TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 05/20 pp1674–1693
DOI: 10.26599/TST.2023.9010083
Volume 29, Number 6, December 2024

Streaming Histogram Publication over Weighted Sliding Windows Under Differential Privacy

Xiujun Wang*, Lei Mo, Xiao Zheng, and Zhe Dang

Abstract: Continuously publishing histograms in data streams is crucial to many real-time applications, as it provides not only critical statistical information, but also reduces privacy leaking risk. As the importance of elements usually decreases over time in data streams, in this paper we model a data stream by a sequence of weighted sliding windows, and then study how to publish histograms over these windows continuously. The existing literature can hardly solve this problem in a real-time way, because they need to buffer all elements in each sliding window, resulting in high computational overhead and prohibitive storage burden. In this paper, we overcome this drawback by proposing an online algorithm denoted by Efficient Streaming Histogram Publishing (ESHP) to continuously publish histograms over weighted sliding windows. Specifically, our method first creates a novel sketching structure, called Approximate-Estimate Sketch (AESketch), to maintain the counting information of each histogram interval at every time instance; then, it creates histograms that satisfy the differential privacy requirement by smartly adding appropriate noise values into the sketching structure. Extensive experimental results and rigorous theoretical analysis demonstrate that the ESHP method can offer equivalent data utility with significantly lower computational overhead and storage costs when compared to other existing methods.

Key words: differential privacy; randomized algorithm; streaming data publication; weighted sliding window; approximate statistics; data usability; computational complexity

1 Introduction

With the rapid development of big data[1–6] and the Internet of Things (IoTs)[7–14], streaming data (data streams) are commonly found in diverse real–world scenarios, such as real-time traffic streams[15–18] and hospital patient information data streams[19, 20]. Histograms, which are statistical methods used to capture the distributional information of streaming data, play a crucial role in real-time analysis applications in these scenarios. However, publishing histograms without privacy protection can pose a significant risk of exposing sensitive personal information to malicious adversaries who could exploit the published data. Therefore, there is a growing need to continuously construct publicly publishable histograms over streaming data, particularly in applications where data privacy is a concern.

Generally, people tend to take more interest in recent data than older ones, this is because the importance of the element decreases as the element ages. Thus, in this

• Xiujun Wang and Xiao Zheng are with School of Computer Science and Technology, Anhui University of Technology, Ma’anshan 243032, China, and also with Institute for Artificial Intelligence, Hefei Comprehensive National Science Center, Hefei 230088, China. E-mail: {xjwang, xzheng}@ahut.edu.cn.
• Lei Mo is with Baosight Software (Anhui) Co. Ltd., Ma’anshan 243000, China. E-mail: ahut_molei@sina.com.
• Zhe Dang is with School of Electrical Engineering and Computer Science, Washington State University, Pullman 99164, WA, USA. E-mail: zdang@wsu.edu.
∗ To whom correspondence should be addressed.
Manuscript received: 2023-05-17; revised: 2023-07-10; accepted: 2023-08-01

paper, we focus on the weighted sliding window model[21, 22], which generalizes the traditional sliding window model by incorporating a weighting factor $\gamma \in [0,1]$. This model naturally has a broad range of applications[23, 24]. For example, considering a monitoring system in a certain network scenario, recent packets are considered more important than the older ones, as they reflect the network status more accurately and timely. So, the weighted sliding window model is frequently used in these online monitoring systems[25] for data streams. As another example, let us look at social network scenarios. In this case, it is clear that recent data outweigh old data when building a trust relationship among users[26]. For more practical examples, please refer to Refs. [27–30].

Nowadays, there are many research works focusing on the data distribution of static data sets or dynamic data streams[31–40]. Nowadays, a number of methods have been proposed for generating safe histograms that do not jeopardize individual user privacy. Most of them are designed to handle static datasets[31–36]. Others are designed to handle dynamic data streams where time and space costs are also considered to be a main factor[37–40].

However, the existing literature on data analysis techniques lacks adequate methods for addressing the weighted sliding window model, where the relevance of data diminishes over time. This is because these methods assign equal importance to all data points in a data stream, irrespective of their relevance and time of occurrence. However, this approach is not suitable for many practical scenarios, such as predicting the likelihood of diseases based on the most recent data in a health app, forecasting the popularity of existing apps using the latest data from Apple, and monitoring and analyzing the most recent financial industry data to mitigate risks. A scenario where a weighted data stream generates a noise histogram is shown below in Fig. 1.

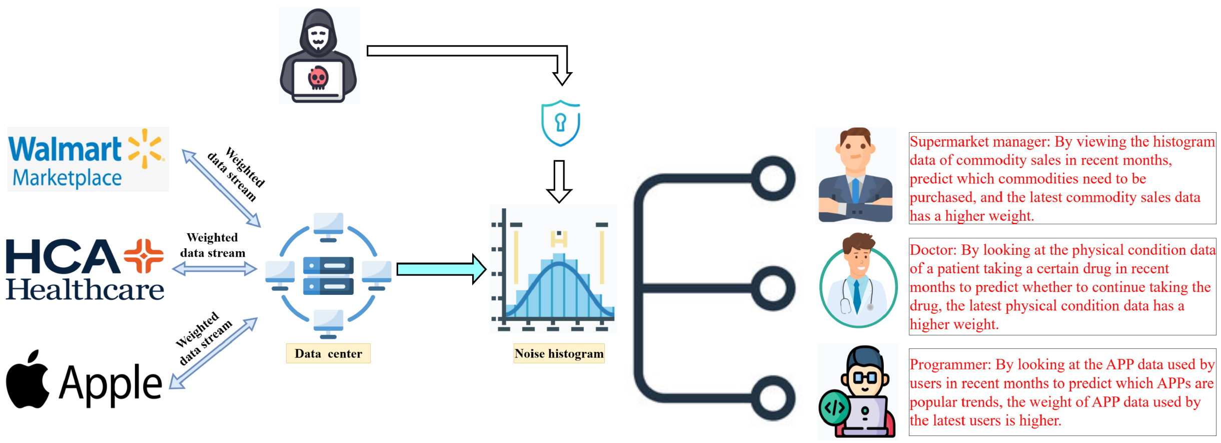
In these common scenarios, in particular, they suffer from two drawbacks when they are used to publish histograms over weighted sliding windows:

(1) The existing methods are not incorporated with an effective mechanism, which places more emphasis on the new elements than the old ones;

(2) The existing methods have high computational overhead and prohibitive storage burden, as they need to repeatedly scan each element contained in each sliding window when constructing a publishable histogram (a safe one that reveals the trending information of a sliding window but does not jeopardize individual users' privacy).

In this paper, we overcome these drawbacks by modelling a data stream by a sequence of weighted sliding windows and storing each weighted sliding window with a novel sketching data structure. Specifically, (1) in the weighted sliding window model, we can characterize the data importance by a weighted factor $\gamma \in [0,1]$, which is pre-specified according to practical safe requirements, (2) with the novel sketching data structure, we can not only significantly reduce the computational cost, but also adaptively control the difference between real data and noisy data, to guarantee competitive usability of query data in most cases for the weighted sliding window model.

Please note that we follow the typical and widely used



Supermarket manager: By viewing the histogram data of commodity sales in recent months, predict which commodities need to be purchased, and the latest commodity sales data has a higher weight.

Doctor: By looking at the physical condition data of a patient taking a certain drug in recent months to predict whether to continue taking the drug, the latest physical condition data has a higher weight.

Programmer: By looking at the APP data used by users in recent months to predict which APPs are popular trends, the weight of APP data used by the latest users is higher.

Fig. 1 Scenarios for weighted data streams. Note: a weighted data stream from Walmart can be a series of shopping records weighted by the time of goods sales; a weighted data stream from HCA can be a list of patient records weighted by their freshness; and a weighted data stream from Apple can be a series of app records weighted by the popularity of the app.

definition of weighted sliding windows from Ref. [5]. Specifically, let $\mathcal{D}$ represent a data stream, where each element e_t arrives at time point t (t is called the timestamp of e_t). Given a window size of w, the sliding window at time point t, denoted by D_w^t, is defined as the w most recent elements in $\mathcal{D}$, i.e., $D_w^t = e_t, e_{t-1}, \ldots, e_{t-w+1}$. Then, for any given weighted factor $\gamma \in [0,1]$, the weighted sliding window $D_w^t(\gamma)$ at time point t refers to the combination of D_w^t and w importance weights assigned to each element contained in D_w^t. The importance weight of e_t is γ^0, the importance weight of e_{t-1} is γ, and at last, the importance weight of e_{t-w+1} is γ^{w-1}. Clearly, the importance weight of each element e_t in D_w^t has its importance weight controlled by the difference between its timestamps t and w, and the weighted factor γ. With this weighted sliding window model, we propose an on-line algorithm denoted by Efficient Streaming Histogram Publishing (ESHP), which enables a continuous publication of histograms over a sequence of weighted sliding windows (i.e., $D_w^t(\gamma)$, $D_{w+1}^t(\gamma)$, ...). The proposed ESHP algorithm first divides the value range of each element from a data stream $\mathcal{D}$ into multiple histogram intervals; then, ESHP stores the critical statistical information of each element from $\mathcal{D}$ into a memory-efficient sketching data structure via scanning the data stream $\mathcal{D}$ only once; lastly, ESHP generates an approximate histogram based on the sketching data structure and then makes it publishable by adding an appropriate Laplacian noise.

In order to make the designed ESHP algorithm efficient in terms of execution time and make ESHP competitive in terms of data usability (i.e., satisfactory accuracy in the generated histogram) in a one-pass scan of the data stream, we face two major technical challenges.

The first technical challenge is to design a novel sketching data structure that can extract critical statistical information from the data stream $\mathcal{D}$ in a one-pass scan, and then generate an approximate histogram rapidly. The second technical challenge is how to secure competitive usability by adding a suitable level of Laplacian noise to the generated histogram.

To solve those technical challenges, we need to deal with these problems.

(1) The solution to the first challenge involves selectively compressing the count information and timestamps of elements in the dataset $\mathcal{D}$ into a two-dimensional array. Additionally, it requires efficiently collecting the data distribution within each weighted sliding window to achieve continuous histogram updates.

(2) The solution to the second challenge is to balance the privacy protection and usability by randomly choosing the release counting information from a weighted sliding window. This solution also incorporates a mechanism that leverages the difference between approximate statistical information for each histogram interval and a noise value for selective release.

In summary, there are three major contributions in this paper as follows:

(1) Our approach creates a novel sketching structure called Approximate-Estimate Sketch Approximate-Estimate Sketch (AESketch), which is suitable for the weighted sliding window model and can be used to continuously publish histograms;

(2) Our approach proposes a selective publishing mechanism. This mechanism uses the difference between the approximate statistical information of each histogram interval and the noise value to select better counting information, and utilizes greedy grouping for all interval counting in this weighted sliding window to guarantee the same data utility of query data in most cases;

(3) Rigorous theoretical and extensive experimental results demonstrate that the proposed ESHP method can provide the same data utility with a substantially reduced computational overhead and storage cost substantially as compared with other existing methods.

This paper is organized as follows: Section 2 reviews the existing literature on differential privacy histogram publication methods. In Section 3, we provide a concise introduction to the theoretical foundations and relevant definitions. Section 4 presents our proposed ESHP algorithm, which comprises two main components. In Section 5, we thoroughly evaluate the privacy aspects of the ESHP algorithm, and analyze its space and time complexity. Section 6 consists of simulations of the ESHP algorithm, where we compare its performance with that of related approaches and provide a detailed analysis of the experimental results. Finally, in Section 7, we conclude the paper by summarizing the key findings and contributions of our research.

2 Related Work

The existing differential privacy histogram publication methods can be mainly divided into two categories.

2.1 Histogram publication method for static data

Several recent studies have focused on analyzing and improving publication methods for static data histograms.

For instance, Xu et al.[31] introduced an approach aiming at reducing query error by compressing similar frequency intervals heuristically. They presented the StructureFirst algorithm, which demonstrates reduced query error through theoretical analysis and empirical experiments. In a different work, Zhang et al.[32] proposed a histogram publishing method known as Differentially private Histogram Release (DiffHR). Their approach involves clustering the original data into distinct clusters, and subsequently adding Laplace noise to protect the clusters. They carefully balanced the error induced by clustering with the error caused by Laplace noise, and showed that the generated histograms using DiffHR can improve the usability of the data. Tang et al.[33] introduced APB, an adaptive privacy budget allocation strategy for histogram publishing. This algorithm optimizes the allocation model for privacy budget weights by determining the weight allocation ratio that minimizes the total error based on the model. The results are then grouped using a greedy approach. The APB method effectively balances the trade-off between noise error and reconstruction error, enhancing the published data's usability. Another study by Chen et al.[34] proposed a method that injects noise into Haar wavelet coefficients using the Gaussian mechanism. By leveraging the Haar wavelet transform and the Gaussian mechanism, this approach ensures differential privacy for input data and arbitrary range query. Notably, the noise variance achieved by this method is significantly smaller compared to using the Gaussian mechanism alone.

Overall, while these methods make valuable contributions to achieving privacy in histogram publication, their practical utility may be limited in situations where efficiency and scalability are paramount. Hence, further research endeavours are necessary to explore alternative approaches that strike a better balance between privacy preservation and the accompanying space and time considerations. Such efforts should focus on developing more efficient and scalable techniques for private histogram generation.

2.2 Histogram publication method for dynamic data

There are some research works proposed recently for histogram publication methods for dynamic data.

For instance, Fan and Xiong[41] proposed a real-time aggregation statistics framework FAST based on filtering and adaptive sampling under differential privacy. Lin et al.[36] proposed a privacy protection publishing algorithm PTDSS-SW for two-dimensional spatial data streams. The proposed algorithm uses low space overhead to approximate the two-dimensional data streaming information and adds appropriate noise to the statistical results. Zhang and Meng[37] proposed a partitioning-based method, called Streaming Histogram Publication (SHP). First, the bucket count in the sliding window is divided into different groups, and then the privacy parameters are adaptively allocated according to different data sampling results. This method can reduce the overall privacy budget, and can not quickly consume the privacy budget. Wu et al.[38] proposed a histogram publishing algorithm using Kullback-Leibler (KL) divergence as a measurement method. The algorithm uses the Kullback-Leibler divergence to calculate the amount of change between two adjacent data and adds different noise values to the published data according to the different values of the Kullback-Leibler divergence calculation to reduce noise errors. Sun et al.[39] proposed a complete algorithm for differential private real-time streaming data publication by putting the Fenwick-tree and matrix optimization together. The algorithm effectively improves query efficiency while ensuring query quality.

In addition, we notice that negative surveys[42] can be used to protect privacy. To ensure the completeness of our paper, we have reviewed the literature related to negative surveys[43–45]. For example, Ref. [43] introduced the definition of negative surveys for the first time, proposed several state-of-the-art methods dealing with prevention and coding strategies in negative surveys, and demonstrated the potential impact of negative surveys in the field of social sciences. Jiang et al.[44] proposed a method based on negative surveys for collecting the time-series data from users and employing it to collect power consumption data, and presented an approach that can provide aggregated power consumption data to the smart grid for load monitoring. Yang et al.[45] proposed a privacy-preserving scheme based on negative surveys to protect the privacy of vehicle fuel consumption data. However, most existing methods[43–48] based on negative surveys may incur a high computational overhead and a heavy storage burden when processing data streams and generating histograms for sliding windows. Because these methods typically require storing all data points (i.e., the false categories of all respondents) contained in each sliding window at each time point, which incurs a heavy storage cost and a large computational cost.

To summarize, the existing methods heavily depend on buffering to store each sliding window, leading to an unacceptably high cost in terms of storage space and computational complexity. This approach poses challenges in terms of scalability and efficiency, particularly when dealing with large datasets. The reliance on buffering in these methods necessitates the allocation of significant memory resources to accommodate the sliding windows. As the dataset size increases, the storage space required grows proportionally, making it impractical for scenarios with limited memory availability. To address these limitations, exploring novel approaches that alleviate the storage and computational burdens will be crucial to achieving scalable and efficient solutions for handling sliding window data in various applications.

3 Definition and Model

In this section, we introduce the definition of differential privacy and the concept of a weighted sliding window model.

3.1 Definitions for data streams and differential privacy

A data stream, denoted as $\mathcal{D}=\langle e_1, e_2, e_3, \ldots, e_n, \ldots\rangle$, is an infinite sequence of elements. Each element e_t arrives with a timestamp t and is chosen from a large universe $U=\{1,2,\ldots,u\}$, representing the possible values an element can take.

Differential privacy, initially proposed by Dwork[42], has emerged as a powerful approach to privacy protection, underpinned by a robust mathematical foundation. It offers distinct advantages over traditional data protection methods, which often rely on encryption. Two key characteristics differentiate differential privacy:

(1) Protection through noise addition: Differential privacy safeguards data by injecting noise using randomized algorithms. By doing so, it provides a probabilistic guarantee of privacy while preserving the overall statistical properties of the data.

(2) Independence from background knowledge: Differential privacy effectiveness remains unaffected by the background knowledge possessed by potential attackers. It ensures the same level of privacy protection, regardless of the amount of information known to an adversary.

In the subsequent section, we will delve into the essential definitions of data streams and differential privacy. These conceptual frameworks lay the groundwork for our proposed solution, enabling the development of privacy-preserving mechanisms for data streams.

In this paper, this threat model of differential privacy[42] involves a powerful adversary with the ability to extract privacy-sensitive information about a user record by comparing query results obtained from two neighbouring datasets, denoted as D and D'. These datasets comprise individual user records, with the sole distinction being the inclusion of a single user record, denoted as u in one dataset (i.e., $D-D'=u$).

The threat model assumes that the attacker has access to all records in dataset D, except for one user record u. Consequently, the attacker is aware of the dataset $D'=D-u$, which excludes the record u. The attacker can obtain the query result x' on D' (i.e., $x'=f(D')$, where $f(\cdot)$ represents the query function known to the public). Additionally, the attacker can also acquire the query results x published by the system on dataset D (i.e., $x=f(D)$). The objective of the attacker is to gain insights into user privacy by comparing the discrepancy between x and x' in order to determine whether the user record u is present in dataset D or not.

Definition 1 ε-differential privacy[42]: Algorithm A is a data processing algorithm. Given two neighboring data streams: $\mathcal{D}$ and $\mathcal{D}'$, Algorithm A achieves the differential privacy protection requirement if and only if for any output set $O\subseteq \text{Range}(A)$, where $\text{Range}(A)$ denotes the set of possible outputs of Algorithm A the following inequality stands:

$$\Pr[A(\mathcal{D})=O]\leqslant e^{\varepsilon}\times\Pr[A(\mathcal{D}')=O] \tag{1}$$

where ε represents the privacy budget. It is important to also note that the probability $\Pr[\,]$ is derived from the internal randomness of algorithm A. Clearly, increasing the privacy budget ε results in a stronger intensity of privacy protection.

Definition 2 Global sensitivity[42]: For any function f, the global sensitivity of function f is defined as

$$\triangle f=\max\left\|f(\mathcal{D})-f(D')\right\| \tag{2}$$

Definition 3 Laplace noise mechanism[42]: Given a data stream $\mathcal{D}$ and a publicly known query function $f:\mathcal{D}\rightarrow\mathbf{R}^d$, Algorithm A satisfies the ε-differential privacy protection requirement if the following equality is true:

$$A(\mathcal{D})=f(\mathcal{D})+\left\langle \text{Lap}_1\left(\frac{\triangle f}{\varepsilon}\right), \text{Lap}_2\left(\frac{\triangle f}{\varepsilon}\right), \ldots, \text{Lap}_d\left(\frac{\triangle f}{\varepsilon}\right)\right\rangle \tag{3}$$

where d represents the query dimension of f.

3.2 Definition of weighted sliding window model and its preprocessing

This section begins by introducing the concept of the weighted sliding window model, as described in Lee et al.'s work in 2014[23]. The weighted sliding window model extends the conventional sliding window approach by incorporating user-specific weighted factors. These factors are assigned to individual data elements, allowing for differentiation based on their relative importance. By introducing these weights, the model offers a finer granularity in capturing the significance of data within the sliding window*.

Expanding on the foundations of the weighted sliding window model, our proposed technique focuses on converting a data stream into multiple binary data streams. This conversion process involves transforming the original data stream into several binary streams, each representing a specific aspect or characteristic of the data. By decomposing the data stream in this manner, we gain the ability to independently analyze and process different aspects of the data. This approach opens up new opportunities for tailored analysis and the implementation of privacy-preserving mechanisms.

In other words, by assigning weights to individual data elements within the sliding window, the weighted sliding window model allows for a more nuanced understanding of their importance. Based on this model, we propose a technique that takes this concept further by converting a data stream into multiple binary data streams. This conversion involves breaking down the original data stream into several binary streams, with each stream representing a distinct aspect or characteristic of the data. By doing so, we can analyze and process different aspects of the data independently, enabling customized analysis and the integration of privacy-preserving measures.

*The difference between the original sliding window model and the weighted sliding window model leads to the limitation of existing methods designed for the original sliding window model. These methods usually cannot handle the weighted sliding window model where each element has a different weight.

Definition 4 Sliding window: Given a data stream $\mathcal{D}=\langle e_1, e_2, e_3, \ldots, e_n, \ldots\rangle$ and a window size w, the sliding window D_w^t at current timestamp t contains the latest w elements seen so far. In other words,

$$D_w^t=\begin{cases} e_{t-w+1}, e_{t-w+2}, \ldots, e_t, & \text{if } t\geqslant w;\\ e_1, e_2, \ldots, e_t, & \text{if } t<w \end{cases} \tag{4}$$

Traditionally, the sliding window model considers the latest w elements equally important, assigning them equal weights. This equal weighting assumption limits the ability to differentiate the importance levels among the w elements. To address this limitation, we introduce a weighted factor denoted as γ, which enables the assignment of larger weights to newer elements compared to older ones.

For the sake of clarity in the subsequent discussion, we represent the importance of an element e belonging to the sliding window D_w^t as $I(e)$. This representation allows us to quantify and compare the importance levels of individual elements within the sliding window. By incorporating the weighted factor γ, we can adjust the weights assigned to elements based on their relative positions within the window, facilitating a more flexible and fine-grained modelling of data importance.

Definition 5 Weighted sliding window[23]: Given a data stream $\mathcal{D}=\langle e_1, e_2, e_3, \ldots, e_n, \ldots\rangle$, a window size w, a user-specific weighted factor γ, and a time instance $t>0$, the weighted sliding window at time instance $t>0$ is constituted two the following two parts:

(1) the sliding window at time instance t: D_w^t;

(2) a weight array $\left\langle\gamma^{w-1}, \gamma^{w-2}, \ldots, \gamma^0\right\rangle$ which assigns each element $e_t\in D_w^t$ is associated with a different weight $I(e_t)=\gamma^{t-i}$.

The schematic diagram of the weighted sliding window is shown in Fig. 2. From the definition of the weighted sliding window, it is clear that the newest element $e^t\in D_w^t$ has the largest weight of 1, the second newest element $e^{t-1}\in D_w^t$ has the second largest weight of γ, and so on. For example, suppose that $\gamma=0.95$, the window size $w=3$, and the current time instance is $t=4$, the weighted sliding window contains two parts: (part-a) $D_w^t=\langle e_2, e_3, e_4\rangle$, and (part-b) $\left\langle\gamma^2, \gamma^1, \gamma^0\right\rangle$ which assigns e_2 a weight of γ^2, e_3 a weight of γ^1, and e_4 a weight of γ^0.

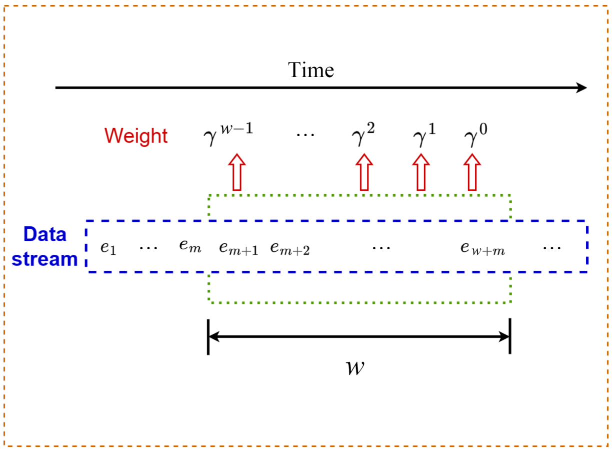

Time
Weight
γ^{w-1} ⋯ γ^2 γ^1 γ^0
Data stream
e_1 ⋯ e_m e_{m+1} e_{m+2} ⋯ e_{w+m} ⋯
w

Fig. 2 Schematic diagram of a weighted sliding window.

Definition 6 Transforming a data stream into multiple binary data streams according to the pre-determined intervals: Given a data stream $\mathcal{D}$, and M intervals: $[l_1, r_1], [l_2, r_2], \ldots, [l_M, r_M]$. Before processing $\mathcal{D}$, we always transform a numerical data stream:$\mathcal{D} = \langle e_1, e_2, \ldots, e_n, \ldots\rangle$ into M separate binary data streams $\langle \mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_M\rangle$ that corresponds to the M intervals. Specifically, the i-th binary data stream $\mathcal{D}_i$ corresponds to the i-th interval $[l_i, r_i]$, and is defined as

$$\mathcal{D}_i = \left\langle I_{e_1\in[l_i, r_i]}, I_{e_2\in[l_i, r_i]}, \ldots, I_{e_n\in[l_i, r_i]}, \ldots \right\rangle \tag{5}$$

where $I_{e_n\in[l_i, r_i]}$ denotes the value belonging to the interval $[l_i, r_i]$. $\forall\, n \geqslant 1$, $I_{e_n\in[l_i, r_i]} = 1$, if element $e_n \in \mathcal{D}$ does belong to $[l_i, r_i]$; and $I_{e_n\in[l_i, r_i]} = 0$, otherwise. Then, the count of a weighted sliding window in each data stream $\mathcal{D}_i$, $i = 1, 2, \ldots, M$ (each of these M binary data streams), denoted by G_i, $G_i = \sum_{j=t-w+1}^{t} I_{e_j\in[l_i, r_i]} \times \gamma^{t-j}$.

Hereafter, to offer an unambiguous illustration, we use

$$\mathcal{S}_i = \langle x_1, x_2, \ldots, x_n, \ldots\rangle \tag{6}$$

to represent the i-th binary data stream ($i = 1$–M), when designing a novel sketching structure to maintain the counting information of each histogram interval at every time instance.

4 Differential Privacy Data Stream Publishing Algorithms Based on Weighted Sliding Window Model

In this section, we outline the design of the ESHP algorithm. The roadmap for this section is as follows:

(1) Creation of AESketch: We propose a introduce a novel sketch structure (AESketch), which enables the maintenance of counting information for each histogram interval by scanning the data stream only once.

(2) Development of the ESHP Algorithm: Building upon AESketch, we present the ESHP algorithm, specifically designed to generate publishable histograms over weighted sliding windows.

(3) Advantages of the ESHP method: It boasts advantages, such as reduced computational overhead and preserved data utility, making it a highly efficient and practical approach for various data processing and analysis tasks.

4.1 AEsketch

In this section, we begin by introducing several fundamental symbols that will be used throughout the paper. We then proceed to provide a detailed analysis of the structure of the sketch.

- w represents the size of a sliding window. It determines the number of most recent elements considered within the window.
- θ represents the weighted approximation error threshold. It is a parameter that controls the acceptable level of error in the weighted sliding window model.
- $\mathcal{S}_i\ [t-w+1, t]$ denotes a sliding window of the i-th binary data stream. It encompasses the w most recent elements of the stream, starting from timestamp $t-w+1$ to timestamp t.
- γ represents the weighted factor. It is a parameter that allows for assigning different weights to individual elements based on their relative importance within the sliding window.
- β represents the approximate error factor. It is a parameter that controls the level of approximation allowed in the estimation of histogram intervals.
- B denotes a two-dimensional array used for storing the counting information of weighted sliding windows. The first one-dimensional sub-array, denoted as $B_1\,(i)$, records the timestamp of the current element in a data stream. The second one-dimensional sub-array, denoted as $B_2\,(i)$, stores the count G_i for the i-th binary data stream $\mathcal{S}_i$.

Table 1 summarizes the important symbols introduced in this section, providing a convenient reference for their meanings and usage throughout the remaining sections of the paper.

Our approach is to create a novel sketching structure AESketch, which is suitable for the weighted sliding window model and can then be used to publish

Table 1 Important symbols in this paper.

Notation	Description
w	Size of a weighted sliding window
$\mathcal{S}_i$	The i-th binary data stream
M	Number of intervals for a histogram
γ	Weighted factor
α	Privacy budget allocation ratio
β	Approximate error factor
θ	Weighted approximation error threshold
B	Two-dimensional array
k	Length of array B
$\hat{G}$	Interval approximate count in current timestamp
$\tilde{G}$	Interval noise value in current timestamp

histograms continuously.

Our approach is based on the following core idea. Accurately tracking the count G_i of each binary stream within a sliding window would require storing all the elements, resulting in a significant computational burden and prolonged computation time. To address this challenge, we propose the design of AESketch, a sketch structure that maintains an approximate estimate $\hat{G}_i$ of G_i. The key criterion is that the difference between $\hat{G}_i$ and G_i should always be less than the user-defined error threshold θ, i.e., $|\hat{G}_i - G_i| \leqslant \theta$ at all times.

Although using AESketch leads to a decrease in accuracy, we believe it is a justifiable trade-off for two main reasons. First, it aligns with the fundamental concept of differential privacy, as all methods for publishing differential privacy data involve introducing random noise to protect privacy. Second, the estimated value $\hat{G}_i$ derived from AESketch can be viewed as G_i augmented by a random noise component. Consequently, the discrepancy between $\hat{G}_i$ and G_i always falls within the error threshold defined by the user (i.e., $|\hat{G}_i - G_i| \leqslant \theta$).

By striking a balance between accuracy and privacy, we consider this compromise to be a reasonable approach for safeguarding the privacy of data while efficiently maintaining approximate counts of binary streams. In the subsequent sections, we will delve into the details of AESketch, conduct a comprehensive analysis, and present experimental results to validate its effectiveness.

The detailed steps of how AESketch processes a data stream $\mathcal{S}_i$ and then calculates the estimated value $\hat{G}_i$ for $\mathcal{S}_i$ are presented in Algorithm 1. Let us now illustrate Algorithm 1 with an example. Suppose that $\mathcal{S}_i = \langle 1, 1, 1, 1\rangle$, weighted factor $\gamma = 0.95$, the window size $w = 3$, approximate error factor $\beta = 0.5$, and the current time instance $t = 4$. With a simple calculation, we can observe that Algorithm 1 will obtain $\theta = 1.4262$. Next, We can calculate the interval count for the current timestamp t by Line 14 of Algorithm 1 (the first case is that the data stream has expired elements). The detailed values of the related parameter at each time instance in this example are given in Table 2. With this, we can obtain: $\hat{G}_i = 0.8525 \times (0.95)^{4-2} + 0.8099 \times (0.95)^{3-2} + 0.7694 \times (0.95)^0 + 1 - 0.5 \times 0.8525 \times 0.95^{4-2} = 2.9235$. It is also clear that the true count value at the current moment is $G_i = 2.8525$. Therefore, we know that $|G_i - \hat{G}_i| = 0.071$ is not more than $\theta/2 = 0.7131$. In Theorem 4 in the following, we shall give a formal proof of this fact for general cases.

Theorem 1 The maximum value of G_i in the weighted sliding window is $\dfrac{1-\gamma^w}{1-\gamma}$.

Proof From the schematic diagram of the weighted sliding window in Fig. 2, it can be observed that when each position in the sliding window is set to 1, the maximum count is w. So the maximum value of G_i is equal to $\dfrac{1-\gamma^w}{1-\gamma}$. ■

Theorem 2 The number of blocks k of the array B used by Algorithm 1 does not exceed $1/\beta$.

Proof First, we have two facts as follows:

(1) By the properties of a weighted sliding window, the maximum count value is

$$\frac{1-\gamma^w}{1-\gamma} \tag{7}$$

(2) Algorithm 1 uses the block threshold in the following:

$$\left(\frac{1-\gamma^w}{1-\gamma}\right) \times \beta \tag{8}$$

Based on the two cases Formulas (7) and (8), the maximum number of blocks used in array B is

$$k = \frac{\dfrac{1-\gamma^w}{1-\gamma}}{\left(\dfrac{1-\gamma^w}{1-\gamma}\right) \times \beta} = \frac{1}{\beta} \tag{9}$$

Theorem 3 For Algorithm 1, there are no two data

Table 2 Parameter status at each time instance.

Parameter	t			
	1	2	3	4
$\mathcal{S}_i$	1	1	1	1
y	1	1	1	1
$B_1(i)$	2	3	4	0
$B_2(i)$	0.8525	0.8098	0.7694	0

blocks that expire at the same time.

Proof To illustrate this fact, let us employ a proof-by-contradiction approach. We start by assuming the existence of two blocks within the weighted sliding window, and both of these blocks contain elements that have already expired. According to the weighted sliding window model described in Algorithm 1, any new block inserted into the sliding window must be positioned before an older block. If the newly inserted block includes an expired element, it logically follows that all elements within the older block must have also expired. This conclusion arises because the older block was inserted into the sliding window prior to the new block, implying that the expiration time of its elements must be earlier. Therefore, we can infer that it is impossible to have two data blocks within the weighted sliding window model that simultaneously expire. This outcome contradicts the initial assumption we made.

By employing a proof by contradiction, we have successfully demonstrated that, within the framework of the weighted sliding window model as described in Algorithm 1, the occurrence of two data blocks expiring simultaneously is impossible. ■

Algorithm 1 How AESketch processes binary data stream

```
Input: $\mathcal{S}_i$: $i$-th binary data streams; $x_t$: element at the current time instance $t$ in the $\mathcal{S}_i$, $\gamma$: weighted factors, $\beta$: approximate error factor, $w$: windows size, $\theta$: weighted approximation error threshold, $B$: two-dimensional array, $k$: length of array $B$.
Output: approximate count $\hat{G}_i$ belonging to the $i$-th interval at the current timestamp $t$.
1: Calculate the threshold of the approximate error of the weighted
   sliding window: $\theta = \left(\frac{1-\gamma^w}{1-\gamma}\right)\times\beta$;
2: Calculate the weighted count at the current timestamp $t$:
   $y = y\times\gamma + x_t$;
3: To assess whether the weighted sliding window value at the
   current time instance ($t$) surpasses the threshold value of the
   weighted sliding window:
4: if $y\times\gamma + x_t \leqslant \theta$ then
5:     $y = y\times\gamma + x_t$;
6: else
7:     $B_1(i) = t$; $B_2(i) = y$; $i = i+1$; $y = x_t$;
8: end if
9: Determine whether the array $S$ has any expiration data element
   at the current timestamp $t$:
10: if ($B_1(i) < t-w+1$) and $B_1(i) > t$) then
11:     $B_1(i) = 0$; $B_2(i) = 0$;
12: end if
13: Calculate the interval counting information for the current
    timestamp $t$ by the following 2 steps:
14: Step 1: It generates expired elements:
        $\hat{G}_i = \text{sum}\,(B_2(i)\times\gamma^{t-B_1(i)}) + y -$
            $0.5\times\text{old}\,(B_2(i))\times(\gamma)^{t-\text{old}}(B_1(i))$;
        //remove any bias from the estimation by subtracting half
        of the oldest block
15: Step 2: There is no expired element in the data stream:
        $\hat{G}_i = \text{sum}\,(B_2(i)\times\gamma^{t-B_1(i)}) + y$.
```

Theorem 4 For each interval in the histogram, the result of Algorithm 1 is $|G_i - \hat{G}_i| \leqslant \theta/2$.

Proof Please recall the equation $\theta = \frac{1-\gamma^w}{1-\gamma}\times\beta$ as presented in Algorithm 1. There are two possible situations for the current sliding window:

(1) When there is no expired element:

By Line 15 of Algorithm 1, we can know

$$|G_i - \hat{G}_i| = 0 \tag{10}$$

Specifically, when there is no expired element, the array B records only the elements in the current sliding window.

(2) When there are expired elements:

By Theorem 3, we know the count in the most recent block is 1, and there is only one block containing expired elements. By Line 14 of Algorithm 1, $\hat{G}_i$ will minus half of the count in the oldest block (see $\hat{G}_i = \text{sum}\,(B_2(i)\times\gamma^{t-B_1(i)}) + y - 0.5\times\text{old}\,(B_2(i))\times(\gamma)^{t-\text{old}\,(B_1(i))}$). Moreover, since only the oldest block contains expired elements, we can obtain

$$|G_i - \hat{G}_i| \leqslant \theta - 1 - \frac{1}{2}(\theta-1) = \frac{\theta}{2} - \frac{1}{2} \leqslant \frac{\theta}{2} \tag{11}$$

Based on the two cases (Eq. (10) and Formula (11)), we have $|G_i - \hat{G}_i| \leqslant \theta/2$.

Theorem 5 The AESketch algorithm has a computational cost of $O(1/\beta)$ and a storage cost of $O(1/\beta)$.

Proof From Algorithm 1, we can observe: the main computational cost and storage cost are determined by the size of array B. So, the computational cost and storage cost is $O(1/\beta)$.

It is worth noting that the existing algorithms that cache the entire window have a storage cost of $O(w)$ and a computational cost of $O(w)$, which are larger than that of the AESketch algorithm since β is usually less than $1/w$.

4.2 ESHP

AESketch provides us with M estimated counts:

$\left\langle \hat{G_1}, \hat{G_2}, \ldots, \hat{G_M} \right\rangle$ for the M binary data streams: $\langle \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_M \rangle$. These M values are estimated values of the M true counts: $G_1, G_2, \ldots, G_M$. However, they are not good enough to prevent the leak of private information contained in the original data stream $\mathcal{S}$. Considering this fact, if we need to further protect $\hat{G_1}, \hat{G_2}, \ldots, \hat{G_M}$ by adding appropriate noise random variables to them.

In order to ensure the competitive availability of the published data, we combine AESkech with differential privacy, and then propose an efficient streaming histogram publishing algorithm, represented by ESHP algorithm.

The ESHP algorithm follows a sequential process to accomplish its objectives.

Initially, it generates a sketching structure known as AESketch to approximate the distribution information of data within the weighted sliding window at the subsequent timestamp.

The subsequent step entails proposing an adaptive selective publishing mechanism that intelligently determines an appropriate value for each interval. This is achieved by comparing the disparity between the estimated value and the noise value. The algorithm assesses this difference and decides whether to publish the estimated value or the noisy value. The detailed implementation is available in Algorithm 2, particularly in Lines 2–6.

In the third step, the ESHP algorithm incorporates Laplace noise into the count value of each interval. This ensures privacy protection while preserving statistical accuracy. Moreover, the algorithm constructs a publishable histogram over the weighted sliding windows using a greedy grouping strategy. The implementation details of this strategy can be observed in Algorithm 2, specifically in Lines 7–21.

The specific implementation of the greedy grouping strategy involves the following actions:

(1) Sorting the optimal sets of noise values and estimation values based on their error relationships. This facilitates grouping together similar or closely related data.

(2) If the current error incurred by not grouping a particular interval is smaller than the error obtained by grouping it, the algorithm decides to place this interval in a separate group. Conversely, if grouping the interval results in a smaller error, it is placed within the current group.

By employing this greedy grouping strategy, the ESHP algorithm optimizes the grouping of intervals to minimize overall overall errors and enhance the accuracy of the published histogram.

These steps collectively empower the ESHP algorithm to effectively estimate distribution information, selectively publish values, incorporate privacy-preserving noise, and construct a publishable histogram over weighted sliding windows.

Note: The third step of this approach ensures that each interval is grouped in a manner that minimizes errors and enhances data usability.

Overall, the ESHP algorithm is a three-step process that utilizes a sketching structure, an adaptive selective publishing mechanism, and Laplace noise to construct a publishable histogram over weighted sliding windows. These steps are designed to ensure that data privacy is maintained while still providing useful insights into the distribution of data.

Algorithm 2 ESHP algorithm

```
Input: $\mathcal{D}$: data stream, $t$: current timestamp, $\varepsilon$: privacy budget, $\alpha$: privacy budget allocation ratio.
Output: noise histogram $\tilde{G}_t$ at the current timestamp.
1: Calculate all interval count $\hat{G}_t$ for a histogram by AESketch
   in the current timestamp $t$;
2: Allocate privacy budget: $\varepsilon_1 = \alpha\varepsilon$ and $\varepsilon_2 = (1-\alpha)\varepsilon$;
3: Determine whether to add Laplace noise to the current data:
4:    if $|\hat{G}_t - G_t| \geqslant \frac{\sqrt{2}\theta}{\varepsilon_1}$ then
5:        $\hat{G}_t = G_t + \mathrm{Lap}\left(\frac{\theta}{\varepsilon_1}\right)$;
6:    end if
7: Use grouping method based on greedy clustering:
8:    $\hat{G}_t = \mathrm{sort}\,(\hat{G}_t)$;
9:    $C = \mathrm{Clustering}\,(\hat{G}_t)$;
10: Calculate the mean of the current histogram:
11:    for $C_i \in C$ do
12:        $\tilde{C}_i = \sum_{\hat{H} \in C_i} \frac{H_j}{|C_i|}$;
13:    end for
14: Calculate the noise value of the current histogram:
15:    for every $\hat{H}_j \in \hat{G}_t$ do
16:        $\tilde{H}_j = \tilde{C}_i + \mathrm{Lap}\left(\frac{\theta}{\varepsilon_2}\right)/|C_i|$;
17:    end for
18: Non-negative constraint on the count of each histogram
    interval:
19:    if $\hat{G}_t \leqslant 0$ then
20:        $\hat{G}_t = 0$;
21:    end if
```

4.3 Advantages of the ESHP method

The ESHP method provides several significant advantages over existing methods.

(1) The ESHP method utilizes the innovative AESketch structure to reduce computational overhead significantly. This allows for the real-time release of histograms without the need to buffer all elements within each sliding window, making it suitable for processing data streams in real-time. It also minimizes storage costs and alleviates the burden of continuously releasing histograms through weighted sliding windows.

(2) The ESHP method can maintain comparable data utility to existing methods. The mechanism leverages the difference between approximate statistical information for each histogram interval and the noise value to select better counting information. It employs a greedy grouping approach to count all intervals within the weighted sliding window, ensuring that the data utility for querying data remains similar in most cases.

In conclusion, the ESHP method excels in the continuous real-time publication of histograms, ensuring differential privacy, accommodating data streams with different weights and preserving data utility. It serves as an efficient and privacy-preserving approach to histogram generation that finds applicability in various scenarios involving data processing.

5 Theoretical Analysis

In this section, we analyze the space complexity and time complexity of the ESHP algorithm and verify the privacy of the ESHP algorithm.

Theorem 6 ESHP algorithm requires $k\times\log(w)\times M$ bits of memory.

Proof The space overhead of the ESHP algorithm is mainly determined by the AESketch data structure. From Theorem 2, we know that the binary data stream $\mathcal{S}_i$ requires $k\times\log_2(w)$ bits. Specifically, the array B in the AESketch data structure should record the timestamp of each element from the data stream, and we need $\log_2(w)$ bits to accurately represent the w different timestamps of the w elements contained in a sliding window. For M intervals, the size of the memory space we need is $k\times\log_2(w)\times M$ bits. Finally, since there are M intervals and each interval requires one array B, we can conclude that the ESHP method requires a total memory $k\times\log_2(w)\times M$ bits. ■

Theorem 7 Assuming that the number of packets in the ESHP algorithm is M, the time cost of the ESHP algorithm is $O(M+k)$.

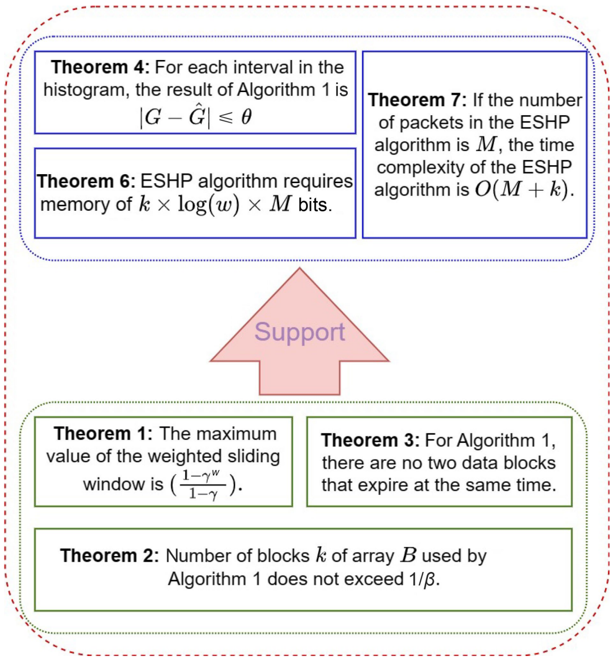
Proof The ESHP method utilizes a data structure called AESketch to store each element in the current sliding window and then employs it to generate a differential private histogram. Specifically, when a new element e_t arrives, the ESHP method identifies to which e_t belongs and stores crucial information about e_t in a block containing $O(k)$ bits of AESketch data structure (see Theorem 2). Subsequently, the ESHP method generates a differential private histogram by scanning the array B, which contains M blocks (each block corresponds to an interval). Here, M represents the number of intervals, and the computational complexity of the ESHP method is $O(M+k)$.

Please note that Theorems 1–3 lay the foundation for the theorems proved in this section, and we show this relationship in Fig. 3. ■

Theorem 8 The proposed ESHP algorithm satisfies ε-differential privacy.

Proof For the ESHP algorithm, it is composed of two combined algorithms. The first part is the adaptive selection algorithm based on AESketch, and the second part is the noisy grouping algorithm based on the greedy approach.

Our proof involves two facts as follows:



Theorem 4: For each interval in the histogram, the result of Algorithm 1 is $|G-\hat{G}|\leqslant\theta$
Theorem 6: ESHP algorithm requires memory of $k\times\log(w)\times M$ bits.
Theorem 7: If the number of packets in the ESHP algorithm is M, the time complexity of the ESHP algorithm is $O(M+k)$.
Support
Theorem 1: The maximum value of the weighted sliding window is $(\frac{1-\gamma^w}{1-\gamma})$.
Theorem 3: For Algorithm 1, there are no two data blocks that expire at the same time.
Theorem 2: Number of blocks k of array B used by Algorithm 1 does not exceed $1/\beta$.

Fig. 3 Diagram of the relationship between theorems.

(1) Adaptive selection algorithm based on AESketch: The proposed ESHP algorithm selects an appropriate value by comparing the difference between the estimated value and the noise value in the adaptive selection section (the adaptive selection section satisfies ε_1-differential privacy).

(2) Noisy grouping algorithm based on greedy approach: The ESHP algorithm uses a greedy grouping strategy to add Laplace noise into the grouping section (thus, the grouping section satisfies ε_2-differential privacy).

With these two facts and $\varepsilon = \varepsilon_1 + \varepsilon_2$, we can conclude that the ESHP algorithm satisfies ε-differential privacy according to the combined properties of differential privacy[49].

6 Experiment

In this section, we present the experiment setting and the used datasets. We compare the proposed method: the ESHP algorithm with three typical data publishing for data stream methods: APB algorithm[33], RTP_DMM algorithm[39], and FAST algorithm[41], and the most recent negative survey algorithm proposed in Ref. [45] to process time-series data. For brevity, we denote the negative survey algorithm in Ref. [45] by NS in the following.

6.1 Experiment setting

The experimental hardware setup comprises an AMD Ryzen R7 5800X3D 3.4 GHz processor with eight cores, 16 GB of RAM, and 750 GB of hard disk storage. The software environment used for the experiment is Windows 10 operating system.

All algorithms utilized in the experiment have been implemented in the Matlab programming language.

The experimental evaluation is conducted using two real datasets.

The first dataset consists of traffic accident information extracted from the UK car accident dataset[50]. Each stream element in the dataset contains records such as the age of injury and death. The age ranges are set to [0, 20], [20, 40], [40, 60], [60, 80], and [80, 100]†.

The second dataset consists of records of the number of passengers in a car extracted from the NYC yellow taxi trip dataset[51]. The number of passengers ranges from 1 to 6‡.

To evaluate the effectiveness of the proposed algorithms, we have implemented five different algorithms in the Matlab programming language. We use Mean Square Error (MSE) to measure the usability and accuracy of the published data. A smaller value of MSE indicates better usability of data. We experiment with three different privacy budgets of 0.5, 1, and 1.5. To ensure reproducibility and eliminate randomness, each evaluation result is an average of 30 independent runs.

MSE is calculated as follows:

$$\mathrm{MSE}\,(Q) = \frac{\sum\limits_{H \in G} (\tilde{H} - H)^2}{|Q|} \tag{12}$$

where $|Q|$ is the number of queries, $\tilde{H}$ represents the statistical result of adding noise to an interval, and H represents the statistical result of an interval.

We conduct an experiment to evaluate the impact of different methods for allocating the privacy budget on the accuracy of our algorithm, utilizing the NYC yellow taxi trip dataset. Specifically, we vary the allocation ratio α of the privacy budget from 0.1 to 0.9, and assess the accuracy of our algorithm using MSE, as illustrated in Fig. 4.

From Fig. 4, we can observe that the MSE of the published data decreases when the privacy budget allocation parameter α increases. This observation can be attributed to two factors. Firstly, our ESHP algorithm effectively reduces the overall algorithmic error by approximating the difference between the statistical value and the noise value, and selecting an appropriate value of α through an approximate estimation method. This selection process mitigates the impact of Laplace noise on the data. Secondly, the ESHP algorithm incorporates Laplace noise using the privacy budget. With an increasing privacy budget

†The UK Traffic Accidents dataset (2014): This dataset is obtained from pertinent governmental organizations in the United Kingdom and encompasses comprehensive road safety information regarding personal injury traffic accidents that occurred in the UK between 2005 and 2014. It includes detailed data, such as accident date, time, location, severity of injuries, weather conditions, and types of vehicles involved. These data hold substantial value in investigating the incidence of traffic accidents, analyzing their causative factors, and formulating effective traffic safety policies.

‡The New York City Yellow Taxi Trip dataset (2019): This dataset is acquired from pertinent governmental agencies in New York City and provides extensive information about yellow taxi trips undertaken within the city during the year 2019. It comprises various fields, such as the trip date and time, pickup and drop-off locations, trip distance, itemized fares, rate types, payment methods, and passenger count as reported by the driver. This dataset serves as a significant resource for comprehending the functioning of taxi services in New York City, analyzing passenger behavior, and studying traffic congestion, among other aspects.

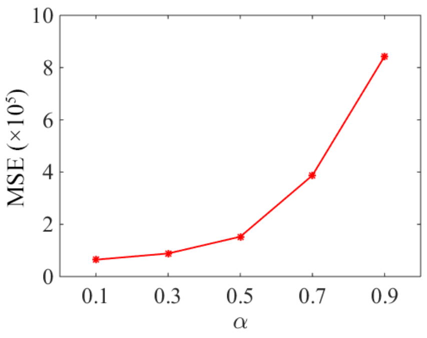


Fig. 4 MSE on NYC yellow taxi trip dataset under different privacy budget allocations using ESHP algorithm.

allocation ratio α, the noise value associated with histogram statistics grows, resulting in an increase in the total error of the ESHP algorithm.

Based on the aforementioned findings, we have set the privacy budget allocation ratio α in our algorithm to 0.1 for all subsequent experiments conducted in this paper. This decision is supported by our results, which indicate that this ratio strikes a favourable balance between privacy protection and the accuracy of the published data.

6.2 Comparisons for average running time

In our experiment, we aimed to analyze the influence of different window sizes on data usability by comparing various algorithms. To ensure consistency, we set the weight factor γ to 0.91 and considered sliding window sizes of 2000, 4000, 6000, 8000, and 10 000. We evaluat the algorithms' values of average running time using the UK car accident dataset and the NYC yellow taxi trip dataset, and the experimental results are presented in Figs. 5 and 6.

The comparison of five algorithms on their values of average running time under different window sizes reveals some interesting findings. For the FAST, APB, and RTP_DMM algorithms, the total execution time increases as the sliding window size increases. On the other hand, the NS algorithm's execution time depends on the sample set size rather than the window size.

For example, with a window size of 6000 (w = 6000) and ε = 1 on the UK car accident dataset, the average

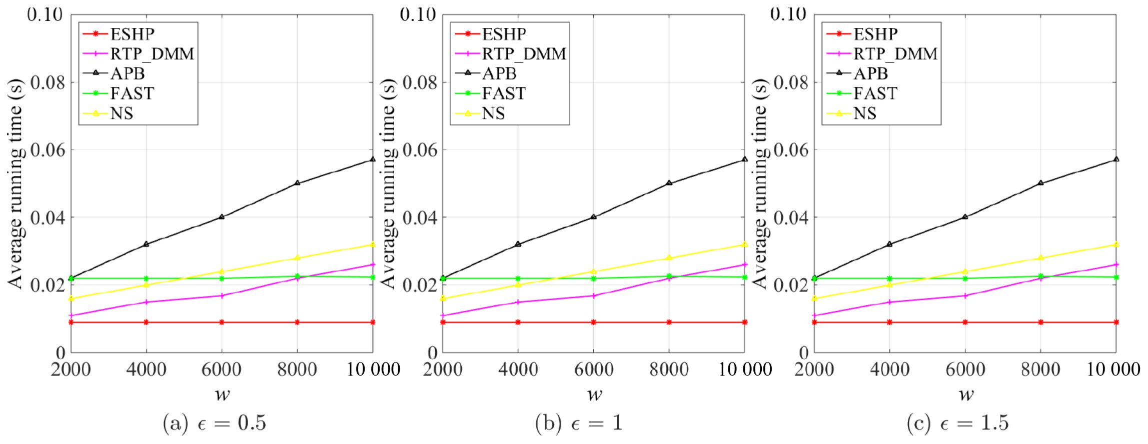


Fig. 5 Average running time under different sliding window sizes (UK car accident dataset).

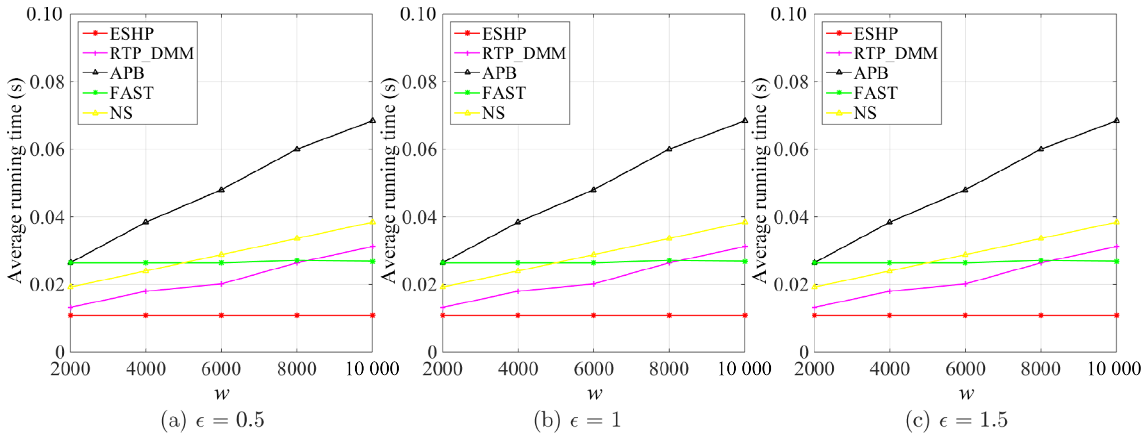


Fig. 6 Average running time under different sliding window sizes (NYC yellow taxi trip dataset).

values of running time of ESHP, RTP_DMM, APB, FAST, and NS algorithms are approximately 9×10^{-3} s, 1.68×10^{-2} s, 4×10^{-2} s, 2.2×10^{-2} s, and 2.4×10^{-2} s, respectively. Similarly, with the same parameters on the NYC yellow taxi trip dataset, the average running values of time of ESHP, RTP_DMM, APB, FAST, and NS algorithms are approximately 1.08×10^{-2} s, 2.02×10^{-2} s, 4.80×10^{-2} s, 2.64×10^{-2} s, and 2.88×10^{-2} s, respectively.

In contrast, the ESHP algorithm's total execution time is almost unaffected by the window size. The processing time of the ESHP algorithm follows a complexity of $O(k+M)$, where k is the number of bins and M is the number of data points. Although the window size impacts the space complexity of the ESHP algorithm, it does not affect the total execution time significantly. The RTP_DMM algorithm, which employs a binary tree, requires $O(w+M)$ space complexity. The NS algorithm needs to cache the entire sliding window, resulting in a time complexity of $O(w+s+M)$, where s is the sample set size. Both the APB and FAST algorithms spend considerable time caching all data in the window when publishing histograms, resulting in a logarithmic linear relationship with the window size.

Compared to the NS algorithm, our ESHP algorithm reduces the average execution time by 62%. These results highlight the efficiency of the ESHP algorithm, which is especially notable in scenarios where the window size varies.

6.3 Comparisons for memory usage

In this experiment, we aim to analyze the impact of different window sizes on data usability by comparing statistics histograms obtained from each time stamp in the sliding window. To maintain consistency, we set the weighted factor γ to a fixed value of 0.91. The sliding window sizes we considered are 2000, 4000, 6000, 8000, and 10 000.

We evaluate five algorithms, namely ESHP, FAST, RTP_DMM, APB, and NS, using two datasets: the UK accident dataset and the NYC yellow taxi trip dataset. Our focus is on examining the memory consumption of these algorithms across various window sizes. The experimental results, depicted in Figs. 7 and 8, clearly indicate that the values of memory usage of these algorithms increase with larger window sizes.

For instance, when the window size is set to 6000 ($w=6000$) and the value of ε is fixed at 1, the average values of memory usage of ESHP, RTP_DMM, APB, FAST, and NS for the UK car accident dataset are approximately 6×10^{3} bits, 1.9×10^{4} bits, 9×10^{3} bits, 9×10^{3} bits, and 1×10^{4} bits, respectively.

Similarly, for the NYC yellow taxi trip dataset with a window size of 6000 ($w=6000$) and $\varepsilon=1$, the average values of memory usage of ESHP, RTP_DMM, APB, FAST, and NS are approximately 6.7×10^{3} bits, 2.1×10^{4} bits, 1×10^{3} bits, 1×10^{3} bits, and 1.1×10^{4} bits, respectively.

ESHP employs a novel sketch structure that requires $k\times\log_2(w)\times M$ memory space. FAST and APB need to cache all the data within the window, thus requiring $w\times M$ memory space. RTP_DMM utilizes a binary tree structure, which necessitates $w\times\dfrac{1-(1/2)^{\log_2(w+1)-1}}{1-1/2}$ memory space. On the other hand, the NS algorithm requires $(w+s)\times M$ memory space. Comparatively, our algorithm exhibits an average reduction of 76% in memory consumption when compared to the NS algorithm.

Overall, these findings highlight the trade-off between window size and memory usage,

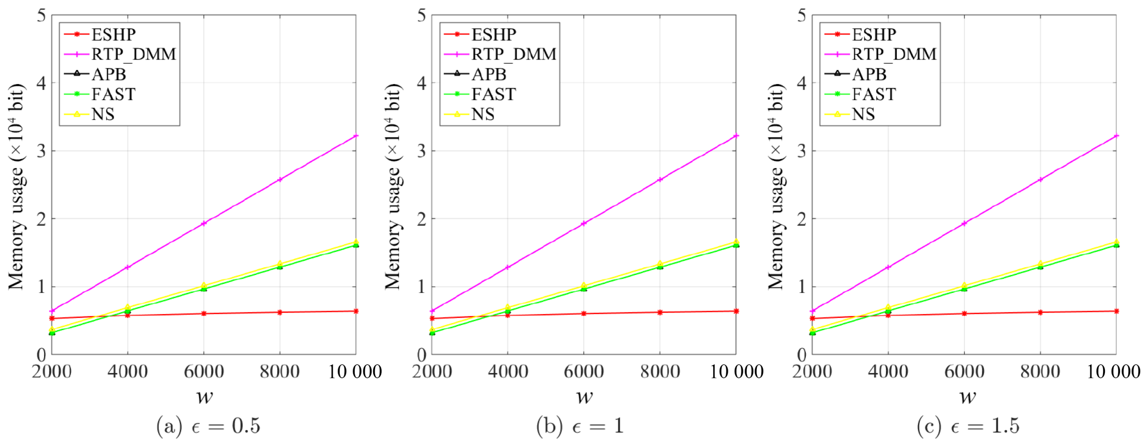


Fig. 7 Memory usage under different window sizes (UK car accident dataset).

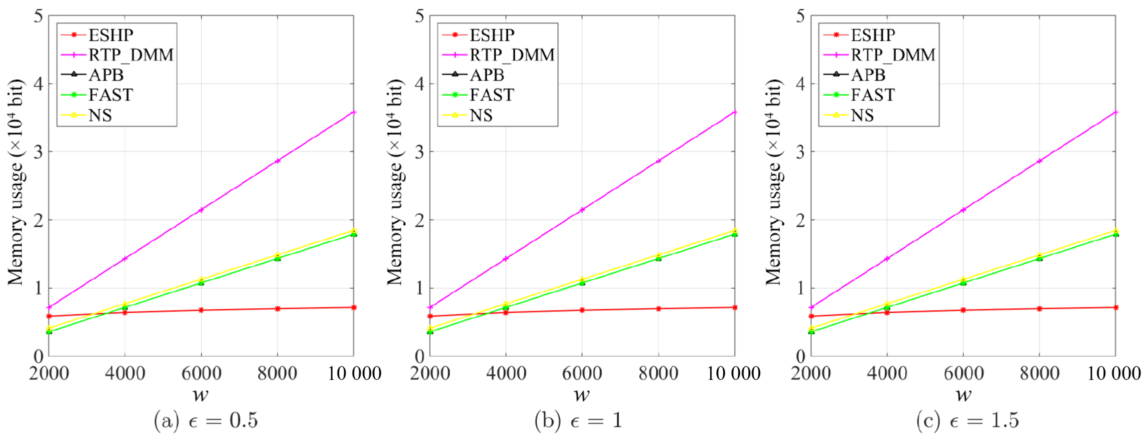


Fig. 8 Memory usage under different window sizes (NYC yellow taxi trip dataset).

demonstrating the influence of different window sizes on the usability of data.

6.4 Comparisons for data accuracy

All four algorithms are compared between these two datasets to evaluate the values of accuracy of the data.

6.4.1 Accuracy under different window sizes

In this experiment, our objective is to analyze the impact of different window sizes on data usability. To maintain consistency, we keep the weighted factor γ fixed at a value of 0.91. The sliding window sizes considered are 2000, 4000, 6000, 8000, and 10 000. The experimental results, presented in Figs. 9 and 10, offer a comparison among these window sizes.

From the analysis of Figs. 9 and 10, it is evident that ESHP exhibits the smallest MSE among the four algorithms as the window size increases. On the other hand, FAST introduces noise to individual elements and thus has a relatively higher MSE. The RTP_DMM algorithm introduces noise to the statistical information within the weighted sliding window, resulting in a smaller MSE compared to FAST. In the case of ESHP, due to the window size approaching a constant value, the total MSE tends to stabilize at a constant value. ESHP employs an adaptive noise-adding method, which avoids significant errors caused by directly adding noise to the sliding window.

Furthermore, when the weighted sliding window is small, the MSE of APB is smaller than that of ESHP. However, as the window size increases, the MSE of ESHP becomes smaller than those of APB, FAST, and RTP_DMM.

For instance, when the window size is set to 6000 and ε is fixed at 1, the MSE of the NS algorithm is approximately 5.9×10^4, while the ESHP algorithm achieves a lower MSE of approximately 6.2×10^4.

It is noteworthy that the NS algorithm satisfies the conditions of differential privacy while ensuring data usability, indicating that both ESHP and NS algorithms are capable of meeting the error range required by users. In most cases, the ESHP algorithm ensures competitive data usability when compared to the other

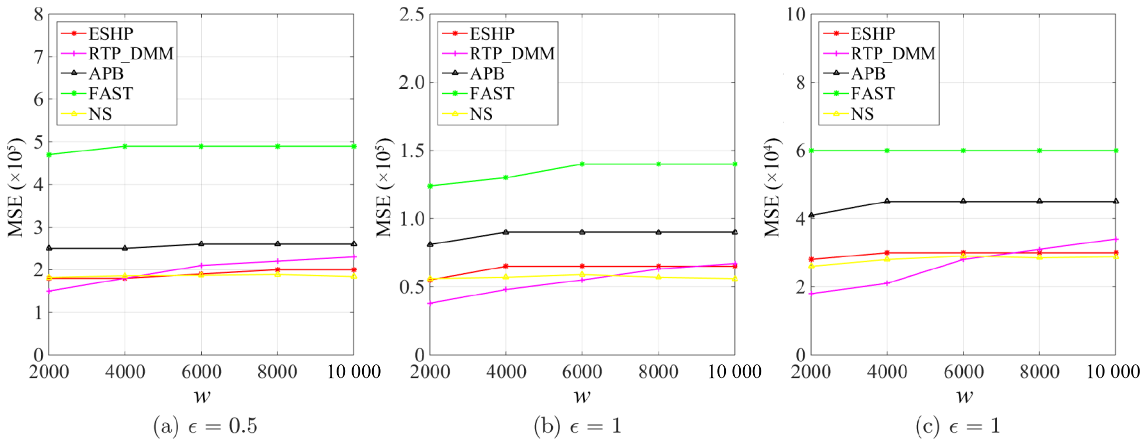


Fig. 9 Accuracy under different window sizes (UK car accident dataset).

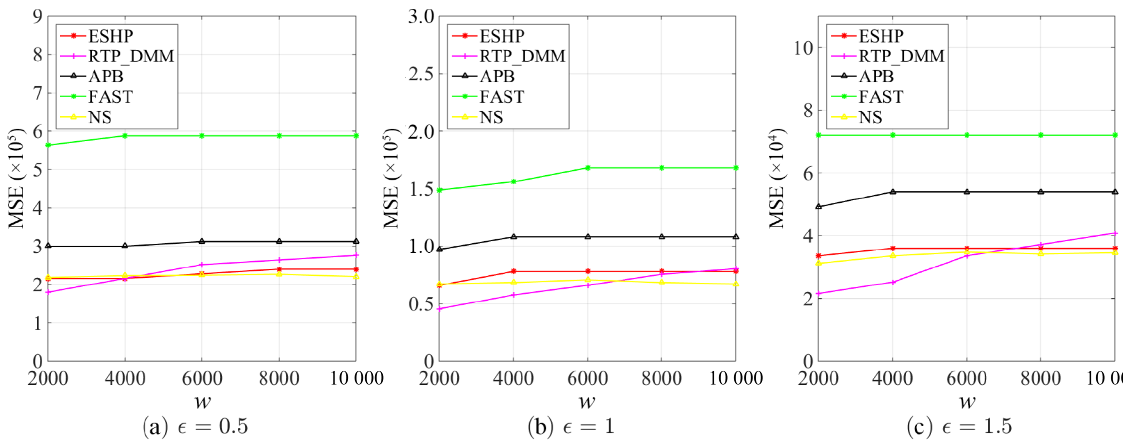


Fig. 10 Accuracy under different window sizes (NYC yellow taxi trip dataset).

algorithms.

6.4.2 Accuracy under different weighted factors

In this experiment, we aim to analyze the influence of different weighted factors on data usability. We consider weighted factors γ of 0.91, 0.93, 0.95, 0.97, and 0.99. The experimental results, depicted in Figs. 11 and 12, illustrate the comparison among these weighted factors.

From the analysis of Figs. 11 and 12, it is evident that the values of MSE of the ESHP and APB algorithms are positively correlated with the weighted factor, whereas the values of MSE of the FAST and RTP_DMM algorithms remain constant, independent of the weighted factor.

When the weighted factor is relatively small, the MSE of the ESHP algorithm is lower than those of FAST, RTP_DMM, and APB algorithms. This can be attributed to the fact that increasing the weighted factor changes the global sensitivity and affects the level of noise added to the data.

For instance, when the weighted factor is set to 0.95 and ε is fixed at 1, the MSE of the NS algorithm is approximately 2×10^4, while the ESHP algorithm achieves a lower MSE of approximately 1.9×10^4.

It is noteworthy that the NS algorithm satisfies the conditions of differential privacy while ensuring data usability, indicating that both the ESHP and NS algorithms can meet the error range required by users. Furthermore, as the weighted factor increases, the ESHP algorithm consistently provides competitive data usability in most cases.

These findings highlight the influence of the weighted factor on data usability, with the MSE of the ESHP and APB algorithms being positively affected by the weighted factor. On the other hand, the FAST and RTP_DMM algorithms maintain a constant MSE regardless of the weighted factor.

6.4.3 Accuracy under different approximate error factors

In this experiment, we aim to analyze the influence of different approximate error factors on data usability while keeping the weighted factor fixed at 0.91. The approximate error factors β considered are 0.1, 0.3, 0.5, 0.7, and 0.9. The experimental results, illustrated in Figs. 13 and 14, provide a comparison among these approximate error factors.

From the analysis of Figs. 13 and 14, it is evident that the values of query error of the ESHP and APB algorithms are positively correlated with the approximate error factor, while the values of MSE of the FAST and RTP_DMM algorithms remain constant and independent of the approximate error factor.

When the approximate error factor is relatively small, the MSE of the ESHP algorithm is significantly lower than that of the APB algorithm. This can be attributed to two reasons. Firstly, reducing the approximate error factor modifies the global sensitivity and the level of Laplacian noise added, resulting in a decrease in MSE. Secondly, the ESHP algorithm utilizes an adaptive selection strategy to avoid directly adding Laplacian noise to the weighted sliding window, leading to a reduction in MSE.

The MSE of the RTP_DMM algorithm remains constant regardless of the approximate error factor. However, as the approximate error factor β increases, the MSE of the ESHP algorithm becomes larger than that of the RTP_DMM algorithm.

For instance, when γ = 0.91, β = 0.5, and ε = 1, the average values of MES of ESHP, RTP_DMM, APB,

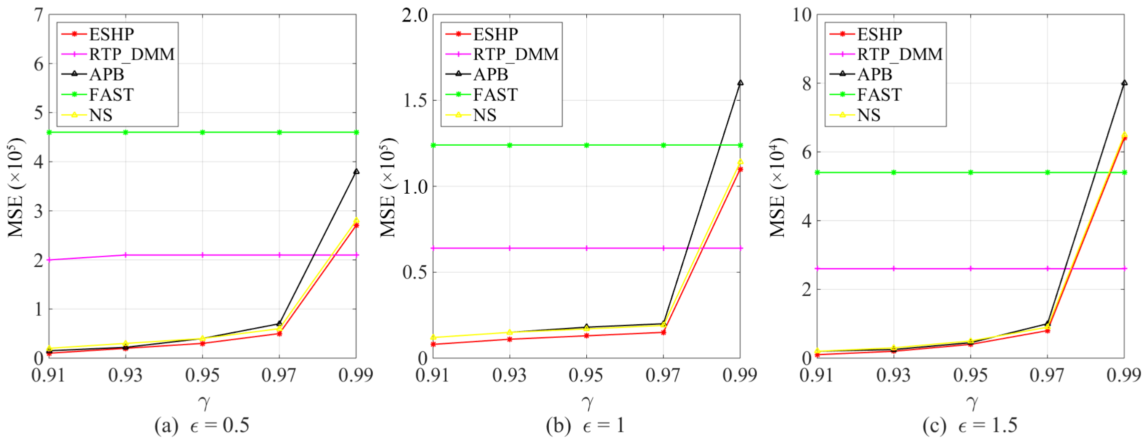


Fig. 11 Accuracy under different weighted factors (UK car accident dataset).

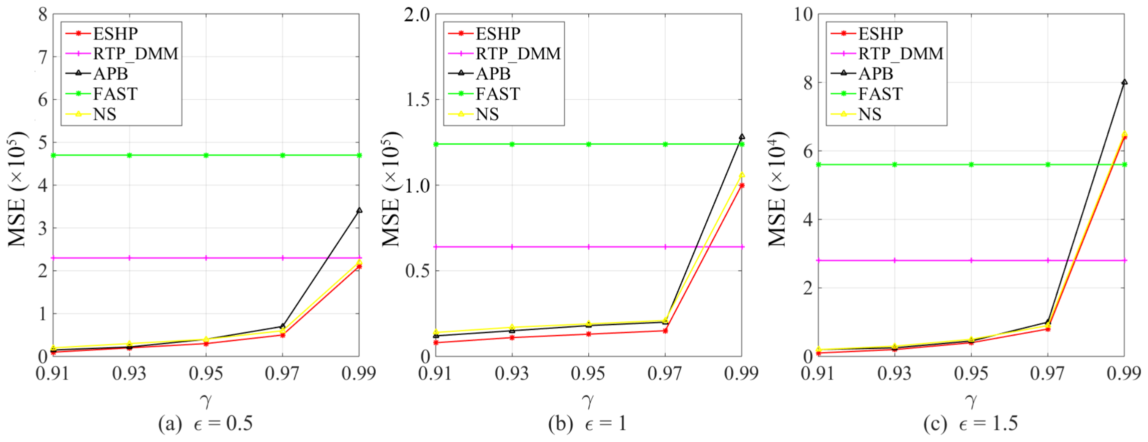


Fig. 12 Accuracy under different weighted factors (NYC yellow taxi trip dataset).

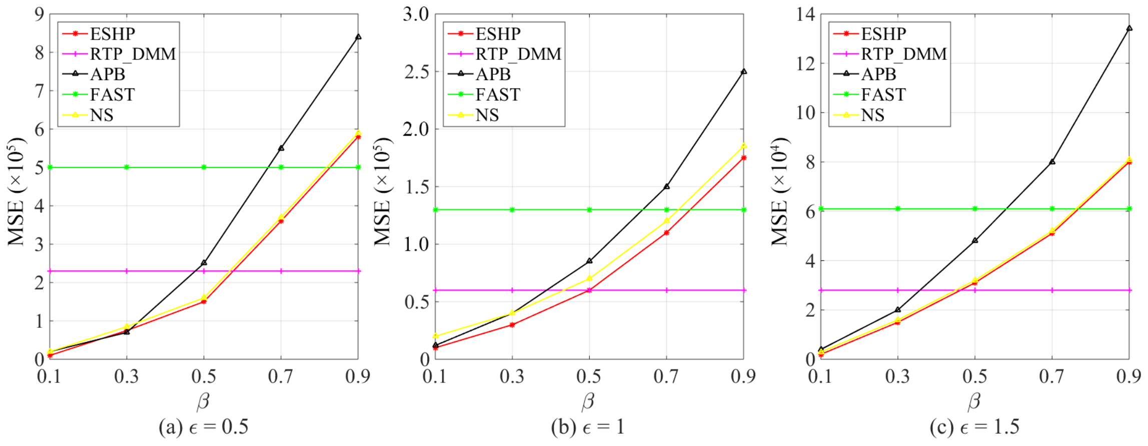


Fig. 13 Accuracy under different approximate error factors (UK car accident dataset)

FAST, and NS for the UK car accident dataset are approximately 6×10^4, 6×10^4, 8.5×10^4, 1.3×10^5, and 7×10^4, respectively. Similarly, for the NYC yellow taxi trip dataset with γ = 0.91, β = 0.5, and ε = 1, the average values of MES of ESHP, RTP_DMM, APB, FAST, and NS are approximately 7.2×10^4, 7.2×10^4, 1.02×10^5, 1.56×10^5, and 8.4×10^4, respectively.

It is noteworthy that the NS algorithm satisfies the conditions of differential privacy while ensuring data usability, indicating that both the ESHP and NS algorithms can meet the error range required by users. However, Figs. 13 and 14 clearly demonstrate that the ESHP algorithm consistently provides competitive usability of published data across a wide range of

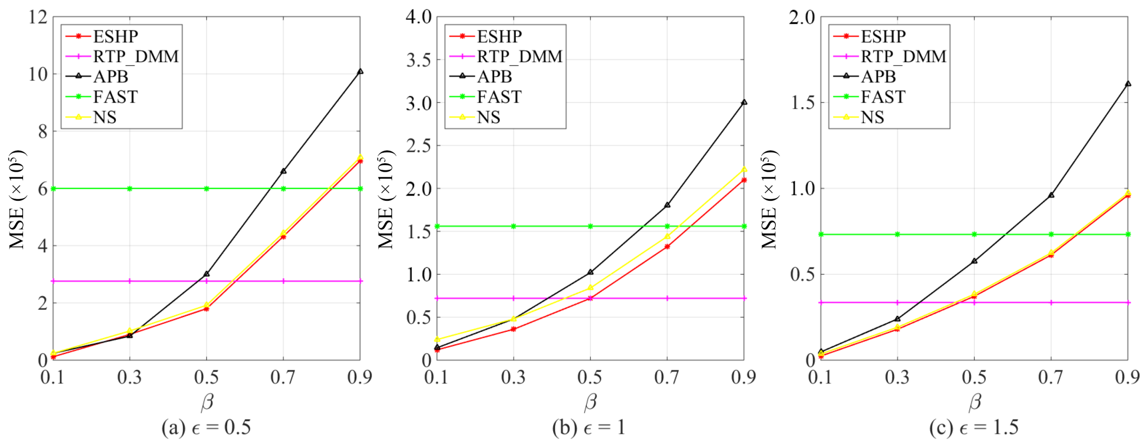


Fig. 14 Accuracy under different approximate error factors (NYC yellow taxi trip dataset).

values for γ and β.

Overall, the results emphasize that the ESHP algorithm delivers enhanced data usability for most combinations of weighted factors and approximate error factors.

7 Conclusion

This research paper presents ESHP, an online algorithm designed for continuously publishing histograms over weighted sliding windows. The algorithm capitalizes on the concept of approximate statistics in data streams and introduces a novel sketching structure called the Approximate-Estimate Sketch. This sketching structure effectively maintains counting information for each histogram interval at every time instance. To ensure the competitiveness of query data in most cases, a greedy clustering algorithm is employed to group and add Laplace noise. The rigorous theoretical analysis and extensive experimental evaluation demonstrate the effectiveness of the proposed ESHP method. It achieves the same level of data utility while significantly reducing computational overhead and storage costs compared to existing methods. These results highlight the potential of ESHP for practical applications involving histogram publishing in data streams. In future work, we aim to explore the integration of reinforcement learning techniques with data stream privacy protection.

Acknowledgment

This research was supported by the Program for Synergy Innovation in the Anhui Higher Education Institutions of China (No. GXXT-2020-012), the National Natural Science Foundation of China (No. 62172003), the Anhui Provincial Natural Science Foundation (No. 2108085MF218), the Anhui Province University Natural Science Research Project (No. 2022AH040052), and the Science and Technology Innovation Program of Ma'anshan, China (No. 2021a120009).

References

[1] A. McAfee and E. Brynjolfsson, Big data: The management revolution, *Harv. Bus. Rev.*, vol. 90, no. 10, pp. 60–68, 2012.

[2] X. Wang, Z. Liu, Y. Gao, X. Zheng, Z. Dang, and X. Shen, A near-optimal protocol for the grouping problem in RFID systems, *IEEE Trans. Mobile Comput.*, vol. 20, no. 4, pp. 1257–1272, 2021.

[3] Z. Hu and D. Li, Improved heuristic job scheduling method to enhance throughput for big data analytics, *Tsinghua Science and Technology*, vol. 27, no. 2, pp. 344–357, 2022.

[4] R. Pan, Z. Li, J. Cao, H. Zhang, and X. Xia, Electrical load tracking scheduling of steel plants under time-of-use tariffs, *Comput. Ind. Eng.*, vol. 137, p. 106049, 2019.

[5] A. Belhassena and H. Wang, Trajectory big data processing based on frequent activity, *Tsinghua Science and Technology*, vol. 24, no. 3, pp. 317–332, 2019.

[6] C. Zhan, H. Hu, Z. Liu, Z. Wang, and S. Mao, Multi-UAV-enabled mobile-edge computing for time-constrained IoT applications, *IEEE Internet Things J.*, vol. 8, no. 20, pp. 15553–15567, 2021.

[7] D. Wei, H. Ning, F. Shi, Y. Wan, J. Xu, S. Yang, and L. Zhu, Dataflow management in the internet of things: Sensing, control, and security, *Tsinghua Science and Technology*, vol. 26, no. 6, pp. 918–930, 2021.

[8] I. Lee and K. Lee, The internet of things (IoT): Applications, investments, and challenges for enterprises, *Bus. Horiz.*, vol. 58, no. 4, pp. 431–440, 2015.

[9] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, Internet of things (IoT): A vision, architectural elements, and future directions, *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.

[10] M. A. Khan and K. Salah, IoT security: Review,

blockchain solutions, and open challenges, *Future Gener. Comput. Syst.*, vol. 82, pp. 395–411, 2018.

[11] Z. Liu, Q. Li, X. Chen, C. Wu, S. Ishihara, J. Li, and Y. Ji, Point cloud video streaming: Challenges and solutions, *IEEE Netw.*, vol. 35, no. 5, pp. 202–209, 2021.

[12] Z. Liu, C. Zhan, Y. Cui, C. Wu, and H. Hu, Robust edge computing in UAV systems via scalable computing and cooperative computing, *IEEE Wirel. Commun.*, vol. 28, no. 5, pp. 36–42, 2021.

[13] Z. Liu, J. Li, X. Chen, C. Wu, S. Ishihara, Y. Ji, and L. Li, Fuzzy logic-based adaptive point cloud video streaming, *IEEE Open J. Comput. Soc.*, vol. 1, pp. 121–130, 2020.

[14] C. Xiang, W. Cheng, C. Lin, X. Zhang, D. Liu, X. Zheng, and Z. Li, *LSTAloc*: A driver-oriented incentive mechanism for mobility-on-demand vehicular crowdsensing market, *IEEE Trans. Mobile Comput.*, doi: 10.1109/TMC.2023.3271671.

[15] A. Cela, T. Jurik, R. Hamouche, R. Natowicz, A. Reama, S. I. Niculescu, and J. Julien, Energy optimal real-time navigation system, *IEEE Intell. Transp. Syst. Mag.*, vol. 6, no. 3, pp. 66–79, 2014.

[16] H. Yi, H. Jung, and S. Bae, Deep neural networks for traffic flow prediction, in *Proc. 2017 IEEE Int. Conf. Big Data and Smart Computing*, Jeiu, Republic of Korea, 2017, pp. 328–331.

[17] L. Zhao, K. Yang, Z. Tan, X. Li, S. Sharma, and Z. Liu, A novel cost optimization strategy for SDN-enabled UAV-assisted vehicular computation offloading, *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 6, pp. 3664–3674, 2020.

[18] C. Wu, Z. Liu, F. Liu, T. Yoshinaga, Y. Ji, and J. Li, Collaborative learning of communication routes in edge-enabled multi-access vehicular environment, *IEEE Trans. Cognit. Commun. Netw.*, vol. 6, no. 4, pp. 1155–1165, 2020.

[19] B. Almadani, B. Saeed, and A. Alroubaiy, Healthcare systems integration using real time publish subscribe (RTPS) middleware, *Comput. Electr. Eng.*, vol. 50, pp. 67–78, 2016.

[20] C. B. Rjeily, G. Badr, A. H. El Hassani, and E. Andres, Medical data mining for heart diseases and the future of sequential mining in medical field, in *Machine Learning Paradigms*, G. A. Tsihrintzis, D. N. Sotiropoulos, and L. C. Jain, eds. Cham, Switzerland: Springer, 2019, pp. 71–99.

[21] C. Xu, Y. Chen, and R. Bie, Sequential pattern mining in data streams using the weighted sliding window model, in *Proc. 2009 15th Int. Conf. Parallel and Distributed Systems*, Shenzhen, China, 2009, pp. 886–890.

[22] P. S. M. Tsai, Mining frequent itemsets in data streams using the weighted sliding window model, *Exp. Syst. Appl.*, vol. 36, no. 9, pp. 11617–11625, 2009.

[23] G. Lee, U. Yun, and K. H. Ryu, Sliding window based weighted maximal frequent pattern mining over data streams, *Exp. Syst. Appl.*, vol. 41, no. 2, pp. 694–708, 2014.

[24] D. H. Vu, K. M. Muttaqi, A. P. Agalgaonkar, and A. Bouzerdoum, A multi-feature based approach incorporating variable thresholds for detecting price spikes in the national electricity market of Australia, *IEEE Access*, vol. 9, pp. 13960–13969, 2021.

[25] M. Yu, L. Wang, G. Xie, and T. Chu, Stabilization of networked control systems with data packet dropout via switched system approach, in *Proc. 2004 IEEE Int. Conf. Robotics and Automation*, Taipei, China, 2004, pp. 362–367.

[26] J. Jiang, H. Wang, and W. Li, A trust model based on a time decay factor for use in social networks, *Comput. Electr. Eng.*, vol. 85, p. 106706, 2020.

[27] F. Liu, T. Yang, J. Zhou, W. Deng, Q. Yu, P. Zhang, and G. Cheng, Spatial variability and time decay of rock mass mechanical parameters: A landslide study in the Dagushan open-pit mine, *Rock Mech. Rock Eng.*, vol. 53, no. 7, pp. 3031–3053, 2020.

[28] A. Erramilli, O. Narayan, and W. Willinger, Experimental queueing analysis with long-range dependent packet traffic, *IEEE/ACM Trans. Netw.*, vol. 4, no. 2, pp. 209–223, 1996.

[29] Y. Ren, Z. Zeng, T. Wang, S. Zhang, and G. Zhi, A trust-based minimum cost and quality aware data collection scheme in P2P network, *Peer Peer Netw. Appl.*, vol. 13, no. 6, pp. 2300–2323, 2020.

[30] A. Golatkar, A. Achille, and S. Soatto, Time matters in regularizing deep networks: Weight decay and data augmentation affect early learning dynamics, matter little near convergence, in *Proc. 33rd Conf. Neural Information Processing Systems*, Vancouver, Canada, 2019, pp. 10678–10688.

[31] J. Xu, Z. Zhang, X. Xiao, Y. Yang, G. Yu, and M. Winslett, Differentially private histogram publication, *VLDB J.*, vol. 22, no. 6, pp. 797–822, 2013.

[32] X. Zhang, C. Shao, and X. Meng, Accurate histogram release under differential privacy, (in Chinese), *J. Comput. Res. Dev.*, vol. 53, no. 5, pp. 1106–1117, 2016.

[33] H. Tang, G. Yang, and Y. Bai, Histogram publishing algorithm based on adaptive privacy budget allocation strategy under differential privacy, (in Chinese), *Appl. Res. Comput.*, vol. 37, no. 7, pp. 1952–1957&1963, 2020.

[34] D. Chen, Y. Li, J. Chen, H. Bi, and X. Ding, Differential privacy via Haar wavelet transform and Gaussian mechanism for range query, *Comput. Intell. Neurosci.*, vol. 2022, pp. 8139813, 2022.

[35] S. Zhang and X. Li, Differential privacy medical data publishing method based on attribute correlation, *Sci. Rep.*, vol. 12, no. 1, p. 15725, 2022.

[36] F. Lin, Y. Wu, Y. Wang, and L. Sun, Differentially private statistical publication for two-dimensional data stream, (in Chinese), *J. Comput. Appl.*, vol. 35, no. 1, pp. 88–92, 2015.

[37] X. J. Zhang and X. F. Meng, Streaming histogram publication method with differential privacy, (in Chinese), *J. Softw.*, vol. 27, no. 2, pp. 381–393, 2016.

[38] X. Wu, N. Tong, Z. Ye, and Y. Wang, Histogram publishing algorithm based on sampling sorting and greedy clustering, in *Proc. 1st Int. Conf. Blockchain and Trustworthy Systems*, Guangzhou, China, 2020, pp. 81–91.

[39] L. Sun, C. Ge, X. Huang, Y. Wu, and Y. Gao, Differentially private real-time streaming data publication based on sliding window under exponential decay,

Comput. Mater. Con., vol. 58, no. 1, pp. 61–78, 2019.

[40] G. Yang, T. Dong, X. Fang, and S. Su, Association data release with the randomised response based on Bayesian networks, *Int. J. Comput. Sci. Eng.*, vol. 20, no. 1, pp. 120–129, 2019.

[41] L. Fan and L. Xiong, An adaptive approach to real-time aggregate monitoring with differential privacy, *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 9, pp. 2094–2106, 2014.

[42] C. Dwork, Differential privacy: A survey of results, in *Proc. 5th Int. Conf. Theory and Applications of Models of Computation*, Xi'an, China, 2008, pp. 1–19.

[43] F. Esponda, Negative surveys, arXiv preprint arXiv: math/0608176v1, 2006.

[44] H. Jiang, W. Luo, and Z. Zhang, A privacy-preserving aggregation scheme based on immunological negative surveys for smart meters, *Appl. Soft Comput.*, vol. 85, p. 105821, 2019.

[45] W. Yang, X. Chen, Z. Xiong, Z. Xu, G. Liu, and X. Zhang, A privacy-preserving aggregation scheme based on negative survey for vehicle fuel consumption data, *Inf. Sci.*, vol. 570, pp. 526–544, 2021.

[46] H. Jiang and W. Luo, Multi-question negative surveys, in *Proc. 3rd Int. Conf. Data Mining and Big Data*, Shanghai, China, 2018, pp. 503–512.

[47] H. Liao, A study of negative surveys with background knowledge, in *Proc. 2019 IEEE 9th Int. Conf. Electronics Information and Emergency Communication*, Beijing, China, 2019, pp. 301–302.

[48] H. Jiang, W. Luo, B. Duan, and C. Wu, Enhancing the privacy of negative surveys using negative combined categories, *Appl. Soft Comput.*, vol. 96, p. 106578, 2020.

[49] F. D. McSherry, Privacy integrated queries: An extensible platform for privacy-preserving data analysis, in *Proc. 2009 ACM SIGMOD Int. Conf. Management of Data*, Providence, RL, USA, 2009, pp. 19–30.

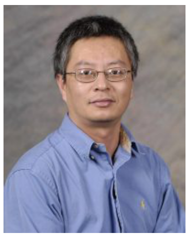
[50] The UK Government, The UK car accident dataset, https://data.gov.uk/dataset/road-accidents-safety-data, 2014.

[51] Taxi & Limousine Commission, NYC yellow taxi trip dataset, https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page, 2019.

Xiujun Wang received the BEng degree in computer science and technology, from Anhui Normal University, China in 2005, and the PhD degree in computer software and theory from University of Science and Technology of China, China in 2011. He is currently a lecturer at School of Computer Science and Technology, Anhui University of Technology. His research interests include data stream processing, randomized algorithms, and the Internet of Things (IoTs).

Xiao Zheng received the BEng degree from Anhui University, China in 1997, the MEng degree from Zhejiang University of Science and Technology, China in 2003, and the PhD degree in computer science and technology from Southeast University, China in 2014. He is currently a professor at School of Computer Science and Technology, Anhui University of Technology, Anhui, China. He is also a guest professor at Institute of Artificial Intelligence, Hefei Comprehensive National Science Center, China. His research interests include service computing, mobile cloud computing, and privacy protection. He has been a guest editor of *IEICE Transactions on Communications*. He is a senior member of CCF, and a member of IEEE and ACM.

Lei Mo received the BEng degree in computer science and technology from Tongling University, China in 2018, and the MEng degree in computer science and technology from Anhui University of Technology, China in 2021. He is currently a system engineer at Baosight Software (Anhui) Co. Ltd. His research interests include data streaming processes and differential privacy.



Zhe Dang received the BEng degree from Nanjing University, China in1987, and the MEng and PhD degrees in computer science from University of California, Santa Barbara, CA, USA in 1998 and 2000, respectively. He is currently an associate professor at School of Electrical Engineering and Computer Science, Washington State University, WA, USA. His current research interests include model-checking and testing for infinite-state and/or real-time systems and randomized algorithms.