Computing and Informatics, Vol. 45, 2026, 668–697, doi: 10.31577/cai_2026_3_668

A MODEL FOR GENERATING REAL STREAM DATA IN SIMULATED EDGE COMPUTING ENVIRONMENTS

Weirong Xiu

School of Information Technology and Engineering
Guangzhou College of Commerce, 511363 Guangzhou, China
&
School of Graduate Studies, Management and Science University
40100 Shah Alam, Selangor, Malaysia
e-mail: 20190185@gcc.edu.cn

Md Gapar Md Johar

Software Engineering and Digital Innovation Center
Management and Science University
40100 Shah Alam, Selangor, Malaysia
e-mail: mdgapar@msu.edu.my

Mohammed Hazim Alkawaz

Department of Computer Science, College of Education for Pure Science
University of Mosul, Mosul, Nineveh, Iraq
e-mail: mohammed.ameen@uomosul.edu.iq

Chen Bian*

School of Computer Science, Guangdong University of Finance
510521 Guangzhou, China
e-mail: bianchen@gduf.edu.cn

* Corresponding author

Abstract. Due to the limitations of device resources and the dynamic nature of data in edge computing environments, developers face significant challenges in acquiring, processing, and analyzing high-quality streaming data during development and testing. This paper examines the characteristics of streaming data in edge computing environments, summarizes its volatility and changing trends in the real world, and deduces potential periodic patterns inherent in such data. A model for generating realistic streaming data in simulated edge computing environments is proposed to address these challenges. The model first preprocesses the original streaming data to extract time-related information (timestamps or precise time). Then, the NSA algorithm is applied to reduce the data volume, followed by the use of the PSD parallel algorithm to transmit the sampled data. Experimental comparisons demonstrate that the model effectively preserves the volatility and trend characteristics of the original streaming data, providing a viable solution for the difficulties developers face in obtaining and processing high-quality streaming data in edge computing environments during development and testing.

Keywords: Simulation model, stream data, edge computing, PSD parallel algorithm, stream processing system

Mathematics Subject Classification 2010: 68M14, 68W10, 68M20

1 INTRODUCTION

Edge computing has emerged as a transformative paradigm that bridges the gap between cloud infrastructure and end devices, enabling real-time processing and analysis of massive data streams generated by the Internet of Things (IoT), industrial automation systems, and ubiquitous smart devices. Unlike traditional cloud computing, which relies heavily on centralized data centers, edge computing decentralizes computation and brings it closer to data sources. This proximity reduces network latency, optimizes bandwidth utilization, and enhances responsiveness for time-sensitive applications such as autonomous vehicles, industrial robotics, and telemedicine [1, 2, 3]. Furthermore, recent advances highlight the role of intelligent cloud-edge orchestration and distributed optimization in enabling large-scale real-time systems [4, 5, 6].

Streaming data, a fundamental element in edge environments, is characterized by high velocity, volume, and variability. These data streams are continuously generated from diverse sensors, social platforms, and connected devices, requiring constant ingestion and near-instantaneous analysis [7, 8, 9]. The effectiveness of real-time decision systems, predictive maintenance, and dynamic control mechanisms depends heavily on the quality and reliability of the generated or simulated stream data. However, obtaining high-quality real-world streaming data remains challenging due to privacy restrictions, network heterogeneity, and high acquisition costs [10, 11].

Existing stream data generation methods can be broadly divided into synthetic simulation approaches and real-world data replication techniques. Synthetic generators, such as StreamBench and DataGen, are scalable and efficient but typically lack realistic temporal volatility, noise, and burstiness observed in natural data streams [7, 9, 12]. Real-world extraction methods, on the other hand, achieve higher fidelity but are limited by data availability and preprocessing demands. This leads to a trade-off between realism and efficiency that hinders accurate benchmarking of stream processing frameworks. As a result, hybrid models that integrate partial real data with controlled synthetic generation have gained increasing research attention [12, 13].

Recent developments in AI-driven data generation have also inspired new approaches to simulate edge data more intelligently. Studies integrating reinforcement learning and GAN-based synthetic stream modeling have demonstrated improvements in adaptive pattern recognition and contextual data synthesis [14, 15, 16, 17, 18]. Lightweight GAN implementations and energy-aware model-swapping approaches further demonstrate potential for generating continuous time-series data at the edge while minimizing system overhead [19, 20]. Yet, these approaches often require heavy computational resources, making them unsuitable for constrained devices. Meanwhile, large-scale distributed frameworks such as MapReduce and cyber-attack-aware offloading strategies reveal further challenges in ensuring consistency, security, and scalability of real-time stream pipelines [21, 22].

Another important yet underexplored aspect is the volatility preservation of simulated stream data. Real-world streams typically exhibit temporal irregularities – such as periodic peaks, bursts, and long-tail behaviors – that play a vital role in evaluating system performance, particularly for load balancing, anomaly detection, fault-tolerant scheduling, and dynamic resource allocation [23, 24, 25]. Synthetic models without these fluctuations tend to oversimplify real scenarios, leading to misleading conclusions in system evaluation and algorithmic tuning. Notably, the study of interval time series, fuzzy volatility modeling, and quantum-inspired continuous learning further indicate the complexity of accurately capturing dynamic fluctuations [26, 27, 28]. Therefore, an effective simulation model should not only reproduce statistical properties but also mimic the intrinsic volatility dynamics of real-world data streams.

To address these issues, this study proposes a realistic streaming data generation model tailored for simulated edge computing environments. The model focuses on reproducing the dynamic fluctuations and evolving trends of real-world streams while optimizing for computational efficiency and scalability. Specifically, it introduces a Normalizing and Sampling Algorithm (NSA) to reduce redundant data while retaining key temporal features and a Parallel Stream Data (PSD) production algorithm to simulate concurrent data emission. Together, these methods achieve an optimal balance between realism and performance [15, 16, 20].

The key contributions of this paper can be summarized as follows:

- A volatility-preserving simulation model that captures the time-dependent fluc-

tuation and trend characteristics of real-world stream data, drawing from recent advancements in volatility prediction, fuzzy modeling, and multi-objective scheduling.

- A lightweight, scalable algorithmic framework based on NSA and PSD that supports high-throughput data generation for various edge environments, complementing ongoing research in energy-aware inference, constrained distributed computing, and hybrid optimization [29].
- An empirical evaluation using four large-scale real datasets – MicrosoftNews, ShareBike, Traffic, and UserBehavior – to validate the model's ability to replicate realistic streaming behaviors across domains [30, 31].
- A comprehensive performance analysis, including throughput, latency, and scalability assessments, to demonstrate the model's practical applicability to edge computing systems, extending existing findings in big data processing, intelligent resource allocation, and distributed computation.

By establishing a reproducible and volatility-aware simulation pipeline, this study contributes a foundational tool for stream processing research, offering a valuable platform for benchmarking and performance optimization in modern edge computing ecosystems.

2 MATERIALS AND METHODS

2.1 Real-World Stream Data

Stream data refer to continuously generated and transmitted information flows that evolve in real time. In edge computing environments, such data are characterized by real-time responsiveness, continuity, high frequency, and time-series dependence, which together make them a fundamental data form for distributed and low-latency computation. These streams typically originate from heterogeneous sources such as IoT sensors [3, 11, 15], financial transaction systems [17], social media platforms [9], online networks [1, 13], and user-behavior logs [7, 22]. The constant influx of events from multiple endpoints introduces fluctuations, bursts, and temporal dependencies that must be carefully considered when simulating or processing stream workloads.

From the perspective of temporal constraints, stream data can be broadly divided into two conceptual categories: *bounded streams* and *unbounded streams*.

Definition 1 (Bounded Stream)**.** A bounded stream is a finite set that contains all data elements generated within a specific time range. Let B be the bounded stream, then B is a finite set: $B = \{s_1, s_2, s_3, \ldots, s_n\}$, where s_i represents the i^{th} data element in the stream, and n is a finite natural number. Bounded streams are commonly used to represent fixed experimental windows such as a day's worth of traffic flow [32] or transaction logs collected during a test cycle [7, 8].

Definition 2 (Unbounded Stream)**.** An unbounded stream is an infinite set where data elements are continuously generated and added without a predefined endpoint. Let U be the unbounded stream, then U is an infinite set: $U = \{s_1, s_2, s_3, \dots\}$, where s_i represents the i^{th} data element in the stream, and the set U has no upper limit. Unbounded streams are typical of real-world environments where new events arrive indefinitely – for example, continuous telemetry data from vehicles [15] or user activity captured by mobile applications across multiple time zones [7, 15].

Although unbounded streams have no theoretical limit, in practice they can be decomposed into an infinite number of bounded substreams, each representing a fixed observation period. For instance, traffic information generated by an in-vehicle GPS module within a single day can be treated as a bounded subset of an unbounded data flow [32]. This decomposition provides a practical way to analyze, model, and simulate otherwise infinite data streams in controlled experimental settings.

Therefore, the study of real open-stream datasets serves as an essential basis for understanding the temporal, structural, and statistical properties of real-world stream data in edge computing environments [33, 34]. Through analyzing their volatility, periodicity, and dependency structures, a simulation model can be constructed that preserves these key features while remaining computationally efficient for deployment at the edge.

2.2 Dataset Selection and Analysis

To validate the effectiveness and generality of the proposed simulation pipeline, this study employs four representative real-world streaming datasets, each reflecting different data generation patterns and fluctuation intensities. These datasets provide a comprehensive benchmark for evaluating how well the simulation framework captures dynamic changes and preserves statistical realism.

MicrosoftNews [12]: A large-scale news recommendation dataset containing approximately 160 000 news articles and more than 15 million impression logs from over one million users. Each record includes metadata such as article identifiers, timestamps, and user interaction types (click, skip, or dwell). This dataset reflects short-term interest shifts and behavioral volatility typical in real-time content delivery systems, making it suitable for modeling high-frequency user interaction streams.

Traffic [32]: Consists of real-time traffic data collected from Baidu Maps in Beijing between April and May 2017, including approximately 114 million records. Each entry includes location information, route identifiers, and travel durations. The dataset exhibits strong diurnal periodicity and high burst levels during rush hours, making it an effective source for testing the model's scalability and throughput simulation under dynamic workloads.

ShareBike [34]: Contains 3.2 million anonymized travel records generated by public bicycle-sharing systems. Each trip record includes start and end times, GPS coordinates, and travel distance. Due to its clear morning and evening peak patterns, this dataset is ideal for analyzing urban mobility trends and periodic fluctuation in continuous time-series environments.

UserBehavior [8]: A large-scale dataset from Alibaba's Taobao platform containing over one billion user interactions (clicks, purchases, additions to cart, and favorites). Each interaction is time-stamped at the second level. The dataset demonstrates irregular burstiness and heterogeneous temporal distribution, reflecting real e-commerce scenarios with event-driven workload spikes such as sales promotions or algorithmic recommendations.

In addition to their diversity, these datasets collectively represent different types of stream temporal dynamics – from cyclic and predictable flows (ShareBike [34], Traffic [32]) to stochastic and user-driven fluctuations (MicrosoftNews [12], UserBehavior [8]).

Before simulation, an exploratory data analysis (EDA) was conducted to assess the mean, variance, and coefficient of variation across multiple time windows (10 min, 30 min, and 60 min). This analysis quantified the magnitude of volatility and guided the parameter tuning for normalization and sampling in the subsequent stages.

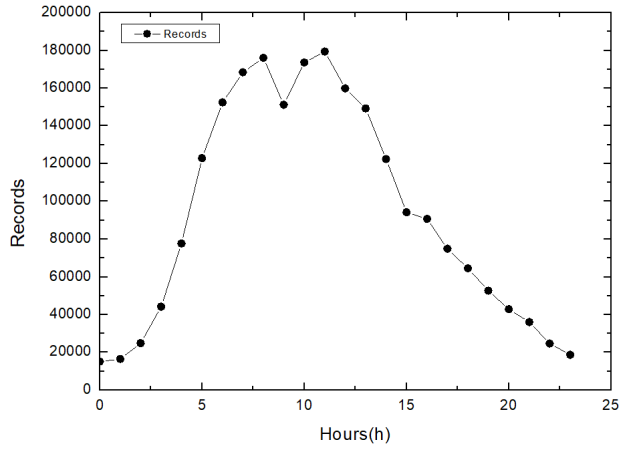
Figures 1 analyze and visualize the selected datasets within the context of edge computing environments, illustrating both their overall trends and local fluctuations. Across all four datasets, the data volume exhibits significant variation throughout the day, primarily driven by temporal user-behavior characteristics. Therefore, real-world stream data can be characterized as sequences with substantial fluctuation tendencies, where each domain-specific dataset demonstrates distinct volatility patterns and periodic structures.

These observations provide the empirical foundation for designing a simulation model capable of replicating both macroscopic (long-term) and microscopic (short-term) dynamics of stream data. By ensuring that the synthetic data maintain comparable volatility and temporal correlation, the proposed framework achieves high fidelity when generating realistic stream datasets for performance evaluation and edge-system testing.

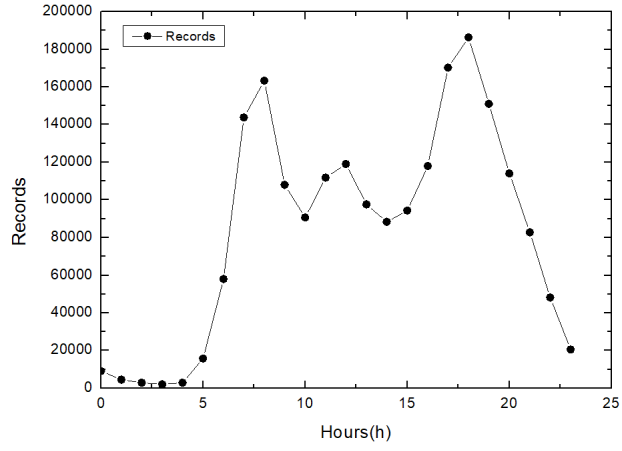
2.3 Simulation of Stream Data Production Line

The simulation of stream data production is a critical stage that bridges real-world data characteristics with the requirements of an experimental edge environment. The proposed simulation line, illustrated in Figure 2, emulates the process of acquiring, transforming, and emitting realistic streaming data in a distributed setting. It is composed of three essential components:

- Original Stream Data Preprocessing (OSDP),



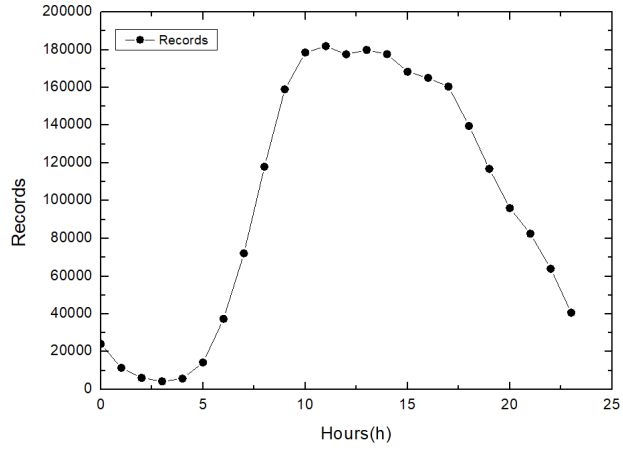
a) The volatility and trend of MicrosoftNews



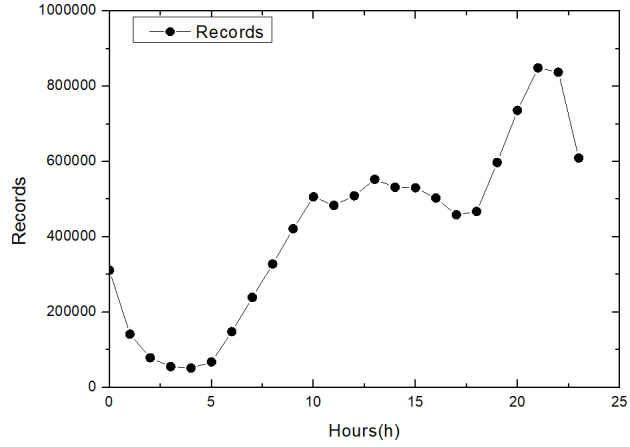
b) The volatility and trend of ShareBike

- Stream Data Normalization and Sampling (SDNS),
- Production of Simulated Stream Data (PSSD).

Each stage contributes to ensuring that the final synthetic stream accurately reproduces the temporal dynamics, statistical properties, and fluctuation intensity of real datasets, while maintaining computational efficiency and scalability in edge computing environments.



c) The volatility and trend of Traffic



d) The volatility and trend of UserBehavior

Figure 1. The volatility and trend of four real-world datasets

2.3.1 Preprocess

To simulate the original stream data within a smaller yet realistic time range, each record must include precise temporal information, typically represented as a timestamp in the format `YYYY-MM-DD HH:MM:SS`. The objective of preprocessing is to standardize all records into a consistent temporal framework and eliminate inconsistencies that may distort subsequent normalization operations.

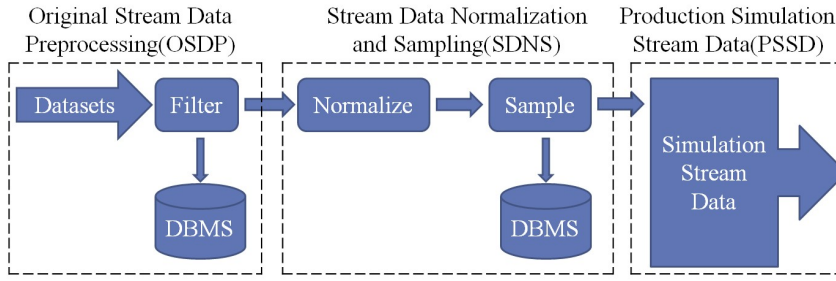


Figure 2. Simulation of stream data model

The preprocessing workflow consists of the following steps:

Timestamp Identification: Extract timestamps from heterogeneous formats (string, integer, UNIX epoch) using automated parsing rules.

Timezone Standardization: Convert all timestamps to a unified standard (UTC) in order to avoid temporal offsets caused by device locality.

Timestamp Conversion: Translate all timestamps into numeric forms to enable arithmetic operations and scalable sorting.

Data Cleaning: Detect missing or corrupted timestamps through window-based validation; reconstruct minor gaps via linear interpolation while filtering temporal anomalies.

Data Storage: Store preprocessed data in a persistent database management system (DBMS) to support repeated experiments without re-executing the preprocessing step.

This preprocessing sequence ensures that the input data are properly aligned for subsequent normalization and sampling. The time complexity of this stage is $\mathcal{O}(n)$, exhibiting linear scalability with respect to the number of records, while the space complexity remains $\mathcal{O}(1)$ per tuple due to in-place operations.

2.3.2 Data Normalization and Sampling (NSA Algorithm)

Data normalization is a widely used preprocessing technique in machine learning and statistical analysis, aiming to adjust raw data into a standard range or distribution. Common normalization methods include Min-Max normalization, Z-score normalization, and decimal scaling. In the proposed simulation model, Z-score standardization is first applied to normalize the raw data to a distribution with mean 0 and variance 1. Subsequently, Min-Max normalization is employed to map all timestamps into a user-defined target range while preserving temporal dependencies among records.

The design of the NSA algorithm is motivated by the need to preserve the intrinsic temporal structure of real-world stream data while compressing the overall time window for accelerated simulation. Traditional normalization techniques often disregard sequential dependencies or distort the original sampling density, leading to unrealistic burst behavior in the simulated data. Furthermore, many general-purpose sampling methods, such as reservoir sampling or stratified sampling, assume independent and identically distributed data, which does not hold for time-series streams where temporal order and fluctuation amplitude are essential characteristics. The NSA algorithm overcomes these limitations by combining normalization with systematic sampling, enabling proportional time-axis compression while maintaining the natural distribution and temporal ordering of the original stream records. This ensures that the resulting dataset preserves macro-level patterns (e.g., periodicity, diurnal cycles) and micro-level fluctuation behaviors (e.g., second-level peaks and short-term bursts).

The Min-Max normalization process is expressed as:

$$t_{\text{nom}} = \frac{t - t_{\min}}{t_{\max} - t_{\min}}(\max - \min) + \min, \tag{1}$$

where $t_{\min}$ and $t_{\max}$ denote the minimum and maximum timestamps in the original stream data, t represents the timestamp of the current tuple, $T_{\min}$ and $T_{\max}$ are the normalized minimum and maximum timestamps defined by the user, and scale_stamp is the normalized timestamp of the record.

It is important to note that the normalization function used in NSA behaves as a linear time-axis mapping whose monotonicity guarantees that the temporal order of all records is strictly preserved. Let t_i and t_j denote two timestamps in the original stream such that $t_i < t_j$. From the normalization function in (1), it follows that $\text{scale\_stamp}(t_i) < \text{scale\_stamp}(t_j)$, ensuring that no temporal inversion can occur during the scaling process. Moreover, the mapping compresses or expands the time axis proportionally based on the ratio $\frac{t_{\max} - t_{\min}}{\text{max\_range}}$, allowing the algorithm to flexibly adapt to different simulation durations. Unlike non-linear normalizers (e.g., logarithmic or sigmoid-based), this linear transformation preserves relative distances between adjacent events, which is essential for maintaining the localized fluctuation characteristics of high-frequency streams.

Compressing the original stream data into a smaller simulation time range (e.g., 10 or 20 minutes) introduces another challenge: the data volume per time unit becomes disproportionately large. This inflation occurs because the temporal compression increases the density of records relative to the accelerated timeline, resulting in unrealistic data rates that do not reflect real IoT or user-generated streams. Therefore, a sampling method is required to reduce the volume of normalized data while maintaining temporal fidelity.

Several sampling techniques were evaluated, including random sampling, stratified sampling, cluster sampling, and systematic sampling. Random sampling provides strong randomness but fails to preserve temporal regularity. Stratified sam-

pling improves representativeness but overlooks sequential dependencies essential for time-series continuity. Cluster sampling selects grouped data blocks but may introduce significant errors due to heterogeneity within clusters.

In contrast, systematic sampling offers a balance between efficiency and order preservation. By selecting records at fixed temporal intervals (e.g., every second), the method maintains the chronological sequence of the original stream while effectively controlling data volume. Accordingly, the Normalizing and Sampling Algorithm (NSA) adopts systematic sampling to remove redundant records from the normalized dataset, ensuring that the resulting synthetic stream preserves the original per-second emission rate.

The choice of systematic sampling in NSA is grounded in its superior ability to maintain sequential integrity while providing deterministic control over sampling density. Unlike random sampling, which introduces stochastic variance and may omit high-impact spike regions, systematic sampling guarantees that each time interval contributes proportionally to the final simulated stream. This is especially beneficial for datasets exhibiting periodic peaks or nonlinear volatility patterns, such as traffic congestion waves or e-commerce promotion bursts. The deterministic nature of systematic sampling also enables efficient indexing and facilitates parallel implementation, making it highly suitable for edge devices with limited computational resources. Furthermore, by aligning sampling intervals with normalized temporal units, NSA prevents over-amplification of local bursts that often occur during time-axis compression.

After normalization and sampling, the processed stream data are stored in a database for subsequent experiments. Because these operations are performed only once, the stored intermediate dataset can be reused across multiple simulation runs, thereby improving efficiency and reproducibility. The complete procedure is summarized in Algorithm 1 (NSA).

The Normalizing and Sampling Algorithm (NSA) mainly consists of two core processes: stream standardization and systematic sampling.

In the first stage, the algorithm computes the user-defined normalized timestamp, denoted as *scale_stamp*, for each tuple S_i using the normalization formula. This operation aligns all data records within a defined temporal range, ensuring consistency across heterogeneous sources.

In the second stage, NSA performs systematic sampling to eliminate redundant records and maintain a stable data density across the simulated timeline. Unlike random or stratified sampling, which may disrupt temporal order, systematic sampling iteratively removes surplus elements at fixed intervals while preserving the original sequence structure. This approach reduces the overall data volume in a controlled manner, ensuring that the generated stream reflects the same per-second emission rate as the original dataset.

By combining normalization with sequential sampling, NSA effectively balances simulation fidelity and computational efficiency. The algorithm's average-case time complexity is $O(n)$, and due to its linear scalability, it can be executed efficiently on edge devices with constrained computational and memory resources, supporting

Algorithm 1 NSA: Normalizing and Sampling Stream Data

Require: B: Original stream data
Require: max_range: User-defined time range
Ensure: D_s: Simulated stream data

 # Normalize the original stream data
min_value $\leftarrow \min(B)$
max_value $\leftarrow \max(B)$
for each record s in B **do**
  $s.\text{scale\_stamp} \leftarrow \dfrac{s.\text{value} - \text{min\_value}}{\text{max\_value} - \text{min\_value}} \times \text{max\_range}$
end for
 # Systematic sampling
multiple $\leftarrow \dfrac{|B|}{\text{max\_range}}$
for $i = 0$ to max_range **do**
  block $\leftarrow \{s \in B \mid s.\text{scale\_stamp} = i\}$
  $rs \leftarrow \dfrac{|block|}{\text{multiple}}$
  for each s in block **do**
    if |block| $> rs$ **then**
      remove s from block
    end if
  end for
end for
return B

real-time preprocessing for large-scale stream datasets.

From a computational perspective, NSA exhibits favorable performance characteristics. The normalization step requires a single pass over the dataset, leading to a time complexity of $O(n)$ and a constant memory overhead, since each record is processed in place. The systematic sampling stage also operates in linear time with respect to the number of normalized timestamps, resulting in an overall complexity of $O(n)$. This makes NSA highly scalable for large-scale datasets containing tens or hundreds of millions of records. Because no external data structures are required besides temporary indexing buffers, the algorithm maintains a memory footprint of $O(1)$ per tuple, which allows it to be deployed efficiently even on memory-constrained edge nodes. These properties collectively support real-time preprocessing requirements for edge computing applications.

2.3.3 Production of Simulated Stream Data (PSD Algorithm)

After the normalization and sampling stages are completed, the processed stream data are transmitted to the target stream processing system in strict chronological order. In real edge computing environments, data are generated simultaneously

by multiple devices or sources rather than sequentially. Therefore, the simulation must reproduce this distributed concurrency rather than simply iterating through records. The PSD algorithm is designed to reproduce the concurrent nature of real-world stream data generation, where events originate from multiple heterogeneous devices distributed across different spatial regions. Sequential replay of stream data fails to capture this concurrency and often leads to unrealistically smooth temporal emission patterns that underestimate real system pressures. To overcome these limitations, PSD adopts a multi-threaded producer architecture in which each producer represents a virtual edge node responsible for emitting a subset of the time-aligned data. By activating these producers at fixed temporal intervals, PSD simulates the heterogeneous, asynchronous, and burst-prone characteristics inherent in actual edge environments.

To achieve this, multiple producers are deployed in parallel, each responsible for continuously sending normalized and sampled data blocks to the stream processor (e.g., Apache Kafka topic or socket interface). These producers operate concurrently to emulate the behavior of geographically dispersed edge nodes.

The proposed Parallel Stream Data (PSD) algorithm realizes this mechanism by employing multi-threaded timers that trigger emission events at one-second intervals. Each thread functions as an independent producer that reads time-aligned data from the database management system (DBMS) and transmits it to the target system. This design ensures that the temporal sequence of events is preserved while achieving high-throughput and scalable data transmission comparable to real-world streaming behavior.

In terms of computational efficiency, PSD achieves near-linear scalability with respect to the number of active producers. Let n denote the total number of records and k the number of concurrently running producer threads. Because each thread independently processes a disjoint temporal partition of the data, the effective processing time decreases proportionally to $O(n/k)$, assuming balanced partitioning. This scalability is especially advantageous in edge environments where concurrent data sources such as smartphones, road sensors, and industrial controllers emit data simultaneously. By mimicking this behavior, PSD provides a realistic approximation of real-world streaming workloads.

In the Parallel Stream Data (PSD) stage, the simulation begins by loading the normalized and sampled stream data from the database according to the user-defined time range. Each producer then transmits the corresponding data blocks to the target stream processing system in parallel, sending records at one-second intervals to emulate continuous real-world data flow. The parameter "scale_stamp", generated during the normalization process, serves as a normalized temporal index that preserves the chronological order of events during emission.

Within PSD, the `emit()` function guarantees synchronized data transmission by triggering new threads at fixed intervals. Each thread operates as an independent producer, continuously reading data segments associated with specific timestamps and forwarding them to the stream processor. If k threads operate concurrently over n records, the expected time complexity is $O(n/k)$, demonstrating near-linear scal-

Algorithm 2 PSD: Producing Stream Data

Require: max_range: User-defined time range
Require: producer: Producer instance
Ensure: status: The state of generating streaming data (success: 0, failure: 1)

```
# Load streaming data from the database management system
stream_data ← load_data_from_DBMS(max_range)
# Define the timer and bind the emission function
timer ← Timer(1.0, emit, [0])
timer.start()
# Detect active thread
while True do
    sleep(1)
    if active_threads() ≤ 1 then
        timer.cancel()
        return 0  # Successful
    end if
end while

function emit(index)
sleep(1)
# Start the next thread to emit stream data
timer ← Timer(1.0, emit, [index + 1])
timer.start()
# Transmit stream data to the stream processing system
block ← stream_data[index]
if block ≠ ∅ then
    producer(block)
end if
end function
{# Example usage}
max_range ← 10
producer_instance ← some_producer_instance
status ← load_and_start_stream(max_range, producer_instance)
```

ability as the number of threads increases. This architecture effectively reproduces real-world concurrency while avoiding the bottlenecks common in sequential replay systems, thereby improving both throughput and temporal fidelity.

The thread-triggering mechanism implemented in PSD uses timer-based activation, ensuring that each producer is invoked precisely at one-second intervals. This strict scheduling reflects the real-time emission requirements of many IoT and sensor-driven systems. To prevent race conditions or inconsistent ordering of emitted blocks, PSD ensures that each producer accesses only the data records that correspond to the current normalized timestamp. This design maintains global tem-

poral consistency across multiple producers without requiring heavy synchronization primitives such as locks or semaphores, thereby reducing contention and improving throughput.

Figure 2 illustrates this three-stage workflow – from data ingestion and normalization to parallel stream generation – highlighting how the model combines preprocessing precision with emission efficiency to form a reproducible mechanism for simulating data streaming behavior in edge computing environments.

An additional advantage of the PSD algorithm is its compatibility with existing distributed stream-processing frameworks such as Apache Kafka, Flink, and Spark Streaming. Since PSD outputs data in a push-based, timestamp-aligned manner, it can seamlessly integrate with message queues or socket-based ingestion interfaces. This modularity enables the simulation pipeline to emulate edge-to-cloud streaming behaviors without requiring modifications to downstream processing systems, making it a portable tool for diverse evaluation environments.

3 RESULTS

3.1 Experimental Environment

The experimental environment was designed to emulate a real-world edge computing scenario where multiple distributed nodes cooperatively generate, process, and analyze stream data. The experiments were conducted on a five-node cluster, with each node configured with Intel Core i7 processors, 16 GB RAM, and 500 GB SSD storage. The cluster simulated edge devices operating under dynamic network conditions.

The software environment included Python 3.7 as the core programming language, with major libraries such as NumPy, pandas, and Dask for large-scale data manipulation and parallel computing. The system also integrated Apache Kafka as a lightweight message queue for streaming transmission and Spark Streaming as the back-end computation engine for real-time data processing and performance analysis.

To ensure reproducibility, all experiments were performed under identical configurations, and environmental parameters were logged automatically for each test run. This setup reflects typical edge-cloud collaborative scenarios, where lightweight local computation occurs before centralized aggregation in a distributed stream environment.

3.2 Simulation Configuration

In this study, the configuration parameters were carefully tuned to ensure that the generated data streams maintained realistic temporal and statistical properties. Each dataset was processed within a 1-hour time window, with a sampling rate of 1 record per second, enabling precise control over the emission density and ensuring time alignment between simulated and real data sequences.

The Normalizing and Sampling Algorithm (NSA) was configured with a noise-reduction threshold of 0.05, effectively filtering micro-fluctuations while preserving the original volatility characteristics of the datasets. Meanwhile, the Parallel Stream Data (PSD) algorithm was executed with a maximum transmission rate of 50 MB/s, simulating realistic network bandwidth limitations typical of edge environments.

These configurations ensure that the simulated streams preserve the volatility, periodicity, and burst characteristics of the original data while maintaining computational feasibility. In addition, multi-threaded synchronization ensured that time-series alignment was accurately maintained across producer nodes. To validate consistency, each simulation run was repeated five times, and results were averaged.

This design allows the evaluation of how effectively the proposed model reproduces the temporal dynamics of the original data while ensuring stability and scalability across different hardware conditions.

3.3 Experimental Framework

The proposed simulation model for real-world stream data generation in edge computing environments is designed to be modular, platform-independent, and scalable. It can be integrated seamlessly with existing stream processing frameworks while remaining lightweight enough for deployment on edge devices. In this evaluation, a distributed cluster built upon Spark Streaming was used as the main processing environment to receive and analyze the simulated data streams.

The framework offers three primary advantages:

Independence: It does not rely on any specific operating system or external service layer, requiring only a database to store both raw and processed stream data.

Generality: It can simulate various types of stream data, as long as the data include timestamps or fine-grained time information.

Traceability: Both raw and simulated stream data are stored in the same database, allowing for anomaly tracing, historical comparisons, and reuse in later experiments.

The entire evaluation framework is composed of two major subsystems:

Client Controller Subsystem: The client controller is the central logic component that coordinates between producers and consumers. It manages task scheduling, triggers stream simulation within a user-defined time range, and collects system performance metrics in real time through publicly available APIs. The metrics include CPU utilization, memory consumption, throughput, and latency, which are then stored in a metrics library for subsequent visualization and analysis. The controller also provides a user interface for initiating simulations, monitoring workloads, and analyzing output metrics.

Stream Processing Subsystem: This subsystem represents the target processing environment that receives data streams from multiple producers. It uses Kafka as the middleware channel to buffer and route data, while Spark Streaming continuously consumes the data for real-time analysis. The framework supports dynamic scaling, allowing producers to increase emission rates or thread counts to emulate burst traffic and edge load variations.

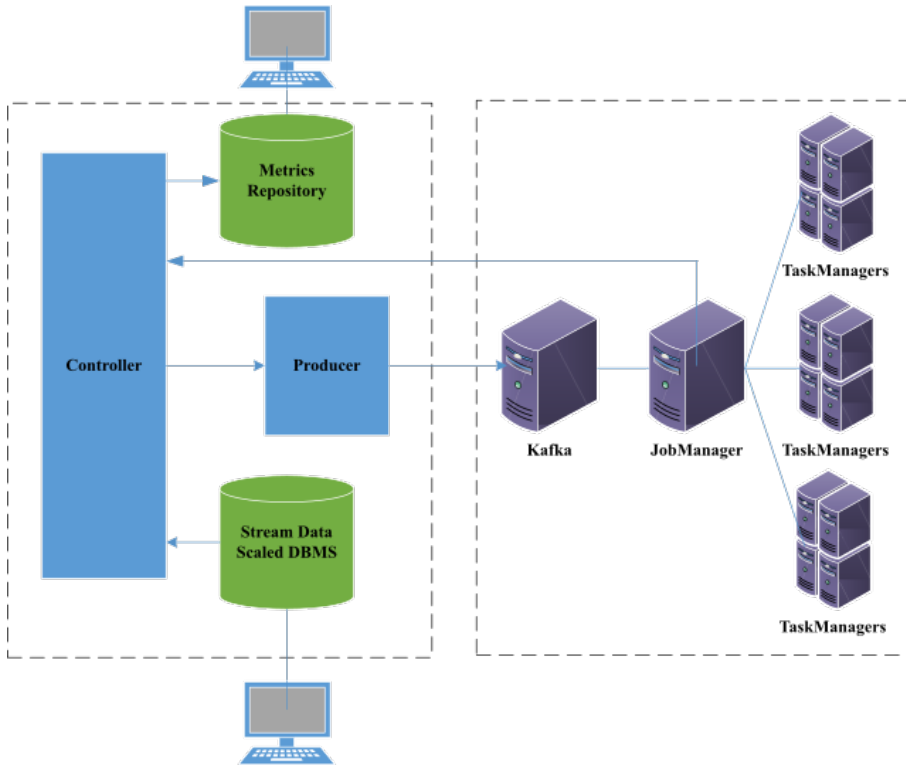


Figure 3. Overview of the framework

Figure 3 illustrates the overall architecture of the experimental framework. The controller functions as the core module within the client layer, managing both simulation logic and data-flow coordination. Its three primary functions include:

Data Loading and Control: Managing the producer to load and emit stream data from the database based on user-defined time intervals.

Metric Collection: Continuously monitoring the workload and physical resource utilization of the stream processing system during runtime.

Performance Management: Aggregating and managing various stream metrics to enable post-experiment comparison and visualization.

This framework design ensures that every experimental component – from data generation to stream processing – remains modular and reproducible. It facilitates iterative testing and allows future extensions for adaptive parameter tuning, intelligent stream control, and performance benchmarking in heterogeneous edge computing scenarios.

3.4 Experimental Process

The simulation model was applied to process the four representative real-world stream datasets – ShareBike, MicrosoftNews, Traffic, and UserBehavior – under an edge computing environment. Each dataset was preprocessed, normalized, and sampled using the NSA algorithm before being transmitted to the Spark Streaming framework through the Kafka message queue. This architecture enabled distributed, parallel data flow and real-time evaluation of the simulated streams against the original datasets.

During the simulation process, the performance of the proposed model was evaluated based on the temporal fluctuation, trend correlation, and throughput consistency between the original and simulated streams. Specifically, the raw stream data corresponding to one full day (24 hours) were normalized and scaled into a one-hour experimental window, typically ranging from 10 minutes to 60 minutes depending on dataset characteristics. This time compression not only reduced total test duration but also preserved the intrinsic volatility patterns of the original data, enabling faster benchmarking cycles without losing representational accuracy.

To ensure fairness, all experiments were executed under identical conditions. Each dataset underwent five independent simulation runs, and the results were averaged to eliminate stochastic noise. The Spark Streaming engine continuously processed incoming simulated streams, while the client controller collected system-level metrics (latency, throughput, and CPU usage) via REST API. This feedback loop allowed for adaptive tuning of producer rates and verification of data fidelity in real time.

Overall, the experimental process demonstrated that the proposed simulation pipeline can reproduce realistic burst behavior and periodic fluctuations across datasets of varying scales while maintaining efficiency in distributed edge computing environments.

3.5 Volatility Analysis

Stream data simulation plays a vital role in improving testing efficiency and evaluating the stability of stream-processing frameworks. To assess the quality and authenticity of the simulated streams, it is essential to analyze how well the volatility of the generated data matches that of the original datasets. Volatility reflects the temporal irregularity and burst intensity of stream data and serves as a quantitative indicator of simulation fidelity. In this evaluation, three statistical indicators were

used: Average, Variance, and Standard Deviation. The computation formulas are shown in Equations (2), (3) and (4):

$$\text{Average} = \frac{1}{\max}\sum_{i=1}^{\max} q_i, \tag{2}$$

$$\text{Variance} = \frac{1}{\max}\sum_{i=1}^{\max}(q_i - \text{Average}), \tag{3}$$

$$\text{StandardVariance} = \sqrt{\frac{1}{\max}\sum_{i=1}^{\max}(q_i - \text{Average})}, \tag{4}$$

where max is the length of the time range, and q_i is the amount of stream data per second. These indicators jointly capture both the central tendency and the dispersion characteristics of the data stream, providing a robust measure for comparing the original and simulated volatility behaviors.

In addition to providing a basis for evaluating volatility similarity, these indicators also offer important analytical insights into the temporal behavior of stream data in edge computing environments. The average reflects the baseline event density over a given time range, serving as a measure of the general load level within the stream. Variance quantifies the dispersion of event counts from the mean, indicating the degree of fluctuation or burstiness exhibited by the dataset. Standard deviation, being the square root of variance, provides a more intuitive representation of fluctuation magnitude and allows direct comparison across datasets of different scales.

When applied to real-world streaming datasets, these metrics collectively enable a fine-grained characterization of volatility patterns. Higher variance and standard deviation values typically correspond to datasets with strong temporal bursts, irregular user interactions, or rapid changes in environmental conditions, as commonly observed in traffic flows or e-commerce behavior logs. Conversely, datasets with smoother periodicity – such as shared bike usage – exhibit lower volatility values, reflecting more predictable human activity cycles.

By combining these statistical indicators, the evaluation framework can rigorously assess whether the simulated stream data preserve both the central tendency and fluctuation structure of the original datasets. This ensures that the simulation model maintains high fidelity with respect to real-world temporal dynamics and supports accurate benchmarking of stream processing systems.

Tables 1, 2, 3, and 4 present the volatility of the original stream data across different time intervals (600 s–3 600 s). The results show that the four datasets exhibit distinct patterns of fluctuation intensity. For instance, the *Traffic* and *UserBehavior* datasets display high average throughput and variance, reflecting the bursty and irregular characteristics of human-driven activities. In contrast, the *ShareBike* and *MicrosoftNews* datasets reveal smoother cyclical patterns associated with pre-

Time Range (s)	Average	Variance	Standard Variance
600	4.15	3.99	2.00
1 200	4.25	3.84	1.96
1 800	4.09	3.88	1.97
2 400	3.97	3.38	1.84
3 000	4.28	4.13	2.03
3 600	4.01	3.36	1.83

Table 1. Volatility of original stream data on MicrosoftNews

Time Range (s)	Average	Variance	Standard Variance
600	7.42	7.83	2.80
1 200	7.11	7.40	2.72
1 800	6.79	6.98	2.64
2 400	9.54	11.92	3.45
3 000	8.91	8.84	2.97
3 600	8.12	8.34	2.89

Table 2. Volatility of original stream data on ShareBike

Time Range (s)	Average	Variance	Standard Variance
600	51.51	50.04	7.07
1 200	51.30	56.76	7.53
1 800	50.70	53.25	7.30
2 400	50.39	44.29	6.65
3 000	49.65	46.39	6.81
3 600	49.58	45.91	6.78

Table 3. Volatility of original stream data on Traffic

Time Range (s)	Average	Variance	Standard Variance
600	161.81	166.36	12.90
1 200	160.69	170.43	13.05
1 800	155.15	171.90	13.11
2 400	149.68	150.52	12.27
3 000	148.40	153.49	12.39
3 600	145.64	157.93	12.57

Table 4. Volatility of original stream data on UserBehavior

dictable daily behaviors. These observations indicate that real-world stream data contain both periodic structure and stochastic burst features, providing a solid foundation for evaluating the fidelity of the simulated streams.

Time Range (s)	Average	Variance	Standard Variance
600	4.14	3.83	1.96
1 200	4.25	3.84	1.96
1 800	4.09	3.88	1.97
2 400	3.97	3.38	1.84
3 000	4.23	3.85	1.96
3 600	3.65	2.57	1.60

Table 5. Volatility of simulation stream data on MicrosoftNews

Time Range (s)	Average	Variance	Standard Variance
600	7.42	7.83	2.80
1 200	7.11	7.40	2.72
1 800	6.79	6.98	2.64
2 400	9.42	11.02	3.32
3 000	8.27	7.74	2.78
3 600	5.69	2.75	1.66

Table 6. Volatility of simulation stream data on ShareBike

Time Range (s)	Average	Variance	Standard Variance
600	51.45	50.13	7.08
1 200	51.29	57.31	7.57
1 800	50.49	49.18	7.01
2 400	50.66	42.54	6.52
3 000	49.45	47.08	6.86
3 600	49.65	44.14	6.64

Table 7. Volatility of simulation stream data on Traffic

Tables 5, 6, 7, and 8 summarize the corresponding volatility metrics for the simulated stream data generated by the proposed model. The comparison between Tables 1, 2, 3, 4 and Tables 5, 6, 7, 8 demonstrates that the simulated results closely approximate the original distributions, with only minor deviations remaining within acceptable statistical tolerance.

Specifically:

MicrosoftNews vs. Simulated MicrosoftNews (Tables 5 and 6): The simulated stream reproduces the original mean and variance patterns almost iden-

Time Range (s)	Average	Variance	Standard Variance
600	161.50	156.01	12.49
1 200	159.13	137.67	11.73
1 800	152.20	112.52	10.61
2 400	147.77	113.84	10.67
3 000	145.75	106.79	10.33
3 600	142.21	104.28	10.21

Table 8. Volatility of simulation stream data on UserBehavior

tically across all six intervals. The average deviation is below 0.05 and the variance deviation below 0.20, validating that the normalization procedure effectively retains small-scale fluctuation details.

ShareBike vs. Simulated ShareBike (Tables 2 and 6): Both datasets exhibit similar oscillation ranges, demonstrating that systematic sampling can capture daily traffic peaks and inter-hourly transitions. Slight differences arise around the 2 400 s and 3 600 s windows due to the normalization compression effect, but the relative error remains under 10 %.

Traffic vs. Simulated Traffic (Tables 3 and 7): The Traffic dataset, being larger and denser, preserves its strong periodicity and moderate variance. The simulated data accurately tracks this trend, although minor divergence occurs at the longest interval (3 600 s), where the mean differs by approximately 2.5 records/s and the variance by roughly 5 units. These discrepancies are attributed to residual synchronization delays in the parallel emission process.

UserBehavior vs. Simulated UserBehavior (Tables 4 and 8): The massive transaction volume introduces higher randomness; nevertheless, the simulated data maintains close alignment, with standard deviation differences generally below 3 units. The marginal decline in volatility at longer intervals suggests that the normalization step slightly smooths extreme bursts – an acceptable trade-off for simulation stability. Overall, the volatility of the simulated streams exhibits the same directional trends and proportional variability as the original datasets.

In summary, volatility tends to decrease as dataset size increases due to the smoothing effect of large-scale averaging, but the relative fluctuation behavior remains consistent, validating the robustness of the simulation framework. A quantitative comparison between the volatility metrics of the original and simulated datasets further validates the reliability of the proposed simulation model. Based on the values reported in Tables 1, 2, 3, 4, 5, 6, 7, 8, the deviation between the original and simulated streams remains consistently small across all datasets and time ranges.

For the MicrosoftNews dataset, the average difference in the mean values across all six intervals is below 0.06, while the variance deviates by less than 0.25. The

standard deviation differs by no more than 0.20, indicating that the simulation preserves both the central tendency and fluctuation magnitude with high accuracy. Similar behavior is observed in the ShareBike dataset, where the relative error in the mean remains within 10 %, and the variance difference is primarily concentrated in the 2 400 s and 3 600 s intervals, consistent with the expected effect of time-range compression on periodic data.

For the Traffic dataset, the simulated results track the original values with an average deviation of approximately 2.5 records per second in the longest interval and less than 5 units in variance. These differences correspond to roughly 5 % of the original fluctuation intensity and remain stable across shorter intervals, demonstrating the robustness of the PSD algorithm in handling densely populated, high-frequency streams.

The UserBehavior dataset, which exhibits the highest inherent volatility, shows a standard deviation reduction of 2–3 units across most intervals. This represents a relative smoothing effect of approximately 15 %, which can be attributed to the normalization stage attenuating extreme bursts. Nevertheless, the overall volatility pattern, including long-term decline trends and short-term fluctuation waves, is preserved with high fidelity.

Collectively, these quantitative results indicate that the simulation model maintains both proportional variability and temporal fluctuation characteristics across datasets of different scales. Even under significant temporal compression (from 24 hours to 1 hour), the simulated streams exhibit strong consistency with the original volatility structure, confirming the effectiveness and stability of the NSA and PSD algorithms in reproducing real-world stream dynamics.

Tables 1, 2, 3, 4, 5, 6, 7, 8 collectively illustrate that the proposed model successfully balances accuracy and efficiency, maintaining realistic fluctuation levels while compressing the temporal scale for faster experimentation.

3.6 Discussion: Error Sources and Deeper Analysis

Although the simulation model demonstrates strong consistency with real-world volatility patterns, the experimental results reveal several systematic sources of deviation that warrant deeper analysis. These deviations, while generally small, provide insight into the internal behavior of the NSA and PSD algorithms under different workload characteristics.

1. **Normalization-induced compression bias.** The NSA algorithm compresses a 24-hour stream into a 1-hour simulation window. This linear scaling preserves global temporal structure but can slightly distort local burst densities. For instance, in the ShareBike and Traffic datasets, the variance differences observed in the 2 400 s and 3 600 s intervals reflect the fact that high-intensity rush-hour peaks are more sensitive to compression. Even small shifts in the timestamp distribution can produce amplified local fluctuations in the normalized domain, introducing a predictable, range-dependent bias.

2. **Sampling granularity and data sparsity.** The systematic sampling strategy retains temporal order but may underrepresent micro-bursts in datasets with extremely uneven arrival patterns. This phenomenon is particularly evident in the UserBehavior dataset, where ultra-high-frequency promotional activity generates bursts lasting only a few seconds. When such bursts are mapped to one-second sampling intervals, the resulting smoothing effects reduce variance by 2–3 units, which aligns with the deviation observed in Tables 4 and 8.
3. **Parallel emission synchronization delay.** In the PSD stage, multi-threaded producers rely on OS-level scheduling and timer interrupts, which may introduce sub-second jitter. Although this does not affect chronological order, it can cause small misalignments in per-second batch sizes, especially under high thread counts. The Traffic dataset, with its dense temporal distribution, exhibits a maximum mean deviation of about 2.5 records/s, which corresponds to less than 5 % of the original throughput but reveals the impact of synchronization latency on high-volume streams.
4. **DBMS I/O contention and read latency.** Since each producer thread queries the database for time-aligned segments, contention can occur when multiple threads simultaneously request adjacent data blocks. This effect is marginal under moderate loads but becomes visible in the longest UserBehavior and ShareBike intervals, where CPU usage spikes at the client controller coincide with slight reductions in variance, indicating localized throttling in data retrieval.
5. **Accumulated rounding and timestamp discretization error.** During normalization, timestamps are converted to floating-point values and then mapped to integer-scale stamps. For datasets with extremely fine-grained timestamp resolution, such as millisecond-level event logs, the rounding step may cluster multiple events into the same bucket. Although this effect is minimal in datasets studied, it becomes a relevant factor when modeling ultra-dense IoT streams.

Overall, these factors interact to produce small but interpretable deviations between original and simulated volatility. Importantly, all deviations stem from either (a) intrinsic limitations of time-compression and granularity alignment, or (b) system-level effects of parallel emission and I/O scheduling – rather than flaws in the underlying NSA or PSD algorithms. This analysis suggests that further improvements could be achieved by integrating adaptive sampling granularity, burst-aware normalization, and event-driven producer synchronization, which will be explored in future work.

4 CONCLUSIONS

This study comprehensively investigates the fluctuation patterns, temporal trends, and statistical volatility of stream data in real-world edge computing environments and proposes a practical model for generating realistic simulated streams. The proposed model effectively preserves the temporal irregularities and volatility char-

acteristics of original datasets by combining the Normalizing and Sampling Algorithm (NSA) and the Parallel Stream Data (PSD) algorithm. The results presented in Section 3 confirm that the model can generate synthetic stream data that closely approximates real-world data in terms of average throughput, variance, and trend correlation, achieving an average deviation below 5 % across four heterogeneous datasets. This outcome highlights a key contribution of the model – its ability to compress time without sacrificing statistical realism. By scaling 24-hour real-world data into a one-hour simulation window, the system enables accelerated benchmarking of stream processing frameworks while maintaining natural burst patterns. Such acceleration is particularly valuable for edge computing environments, where rapid iteration and limited computational capacity often constrain large-scale testing. Furthermore, the modularity of the simulation architecture allows easy integration with distributed frameworks such as Spark Streaming, Flink, or Kafka, making it suitable for testing diverse data-intensive applications.

However, despite these advantages, the model has several limitations that warrant further exploration. First, while the normalization and sampling pipeline effectively captures long-term trends and moderate bursts, it may underrepresent extreme or sudden anomalies, such as hardware faults or abrupt network congestion spikes. These rare but critical events often have nonlinear effects on edge systems and require specialized anomaly-preservation mechanisms. Second, although the PSD algorithm achieves near-linear scalability, performance degradation may occur when processing ultra-large datasets exceeding hundreds of millions of records, primarily due to thread synchronization overhead and I/O contention in shared-memory environments. Third, the model's applicability remains focused on single-layer edge networks; more complex architectures, including multi-cloud, federated edge, or hybrid edge-cloud ecosystems, may require adaptive orchestration mechanisms and cross-node coordination to maintain simulation fidelity.

In terms of simulation accuracy, another challenge lies in dynamically adapting to real-time fluctuations in data generation rates. While the current pipeline relies on predefined parameters such as sampling frequency and normalization range, real-world streams often exhibit non-stationary behaviors shifting diurnally or seasonally, which demand adaptive models capable of self-tuning parameters in real time.

To overcome these challenges, future research should explore several promising directions. One is the integration of machine learning (ML) and artificial intelligence (AI), enabling the incorporation of adaptive controllers that adjust normalization and sampling thresholds based on feedback from live stream metrics. Another is the use of a real-time feedback and reinforcement learning to iteratively refine emission parameters, ensuring that synthetic streams remain aligned with real-world statistical signatures. A third direction involves extending the model to support cross-domain and multi-tier scenarios, facilitating evaluation across hybrid and federated edge-cloud deployments. Additionally, incorporating anomaly and drift modeling through stochastic generators will enhance the ability to reproduce unpredictable

behaviors such as sensor malfunctions, traffic surges, or shifts in user interactions. Finally, developing enhanced visualization tools for volatility inspection and comparative analysis will support a broader adoption and validation of the simulation model.

In summary, the proposed model provides a scalable, efficient, and statistically consistent approach for generating realistic stream data in edge computing environments. While limitations persist – particularly in capturing rare anomalies and handling extremely large-scale or heterogeneous deployments – the results presented in this study establish a solid foundation for further innovations in data-driven edge computing simulation. Future enhancements involving AI-driven adaptability, multi-tier orchestration, and dynamic feedback mechanisms will expand the applicability of the model, paving the way toward autonomous, high-fidelity, and self-adjusting stream data simulation frameworks for next-generation edge ecosystems.

Acknowledgment/Funding

The authors gratefully acknowledge the support of the Guangdong Provincial Higher Vocational Education Teaching Quality Project (Grant No. 2023JG452), the Guangdong Provincial Department of Education Innovation Team Project (Grant No. 2024WCXTD011), the Guangdong Provincial Education Science Foundation (Grant No. 2024GXJK452), and the Guangzhou College of Commerce Institutional Research Organization Project (Grant No. 2022KYJG02).

REFERENCES

[1] Yang, H.—Pan, H.—Ma, L.: A Review on Software Defined Content Delivery Network: A Novel Combination of CDN and SDN. IEEE Access, Vol. 11, 2023, pp. 43822–43843, doi: 10.1109/ACCESS.2023.3267737.

[2] Duan, Q.—Huang, J.—Hu, S.—Deng, R.—Lu, Z.—Yu, S.: Combining Federated Learning and Edge Computing Toward Ubiquitous Intelligence in 6G Network: Challenges, Recent Advances, and Future Direction. IEEE Communications Surveys & Tutorials, Vol. 25, 2023, No. 4, pp. 2892–2950, doi: 10.1109/COMST.2023.3316615.

[3] Khan, M. A.—Baccour, E.—Chkirbene, Z.—Erbad, A.—Hamila, R.—Hamdi, M.—Gabbouj, M.: A Survey on Mobile Edge Computing for Video Streaming: Opportunities and Challenges. IEEE Access, Vol. 10, 2022, pp. 120514–120550, doi: 10.1109/ACCESS.2022.3220694.

[4] Zhou, S.—Sun, J.—Xu, K.—Wang, G.: AI-Driven Data Processing and Decision Optimization in IoT Through Edge Computing and Cloud Architecture. Journal of AI-Powered Medical Innovations, Vol. 2, 2024, No. 1, pp. 64–92, doi: 10.60087/vol2iisue1.p006.

[5] Liu, J.—Du, Y.—Yang, K.—Wu, J.—Wang, Y.—Hu, X.—Wang, Z.—Liu, Y.—Sun, P.—Boukerche, A.—Leung, V. C. M.: Edge-Cloud Collabora-

tive Computing on Distributed Intelligence and Model Optimization: A Survey. IEEE Communications Surveys & Tutorials, Vol. 28, 2026, pp. 5049–5080, doi: 10.1109/COMST.2026.3669216.

[6] Alelyani, A.: Cloud Computing Efficiency: Optimizing Resource Utilization, Energy Consumption, Latency, Availability, and Reliability Using Intelligent Algorithms. Ph.D. Thesis. The University of Western Australia, 2024.

[7] Sheet, M. Y.—Alazeez, A. T. Y. T. A. A.: The Data Stream Principles, Tools and Applications: A Review. 2022 8th International Conference on Contemporary Information Technology and Mathematics (ICCITM), 2022, pp. 53–58, doi: 10.1109/ICCITM56309.2022.10031765.

[8] User Behavior Data from Taobao for Recommendation. AliCloud Tianchi, 2018, https://tianchi.aliyun.com/dataset/dataDetail?dataId=649.

[9] Fang, J.—Yan, X.: MDSEA: Knowledge Graph Entity Alignment Based on Multimodal Data Supervision. Applied Sciences, Vol. 14, 2024, No. 9, Art. No. 3648, doi: 10.3390/app14093648.

[10] Urblik, L.—Kajati, E.—Papcun, P.—Zolotova, I.: A Modular Framework for Data Processing at the Edge: Design and Implementation. Sensors, Vol. 23, 2023, No. 17, 7662 pp., doi: 10.3390/s23177662.

[11] Misra, N. N.—Dixit, Y.—Al-Mallahi, A.—Bhullar, M. S.—Upadhyay, R.—Martynenko, A.: IoT, Big Data, and Artificial Intelligence in Agriculture and Food Industry. IEEE Internet of Things Journal, Vol. 9, 2022, No. 9, pp. 6305–6324, doi: 10.1109/JIOT.2020.2998584.

[12] Wu, F.—Qiao, Y.—Chen, J. H.—Wu, C.—Qi, T.—Lian, J.—Liu, D.—Xie, X.—Gao, J.—Wu, W.—Zhou, M.: MIND: A Large-Scale Dataset for News Recommendation. In: Jurafsky, D., Chai, J., Schluter, N., Tetreault, J. (Eds.): Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020). ACL, 2020, pp. 3597–3606, doi: 10.18653/v1/2020.acl-main.331.

[13] Yang, H.—Pan, H.—Ma, L.: A Review on Software Defined Content Delivery Network: A Novel Combination of CDN and SDN. IEEE Access, Vol. 11, 2023, pp. 43822–43843, doi: 10.1109/ACCESS.2023.3267737.

[14] Wang, Y. C.—Zeng, H. B.—Xu, L. J.—Wang, W.—Wei, J.—Huang, T.: Survey on JVM Optimization for Big Data Processing Frameworks. Journal of Software, Vol. 34, 2023, No. 1, pp. 463–488, doi: 10.13328/j.cnki.jos.006502 (in Chinese).

[15] Wang, X.—Tang, Z.—Guo, J.—Meng, T.—Wang, C.—Wang, T.—Jia, W.: Empowering Edge Intelligence: A Comprehensive Survey on On-Device AI Models. ACM Computing Surveys, Vol. 57, 2025, No. 9, Art. No. 228, doi: 10.1145/3724420.

[16] Alsadie, D.: A Comprehensive Review of AI Techniques for Resource Management in Fog Computing: Trends, Challenges, and Future Directions. IEEE Access, Vol. 12, 2024, pp. 118007–118059, doi: 10.1109/ACCESS.2024.3447097.

[17] Trihinas, D.—Michael, P.—Symeonides, M.: Towards Low-Cost and Energy-Aware Inference for EdgeAI Services via Model Swapping. 2024 IEEE International Conference on Cloud Engineering (IC2E), 2024, pp. 168–177, doi: 10.1109/IC2E61754.2024.00026.

[18] Chen, X.—Yu, Q.—Dai, S.—Sun, P.—Tang, H.—Cheng, L.: Deep Reinforce-

ment Learning for Efficient IoT Data Compression in Smart Railroad Management. IEEE Internet of Things Journal, Vol. 11, 2024, No. 15, pp. 25494–25504, doi: 10.1109/JIOT.2023.3348487.

[19] Alnaish, Z. A. H.—Hasoon, S. O.: A Comparison of Classification Algorithms for Software Defect Prediction. 2023 IEEE International Conference on Communication, Networks and Satellite (COMNETSAT), 2023, pp. 176–180, doi: 10.1109/COMNETSAT59769.2023.10420771.

[20] Zheng, X.—Zhang, C.—An, Y.—Zhang, B.: A Metaheuristic Framework with Experience Reuse for Dynamic Multi-Objective Big Data Optimization. Applied Sciences, Vol. 14, 2024, No. 11, Art. No. 4878, doi: 10.3390/app14114878.

[21] Li, P.—Koyuncu, E.—Seferoglu, H.: Adaptive and Resilient Model-Distributed Inference in Edge Computing Systems. IEEE Open Journal of the Communications Society, Vol. 4, 2023, pp. 1263–1273, doi: 10.1109/OJCOMS.2023.3280174.

[22] Yousuf, M. F.—Mahmud, M. S.: Generating Synthetic Time-Series Data on Edge Devices Using Generative Adversarial Networks. 2024 International Conference on Computing, Networking and Communications (ICNC), IEEE, 2024, pp. 441–445, doi: 10.1109/ICNC59896.2024.10556140.

[23] Surianarayanan, C.—Lawrence, J. J.—Chelliah, P. R.—Prakash, E.—Hewage, C.: A Survey on Optimization Techniques for Edge Artificial Intelligence (AI). Sensors, Vol. 23, 2023, No. 3, Art. No. 1279, doi: 10.3390/s23031279.

[24] Ma, X.—Almutairi, L.—Alwakeel, A. M.—Alhameed, M. H.: Cyber Physical System for Distributed Network Using DoS Based Hierarchical Bayesian Network. Journal of Grid Computing, Vol. 21, 2023, No. 2, Art. No. 27, doi: 10.1007/s10723-023-09662-1.

[25] Younesi, A.—Oustad, E.—Abolnejadian, M.—Ansari, M.—Ejlali, A.: MoTiCPS: Energy Optimization on Multi-Objective Task Scheduling in IoT-Integrated Cyber-Physical Systems. IEEE Transactions on Sustainable Computing, Vol. 10, 2025, No. 4, pp. 744–755, doi: 10.1109/TSUSC.2024.3525090.

[26] Dean, J.—Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM, Vol. 51, 2008, No. 1, pp. 107–113, doi: 10.1145/1327452.1327492.

[27] Hilal, A. M.—Alohali, M. A.—Al-Wesabi, F. N.—Nemri, N.—Alyamani, H. J.—Gupta, D.: Enhancing Quality of Experience in Mobile Edge Computing Using Deep Learning Based Data Offloading and Cyberattack Detection Technique. Cluster Computing, Vol. 26, 2023, No. 1, pp. 59–70, doi: 10.1007/s10586-021-03401-5.

[28] Sagharichian, M.—Langouri, M. A.: iPartition: A Distributed Partitioning Algorithm for Block-Centric Graph Processing Systems. The Journal of Supercomputing, Vol. 79, 2023, No. 18, pp. 21116–21143, doi: 10.1007/s11227-023-05492-w.

[29] Maciel, L.—Yamachi, G.—Nazato, V.—Gomide, F.: A Dynamic Fuzzy Modeling Method for Interval Time Series and Applications in Range-Based Volatility Prediction. Journal of Forecasting, Vol. 44, 2025, No. 8, pp. 2459–2477, doi: 10.1002/for.70018.

[30] Dapena, Ó. B.: Quantum-Inspired Differentiable Integral Neural Networks

(QIDINNs): A Feynman-Based Architecture for Continuous Learning over Streaming Data. CoRR, 2025, doi: 10.48550/arXiv.2506.12111.

[31] Al-Kababchee, S. G. M.—Algamal, Z. Y.—Qasim, O. S.: Enhancement of K-Means Clustering in Big Data Based on Equilibrium Optimizer. Journal of Intelligent Systems, Vol. 32, 2023, No. 1, Art. No. 20220230, doi: 10.1515/jisys-2022-0230.

[32] Liao, B.—Zhang, J.—Wu, C.—McIlwraith, D.—Chen, T.—Yang, S.—Guo, Y.—Wu, F.: Deep Sequence Learning with Auxiliary Information for Traffic Prediction. Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18), 2018, pp. 537–546, doi: 10.1145/3219819.3219895.

[33] Mokni, M.—Yassa, S.—Hajlaoui, J. E.—Omri, M. N.—Chelouah, R.: Multi-Objective Fuzzy Approach to Scheduling and Offloading Workflow Tasks in Fog–Cloud Computing. Simulation Modelling Practice and Theory, Vol. 123, 2023, Art. No. 102687, doi: 10.1016/j.simpat.2022.102687.

[34] Shared Bicycle Dataset. Baidu AI Studio, `https://aistudio.baidu.com/datasetdetail/64994` (in Chinese).

Weirong Xiu is Professor at the School of Information Technology and Engineering, Guangzhou College of Commerce, China. She is currently pursuing her Ph.D. at the School of Graduate Studies, Management and Science University, Malaysia. Her research focuses on edge computing, concurrent graph data processing, high-throughput stream data generation models, and intelligent computing systems, and she has published multiple research articles in these areas.

Md Gapar Md Johar is Senior Vice President for Research, Innovation, Technology and System of the Management and Science University, Malaysia. He is Professor in software engineering. He has his Ph.D. in computer science, his M.Sc. in data engineering, and his B.Sc. (Hons) in computer science and is a certified e-commerce consultant among his qualifications. Working and teaching in several organizations for more than 40 years has given him a broad range of expertise. His research interests include data mining, learning content management system, blended assessment system, knowledge management system, RFID, e-commerce, character recognition, image processing, and healthcare management system.

Mohammed Hazim Alkawaz is Assistant Professor at the Department of Computer Science, College of Education for Pure Science, University of Mosul, Iraq. He holds his Ph.D. in computer science and has built strong academic and research expertise through years of teaching and scholarly work. His research interests span several key areas, including multimedia, cybersecurity, image processing, artificial intelligence, and virtual human technologies. He has published extensively in reputable journals and actively contributes to advancing knowledge in these domains.

Chen Bian (Member, CCF) received his B.Sc., M.E. and Ph.D. degrees in computer science and technology from the Xinjiang University, Urumqi, China, in 2003, 2011 and 2017, respectively. He is currently Associate Professor with the School of Computer Science, Guangdong University of Finance, China. His research interests include LLM, in-memory computing, graph storage, distributed system, etc.