1-regularized problems. *SIAM journal on imaging sciences* 2, 2 (2009), 323–343.
[16] Matthew D. Hoffman, David M. Blei, and Francis Bach. 2010. Online Learning for Latent Dirichlet Allocation. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 1* (Vancouver, British Columbia, Canada) *(NIPS'10)*. Curran Associates Inc., Red Hook, NY, USA, 856–864.
[17] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. 2013. Stochastic variational inference. *Journal of Machine Learning Research* (2013).
[18] 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)*. IEEE, 493–504.
[19] Yuanzhen Ji, Hongjin Zhou, Zbigniew Jerzak, Anisoara Nica, Gregor Hackenbroich, and Christof Fetzer. 2015. Quality-driven continuous query execution over out-of-order data streams. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 889–894.
[20] 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*. 631–646.
[21] Nikos R Katsipoulakis, Alexandros Labrinidis, and Panos K Chrysanthis. 2020. Spear: Expediting stream processing with accuracy guarantees. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1105–1116.
[22] 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*. 615–629.
[23] Jin Li, Kristin Tufte, Vladislav Shkapenyuk, Vassilis Papadimos, Theodore Johnson, and David Maier. 2008. Out-of-Order Processing: A New Architecture for High-Performance Stream Systems. *Proc. VLDB Endow.* 1, 1 (aug 2008), 274–288. https://doi.org/10.14778/1453856.1453890
[24] Kaiyu Li, Yong Zhang, Guoliang Li, Wenbo Tao, and Ying Yan. 2018. Bounded approximate query processing. *IEEE Transactions on Knowledge and Data Engineering* 31, 12 (2018), 2262–2276.
[25] Ming Li, Mo Liu, Luping Ding, Elke A Rundensteiner, and Murali Mani. 2007. Event stream processing with out-of-order data arrival. In *27th International Conference on Distributed Computing Systems Workshops (ICDCSW'07)*. IEEE, 67–67.
[26] Yiming Li, Yanyan Shen, and Lei Chen. 2022. Camel: Managing Data for Efficient Stream Learning. In *Proceedings of the 2022 International Conference on Management of Data*. 1271–1285.
[27] Qingzhi Ma and Peter Triantafillou. 2019. Dbest: Revisiting approximate query processing engines with machine learning models. In *Proceedings of the 2019 International Conference on Management of Data*. 1553–1570.
[28] Hongyu Miao, Heejin Park, Myeongjae Jeon, Gennady Pekhimenko, Kathryn S McKinley, and Felix Xiaozhu Lin. 2017. Streambox: Modern stream processing on a multicore machine. In *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)* (Santa Clara, CA, USA) *(Usenix Atc '17)*. USENIX Association, Berkeley, CA, USA, 617–629. http://dl.acm.org/citation.cfm?id=3154690.3154749
[29] Adrian Michalke, Philipp M Grulich, Clemens Lutz, Steffen Zeuch, and Volker Markl. 2021. An energy-efficient stream join for the Internet of Things. In *Proceedings of the 17th International Workshop on Data Management on New Hardware (DaMoN 2021)*. 1–6.
[30] Douglas C Montgomery, Cheryl L Jennings, and Murat Kulahci. 2015. *Introduction to time series analysis and forecasting*. John Wiley & Sons.
[31] Mohammadreza Najafi, Mohammad Sadoghi, and Hans-Arno Jacobsen. 2016. SplitJoin: A Scalable, Low-latency Stream Join Architecture with Adjustable Ordering Precision. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. USENIX Association, Denver, CO, 493–505. https://www.usenix.org/conference/atc16/technical-sessions/presentation/najafi
[32] Mohammadreza Najafi, Mohammad Sadoghi, and Hans-Arno Jacobsen. 2016. SplitJoin: A Scalable, Low-latency Stream Join Architecture with Adjustable Ordering Precision. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. 493–505.
[33] 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*. 185–197.
[34] Amirhesam Shahvarani and Hans-Arno Jacobsen. 2020. Parallel Index-Based Stream Join on a Multicore CPU. 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, 2523–2537. https://doi.org/10.1145/3318464.3380576
[35] Yang Song, Yunchun Li, Hailong Yang, Jun Xu, Zerong Luan, and Wei Li. 2021. Adaptive watermark generation mechanism based on time series prediction for stream processing. *Frontiers of Computer Science* 15 (2021), 1–15.
[36] Binh Tang and David S Matteson. 2021. Probabilistic transformer for time series analysis. *Advances in Neural Information Processing Systems* 34 (2021),

23592–23608.

[37] 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)*. Acm, New York, NY, USA, 625–636. https://doi.org/10.1145/1989323.1989389

[38] Arash Vahdat and Jan Kautz. 2020. NVAE: A deep hierarchical variational autoencoder. *Advances in neural information processing systems* 33 (2020), 19667–19679.

[39] Sifan Wu, Xi Xiao, Qianggang Ding, Peilin Zhao, Ying Wei, and Junzhou Huang. 2020. Adversarial sparse transformer for time series forecasting. *Advances in neural information processing systems* 33 (2020), 17105–17115.

[40] 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*. 277–288.

[41] Steffen Zeuch, Ankit Chaudhary, Bonaventura Del Monte, Haralampos Gavriilidis, Dimitrios Giouroukis, Philipp M. Grulich, Sebastian Breß, Jonas Traub, and Volker Markl. 2020. The NebulaStream Platform for Data and Application Management in the Internet of Things. In *CIDR 2020, 10th Conference on Innovative Data Systems Research, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2020/papers/p7-zeuch-cidr20.pdf

[42] Hao Zhang, Xianzhi Zeng, Shuhao Zhang, Xinyi Liu, Mian Lu, Zhao Zheng, and Yuqiang Chen. 2023. Scalable Online Interval Join on Modern Multicore Processors in OpenMLDB. In *Proceedings of the 39th IEEE International Conference on Data Engineering (ICDE2023)*. IEEE.

[43] Shuhao Zhang, Yancan Mao, Jiong He, Philipp M Grulich, Steffen Zeuch, Bingsheng He, Richard TB Ma, and Volker Markl. 2021. Parallelizing intra-window join on multicores: An experimental study. In *Proceedings of the 2021 International Conference on Management of Data*. 2089–2101.

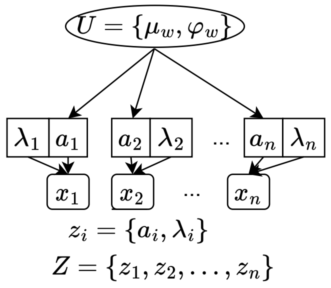

$U = \{\mu_w, \varphi_w\}$
λ_1 a_1
a_2 λ_2
a_n λ_n
x_1 x_2 x_n
$z_i = \{a_i, \lambda_i\}$
$Z = \{z_1, z_2, \ldots, z_n\}$

Figure 12: The variable dependency under an embarrassingly failed analytical instantiation example.

A AN IMPRACTICAL COMPLICATED ANALYTICAL INSTANTIATION

Figure 12 demonstrates our attempt at using a more complicated analytical instantiation. Specifically, we treat the long-tail effects of stream data [43] as the first-class citizen in describing the streaming dynamics. Despite its impractical, we present it here to provide a more comprehensive discussion, in order to show the limitations and difficulties of more sophisticated analytical instantiations.

To cover the long-tail effects, the dependency between the U and Z should be considered. Specifically, each local latent variable z_i has two components independent of each other, i.e., a_i and λ_i. The a_i controls observation x_i to concentrate on a certain value, which is further determined by global variable $U = \{\mu_w, \varphi_w\}$. We further assume that a_i is independent of each other when given U. Different from a_i, λ_i is independent of U, and it controls the long-tail skewed distribution of x_i. More clearly, Eqn 6 is changed into the following.

$$f(a_i|\mu_w, \varphi_w) = e^{-(a_i-\mu_w)^2 \times \varphi_w/2} \times \sqrt{\varphi_w} \times const \quad (16)$$

$$f(x_i|a_i, \lambda_i) = \lambda_i \times e^{-\lambda_i(x_i-a_i)} \times const \quad (17)$$

Eqn 16 enforces that $a_i \sim \mathcal{N}(\mu_w, 1/\varphi_w)$ given the global variable μ_w, φ_w, and a_i is i.i.d to each other, and Eqn 17 models x_i to follow an exponential distribution concentrated on a_i, and have a tail assigned by λ_i. As a result, we rewrite the joint distribution function from Eqn 7 into Eqn 18. Similar to Eqn 8, the $q(\mu_w)$ under this setting is given as Eqn 19. The posterior Gaussian distribution of μ_w is still Gaussian as $\mu_w \sim \mathcal{N}(\frac{\sum_{i=1}^{n}(\mathbb{E}(a_i))\mathbb{E}(\varphi_w)+\mu_0\tau_0}{\mathbb{E}(\varphi_w)n+\tau_0}, 1/(\mathbb{E}(\varphi_w)+\tau_0))$, and we can also acquire the estimated value and credible interval of μ_w as Section 5.1. However, the key difference is that $\mathbb{E}(\mu_w)$ is no longer linear to U (i.e., comparing Eqn 19 and Eqn 8), due to the involved $\mathbb{E}(\varphi_w)$ item, which can be further expanded as Eqn 20 by continuing [17] derivation. It is worth noting that $\mathbb{E}(\lambda_i)$ and x_i are further contained in $\mathbb{E}(a_i)$, and we omitted its details.

$$f(U,Z,X) = (\varphi_w)^{n/2} e^{-\varphi_w \times \sum_{i=1}^{n}((a_i-\mu_w)^2)} \times \prod_{i=1}^{n}(\lambda_i) e^{-\sum_{i=1}^{n}(\lambda_i(x_i-a_i))}$$

$$\times p(\mu_w) \times p(\varphi_w) \times p(z_1 \cap z_2 ... \cap z_n | U) \times const \quad (18)$$

$$q(\mu_w) = \mathbb{E}_{\varphi_w, Z}(f(U,Z,X))$$

$$= const \times e^{-\frac{\mathbb{E}(\varphi_w)+\tau_0}{2} \times (\mu_w - \frac{\sum_{i=1}^{n}(\mathbb{E}(a_i))\mathbb{E}(\varphi_w)+\mu_0\tau_0}{\mathbb{E}(\varphi_w)n+\tau_0})^2} \quad (19)$$

where the prior knowledge $p(\mu_w)$ is given as $\mu_w \sim \mathcal{N}(\mu_0, 1/\tau_0)$

$$\mathbb{E}(\varphi_w) = (n/2+\alpha_\tau)/(\frac{\sum_{i=1}^{n}(\mathbb{E}(a_i)-\mathbb{E}(\mu_w))^2}{2} + \beta_\tau) \quad (20)$$

where the prior knowledge $p(\varphi_w)$ is given as $\varphi_w \sim \Gamma(\alpha_\tau, \beta_\tau)$

It seems that the $\bar{\mu_w} = \mathbb{E}(\mu_w)$ estimation follows some analytical restrictions, and we can still theoretically let analytical VI converge into mean-field equilibria [17]. However, we found that the ELBO optimization (Eqn 4) under this instantiation can not be well supported by common optimizers, such as the ADAM and SGD in *Pytorch* [3]. **Specifically, a full unfold of putting Eqns 19 and 20 into Eqn 4 requires a catastrophically complicated tensor graph, which prevents the common optimizers from automatically computing the gradients**. Considering the extreme difficulty of implementing a custom optimizer from scratch or further reshaping the Eqn 4 under this instantiation, we give up the attempt in this case.