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

XIANZHI ZENG, Singapore University of Technology and Design, Singapore
SHUHAO ZHANG, Nanyang Technological University, Singapore
HONGBIN ZHONG, 4paradigm Inc., Beijing
HAO ZHANG, 4paradigm Inc., Singapore
MIAN LU, 4paradigm Inc., Singapore
ZHAO ZHENG, 4paradigm Inc., Beijing
YUQIANG CHEN, 4paradigm Inc., Beijing

Stream Window Join (SWJ), a vital operation in stream analytics, struggles with achieving a balance between accuracy and latency due to 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 stream oscillation. 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.

CCS Concepts: • **Information systems** → **Stream management**; • **Mathematics of computing** → **Variational methods**.

Additional Key Words and Phrases: data stream, variational methods, out-of-order arrival, error compensation

ACM Reference Format:
Xianzhi Zeng, Shuhao Zhang, Hongbin Zhong, Hao Zhang, Mian Lu, Zhao Zheng, and Yuqiang Chen. 2024. PECJ: Stream Window Join on Disorder Data Streams with Proactive Error Compensation. *Proc. ACM Manag. Data* 2, 1 (SIGMOD), Article 13 (February 2024), 24 pages. https://doi.org/10.1145/3639268

1 INTRODUCTION

Stream Window Join (SWJ) is an operation for joining two input streams within distinct, finite subsets, or 'windows', of infinite streams. SWJ, a crucial component of data stream analytics [49], 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

Authors' addresses: Xianzhi Zeng, Singapore University of Technology and Design, Singapore, xianzhi_xianzhi@mymail.sutd.edu.sg; Shuhao Zhang, Nanyang Technological University, Singapore, shuhao.zhang@ntu.edu.sg; Hongbin Zhong, 4paradigm Inc., Beijing, zhonghongbin@4paradigm.com; Hao Zhang, 4paradigm Inc., Singapore, zhanghao@4paradigm.com; Mian Lu, 4paradigm Inc., Singapore, lumian@4paradigm.com; Zhao Zheng, 4paradigm Inc., Beijing, zhengzhao@4paradigm.com; Yuqiang Chen, 4paradigm Inc., Beijing, chenyuqiang@4paradigm.com.

ACM 2836-6573/2024/2-ART13
https://doi.org/10.1145/3639268

arises from its essential role across various sectors, such as financial markets [13], fraud detection systems [2], and sensor networks [35].

One of the challenges complicating SWJ is the disorderly arrival of data, primarily due to factors like network delays, often termed as *stream oscillation* [6, 7, 9]. The management of these disordered data streams typically involves buffering input data [22, 23], 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 [22, 49].

To address these issues, we propose a novel solution: PECJ (Proactive Error Compensation-Join) 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 promising avenue for achieving significant accuracy enhancements without corresponding increases in latency. Notably, while subjects such as disorder handling parallelization [27, 29, 34] and efficient buffer structures [11] have been thoroughly explored in prior studies, these aspects are orthogonal to our work.

Application Example: Consider a sophisticated online anomaly detection system deployed in a stock exchange data center [3]. This system aims to identify irregular trading behaviors, such as "malicious short-selling" [15], through routine evaluations. It functions within designated time-based windows and employs *intra-window joins*[1] [49] to establish correlations between quotes and trades. Subsequent to this correlation, an aggregation function, commonly $COUNT()$, generates a scalar output that acts as the basis for issuing alerts. The complexity escalates when factoring in stream oscillations, which can be induced by network latencies, data source inconsistencies, and even geopolitical events affecting the timeliness of data streams. For example, consider an overseas transaction potentially aimed at malicious short-selling; it would ideally be processed within a latency as low as 200ms [14]. However, due to the unpredictable effects of stream oscillations, this transaction might experience significant delays, potentially as long as 800ms or more [17]. Traditional methods [9, 22, 23, 29] present two undesirable options: either wait for the delayed data, risking further latency, or proceed with incomplete data, which risks inaccuracy. Both options are problematic in a high-stakes financial environment. PECJ offers a proactive approach for identifying suspect trading activities by integrating predictive analytics for delayed data. By utilizing variational inference methods for estimating the posterior distribution of unobserved data, PECJ achieves a balance between computational efficiency and prediction accuracy unparalleled by existing methods. This enables the system to operate effectively even in latency-sensitive financial contexts.

Contributions and Outline: PECJ aims to augment the reliability of SWJ by proactively accounting for the yet-to-arrive disordered data, without incurring additional latency. The architecture of PECJ is founded on a three-stage approach. In the first stage, we redefine the problem of SWJ with disordered data streams as a *Posterior Distribution Approximation* (PDA) problem. This avoids the pitfalls of single data-point predictions and instead focuses on the collective impact of all unobserved data. This framework is compatible with any scalar-output aggregation functions, such as $SUM()$, and $COUNT()$, without requiring per-tuple decompositions. The second stage focuses on the optimization of our probabilistic model's parameters. Instead of utilizing the conventional but impractical brute-force parameterization, we propose to employ *Variational Inference* (VI)

[1]This is a specific type of SWJ; see Section 2.1 for more details.

techniques to enhance efficiency [21, 42]. The final stage translates these concepts into practice through two implementations: $PECJ_{\text{analytical}}$ for simpler cases and $PECJ_{\text{learning}}$ for more complex scenarios. $PECJ_{\text{analytical}}$ employs low-overhead linear modeling, while $PECJ_{\text{learning}}$ uses neural networks for improving posterior distribution accuracy.

The efficacy of PECJ is principally evaluated through a comprehensive algorithmic comparison, substantiating its advantages over existing methods [9, 22]. As a supplementary validation, PECJ is also integrated into AllianceDB, a multi-threaded SWJ benchmark testbed [49]. This additional evaluation demonstrates PECJ's robustness in mitigating out-of-order processing errors while upholding scalability. Although our primary experiments focus on intra-window joins with $SUM()$ and $COUNT()$ as example aggregations, PECJ's mathematical formulation accommodates a wide array of scalar-output aggregations. Its flexibility also allows for future adaptability to other SWJ variants. Issues concerning computational reuse in alternative types of SWJ are designated for future research [37, 39, 43].

- 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 effects of stream oscillation. 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 severe stream oscillation and gradually progress to a more generalized form (*learning-based*) that employs machine learning for handling complex oscillation cases.
- 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.

2 PRELIMINARY

This section provides a detailed introduction to Stream Window Join (SWJ), including the buffering mechanisms for handling disorder prevalent in existing research. Afterwards, we introduce a better strategy than state-of-art and discuss its technical challenges.

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 $y = \tau_{event}$,κ,v,$\tau_{arrival}$,τ_{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 Zhang et al. [49] to handle infinite stream joins over limited subsets of data. Here, a ***window*** is defined as an arbitrary time range $[t1, t2]$, denoted as $\mathbb{W} = [t1, t2]$. A tuple y is considered part of $\mathbb{W}$ if its timestamp t_e falls within this range. The length of the window is represented as $|\mathbb{W}|$. As discussed in the motivating example in Section 1, we use *intra-window joins* [49] as an example SWJ in this work. 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 result of $R \bowtie_{\mathbb{W}} S$ is subsequently condensed into a scalar output, O, via an *aggregation function*, which commonly either counts the joined tuples—i.e.,

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 [22]
	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 [21]
	$\mathbb{E}_j(k)$	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

$COUNT()$—or performs a sum operation on $R.v$ and $S.v$, denoted as $SUM()$. When O is dispatched to the user at the time point τ_{emit}, we consider the following two performance metrics:

- **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. A larger ϵ means that the O^{opr} is further from the expected outcome O^{exp}.
- **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. A larger l indicates more time to process SWJ and its follow-up aggregation function.

It should be noted that while this paper predominantly employs $COUNT()$ and $SUM()$ as example aggregation functions due to their widespread use, PECJ is versatile enough to support any aggregation function yielding a scalar result. Additionally, other variants of SWJ, such as sliding window joins [37, 39], are also worth mentioning. These alternate approaches often introduce additional computational challenges, particularly in the realm of computational reuse for overlapping windows. While important, these aspects are outside the purview of this paper and are designated as topics for future investigation [37, 39, 43].

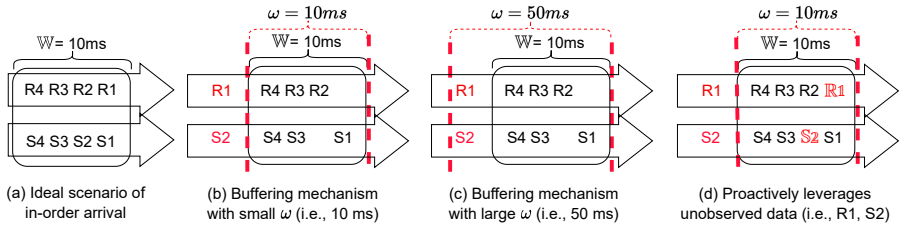


Fig. 1. Disorder handling of SWJ ($R \bowtie_{|\mathbb{W}|=10ms} S$)

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. However, this idealistic case is rare in the real world due to the *stream oscillation* [47, 48].

In contrast, a *disordered* arrival is more common and the $\tau_{arrival}$ sequence diverges from that defined by τ_{event}. Figures 1(b) and (c) illustrate the example scenarios of the disordered arrival. Under these circumstances, ensuring window completeness becomes challenging due to the late-arriving tuples, e.g., $R1$ 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. Join result 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 [9, 22, 23, 29] 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.

2.3 Proactive Incorporation: A Better Strategy?

To avoid the sub-optimal trade-off encountered in the state-of-art, a natural idea is to *proactively* incorporate the unobserved data into the processing workflow of SWJ ahead of its arrival, as showcased in Figure 1(d), rather than merely waiting. This strategy enables improved accuracy under the same ω compared to Figure 1(b), without needing to increase ω as in Figure 1(c). The most straightforward approach to realize such an idea might suggest leveraging time series prediction techniques to anticipate the contributions from each unseen tuple [45]. 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. Moreover, individual estimation of tuples' contribution itself enforces an assumption that the aggregation function is decomposable, which limits the applicability.

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 [45]. 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 commences with the preliminary theoretical foundations for PECJ. Subsequently, it offers an overview of PECJ's conceptual framework, accompanied by illustrative examples.

3.1 Theoretical Foundations for PECJ

To realize the proactive incorporation of unobserved data and address the difficulties discussed in Section 2.3, PECJ solves a *posterior distributions approximation* (PDA) problem, optimized via *variational inference* (VI).

Posterior Distribution Approximation (PDA) is a fundamental problem in Bayesian analysis [10], aiming to update beliefs about probability models' parameters (i.e., the so-called *model parameterization* process) in response to observations, thereby understanding and interpreting uncertainties. Despite its straightforward concept, deriving the exact posterior distribution analytically can be highly challenging or even impossible when dealing with high-dimensional or nonlinear relationships within probability models. This is because exponential growth complexity of summation and integration will be involved.

Variational Inference [8, 21, 42] (VI) is an optimization technique for simplifying model parameterization in PDA. Rather than brutal force computation, it approximates the true posterior distribution with a tractable distribution family of functions (each denoted as $q()$), known as the *variational family*, and a popular choice of variational family is the mean-field variational family. Specifically, VI brings the variational family close to the truth by maximizing the *evidence lower bound* ($ELBO_q$) with the gathered pieces of evidence, leading to significantly improved computation efficiency. VI is superior to traditional approaches in four major aspects. First, VI is less prone to overfitting compared to Maximum Likelihood Estimation (MLE) [10]. MLE often struggles to estimate latent variables in complex models accurately. Second, VI incurs less computational overhead than Markov Chain Monte Carlo (MCMC)) [10], making it more suitable for latency-sensitive applications. Third, unlike regularization methods like L1 and L2 [19], VI can robustly handle evolving observations without the need for additional hyperparameter tuning. Lastly, VI is capable of incrementally integrating new observations into the existing distributions [12] and enables continual learning. Therefore, VI is widely used in latent dirichlet allocation [20], autoencoder construction [44], and concept drift detection [8], etc.

3.2 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

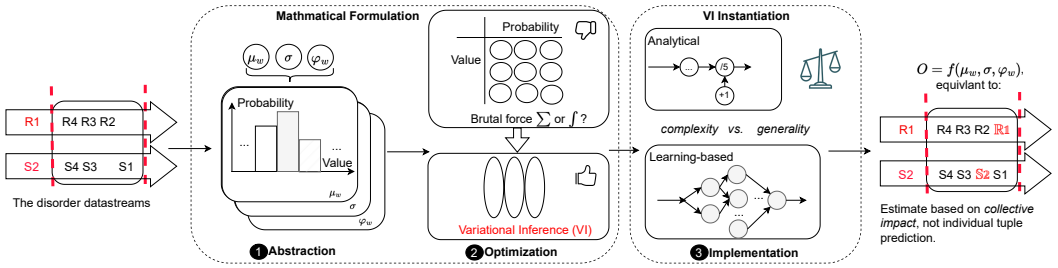


Fig. 2. Conceptual framework of PECJ.

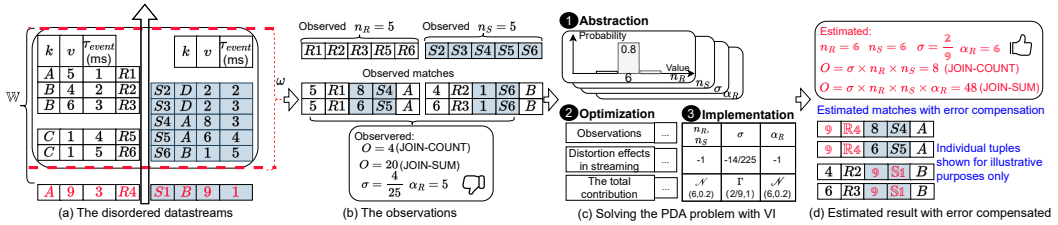


Fig. 3. Running Example of PECJ.

PDA problem, requiring the development of a probability model that is conscious of the stream oscillation.

Optimization: Given the inherent challenges of efficient PDA parameterization, we turn to the VI approach for theoretical optimization (explained in Section 4.2). As discussed in Section 3.1, VI drastically reduces the parameterization overhead of PDA, 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 oscillation patterns. We realize it using both Stochastic Variational Inference [21] (SVI) iterations and an Adaptive Exponential Moving Average Filter [18, 36] (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 implement the core ideas.

3.3 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 [22] 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 oscillation on the observed data, which often distorts the true picture.

Unfortunately, exhaustively computing every potential scenario of stream oscillation via brute-force methods is computationally infeasible. For this reason, PECJ adopts the VI approach and maximizes the evidence lower bound (Section 3.1) of describing stream oscillation. 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, 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).

3.4 Assumptions and Limitations

PECJ is built upon the theoretical framework of VI, and it shares the common assumption that *there exists a variational family of functions, typically the mean-field family, applicable to* PDA *[8, 12]*. In addition, the *analytical* approach makes the further assumption that the mean-field family should converge to a specific analytical form, which reflects the distortion effects of stream oscillation as a reverse linear effect on the central limit theorem. However, as demonstrated in Figure 12, this assumption of a reverse linear distortion to the central limit theorem proves to be quite restrictive and struggles to accommodate situations with severe stream oscillations. In contrast, *learning-based* approach relaxes this assumption and only requires that the mean-field family converges within the capacity defined by the *universal approximation theorem* [10], i.e., the foundation of artificial neural networks. To summarize, our solution is not intended to handle the cases where mean-field family approximation and the universal approximation theorem do not hold, which still remains an untapped territory in the literature.

4 MATHEMATICAL FORMULATION

This section illustrates the detailed mathematical formulation of PECJ. We begin by extracting critical information from the oscillating 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).

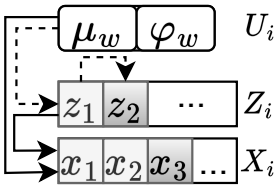


Fig. 4. Probability model.

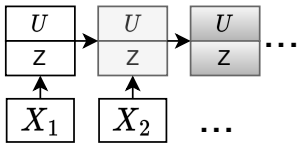


Fig. 5. Parameterization as continual learning.

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 [45]. 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 $\mu_w = \frac{1}{|\mathbb{W}|} f(\mathbb{W})$. Here, $f(\mathbb{W})$ is a scalar function to represent an arbitrary process of the whole $\mathbb{W}$, and it's then normalized by the windowlength $|\mathbb{W}|$ to define a μ_w. $f(\mathbb{W})$ does not have to be decomposable per tuple, in order to support an arbitrary window aggregation with scalar result. 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 3.3. 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 3.3), thereby facilitating proactive compensation for disorder handling errors. With the corresponding observations $X = \{x_1, x_2, ...\}$, we can estimate μ_w by approximate the the posterior distribution $p(\mu_w|X)$. 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 $p(\varphi_w|X)$. In summary, we are to achieve the following approximation objective:

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

The dynamics and randomness caused by stream oscillation can cause significant deviations in the observations X from the global U [8]. Unlike evaluating a static dataset [28, 46], a straightforward statistics approximation will be inaccurate due to the highly distorted observations under stream oscillation. To achieve a better reflection on the effects of stream oscillation, we employ *latent variables* $Z = \{z_1, z_2, ...\}$ in our model. We use directed arrows to denote probabilistic dependencies in our model, as shown in Figure 4. 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. Each variable z_i in Z directly influences specific observations in X, embodying temporal or local dynamics of the stream oscillation. 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 oscillation patterns, 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. By introducing Z, we can expose patterns and trends in the data streams that might not be immediately noticeable when examining X alone, providing a better reflection on the stream oscillation and therefore achieving more accurate U estimation.

4.2 Optimizing Model Parameterization with VI

Despite the better reflection of stream oscillation, latent variables entail undesirable exponential computational complexity for parameterizing the probability model, as discussed in Section 3.1. Moreover, we continuously need to update the model parameters to handle new incoming data and promptly make inferences. Therefore, we employ VI [8, 21, 42] for model parameterization. VI is advantageous for PECJ compared with traditional approaches in both accuracy and latency, and it inherently supports continual learning on stream oscillation (Section 3.1). Specifically, PECJ utilizes VI to approximate the true posterior $p(U|X)$ in Objective 1, without resorting to brute-force integration or summation on analyzing U and Z. Although the successful use of VI in other problems is acknowledged [8, 20, 44], 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 use variation family of $q()$ functions (Section 3.1) to approximate the $p()$ distributions, and use the $\approx$ symbol to indicate an approximation process. We illustrate the approximations to our target distribution $p(U|X)$ (Objective 1), conditional prior distribution of $Z|U$, and joint distribution of U, Z in Equations 1 to 3, respectively. By decomposing each variable into separate distributions during the approximation, we can apply divide and conquer to each variable and avoid brute force summation or integration.

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

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

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

VI solves an *optimization* problem of bringing $q()$ close to $p()$, by maximizing the *evidence lower bound* ($ELBO_q$), as defined in Equation 4. 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. The uncertainties and distortion effects brought by stream oscillation further require the capacity of continual learning, i.e., to assimilate new

information *progressively*, while retaining previously learned knowledge. 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 [12], 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 [32] and designate them as subjects for future research.

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

5 INSTANTIATION OF VI

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 [21], and subsequently examining the *learning-based* approach [8, 42]. For each approach, we present an overview at the beginning, then introduce its 1) key derivation steps, 2) conclusions, and 3) implementation usage.

5.1 Analytical Instantiation

The analytical instantiation is designed to provide a straightforward interpretation of stream oscillation, and it relies on several assumptions about the oscillation patterns to simplify the instantiation process. In particular, we enforce that the latent variable set Z matches the size of the observations X, and each z_i is treated as a scalar. Furthermore, z_i is typically correlated with a certain physical quantity that causes the stream oscillation.

Derivation Steps. We undertake a three-step process to extend the central limit theorem's (CLT) applicability to the context of handling stream oscillation with an intuitive example. **First**, if stream oscillation does not exist, our observations $X = \{x_1, x_2, ..., x_n\}$ should approximately match a Gaussian Distribution with mean μ_w and inverse variance φ_w. This can be expressed as $x_i \sim N(\mu_w, 1/\varphi_w)$ according to the CLT. Several factors as discussed in Section 4.1 support this approximation: 1) We have defined μ_w as one factor of the window-averaged total contribution, and 2) Each x_i observes the same entity, μ_w, and these observations are independently made. To illustrate, suppose $\mu_w = 1$, i.e., $1K$ transactions happening at the remote source per second in Section 1's example. If transaction reporting monopolizes the whole bandwidth, each x_i should be observed as 1 on average by the online anomaly detection system.

Second, we introduce three assumptions of reflecting the stream oscillation: 1) stream oscillation independently affects each observation x_i, 2) it is independent of the set of concerned global variables U, and 3) it can be reflected by single-dimension. For example, let's assume stream oscillation occurs as the number of stock services (using z_i to describe) varies over time, and the bandwidth is equally shared by all services. In this case, x_i will be observed to be $1/2$ on average if $z_i = 2$, as only half of the bandwidth is used for transaction reporting. In essence, we are characterizing stream oscillations by incorporating a reverse linear distortion of Z into the CLT-based approximation.

Specifically, we have $x_i \times z_i \sim N(\mu_w, 1/\varphi_w)$, or equivalently $x_i \sim \mathcal{N}(\mu_w/z_i, 1/(z_i^2\varphi_w))$. Therefore, the conditional probability function of x_i is formulated as Equation 6, and 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 stream oscillations (i.e., reflected under 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 $X = \{x_1, x_2, ..., x_n\}$. Noted that, the $U|Z$ notations in Equation 2 are simplified into disjoint parts, as we have independent U, z_i here.

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

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

$$\begin{aligned} q(\mu_w) &= \mathbb{E}_{\varphi_w, Z}(f(U, Z, X)) \\ &= const \times p(\mu_w) e^{-(\mu_w - g(X,Z))^2 \times (n\mathbb{E}(\varphi_w)/2)} \end{aligned} \tag{8}$$

$$\text{where } g(X, Z) = \sum_{i=1}^{n} \frac{\mathbb{E}(z_i) * x_i}{n}$$

$$\exists \text{vector } K \text{ and scalar } b, s.t., \bar{\mu_w} = \mathbb{E}(\mu_w|X) = KX + b$$

$$\text{where } KX = \frac{ng(X, Z)}{\tau_0 + n}, b = \frac{\tau_0\mu_0}{\tau_0 + n} \tag{9}$$

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

Third, when `VI` converges to a mean-field family of $q()$ and Equation 4 is achieved, an analytical solution [10, 21] 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 $\mu_w \sim \mathcal{N}(\frac{\tau_0\mu_0 + ng(X,Z)}{\tau_0 + n}, \frac{1}{(\tau_0+n)\mathbb{E}(\varphi_w)})$. Suppose we have observed $x_1 = 1/2$ and $x_2 = 1/3$, obtained that $\mathbb{E}(z_1) = 2, \mathbb{E}(z_2) = 3$ through the converging process of the `VI`, and initially held a prior knowledge that $\mu_0 = \tau_0 = 1$. In this context, we can construct the posterior distribution of μ_w and conclude it should be 1 on average, which aligns well with the truth.

Conclusions. We can deduce two crucial insights from the derivation above:

(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 $\bar{\mu_w} \pm 1.96\frac{1}{\sqrt{(\tau_0+n)\mathbb{E}(\varphi_w)}}$.

Implementation Usage. We can use Stochastic Variational Inference (SVI) [21] 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 [18, 36] 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 are possible to represent more complex patterns of stream oscillation, this approach demands significant manual effort and potentially leads to an impractical implementation (see Appendix for details). To overcome the challenges of capturing complex stream oscillation, we refer back to the abstract ELBO definition in Equation 4 for a more universal solution. This approach eliminates the need for prior knowledge or additional assumptions about Z and its interactions with U. Instead, we treat Z as a learnable black box, without knowing about its size, element dimensions, or dependency relationships.

Derivation Steps. We prove the learnable effects of stream oscillation by four major steps of parameter remapping and the divide-and-conquer policy as follows. **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 $q()$ functions, we can conveniently remove the $\mathbb{E}_q$ notations. **Third**, based on the mean-field property [10, 21], $\mathbb{E}_q(log(q(H)))$ 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, and each item in Equation 15 is a scalar value.

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

Conclusions. Similar to Section 5.1's case, the resulting $\mathbb{E}(\mu_w|X)$ and $\mathbb{E}(\varphi_w|X)$ can be directly utilized for the estimated value in PECJ's error compensation, as discussed in Section 4.1. Moreover, Equation 15 can further be leveraged to regulate the behavior of neural networks (NNs), enabling them to conform to the PDA process through an ELBO-driven learning process as follows:

(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, the whole Equation 15 can be employed to optimize NN loss. For example, if gradient descent is implemented via ADAM or SGD [4], 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.

Implementation Usage. In PECJ, we implemented a straightforward multilayer perceptron (MLP) to briefly illustrate this concept, leaving more powerful structures like LSTM [8] or transformer [42] 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 two popular state-of-art algorithms [9, 22], PECJ emerged as more effective, maintaining lower error rates even under intricate disorder arrival patterns and lenient real-time requirements (Section 6.3).
- The efficiency of PECJ was further validated under different workload conditions and algorithm configurations. In particular, PECJ can handle a severe stream oscillation (i.e., where the arrival delay of tuples oscillates from $0 \sim 1000ms$) with low error (i.e., 4.2%) by using learning-based instantiation, i.e., $PECJ_{\text{learning}}$ (Section 6.4).
- Lastly, the integration of PECJ into PRJ and SHJ demonstrated substantial error rate reductions without significantly impacting latency or scalability (Section 6.5).

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 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**. **Stock** is based on a real-world stock exchange dataset [3]. **Micro** and **Rovio** are from recent benchmark studies [26, 49]. **Logistics** and **Retail** datasets were obtained from a recent open source project [48]. **Stock** are the streams of financial quotes and trades, while **Rovio** continuously monitors user actions within a specific game. Additionally, **Logistics** and **Retail** involve streams of online decision augmentation labels and actions in logistics and retail applications, respectively. For more detailed characteristics, please consult Table 3 in [49] and Table 3 in [48]. To simulate a realistic scenario of stream oscillation, 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. By default, we employ the **Stock** datasets, which align with the motivating example presented in Section 1. To ensure a thorough evaluation, other datasets are utilized.

Queries: Three different queries were employed in our evaluation. **Q1**: This query entails a SWJ aggregated by COUNT (Section 3.3), 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 [47] . **Q2**: This query modifies **Q1** by changing the aggregation function to SUM (Section 3.3), 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 [16].

While **Q1** and **Q2** are tailored to require ultra-low latency processing, typically tens of milliseconds or less [1], **Q3** cannot expect such low latency due to the large arrival delay. Nonetheless, the goal is to achieve a latency below $200ms$ as discussed in our motivating example in Section 1.

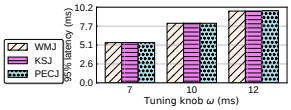
6.2 Implementation Details

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.

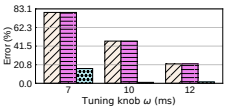
A) Standalone Implementation: In the standalone implementation setup, we're aiming for an algorithmic comparison between PECJ and two existing methodologies, namely **K-Slack-Join (*KSJ*)** [22] and **Watermark-Join (*WMJ*)** [9]. For these standalone implementations, we employed the same C++ codebase 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.

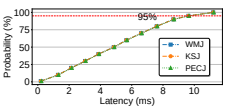
KSJ uses a k-slack buffer approach to manage the disorder in data streams. After data streams are preprocessed through the k-slack buffer, *KSJ* conducts a standard hash-join operation, treating the data as ordered. Importantly, our tuning parameter, ω, is tied to the k-slack buffer's control conditions, as discussed in Section 2. On the other hand, *WMJ* applies the watermark mechanism [9]



(a) 95% latency



(b) ϵ comparison



(c) Latency CDF under $\omega = 12ms$

Fig. 6. End-to-end comparison of Q1.

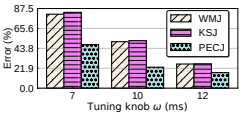
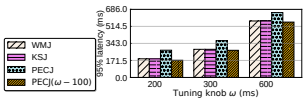
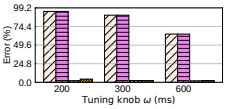


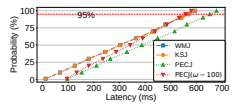
Fig. 7. ϵ comparison of Q2.



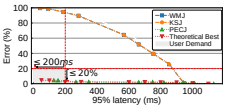
(a) 95% latency



(b) ϵ comparison

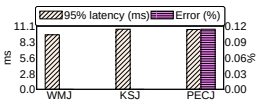


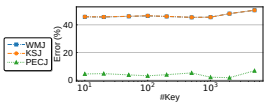
(c) Latency CDF under $\omega = 600ms$



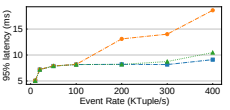
(d) Trade-off space.

Fig. 8. End-to-end comparison of Q3. PECJ (ω-100) refers to subtracting the ω of PECJ by $100ms$.

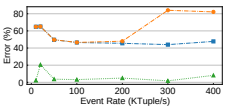




(a) ϵ by varying key number



(b) 95% latency of varying event rate



(c) ϵ by varying event rate

Fig. 9. Evaluation under in-order data.

Fig. 10. Impacts of key number and event rate.

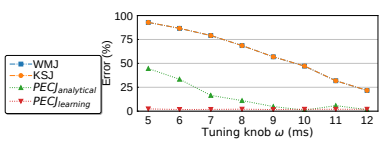
for data preprocessing, eliminating the need for a k-slack buffer. Each watermark indicates the arrival of tuples with $\tau_{event} < T$, enabling the computation to commence early upon watermarks' arrival. However, the emission of O waits until the ω is reached.

B) Integrated Implementations: This setup is designed to assess PECJ's performance when incorporated into an existing multi-threaded stream processing system AllianceDB [49], 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.

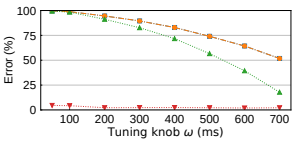
PRJ adopts a 'lazy' approach, delaying the join operation until all tuples have arrived. Conversely, SHJ pursues an 'eager' strategy, initiating the join process as soon as a portion of tuples arrives. Both PRJ and SHJ operate under the assumption of in-order arrival, and consider a window complete when 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**~**Q3** using the Stock dataset. The assumed time point of window completeness ω is fine-tuned to $7ms$, $10ms$, and $12ms$ for each methodology under **Q1** and **Q2**, and to $200ms$, $300ms$, and $600ms$ under **Q3**. We further include an in-order case for a more comprehensive comparison.



(a) Q1



(b) Q3

Fig. 11. Impacts of Algorithm Configuration.

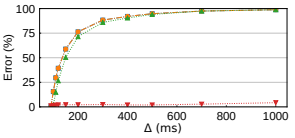
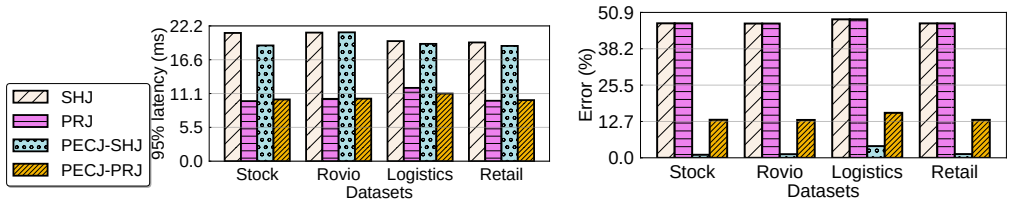


Fig. 12. Impacts of stream oscillation. The larger Δ, the severer oscillation.

Comparison under Q1. We apply the *analytical* instantiation in PECJ, i.e., $PECJ_{\text{analytical}}$ is deployed. We elucidate the ensuing 95% processing latency (95% l) and relative error (ϵ) in Figures 6(a) and 6(b). We also report the cumulative distribution function (CDF) of processing latency under $\omega = 12ms$ in Figure 6(c). 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 Figures 6(a) and 6(c). 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. The extra overhead of $PECJ_{\text{analytical}}$ to conduct error compensation and update its model is within $1ms$ in total, thanks to the proven straightforward linear form (Equations 8, 9, and 10). 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).

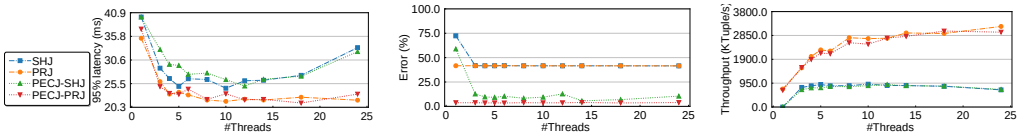
Comparison under Q2. Given the similar latency patterns across PECJ, *KSJ*, and *WMJ*, we primarily present the resulting relative error (ϵ) in Figure 7. Despite **Q2** demanding a more intricate syntax and involving additional parameters compared to **Q1** (Section 3.3), PECJ retains its superior performance, 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.

Comparison under Q3. **Q3** involves much severer stream oscillation than **Q1** and **Q2**, and we adjust PECJ from *analytical* to *learning-based*. The corresponding 95% l and ϵ are shown in Figures 8(a) and 8(b), and we report the latency CDF under $\omega = 600ms$ in Figure 8(c). Our findings show that *WMJ* and *KSJ* fall short in adapting to this scenario, where stream oscillation and its resulting 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 8(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%.



(a) 95% latency (b) ϵ comparison

Fig. 13. Single thread assessment of integrated implementation, using four real-world datasets.



(a) 95% latency (b) ϵ comparison (c) Throughput

Fig. 14. Scaling-up evaluation of integrated implementation, using Stock dataset.

In aligning with the motivation example in Section 1, we vary ω from 50 $\sim$ 1100ms in order to examine the trade-off space of latency and accuracy offered by PECJ, *KSJ*, and *WMJ*, as depicted in Figure 8(d). For a clearer reference, we also plot the user demand (within 200ms l [14] and 20% ϵ [24, 41]) and the theoretical best condition, i.e., when PECJ uses a perfect learning-based instantiation with zero overhead, in Figure 8(d). Note that, the maximum oscillation magnitude of tuples' arrival delay is 1000ms, and each mechanism can observe all data in a window when $\omega > 1010ms$. In the case where *KSJ* and *WMJ* can access all data of a window, their accuracy approaches 100% but latency is too high (i.e., exceeds 800ms) to meet the user's demands (i.e., within 200ms latency). In contrast, PECJ outputs earlier (e.g., about 160ms) with marginal errors (e.g., 4.2%). In other words, the tradeoff provided by PECJ is much more practically useful than alternative solutions.

Comparison under In-order data. We evaluate PECJ, *KSJ*, and *WMJ* under a query with in-order data. This query shares the same settings with **Q1**, except for $\Delta = 0ms$, i.e., the event time (τ_{event}) and arrival time ($\tau_{arrival}$) of each tuple remain consistently synchronized, eliminating any issues related to disordered arrivals or stream oscillations. We allow up to 1ms processing delay for the benchmark program to ingest tuples and set ω to 11ms for each mechanism, the resulting 95% l and ϵ are demonstrated in Figure 9. There are two key observations: 1) The latency of *KSJ*, *WMJ*, and PECJ remains comparable. 2) *KSJ* and *WMJ* lead to zero error while PECJ results in minor, about 0.1% overcompensation, i.e., it may mistakenly think there are still some tuples missing. This over-compensation phenomenon is caused by a lagged response to analyzing stream tendencies, and the impact of historical observation is incrementally fading in PECJ instead of immediately removed. We acknowledge that addressing the challenge of continually learning new tendencies while preventing the catastrophic forgetting of historical observations is a fundamental research challenge, as discussed in [32]. We envision future work aimed at resolving this issue.

6.4 Sensitivity Study

This subsection of the sensitivity study aims to contrast PECJ with the baseline models, *WMJ* and *KSJ*, under a range of characteristics, including 1) the number of join keys, 2) the event rate, and 3) the algorithm configurations, and 4) the magnitude of stream oscillations. By default, we fix ω to $10ms$ and operate under a SWJ with a window length of $10ms$, followed by $SUM()$ aggregation.

Impacts of Join Keys. We utilize the synthetic dataset **Micro** [49] and set the Δ as $5ms$. The number of 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 10(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.

Impacts of Event Rate. 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 10. 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.

Impacts of Algorithm Configurations. We delve 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 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 slight stream oscillation and observation distortion. As illustrated in Figure 11(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 severer stream oscillation 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 11(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.

Impacts of Stream Oscillation. We conducted an investigation into the varying magnitudes of stream oscillation based on the different outcomes observed with $PECJ_{\text{analytical}}$ and $PECJ_{\text{learning}}$ under mild (**Q1**) and severe (**Q3**) stream oscillation scenarios. Specifically, we set the ω to $100ms$, gradually increase the maximum magnitude of tuples' arrival delay (Δ) from $90ms$ to $1000ms$, and keep other settings the same as **Q1**. It's important to note that increasing Δ results in larger magnitudes of stream oscillation. The resulting error is depicted in Figure 12. It is evident that the error of $PECJ_{\text{analytical}}$ increases gradually with Δ, surpassing 50% when Δ reaches $150ms$ or higher. Eventually, it matches the high error levels of WMJ or KSJ when Δ becomes sufficiently large. This behavior occurs because a large magnitude of stream oscillation, such as $\Delta = 500ms$, renders the central limit theorem unsuitable, and the relatively simple analytic forms in Equations 8, 9, and 10 struggle to converge. In contrast, $PECJ_{\text{learning}}$ is capable of handling more severe stream oscillation scenarios (e.g., 4.2% error when $\Delta = 1000ms$). This capability is attributed to Equation 15, which offers a proven general approach to dealing with stream oscillation. Theoretically, it will only fail when mean-field family approximation and the universal approximation theorem do not hold, which is a research area yet to be thoroughly explored in the literature. In this evaluation, our neural network was pre-trained within a mere 2 minutes and required no further modification after deployment. However, we acknowledge the potential need for more complex scenarios in the future, where continuous retraining of the learning-based model may be necessary to further enhance our understanding of this study.

6.5 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 $10ms$. We first conduct a single-thread assessment on four real-world datasets, followed by a scaling-up evaluation by using **Stock** as an example.

Single-thread Assessment. In this assessment, we report the 95% l and ϵ of handling **Stock**, **Rovio**, **Logistics**, and **Retail** datasets under **Q1**, as illustrated in Figure 13. 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.

Scaling-up Evaluation. In the scaling-up evaluation, we keep the **Q1** query, 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 14. 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 [49] 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 instantiation 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 13(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 [43] and the Split Join [37] use a dataflow model to achieve scalability on modern multicore architectures, whereas the *IBWJ* [39] utilizes a shared index structure to expedite tuple matching. An exhaustive experimental study conducted by Zhang et al. [49] 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 [23, 31], watermarks [6, 9, 40], and punctuations [29]. For example, Ji et al. [22] 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 [25, 30]. As data selection is system-controlled, error compensation can be predefined and is relatively stable in AQP. Compensation can use either linear [38] or non-linear formulas [5], depending on the algorithm's subset selection. More advanced AQP approaches employ machine learning [33] and bootstrap methods [46] to tackle ubiquitous queries under static data, albeit with higher computational costs. To address this issue, the *Wander Join* algorithm [28] 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.

Appendix: The data, results, code, scripts and an appendix with more discussions of this work can be downloaded from https://anonymous.4open.science/r/PECJ.

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] 2018. *Shanghai Stock Exchange, http://english.sse.com.cn/*. Last Accessed: 2020-06-29.

[4] 2023. *Pytorch homepage, https://pytorch.org/*.

[5] 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.

[6] 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).

[7] 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).

[8] 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.

[9] 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.

[10] Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern recognition and machine learning*. Vol. 4. Springer.

[11] 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.

[12] 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).

[13] 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.

[14] Martijn de Heus, Kyriakos Psarakis, Marios Fragkoulis, and Asterios Katsifodimos. 2021. Distributed transactions on serverless stateful functions. In *Proceedings of the 15th ACM International Conference on Distributed and Event-based Systems*. 31–42.

[15] Kangqi Ding. 2022. Analysis of Short Selling. In *2022 7th International Conference on Financial Innovation and Economic Development (ICFIED 2022)*. Atlantis Press, 2030–2034.
[16] Roger Dingledine, Nick Mathewson, Paul F Syverson, et al. 2004. Tor: The second-generation onion router.. In *USENIX security symposium*, Vol. 4. 303–320.
[17] Hua Fan and Wojciech Golab. 2021. Gossip-based visibility control for high-performance geo-distributed transactions. *The VLDB Journal* 30, 1 (2021), 93–114.
[18] Behrouz A Forouzan. 2002. *TCP/IP protocol suite*. McGraw-Hill Higher Education.
[19] Tom Goldstein and Stanley Osher. 2009. The split Bregman method for L1-regularized problems. *SIAM journal on imaging sciences* 2, 2 (2009), 323–343.
[20] 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.
[21] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. 2013. Stochastic variational inference. *Journal of Machine Learning Research* (2013).
[22] 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.
[23] 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.
[24] Rasmus Kær Jørgensen and Christian Igel. 2021. Machine learning for financial transaction classification across companies using character-level word embeddings of text fields. *Intelligent Systems in Accounting, Finance and Management* 28, 3 (2021), 159–172.
[25] 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.
[26] 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)*. Ieee, 1507–1518.
[27] 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.
[28] 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.
[29] 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
[30] 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.
[31] 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.
[32] 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.
[33] 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.
[34] 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.
[35] 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.
[36] Douglas C Montgomery, Cheryl L Jennings, and Murat Kulahci. 2015. *Introduction to time series analysis and forecasting*. John Wiley & Sons.
[37] 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

[38] 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.

[39] 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

[40] 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.

[41] Salvatore Stolfo, David W Fan, Wenke Lee, Andreas Prodromidis, and Philip Chan. 1997. Credit card fraud detection using meta-learning: Issues and initial results. In *AAAI-97 Workshop on Fraud Detection and Risk Management*. 83–90.

[42] Binh Tang and David S Matteson. 2021. Probabilistic transformer for time series analysis. *Advances in Neural Information Processing Systems* 34 (2021), 23592–23608.

[43] 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

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

[45] 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.

[46] 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.

[47] 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

[48] 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.

[49] 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.

ACKNOWLEDGMENTS

This work is partially supported by a MoE AcRF Tier 2 grant (MOE-T2EP20122-0010), and a startup grant of NTU (023452-00001). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore. Corresponding author is Shuhao Zhang.

Received July 2023; revised October 2023; accepted November 2023