Information Fusion 80 (2022) 282–294

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

PrivStream: A privacy-preserving inference framework on IoT streaming data at the edge

Dan Wang [a], Ju Ren [b,a,*], Zhibo Wang [c], Yaoxue Zhang [b,a], Xuemin (Sherman) Shen [d]

[a] School of Computer Science and Engineering, Central South University, Changsha, PR China
[b] Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing, PR China
[c] School of Cyber Science and Engineering, Zhejiang University, Hangzhou, PR China
[d] Department of Electronic and Computer Engineering, University of Waterloo, Waterloo, ON, Canada

ARTICLE INFO

Keywords:
Differential privacy
Artificial intelligence
Edge computing
IoT streaming data
Feature extraction

ABSTRACT

Edge computing combining with artificial intelligence (AI) has enabled the timely processing and analysis of streaming data produced by IoT intelligent applications. However, it causes privacy risk due to the data exchanges between local devices and untrusted edge servers. The powerful analytical capability of AI further exacerbates the risks because it can even infer private information from insensitive data. In this paper, we propose a privacy-preserving IoT streaming data analytical framework based on edge computing, called PrivStream, to prevent the untrusted edge server from making sensitive inferences from the IoT streaming data. It utilizes a well-designed deep learning model to filter the sensitive information and combines with differential privacy to protect against the untrusted edge server. The noise is also injected into the framework in the training phase to increase the robustness of PrivStream to differential privacy noise. Taking into account the dynamic and real-time characteristics of streaming data, we realize PrivStream with two types of models to process data segment with fixed length and variable length, respectively, and implement it on a distributed streaming platform to achieve real-time streaming data transmission. We theoretically prove that Privstream satisfies ϵ-differential privacy and experimentally demonstrate that PrivStream has better performance than the state-of-the-art and has acceptable computation and storage overheads.

1. Introduction

1.1. Background and motivation

We are now living in an unprecedented development era of Internet-of-Things (IoT). The massive IoT data produced by the increasing number of IoT devices has fueled the continuous booming of artificial intelligence (AI). The popularity of AI in turn promotes the prosperity of IoT intelligent applications, such as smart healthcare, smart home, and smart city. Most of the IoT data are time series (e.g., speech, video, and sensor data), which come as streams with huge volumes, requiring fast analysis of AI to meet the task requirement of the underlying IoT application. The traditional practices that require uploading all the data to the cloud server before analyzing data would lead to unpredictable latency due to the network bandwidth limitation. An alternative solution is to leverage edge computing techniques [1,2] combining with AI to timely process and analyze the streaming data, which motivates the emergence of edge intelligence (EI) [3].

One of the effective methods to realize EI is deep learning at the edge (DLE), which has shown the benefits of low latency, energy efficiency, and reduced bandwidth consumption due to the physical proximity between computing and the data-generation sources. However, it faces the risk of privacy leakage on account of some data exchanges between unreliable distributed edge servers and terminal devices. More seriously, deep learning has the ability to infer private information from insensitive data. For instance, when an individual uploads his accelerometer data to the edge server for activity recognition, the untrusted server honestly carries out the target task, but it is also curious about the private information of the user, thus infers the user's private attributes e.g., gender and identity) from the same data [4]. Therefore, effective defenses are urgently needed to prevent the untrusted edge server from making sensitive inferences from the IoT streaming data.

* Corresponding author at: Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing, PR China.
E-mail addresses: wang_dan113@csu.edu.cn (D. Wang), renju@tsinghua.edu.cn (J. Ren), zbwang@whu.edu.cn (Z. Wang), zhangyx@tsinghua.edu.cn (Y. Zhang), sshen@uwaterloo.ca (X. Shen).

https://doi.org/10.1016/j.inffus.2021.11.013
Received 11 January 2021; Received in revised form 26 September 2021; Accepted 9 November 2021
Available online 27 November 2021

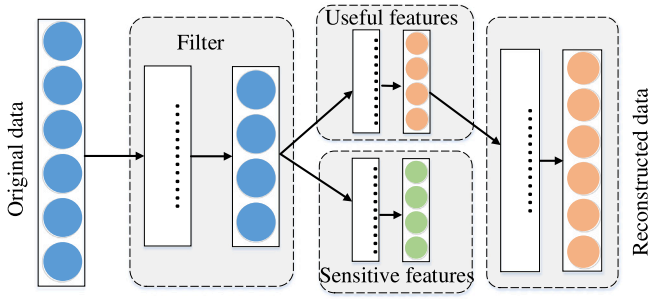


Fig. 1. Sensitive features filtering.

1.2. Limitations of existing works

The traditional privacy-preserving techniques against untrusted edge servers, such as encryption [5,6] and differential privacy (DP) [7] are not suitable for solving this problem. The encryption-based methods provide a high level of privacy protection but they cannot prevent sensitive inference. For example, Apthorpe et al. [8] found that the encrypted network traffic rates of IoT smart home devices can still reveal potentially sensitive user interactions. DP can prevent sensitive inference by injecting a lot of noise at the local device, but it would also destroy the data utility for desire tasks.

Thanks to the advance of feature extraction of deep learning, researchers found that filtering the sensitive information through extracting only the features relevant to useful information is a simple and effective method to resist sensitive inferences [4,9], as shown in Fig. 1. Whereas, most of these works are carried out in an edge server or cloud server with an implicit assumption that the edge/cloud server is trusted, which may be violated in reality.

To solve this problem, some researchers propose private feature filters combined with differential privacy [10,11], which use a pre-trained neural network to filter the private features and then implement differential privacy to the filtered data. These methods provide a privacy-preserving inference framework to protect implicit private information. Nevertheless, most of the existing works target static batch data. Designing a privacy-preserving inference framework for streaming data still faces the following challenges:

- **Variable length.** As we mentioned above, most of the IoT data are streams, which are time-dependent and may have variable lengths (for example, linguistic data). The existing works usually do not consider the context information and are not flexible to process sequences with variable lengths.
- **Streaming processing.** IoT data from multi-sources usually have huge volumes, which need timely processing and analysis. However, to the best of our knowledge, almost all the privacy-preserving inference frameworks transfer batch data periodically. As a result, it is difficult for the existing works to meet the real-time transmission and fast processing requirements of streaming data.
- **Utility preservation.** The DP algorithm protects privacy by injecting noise into data at the local device, which would reduce data utility (e.g., inference accuracy). How to mitigate the utility degradation caused by perturbation while simultaneously preserving privacy is a big challenge for DP algorithm design.

1.3. Contributions

In this paper, we propose a privacy-preserving IoT streaming data inference framework based on edge computing, called PrivStream, to prevent the sensitive inferences on IoT streaming data while enabling the target inference tasks. Specifically, PrivStream is designed as two parts deployed on the IoT device side and edge side, respectively. At the IoT device side, we sample the streaming data, utilize the encoder of a tailored encoder–decoder model to filter the implicit private information from the sampled sequences, and perturb the extracted features by the DP algorithm. The noise is also injected into the framework in the training phase to increase the robustness of PrivStream to differential privacy noise. At the edge side, the perturbed features are decoded into a perturbed sequence, which is then fed to a deep learning model for the target task inference. PrivStream is implemented on a distributed streaming platform, Kafka, to achieve real-time streaming data transmission. To process data segments with fixed length and variable length, we realize PrivStream with two types of deep learning models, the multilayer perceptron (MLP) and the recurrent neural network (RNN), respectively. Our contributions are summarized as follows.

- We propose PrivStream, a privacy-preserving inference framework for IoT streaming data, which can effectively prevent sensitive inferences on IoT streaming data and preserve the data utility for target inference tasks. It uses a deep learning model to filter the sensitive information and integrates with Laplace noise at both training and inference stages to improve the target inference performance while providing the privacy guarantee. In addition, it is deployed on a distributed streaming platform to achieve the goals of real-time transmission and fast streaming processing.
- We realize PrivStream on different deep learning models to compare their performances, including inference accuracy, delay, and resource consumptions.
- We theoretically prove that PrivStream satisfies ϵ-DP. We also execute extensive experiments to evaluate the performances of the proposed framework.

The remainder of the paper is organized as follows. Section 2 introduces some preliminaries. We present the overview of PrivStream in Section 3. We detail the design of PrivStream in Section 4. Implementation is introduced in Section 5. Section 6 exhibits experimental results, and Section 7 reviews related works. Finally, we draw conclusions in Section 8.

Table 1
Frequently used notations.

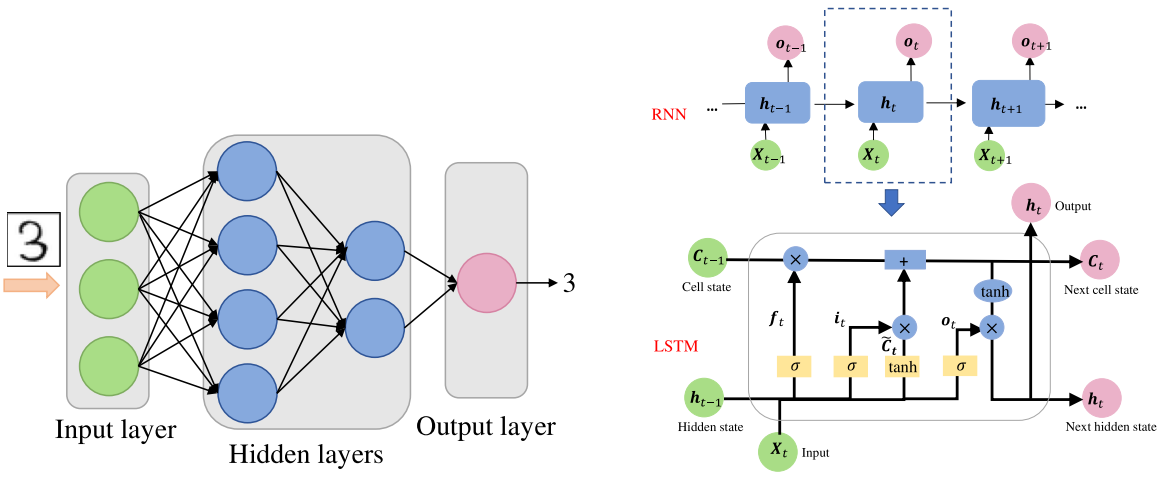
Notation	Description
ϵ	The privacy budget
$\boldsymbol{S}$	The original streaming data
k_n	The nth sampling point
$\boldsymbol{R}$	The sampled streaming data
$\boldsymbol{D}_t$	The tth segment in streaming segment
$\boldsymbol{z}_t$	The feature vector corresponding to $\boldsymbol{D}_t$
$\boldsymbol{z}_t'$	The perturbed feature vector
$\boldsymbol{D}_t'$	The reconstructed segment corresponding to $\boldsymbol{D}_t$
$\dim(\cdot)$	The dimension function
d	The length of extracted feature
w	The window size
N_s	The number of attributes of the input stream
I_n	The sampling interval for k_n

2. Preliminaries

This section describes the basic knowledge of deep learning, and the definitions of DP and perturbation mechanism. The frequently used notations are summarized in Table 1, where the lowercase letters represent scalars, the lowercase bold letters represent vectors, and the capital bold letters represent tensors.

2.1. Deep learning

The core insight of Deep learning is staking multiple layers to extract complex representations from high-dimensional inputs progressively. Two typical deep learning models are multilayer perceptrons



(a) MLP (b) RNN and LSTM

Fig. 2. Illustration of two typical deep learning models.

(MLPs) and recurrent neural networks (RNNs), which are used to process data with fixed length and variable length, respectively.

MLP, as shown in Fig. 2(a), consists of the input layer, hidden layer, and output layer. Each layer is composed of neurons that perform a nonlinear operation. The input layer receives data and propagates them to the hidden layer. Then, each hidden layer transforms its inputs into its outputs through matrix multiplications and nonlinear activation functions e.g., tanh, ReLU, sigmoid) one by one. The output layer receives the data from the hidden layer and outputs the final results.

RNN, as shown in Fig. 2(b), is designed specifically for time series with variable lengths. Unlike the feed-forward neural networks e.g., MLPs), RNNs loop their layer connections to exhibit temporal dynamic characteristics of sequences. That means the outputs are not only controlled by the inputs, but also affected by the previous hidden states. Long and short term memory network (LSTM) [12] is a variant of RNN to process very long sequences. The basic neuron in LSTM is called the memory cell, which includes a cell state $\boldsymbol{C}_t$, a forget gate $\boldsymbol{f}_t$, an input gate $\boldsymbol{i}_t$, and an output gate $\boldsymbol{o}_t$. The cell state acts as the memory of network that carries relevant information throughout the processing of the sequence, the forget gate decides what is relevant to keep from prior steps, the input gate decides what information is relevant to add from the current step, and the output gate determines what the next hidden state should be. These conceptions can be formulated as

$$\boldsymbol{C}_t = \boldsymbol{f}_t \odot \boldsymbol{C}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{C}}_t, \tag{1}$$

$$\tilde{\boldsymbol{C}}_t = \tanh(\boldsymbol{W}_c \cdot \boldsymbol{h}_{t-1} + \boldsymbol{U}_c \cdot \boldsymbol{X}_t), \tag{2}$$

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f \cdot \boldsymbol{h}_{t-1} + \boldsymbol{U}_f \cdot \boldsymbol{X}_t + \boldsymbol{b}_f), \tag{3}$$

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i \cdot \boldsymbol{h}_{t-1} + \boldsymbol{U}_i \cdot \boldsymbol{X}_t + \boldsymbol{b}_i), \tag{4}$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o \cdot \boldsymbol{h}_{t-1} + \boldsymbol{U}_o \cdot \boldsymbol{X}_t + \boldsymbol{b}_o), \tag{5}$$

$$\boldsymbol{h}_t = \tanh(\boldsymbol{C}_t) \odot \boldsymbol{o}_t, \tag{6}$$

where $\boldsymbol{C}_t$ is the cell state that controlled by a weighted sum of candidate cell state at current time step $\tilde{\boldsymbol{C}}_t$ and the cell state at previous step $\boldsymbol{C}_{t-1}$, $\odot$ denotes the element-wise product, $\boldsymbol{f}_t$, $\boldsymbol{i}_t$ and $\boldsymbol{o}_t$ are respectively the forget gate, input gate and output gate that controlled by a sigmoid function σ, $\boldsymbol{W}$ and $\boldsymbol{U}$ are transformation matrix, $\boldsymbol{b}$ is the bias.

Both MLPs and RNNs are trained by back propagation through minimizing loss. This procedure is usually done by Adam or stochastic gradient descent (SGD) algorithm and the loss is task-specific, which can be cross entropy loss for classification tasks or squared loss for regression problems.

2.2. Differential privacy

Differential privacy (DP) is first proposed for privacy-preserving statistics analysis [13]. It can be either applied on a trusted edge server to provide user-level privacy protection, or at the local device to provide event-level privacy protection. The formal definition of ϵ-differential privacy is given as below:

Definition 1 (ϵ-*Differential Privacy*). A randomized algorithm $\mathcal{M}$ satisfies ϵ-differential privacy, where $\epsilon \geq 0$, if for two neighboring inputs $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$ differing in an item, and for any output set $O \subseteq Range(\mathcal{M})$, we have

$$\max_{\boldsymbol{D}_1, \boldsymbol{D}_2, O} \frac{\Pr\{\mathcal{M}(\boldsymbol{D}_1) \in O\}}{\Pr\{\mathcal{M}(\boldsymbol{D}_2) \in O\}} \leq e^{\epsilon}, \tag{7}$$

where ϵ is the privacy budget that trades off the utility and privacy of the output of algorithm $\mathcal{M}$. The notion of the item in the input is application-specific, which may be a segment in a sequence or one data in a segment. A larger ϵ means better utility but weaker privacy protection.

A common technique for numeric queries to satisfy ϵ-differential privacy is the Laplace mechanism, which is defined as

$$\mathcal{M}(\boldsymbol{D}) = f(\boldsymbol{D}) + \left\langle Lap\left(\frac{\Delta f}{\epsilon}\right)\right\rangle^d, \tag{8}$$

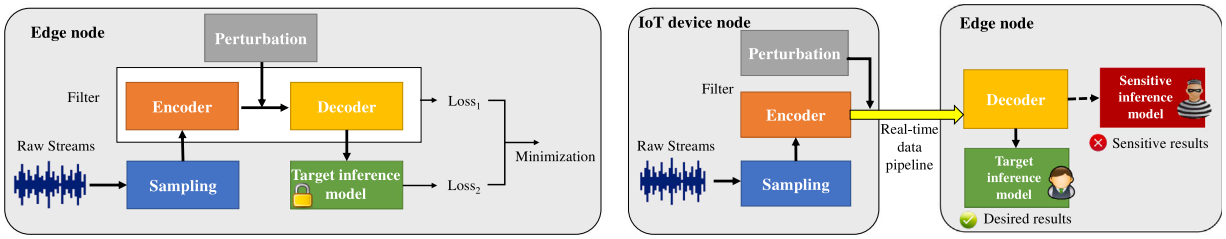
where $f : \boldsymbol{D} \to \mathbb{R}^d$ is a deterministic function, $\Delta f = \max \|f(\boldsymbol{D}_1) - f(\boldsymbol{D}_2)\|_1$ is the sensitivity of f, which is defined as the maximum L_1 distance between the results of the function of two neighboring datasets. $Lap(\Delta f/\epsilon)$ is generated from a zero-mean Laplace distribution with the probability density function $p(x|\lambda) = \frac{1}{2\lambda} e^{-|x|/\lambda}$, where $\lambda = \Delta f/\epsilon$.

In addition, DP has an interesting property of post-processing invariant [14], which means that post-processing an output of a differential privacy algorithm with any complicated operation would not incur any additional privacy loss.

3. Problem definition and system overview

3.1. Problem definition

We assume the IoT data are collected and analyzed at the untrusted edge node. The edge node honestly carries out the target tasks, but also feels curious about the private information of users. Consider a case as below:



(a) Training phase (b) Inference phase

Fig. 3. The high-level overview of PrivStream.

The accelerometer data from wearable devices or mobile phones are collected at an edge node for analysis. The target task of the edge server is to infer the activities of users, such as walking, running, upstairs, and downstairs. Although the data itself is insensitive, however, due to the uniqueness of each user's data, the edge server can still easily infer private information from data, e.g., age, gender, and identification.

In this case study, we define the useful inference as activity and the sensitive inference as all the possible inferences except for the useful inference. Our objective is to prevent sensitive inferences on the IoT streaming data while enabling useful inference with high accuracy.

3.2. Overview

To prevent the sensitive inference while enabling desired useful inference, we propose a privacy-preserving IoT streaming data inference framework, PrivStream, as shown in Fig. 3. PrivStream mainly contains five components: sampling module, encoder, perturbation module, decoder, and inference model. The sampling module adaptively samples the stream to construct input sequences of subsequent modules, the encoder extracts features from high-dimensional input sequences, the perturbation module injects differential privacy noise into the extracted features, the decoder reconstructs the noised features into a noised sequences. Encoder and decoder work together to filter the implicit sensitive information in input sequences. The inference model is a deep learning model used for inferring desired useful information.

We train these models with users' data in the cloud and inferencing at the edge. In the training phase as shown in Fig. 3(a), we first use the sampled sequences and corresponding labels to train an inference model for the desired useful prediction. Then we froze the parameters of the inference model and train the encoder–decoder model. The encoder–decoder model is trained by minimizing the total losses utilizing the SGD or Adam algorithm. Specifically, the total losses include the squared loss between the input of encoder and output of decoder, and the cross entropy loss between the predicted results of inference model and ground truths. Through this supervised learning, the encoder–decoder can efficiently filter the private features and only preserve the features related to the desired inference task. Besides, we add perturbations to the media features between encoder and decoder to increase the robustness of the encoder–decoder model. After the training, the IoT devices download the sampling, encoder, and perturbation modules from the cloud server to run them locally, and the edge server downloads the decoder and inference models.

In the inference phase as shown in Fig. 3(b), the IoT device node executes the sampling, encoder, and perturbation modules, and the edge node implements the decoder and inference models. It first extracts features from the sampled sequences using the encoder model and then perturbs the extracted features with random noise. These processes provide two layers of protection. On the one hand, transforming the sequence and filtering the private features protect both the sensitive data and implicit private attributes embedded in the stream. On the other hand, adding noise to the extracted features protects them against untrusted edge nodes. After that, the perturbed features are transmitted to the edge node through a real-time data pipeline. The decoder reconstructs the perturbed features into perturbed sequences, which have the same dimension as the input sequences to adapt the inference model. Given the perturbed sequences, the edge node can accurately predict desired tasks utilizing the inference model, but cannot obtain any sensitive information.

4. Design of PrivStream

In this section, we first describe each module of PrivStream. In particular, we exhibit the constructions of Encoder–Decoder filter on two types of deep learning models, MLP and RNN, which process data segments with fixed length and variable length, respectively. Then, we theoretically analyze the privacy guarantee.

4.1. Adaptive sampling

Since the streaming data may be real-time and has huge volumes, it is inefficient to process and analyze all the data. Sampling is an effective method to reduce the computing burden and reflect the approximate distribution of streaming data, but it has a problem that how to determine the sampling frequency when the distribution of streaming data is unknown. For example, when the data changes gently, a high sampling frequency would collect too much data that is not necessary, while when the data changes dramatically, a low sampling frequency would lead to information loss. To solve this problem, we adaptively sample the streaming data based on PID control [15], which uses a feedback error between the current and the last sampling point to affect PID error and then control the next sampling interval. The detailed process of the PID controller is introduced in the following.

We denote the streaming data as $\boldsymbol{S} = \{\boldsymbol{r}_1, \boldsymbol{r}_2, \dots\}$, where each element $\boldsymbol{r}_i$ is a vector with dimension N_s. The feedback error at time k_n is defined as

$$E_{k_n} = \frac{1}{N_s} \sum_{j=1}^{N_s} |r^j_{k_n} - r^j_{k_{n-1}}|, \tag{9}$$

where k_n is the current sampling point, and k_{n-1} is the last sampling point.

The feedback error then affects the PID error, which is calculated as follows.

$$\Delta_{err} = K_p E_{k_n} + K_i \frac{\sum_{o=n-\pi-1}^{n} E_{k_o}}{\pi} + K_d \frac{E_{k_n}}{k_n - k_{n-1}}, \tag{10}$$

where the parameters K_p, K_i and K_d are the scale factors representing proportional gain, integral gain, and derivative gain, respectively. The first term in Eq. (10) stands for present error, the second term denotes the accumulation of past error, and the third term is the prediction of future error. π is how many recent errors are taken for integral error.

An intuitive idea is that, when the data changes rapidly, the PID error would be large, so the sampling interval should be small. Thus, the next sampling interval I' can be calculated by

$$I' = \max\{1, I + \rho(1 - e^{\frac{\Delta_{err} - \eta}{\eta}})\}, \tag{11}$$

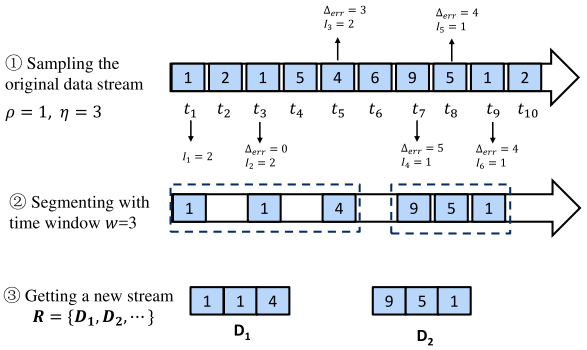


Fig. 4. Example of adaptive sampling on IoT streaming data.

where I and I' are the current and next sampling interval, respectively. ρ and η are pre-determined parameters, respectively. It can be seen from Eq. (11) that the sampling interval decreases as the PID error Δ_{err} increases and vice versa. More details about PID control please refer to [15,16].

Step ① in Fig. 4 is an example to illustrate the adaptive sampling process in 10 timestamps with parameters $\rho = 1$ and $\eta = 3$. For the sake of simplicity, we assume the feedback error is the PID error. Assume the initial sampling point and sampling interval are t_1 and $I_1 = 2$, respectively. t_3 is the next sampling point, and the PID error between t_1 and t_3 is 0. Therefore, the sampling interval is $I_2 = \max\{1, 2+1\cdot(1-e^{\frac{0-3}{3}})\} \approx 2$ (we take the integer part). Thus the third sampling point is t_5 and the PID error is 3. Similarly, $I_3 = \max\{1, 2+1\cdot(1-e^{\frac{3-3}{3}})\} = 2$ and the next sampling point is t_7. However, at the fourth sampling point t_7, the PID error is 5, as a result, the sampling interval becomes 1 and the next sampling point is t_8. Overall, the sampling interval will decrease if the feedback error exceeds a certain threshold and vice versa.

We denote the sampled points as a new stream $\boldsymbol{S}_{samp} = \{\boldsymbol{r}_{k_1}, \boldsymbol{r}_{k_2}, \dots\}$, where $\boldsymbol{r}_{k_i} \in \mathbb{R}^{N_s}$. For further analysis, we follow the common practice [17] to partition them into non-overlapping segments $\boldsymbol{R} = \{\boldsymbol{D}_1, \boldsymbol{D}_2, \dots\}$, as shown in Step ② and Step ③ in Fig. 4, where $\boldsymbol{D}_t = \{\boldsymbol{r}_{k_{w(t-1)}}, \dots, \boldsymbol{r}_{k_{wt-1}}\} \in \mathbb{R}^{w\times N_s}$, w is the size of time window. Note that, every window $\boldsymbol{D}_t$ is an input sample of the subsequent model.

4.2. Encoder–decoder filter

Since the input sample $\boldsymbol{D}_t$ usually contains redundant information, it deservedly can be minimized according to its inherent structure [18]. Based on this fact, a non-trivial idea behind PtivStream is minimizing the input data to get features that contain only useful information. To realize this goal, we have to solve two problems: (1) How to minimize the input data, (2) How to filter the sensitive information.

The Encoder–decoder model [19] is a typical technique to automatically extract features from high-dimensional data by minimizing the reconstruction error between the input and the reconstructed output. But it is a type of unsupervised learning model, of which the extracted features contain both useful and sensitive information. To solve the two problems mentioned above, we designed a tailored encoder–decoder filter based on a multilayer perceptron (MLP) in the prior work [20]. However, it is designed for processing data segments with a fixed length. To adapt the filter for data segments with potential variable lengths, we also realize it on a recurrent neural network (RNN). The two types of filters are detailed in the following. Note that, the encoder–decoder filter is trained offline in the cloud. After training, the encoder and the decoder are split and deployed on the IoT device side and edge server for feature extraction and data reconstruction, respectively. The output of decoder has the same dimension as the input of encoder to adapt to the interface of the subsequent inference model.

4.2.1. Encoder–decoder filter based on MLP [20]

A traditional structure of encoder–decoder model [19] based on MLP is shown in the left of Fig. 5, which consists of *Encoder* and *Decoder* model. The middle layer between *Encoder* and *Decoder* are the features that we aim to extract. As each neuron in MLP stands for a single data, for each input sample $\boldsymbol{D}_t \in \mathbb{R}^{w\times N_s}$, we first flatten $\boldsymbol{D}_t$ into a column vector $\boldsymbol{D}_t \in \mathbb{R}^{wN_s\times 1}$. The *Encoder* maps the input sample $\boldsymbol{D}_t \in \mathbb{R}^{wN_s\times 1}$ into feature representation $\boldsymbol{z}_t \in \mathbb{R}^{d\times 1}$, where d ($d < wN_s$) is the length of extracted features. Then the *Decoder* reconstructs the feature vector into output $\boldsymbol{D}'_t \in \mathbb{R}^{wN_s\times 1}$. The encoder–decoder model is trained by minimizing the cost function

$$\begin{aligned}\min J_0(\boldsymbol{u},\boldsymbol{v}) &= \min \sum_{t=1}^{T} \|\boldsymbol{D}_t - \boldsymbol{D}'_t\|_2 \\ &= \min \sum_{t=1}^{T} \|\boldsymbol{D}_t - Decoder(Encoder(\boldsymbol{D}_t))\|_2,\end{aligned} \tag{12}$$

where T is the length of time for streaming data, $\boldsymbol{u}, \boldsymbol{v}$ are the parameters of *Encoder* and *Decoder*, respectively.

However, the feature vector $\boldsymbol{z}_t$ is related to both useful and sensitive information. To ensure that $\boldsymbol{z}_t$ is as relevant to useful information as possible, we train the encoder–decoder filter combining with a trained useful inference model in a supervised manner. Specifically, we freeze the parameters of the trained inference model, and feed it the output of the *Decoder* as shown in Fig. 5. Then the cost function of the useful inference model is added into the cost function of the encoder–decoder model as follows.

$$\begin{aligned}\min \boldsymbol{J}(\boldsymbol{u},\boldsymbol{v}) &= \min J_0(\boldsymbol{u},\boldsymbol{v}) \\ &- \mu \sum_{t=1}^{T} y_t \log(\boldsymbol{\theta}^T \boldsymbol{D}'_t - y_t) + \mu\lambda \|\boldsymbol{\theta}\|_2,\end{aligned} \tag{13}$$

where the first term on the right side is the cost function of a traditional encoder–decoder model, the second term is the loss function of the useful inference task, and the third term is the regularization term, $\boldsymbol{\theta}$ is the parameter of the useful inference model, y_t is the ground truth of $\boldsymbol{D}_t$, and μ is a parameter that controls the trade-off between the reconstruction loss and the utility penalty.

4.2.2. Encoder–decoder filter based on RNN

As we mentioned above, RNN is more suitable for processing time series with variable length than MPL, because RNN only needs to add a time step instead of changing the graph of neural networks. Since the streaming data can also be regarded as time series, we also design the RNN-based encoder–decoder filter and inference model, of which the structures are shown in Fig. 6.

Suppose the sampled stream is segmented by variable-length windows with the maximum length K. The segment that is shorter than K will be padding to the same length, and the padding value is then masked in RNN. The training of an RNN-based encoder–decoder filter still begins with training an inference model. The inference model is a many-to-one structure (the right side in Fig. 6), that is to say, we feed the segment $\boldsymbol{D}_t = \{\boldsymbol{x}_1^t, \boldsymbol{x}_2^t, \dots, \boldsymbol{x}_K^t\} \in \mathbb{R}^{K\times N_s}$ to the hidden layers step by step, and get the corresponding output $\boldsymbol{O}_t = \{o_1^t, o_2^t, \dots, o_K^t\}$, but only the last output o_K^t (if $\boldsymbol{D}_t$ is not padded) of hidden layer is mapped by a `softmax` function to obtain the inference results $\boldsymbol{p}_t = \{p_1^t, p_2^t, \dots, p_N^t\}$ (the probabilities of N potential categories). Note that, padding and masking are co-existed for all variable-length segments in both encoder–decoder filter and inference model. The loss function of the inference model is cross entropy that can be formulated as

$$L_i = -\sum_{t=1}^{T}\sum_{n=1}^{N} y_n^t log(p_n^t), \tag{14}$$

where N is the total number of categories, and T is the length of time for streaming data. $y_n^t \in \{0,1\}$ is the true label of the sample, and p_n^t is the prediction probability that the sample belongs to category n. We

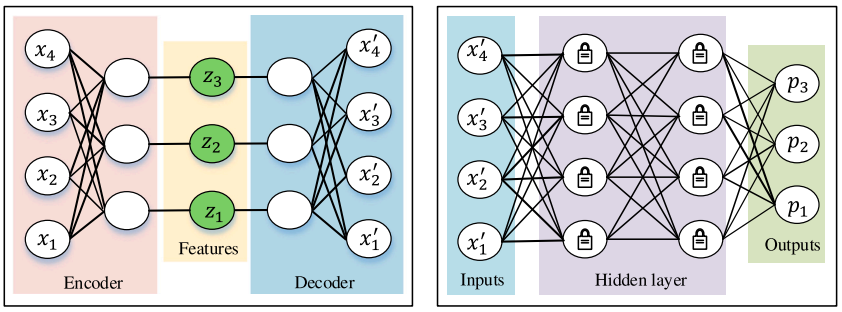


Fig. 5. The structure of the encoder–decoder filter based on MLP.

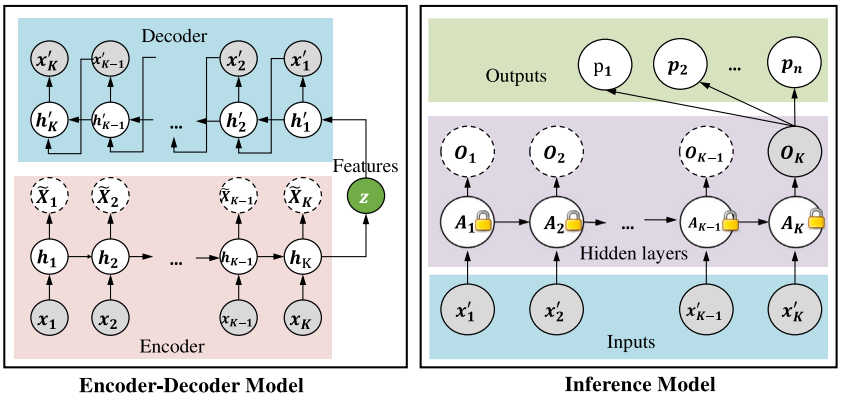


Fig. 6. The structure of the encoder–decoder filter and inference model based on RNN.

employ LSTM as the backbone network of the inference model due to its powerful modeling for a very long sequence. The inference model is trained by minimizing the L_i through gradient descent optimization. After training, the parameters are frozen and used for training the encoder–decoder model.

Similar to the MLP-based encoder–decoder filter, the RNN-based encoder–decoder model also consists of two components: the *Encoder* which encodes the input sequence into a hidden representation and the *Decoder* which reconstructs the input sequence from the hidden representation. We also realize the filter on an LSTM network, where the encoder and decoder are presented as follows.

Encoder. The *Encoder* aims to get the feature vector $\boldsymbol{z}_t$, which is the last hidden state $\boldsymbol{h}_K^t$ on the left side of Fig. 6. Formally, given the input stream $\{\boldsymbol{D}_1, \boldsymbol{D}_2, \dots\}$, each segment $\boldsymbol{D}_t$ at timestamp t are padded to the same length K and consists of $\{\boldsymbol{x}_1^t, \boldsymbol{x}_2^t, \dots, \boldsymbol{x}_K^t\}$ in which $\boldsymbol{x}_k^t \in \mathbb{R}^{N_s}$ denotes the observations at the kth time step of $\boldsymbol{D}_t$. The calculations of hidden state $\boldsymbol{h}_K^t$ of the last time step are in line with the routine forward propagation of a many-to-one LSTM model, which is formulated as follows. Note that, for simplicity, we omit t in the following.

$$\boldsymbol{h}_k = \boldsymbol{o}_k \odot \tanh(\boldsymbol{C}_k), \tag{15}$$

$$\boldsymbol{z} = \boldsymbol{h}_K, \tag{16}$$

where $\boldsymbol{o}_k$ and $\boldsymbol{C}_k$ are the output gate and cell state of the kth step, which can be calculated by Eq. (1)–(5). The calculations of $\boldsymbol{h}_k$ are recurrent until $k = K$. At last, the hidden state $\boldsymbol{h}_K$ at time step K is employed as the feature vector $\boldsymbol{z}$.

Decoder. We deploy another LSTM model as a decoder, which is a little different from the encoder model. Specifically, in the *Encoder* model, only the hidden state $\boldsymbol{h}$ is recurrent, but in the *Decoder* model, both the hidden state $\boldsymbol{h}'$ and output $\boldsymbol{x}'$ are recurrent. Formally, the reconstructed observation $\boldsymbol{x}'_k$ at the kth time step can be modeled as

$$\boldsymbol{x}'_k = \boldsymbol{B} \cdot g(\boldsymbol{A} \cdot \boldsymbol{h}'_k), \tag{17}$$

where $\boldsymbol{A}, \boldsymbol{B}$ are mapping matrices and $g(\cdot)$ is an activation function (Relu, tanh or softmax). The derivative process of hidden state $\boldsymbol{h}'_{k,k\neq 0}$ is similar to that of the encoder (Eq. (15)), but note that the candidate cell state $\tilde{\boldsymbol{C}}_k$ and three gates $\{\boldsymbol{f}'_k, \boldsymbol{i}'_k, \boldsymbol{o}'_k\}$ are slightly different from Eq.((3)–(5)), because the variate $\boldsymbol{x}'_k$ is not from external input but from the output of the previous time step. For any $\boldsymbol{v}_k \in \{\tilde{\boldsymbol{C}}_k, \boldsymbol{f}'_k, \boldsymbol{i}'_k, \boldsymbol{o}'_k\}$, it can be modeled as

$$\boldsymbol{v}_k = f_v(\boldsymbol{W}'_{\boldsymbol{v}} \cdot \boldsymbol{h}'_{k-1} + \boldsymbol{U}'_{\boldsymbol{v}} \cdot \boldsymbol{x}'_{k-1} + \boldsymbol{b}'_{\boldsymbol{v}}). \tag{18}$$

All the variates are calculated step by step from the initially hidden state $\boldsymbol{h}'_0$ and the initial input $\boldsymbol{x}'_0$, which are respectively specified as the feature vector and a start symbol e.g., a character in natural language processing or zero matrix in numeric streaming data) as follows.

$$\boldsymbol{h}'_0 = \boldsymbol{z}, \tag{19}$$

$$\boldsymbol{x}'_0 = start\ symbol. \tag{20}$$

For an unsupervised encoder–decoder model, the parameters of the model are optimized by minimizing the reconstruction error between the reconstructed sequence and the original sequence to guarantee that they are as close as possible. We use the mean absolute error as the loss function, which is formulated as

$$L_a = \sum_{t=1}^{T}\left(\frac{1}{K^{(t)}}\sum_{k=1}^{K^{(t)}} |\boldsymbol{x}_k^{\prime(t)} - \boldsymbol{x}_k^{(t)}|\right). \tag{21}$$

However, this model trained in an unsupervised way cannot effectively filter sensitive information. To extract features that only contain useful information, we feed the output of RNN-based encoder–decoder filter to the pre-trained inference model to get the loss L_i according to Eq. (14). Therefore, the encoder–decoder filter is trained by minimizing the weighted sum of loss functions L_a and L_i:

$$L = \mu L_a + (1-\mu) L_i, \tag{22}$$

where $\mu \in [0, 1]$ is a hyper-parameter that trades off the impact of two models.

4.3. Feature perturbation

Although the feature extraction module has provided a protection layer for private information through extracting features that are specific to useful inferences, the learned features still have some degree of correlations with private information. To protect them against the untrusted edge server, we further apply ϵ-differential privacy (ϵ-DP) to the extracted feature vector $\boldsymbol{z}$ before sending them to the edge server. ϵ-differential privacy can guarantee that no matter how much background knowledge the edge server has, it cannot restore the input sample from the perturbed features.

The perturbation mechanism that we adopted is the Laplace mechanism. We take the *Encoder* model as the query function, and the feature vector as the query result. The perturbed feature vector can be modeled as

$$\boldsymbol{z}' = f(\boldsymbol{D}) + \langle Lap(\Delta f/\epsilon)\rangle^d = \boldsymbol{z} + \langle Lap(\Delta f/\epsilon)\rangle^d, \tag{23}$$

where $\Delta f = \max \|f(\boldsymbol{D}) - f(\boldsymbol{D}')\|_1$ is the global sensitivity defined as the maximum L_1 distance between $f(\boldsymbol{D})$ and $f(\boldsymbol{D}')$, $\boldsymbol{D}$ and $\boldsymbol{D}'$ are two neighboring inputs that only differ at one element, and d is the dimension of the feature vector.

The value of global sensitivity Δf is determined by the output threshold of *Encoder* model, which is computed as the maximal domain of the activation function of the output layer. For example, for the `Sigmoid` and `tanh` functions with finite bounds $[0, 1]$ and $[-1, 1]$, the global sensitivities are 1 and 2, respectively. However, for the activation function with infinite bound, for example, the `ReLU` function with bound $[0, \infty)$, it is difficult to estimate the global sensitivity. To solve this problem, we clip the output of the *Encoder* with a threshold. Specifically, we set the threshold as B, the encoder output $\boldsymbol{z}$ is clipped into

$$\boldsymbol{z} = \boldsymbol{z}/\max\{1, \frac{\|\boldsymbol{z}\|_\infty}{B}\}. \tag{24}$$

It indicates that $\boldsymbol{z}$ will be preserved if $\|\boldsymbol{z}\|_\infty < B$, otherwise it will be cut down to B. Then, the global sensitivity is estimated as B for the *Encoder* with `ReLU` activation function. According to normal practice, the value of B is set as the median of $\|\boldsymbol{z}\|_\infty$ over the training [21].

Applying differential privacy in the inference phase can guarantee that the untrusted edge server cannot acquire any observation in the input sequence from the features, nevertheless, it would decrease the data performance (inference accuracy), which has been proved in our previous work [20]. In order to improve the robustness of PrivStream to noise, we also add random noise into the feature vector $\boldsymbol{z}$ in the training phase. However, it is difficult to determine how much the random noise is suitable because the amount of Laplace noise (controlled the privacy budget ϵ) added during the inference stage is unknown in advance. Wang et al. [7] proposed to inject the worse perturbation into the representation to maximize the deviation from the original data, while trying to minimize the loss in the training. Such a method, however, is not suitable for PrivStream, because the perturbations added to the features $\boldsymbol{z}$ have very complex effects on the total losses, which makes the maximum perturbation difficult to be calculated. Therefore, we estimate the value of perturbation with qualitative experiments instead of quantitative analysis. Specifically, we add Laplace noise $Lap(B/\delta)$ to the features in the training stage. The perturbation degree is controlled by parameter δ. A smaller δ corresponds to the worse perturbation. To found the proper δ, we will evaluate the performances of PrivStream under different (δ, ϵ) pairs in Section 6.

4.4. Reconstruction and inference

The *Decoder* model and inference model are deployed on the edge node. Once the *Decoder* receives the perturbed feature vector $\boldsymbol{z}'$ from the IoT device node, it would reconstruct the feature vector into the data that has the same dimension as the input samples of *Encoder*. The reconstructed data are then sent to the inference model for the target task inference. Note that, the reconstructed data have been cleaned and perturbed, which means that, the adversary cannot get any private information from the reconstructed data, but the inference model can still get accurate inference results of the target task.

Algorithm 1 The first part of PrivStream on the IoT device node

Require: Original streaming data $\boldsymbol{S} = \{\boldsymbol{r}_1, \boldsymbol{r}_2, \dots\}$, privacy budget ϵ,initial sampling interval I, :
Ensure: Perturbed feature representations $\boldsymbol{Z} = \{\boldsymbol{z}_1', \boldsymbol{z}_2', \dots\}$
1: Adaptively sampling $\boldsymbol{S}$ with Eq. (11) to get $\boldsymbol{S}_{samp}$
2: Segmenting $\boldsymbol{S}_{samp}$ to form the input streaming samples $\boldsymbol{R} = \{\boldsymbol{D}_1, \boldsymbol{D}_2, \dots\}$
3: **for** each $\boldsymbol{D}_t \in \boldsymbol{R}$ **do**
4: Extracting features $\boldsymbol{z}_t = Encoder(\boldsymbol{D}_t)$
5: Perturbing features $\boldsymbol{z}_t' = \boldsymbol{z}_t + Lap(\Delta f/\epsilon \boldsymbol{I})$
6: **end for**
7: Transferring the perturbed features $\boldsymbol{Z} = \{\boldsymbol{z}_1', \boldsymbol{z}_2', \dots\}$ to the edge server

Algorithm 2 The second part of PrivStream on the edge node

Require: Perturbed feature representations $\boldsymbol{Z} = \left\{\boldsymbol{z}_1', \boldsymbol{z}_2', \dots\right\}$
Ensure: Target task inference results $\boldsymbol{Y}^* = \{\boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \dots\}$
1: **for** each $\boldsymbol{z}_t' \in \boldsymbol{Z}$ **do**
2: Reconstructing data $\boldsymbol{D}_t' = Decoder(\boldsymbol{z}_t')$
3: Inferring the target task result $\boldsymbol{y}_t^* = Target(\boldsymbol{D}_t')$
4: **end for**

4.5. Algorithms

PrivStream is designed as two parts deployed on the IoT device side and edge side. The key steps on the two sides are summarized in Alg. 1 and Alg. 2, respectively. At the IoT device side, PrivStream adaptively samples the streaming data, extracts features to defend sensitive inference, and perturbs features to prevent from restoring the input sample. While at the edge node, PrivStream reconstructs the perturbed features for the target inference task. With the two algorithms, we then prove that PrivStream satisfies ϵ-DP.

4.6. Privacy analysis

Theorem 1. *For any $\epsilon > 0$, PrivStream satisfies ϵ-DP.*

Proof. According to the definition of DP, a mechanism satisfies DP as long as an adversary cannot distinguish the input from its neighboring data based on the output of this mechanism. Laplace mechanism is a well-known perturbation mechanism that satisfies ϵ-DP, which injects Laplace noise into the results of the query function. We take the *Encoder* of PrivStream as the query function and the feature representation $\boldsymbol{z}_t$ as the output of *Encoder*. PrivStream adds Laplace noise into the features, thus the perturbed features naturally satisfy ϵ-DP. In addition, DP has a character of post-processing invariant [14], which means that post-processing an output of a differentially private algorithm does not incur any additional loss of privacy. We consider the *Decoder* and inference model as the post-processing functions. As a result, the output of PrivStream still satisfies DP. That ends the proof. □

5. Implementation

In order to realize real-time and reliable streaming data transmission, we implement PrivStream on a distributed streaming platform. There are many distributed computing system can process big streaming data in real-time or near real-time, such as Apache Spark [22],

Table 2
Structure of PrivStream-MLP and PrivStream-RNN. Each function with parameters denotes a layer in the deep learning model. w is the size of time window and d is the dimension of extracted features.

	Encoder–Decoder filter		Inference model
PrivStream-MLP	Encoder	Input ($\|50 \times 12\|$) Dense (512) Dense (d)	Input (50, 12, 1) Conv2D (50,(1 × 5)); Conv2D (50,(1 × 3)) Dense (50); Pool(1 × 2); Drop(0.2) Conv2D (40,(1 × 5))
	Decoder	Dense (d) Dense (512) Output ($\|50 \times 12\|$)	Dense (40); Pool (1 × 3); Drop(0.2) Conv2D (20,(1 × 3)); Drop (0.2) Flatten; Dense (400); Drop (0.2) Softmax (4)
PrivStream-RNN	Encoder	Input (w, 12) LSTM (d) LSTM (d)	Masking ((50, 12)) LSTM (50) LSTM (50)
	Decoder	LSTM (d) LSTM (d) Dense (w,12)	Dense (4) Softmax (4)

Storm [23], and Kafka [24]. Here we choose Kafka due to its advantages of lightweight, integration, scalability, and high throughput.

Apache Kafka mainly contains three components: `producer`, `consumer` and `agent`. In our work, the `producer` runs on the IoT device, and the `consumer` runs on the edge server node. `Agent` is the core of Kafka, which serves as a bridge between producer and consumer, and runs as a cluster on the Kafka server (it can be the edge server node or another server). `Agent` is also responsible for balancing the traffic load. If the `consumer` cannot draw the streaming data that come from `producer` in time, `agent` will temporarily store the stream in the Kafka server in a place called `brokers`, and blocks the output of the IoT device. The specific process of streaming data transmission is described as follows.

PrivStream first starts the Kafka server and creates a topic on it using Kafka API. Then the IoT device node and edge server subscribe to this topic through a Kafka Producer API and a Consumer API, respectively. After that, the IoT device begins to run the sampling, encoder, and perturbation module to produce the noisy features, which afterward are published to the topic through a Kafka Producer API. Thereafter, the edge server receives the noisy features from this topic through a Kafka Consumer API and sends these data to the Decoder for data reconstruction. The reconstructed data are then fed into the subsequent inference model to get the desired inference results.

Kafka can integrate with many programming languages. Here we give examples based on Python. Python has the Kafka package, and the Producer and Consumer APIs can be easily implemented by a few lines of code:

```
//The Producer API of IoT device
from kafka import KafkaProducer
producer=KafkaProducer(bootstrap_servers=KAFKA_URI)
producer.send(topic=TOPIC,value=Features)

//The Consumer API of edge server
from kafka import KafkaConsumer
consumer=KafkaConsumer(TOPIC,bootstrap_servers=KAFKA_URI)
for features in consumer:
Decoder(features)
```

6. Experimental results

In this section, we first clarify the dataset and experiment setup, then conduct extensive experiments to evaluate the effects of three modules (adaptive sampling, encoder–decoder filter, perturbation module) on the performances of PrivStream. Finally, we test the computation and storage efficiency of PrivStream on both IoT devices and the edge node.

Table 3
Original inference accuracy of three tasks.

Model	Phase	Activity	Gender	Identity
PrivStream-MLP	Train	0.98	0.99	0.97
	Test	0.95	0.97	0.85
PrivStream-RNN	Train	0.99	0.99	0.98
	Test	0.93	0.97	0.81

6.1. Dataset and experiment setup

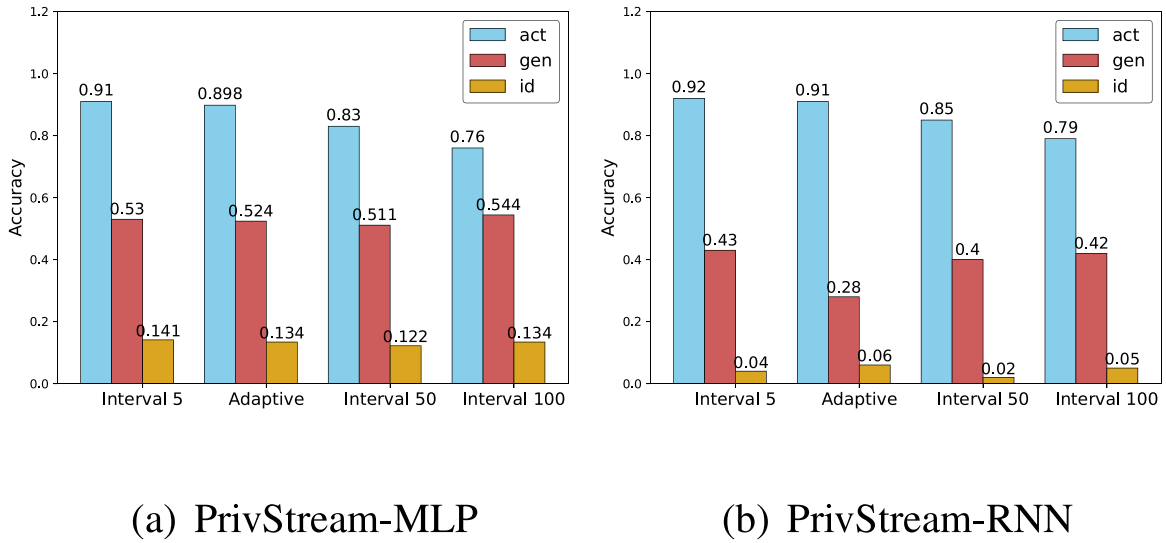
Dataset. The dataset we used is MotionSense [4], which contains 24 participants that perform 4 activities in 15 trials: walking, running, upstairs, and downstairs. The data includes the accelerometer, attitude and gyroscope data, with a total of 12 attributes (i.e., $N_s = 12$), and the labels include their activities, IDs, genders, ages, and weights. We randomly divide the trials into train set, validation set, and test set to 60%, 20%, 20%, respectively.

Metrics. We choose activity recognition as our target inference task (useful inference), while identity recognition and gender recognition as sensitive inferences that we aim to prevent. Privacy metric is defined as the sensitive inference accuracy, and a lower sensitive inference accuracy means higher privacy. The probabilities of random guesses for identity recognition and gender recognition are 1/24 and 1/2, respectively. If the sensitive inference accuracies are close to random guessing, we consider that PrivStream has a high degree of privacy protection. Date utility is measured from two aspects, including the useful inference accuracy and the mean absolute error (MAE) between the reconstructed data and original data. Higher useful inference accuracy and lower MAE indicate better data utility.

Model structure. Since these data are time-series, they are first sampled by the sampling module of PrivStream and segmented with a time window to generate the input segments. For the MLP-based PrivStream (we call it PrivStream-MLP), we set the time window size as $w = 50$, while for the RNN-based PrivStream (we call it PrivStream-RNN), the window size is randomly chosen from $w \in \{30, 40, 50\}$ for each segment. The structures of encoder–decoder filters and inference models of PrivStream-MLP and PrivStream-RNN are displayed in Table 2.

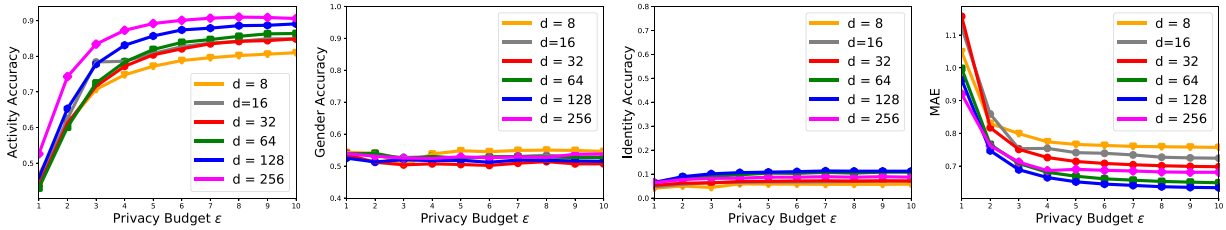
Configuration. In the experiment, we use Raspberry Pi 3B as our IoT device node, and a laptop equipped with quad-core Intel Core i5 processors and 8 GB of RAM as both the Kafka server and edge node. All the programs are written with Python and run on Ubuntu 16.04.

Comparison methods. We compare PrivStream with two mechanisms to prove the effectiveness of encoder–decoder filter and perturbation. The first is GEN [4], which only filters sensitive features without perturbation. The other is the baseline method, which only perturbs the stream without feature extraction. For a fair comparison, the GEN



(a) PrivStream-MLP (b) PrivStream-RNN

Fig. 7. Comparison of different sampling intervals.



(a) Activity (b) Gender (c) Identity (d) MAE

Fig. 8. Effect of encoder–decoder filter of PrivStream-MLP.

and baseline method used in this paper have the same encoder–decoder filter and inference model of PrivStream, respectively.

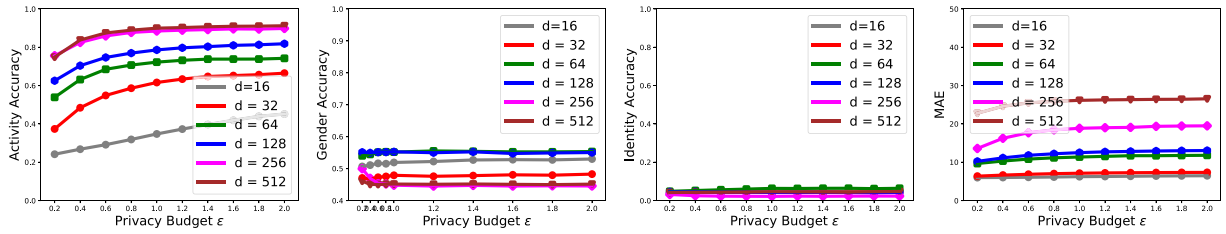
6.2. Effect of adaptive sampling

To exhibit the advantage of adaptive sampling, we compare it with fixed sampling intervals 5, 50, and 100. Specifically, for the adaptively sampling, the parameters K_p, K_i, K_d, ρ and η of PID control in Eq. (10)(11) are set as 0.8, 0.05, 0.1, 2, 0.8, respectively. For the fixed sampling intervals, we first sample the IoT data with assigned fixed sample intervals, and then segment sampled data by time windows $w = 50$. The dimension of features is $d = 512$. The useful and sensitive inferences accuracies are shown in Fig. 7. From this figure, we have the following observations: (1) a lower sampling interval corresponds to a higher activity inference accuracy, since they have stronger temporal correlations and thus exhibit more accurate physical attributes. However, a low sampling interval would generate more data samples and thus result in higher system processing delays. (2) Adaptive sampling has similar performance with sampling interval 5, but has better performance than sampling intervals 50 and 100. It means that adaptive sampling is a better choice, especially when then data distribution is unknown. (3) Comparing with the original inference accuracies of three tasks in Table 3, the useful inference (act) accuracy has few decreases, while the sensitive inference (gender and identity) accuracies have massive recessions. This phenomenon proves that PrivStream can effectively protect sensitive information while preserving useful information.

6.3. Effect of encoder–decoder filter

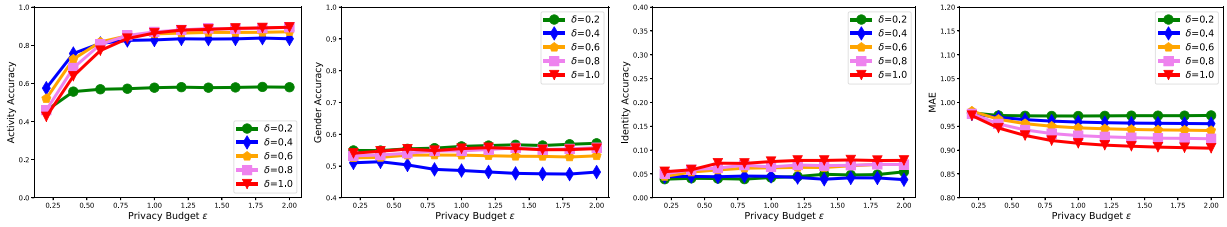
To evaluate the effect of the encoder–decoder filter, we first analyze the useful and sensitive inference accuracies and the MAEs of the outputs of PrivStream-MLP and PrivStream-RNN under various dimensions of features $\boldsymbol{z}$ and different perturbation level ϵ in Figs. 8 and 9. Then we compare the results to analyze the performances of two types of models. The x-axis denotes the privacy budget, and the y-axes stand for inference accuracies and MAE, respectively. Note that, in this evaluation, we only perturb features in the inference phase. For the PrivStream-MLP, the window size is fixed as 50, while for the PrivStream-RNN, since the data segment is length-variable, the data segments have three sizes with 30, 40, and 50.

The performance of PrivStream-MLP is shown in Fig. 8. We vary the extracted feature dimension d from 256 to 8, and the results under different dimensions and privacy budget privacy ϵ show that: (1) A higher privacy budget corresponds to a larger activity accuracy and lower MAE. This phenomenon is consistent with the properties of DP in Definition 1. (2) The inference results of gender and identity under different dimensions and different privacy budgets are close to the random guesses (1/2 and 1/24, respectively). This demonstrates that the sensitive information within the extracted features is low enough. (3) As the feature dimension decreases, the data utility would also go down, i.e., a lower activity accuracy and higher MAE, because more information is lost. But we also note that MAE has a decrease when the feature dimension reduces from 256 to 128, which is contrary to the law. We suspect that the excessive noise introduced by high dimensions has a greater effect on utility than dimension reduction. According to the above analyses, we conclude that the largest feature dimension is not always the best choice. To trade off the utility and privacy, PrivStream should choose a proper feature dimension and a privacy budget. We set the feature dimension as 128 in the following experiments, as it has the best performance than others. For example, when $\epsilon = 5$, the activity accuracy at $d = 128$ is 0.86, which is close to



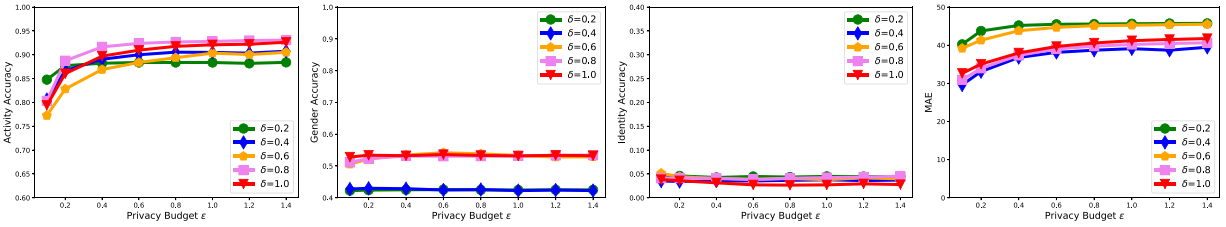
(a) Activity (b) Gender (c) Identity (d) MAE

Fig. 9. Effect of encoder–decoder filter of PrivStream-RNN.



(a) Activity (b) Gender (c) Identity (d) MAE

Fig. 10. Effect of noisy training of PrivStream-MLP.



(a) Activity (b) Gender (c) Identity (d) MAE

Fig. 11. Effect of noisy training of PrivStream-RNN.

0.89 that at $d = 256$, but the MAE at $d = 128$ is 0.65, which is lower than 0.69 that at $d = 256$.

Fig. 9 exhibits the results of PrivStream-RNN, which have similar performances with PrivStreram-MLP, in which the useful inference accuracy increases with the increment of privacy budget ϵ and feature dimension d, while the sensitive inference accuracies are almost unchanged with them, and the MAE drops sharply as the dimension decreases. But we also notice some differences: (1) Under the same feature dimension and same perturbation level, the useful inference accuracy of PrivStream-RNN is higher than PrivStream-MLP. For example, when $\epsilon = 1$ and $d = 256$, the activity inference accuracy of PrivStream-MLP is 0.52, while that of PirvStream-RNN is 0.88. This indicates that PrivStream-RNN is more robust to noise than PrivStream-MLP. (2) The MAE of PrivStream-RNN is much larger than that of PrivStream-MLP. This is because PrivStream-RNN is the serial structure, which would cause a larger disorder of data than the parallel structure of PrivStream-MLP. Even so, the high useful inference accuracy of PrivStream-RNN suggests that PrivStream-RNN is more likely to retain useful information than the data value.

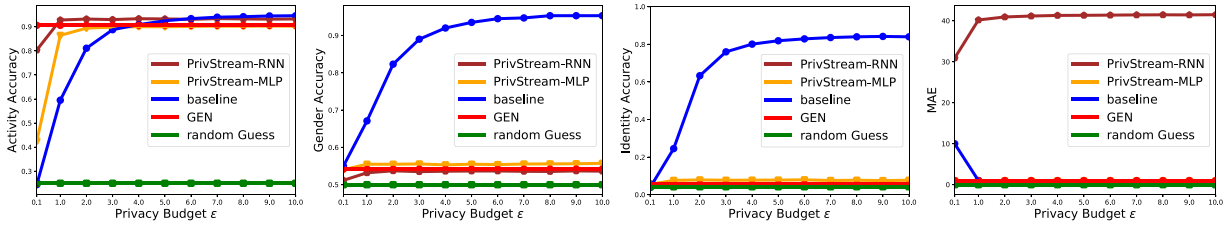
6.4. Effect of noisy training

In order to increase the robustness of PrivStream to perturbation, we also inject noise that is scaled by privacy budget δ to the extracted features in the training phase. The inference accuracies and MAE under different (δ, ϵ) pairs are shown in Figs. 10–11. The feature dimensions of PrivStream-MLP and PrivStream-RNN are 256 and 512, respectively.

Comparing with the experiments without noisy training in Figs. 8–9 under the same feature dimension and privacy budget ϵ, the most obvious observation is that, the useful inference accuracy has a great increment. For example, in Figs. 8–9, the activity accuracies of PrivStream-MLP (d = 256, $\epsilon = 1$) and PrivStream-RNN (d = 512, $\epsilon = 0.2$) are 0.53 and 0.75, respectively, while that in Figs. 10–11 under $(\delta, \epsilon) = (1, 1)$ and $(\delta, \epsilon) = (0.8, 0.2)$ are 0.86 and 0.93. Another observation is that the sensitive inference accuracies have no significant changes, but the MAEs have slight growths, which increase by 0.21 and 11.18 for PrivStream-MLP and PrivStream-RNN, respectively. Even so, the noisy training still has a positive effect on the performance of PrivStream because our primary goal is to preserve the inference accuracy of the target task.

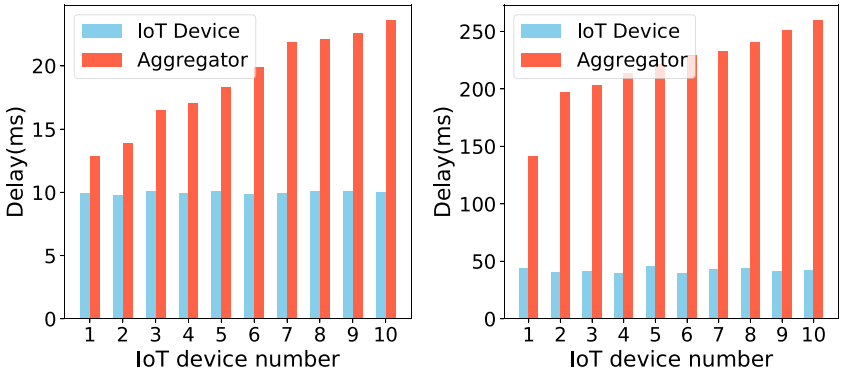
6.5. Trade-off between privacy and utility

In this subsection, we compare PrivStream-MLP and PrivStream-RNN with a baseline method and a state-of-the-art algorithm GEN [4]. The privacy budgets δ in the training phase and the feature dimensions d of PrivStream-MLP and PrivStream-RNN are 1, 256 and 0.8, 512, respectively. From the results shown in Fig. 12, we have the following observations: (1)when the $\epsilon = 0.1$ (a very large perturbation), GEN has the highest useful inference accuracy, because it does not inject noise into the features, but as ϵ increases to 1 (a typical value in differential privacy [16,25]), PrivStream-RNN has the best performance, and when ϵ increases to 3, PirvStream-MLP has the same performance as GEN. This demonstrates that PrivStream can preserve data utility while providing greater privacy protection. (2) The baseline method



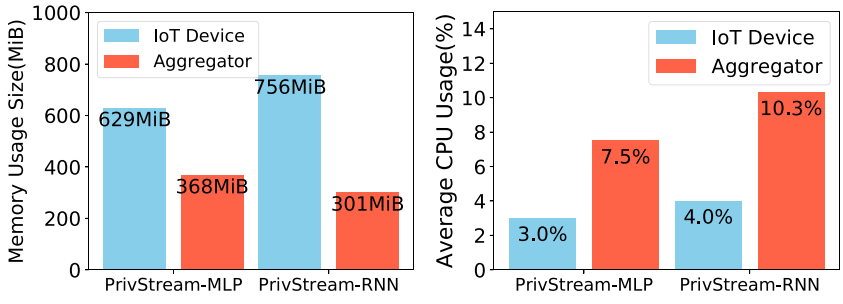
(a) Activity (b) Gender (c) Identity (d) MAE

Fig. 12. The trade-off between data utility and privacy.



(a) PrivStream-MLP (b) PrivStream-RNN

Fig. 13. The latency at IoT device and aggregator (the edge node).



(a) Storage overhead (b) CPU consumption

Fig. 14. (a) Storage overhead on IoT device and aggregator (the edge node); (b) CPU consumption of IoT device and aggregator (the edge node) at loading and running stages, respectively.

has higher sensitive inference accuracies than other methods when $\epsilon > 0.1$. This shows that directly adding noise into the stream without feature filtering cannot prevent sensitive inferences. (3) The MAE of PrivStream-RNN is much larger than other methods, this suggests that PrivStream-RNN is appropriate for the situation that the user wants to hide real data values, while PrivStream-MLP is suitable for preserving the original data value. But both of them can effectively prevent sensitive inferences and provide high inference accuracy of the target task.

6.6. Computation efficiency

The computation efficiency is quantified by the execution time of PrivStream on both the IoT device side and edge node. Each IoT device independently carries out the sampling, feature extraction, and perturbation operations, and the perturbed features from all IoT devices are aggregated on the edge node for reconstruction and the target task inference. From the results in Fig. 13, we can observe that (1) The latency on IoT devices are not affected by the number of IoT device because they are independent of each other. (2) The computation latency of the aggregator (the edge node) increases with the increment of the IoT device number. This is because the aggregator needs more time to process the accumulated data. (3) The delay of PrivStream-RNN is much larger than PrivStream-MLP, as the runtime of the RNN model is longer than the MLP model.

6.7. Storage overhead

The storage overhead and CPU consumption of PrivStream are evaluated in Fig. 14(a) and Fig. 14(b), respectively. From the figures, we can see that PrivStream-RNN takes up higher storage space in IoT devices (Raspberry Pi) but lower space in aggregator (the edge node)

than PrivStream-MLP. In addition, PrivStream-RNN takes up 7.5% and 10.3% CPU cycles of the IoT device and edge node, which are slightly higher than 3% and 7.5% of PrivStream-MLP. Even so, both of them are still within an acceptable range.

7. Related work

7.1. EI with differential privacy

EI is a major trend in the era of the Internet of Everything, due to the powerful ability of AI to quickly analyze the data with huge volumes generated from surrounding IoT devices. But those data may contain private information, such as location, health, gender, and so on. Thus, distributed learning becomes a feasible paradigm, which jointly trains a model while keeping the training data in the local devices. Nevertheless, there is still a risk of privacy leakage in the data exchange between local devices and the untrusted server. To solve this problem, many studies [21,26,27] have combined DP with distributed learning. Shokri et al. [28] proposed a system that enables joint training such that the input data are kept in the local devices and only the model parameters are shared. The shared parameters are further protected by differential privacy before being aggregated in the centralized parameter server. But Abadi et al. [21] found that the total privacy loss in Ref. [28] would be very large. To reduce the total privacy loss, they proposed a new differential privacy based stochastic gradient descent algorithm. In these two works, gradients are the perturbation object. Besides the gradient perturbation, noise can also be added to the objective function [29] or the output [30]. For example, Phan et al. [26] enforced ϵ-differential privacy by perturbing the objective functions of the traditional deep auto-encoder. Jayaraman et al. [31] added noise to the output inside the secure computation after aggregation, and showed that the noise can be reduced in the multi-party setting. However, these methods applied differential privacy in the training stage to prevent EI from learning private information from the data. They are unable to achieve the goal of PrivStream, which is to prevent a malicious model from inferring sensitive attributes.

7.2. Defenses against sensitive inference

Sensitive inferences, which mean inferring private attributes from insensitive public data, have been demonstrated that exist in a variety of applications, such as recommender systems [32], social media [33–35], and mobile applications [36,37]. The methods that defend against those attacks can be roughly divided into three categories.

Correlation-based perturbations, which added perturbations to the items that are most relevant to the private attributes. For example, Weinsberg et al. [32] used a logistic regression classifier to learn the correlations between items and private attributes, and then perturb the top k items with the largest correlations. Chen et al. [38] computed the correlations based on chi-square statistics. Jia et al. [39] proposed AttriGuard, which consisted of two phases. It first found a minimum noise for each attribute value, and then randomly selected one of the noises to mislead the attacker's inference. The limitations of these methods are low utility or high computational overhead.

Feature extraction, which filters sensitive information by extracting features only related to useful information. For example, Malekzadeh et al. [4,15] propose a framework, called GEN, to filter the sensitive attributes and simultaneously preserve the useful attributes. Hamm et al. [9] proposed a minimax filter to filter sensitive information. However, most of these works are carried out in an edge server or cloud server with an implicit assumption that the edge/cloud server is trusted, which may be violated in reality. To solve this problem, some works propose to combine feature extraction with differential privacy.

Differential privacy [25,40], which protects a data record by locally injecting noise to guarantee that the adversary cannot distinguish this data record from its neighbor by observing the perturbed outputs. However, since streaming data has high dimensions, directly applying differential privacy to the streaming data would incur too much noise on the private attributes. Fortunately, after the feature extraction, the dimension of the original data is greatly reduced, and then adding differential privacy noise can not only protect the privacy from being leaked, but also effectively retain the data utility. Wang et al. [7] propose a framework enabling deep learning on mobile devices, which offloads the network training and complex inference to the cloud centers, and applies differential privacy at the local neural network to protect the privacy. Refs. [10,20] offline pre-train an autoencoder, and then separate the model into encoder and decoder that deployed at the local and cloud server, respectively. Noise is injected into the output of encoder at the local side. In this work, we also use feature extraction and differential privacy However, the difference is that our proposed framework can process variable-length streaming data and realize real-time transmission and fast processing, whereas the previous works only focus on batch data with fixed length.

8. Conclusion

In this paper, we propose a privacy-preserving IoT streaming data inference framework based on edge computing, called PrivStream, to prevent sensitive inferences while enabling the accurate target inference task. We realize PrivStream with MLP model and RNN model to process data with fixed length and variable length, respectively. To effectively utilize the computing resources and defense against the untrusted edge server, PrivStream is separated into two parts that are deployed on the IoT device side and the edge server, respectively. The IoT device side carries out the adaptive sampling, feature extraction, and perturbation operations, which provides two-layer privacy protection for the streaming data to effectively prevent the untrusted edge server from inferring any sensitive information of users. The edge server executes the data reconstruction for the target task inference with high accuracy. To realize the real-time transmission of streaming data between IoT devices and the edge server, PrivStream is implemented on a distributed streaming platform. In order to increase the robustness of PrivStream to noise, we also inject noise into the framework in the training phase. We theoretically prove that PrivStream can achieve ϵ-DP. Extensive experiments demonstrate that PrivStream has better performance than the state-of-the-art and has acceptable computation and storage overheads.

CRediT authorship contribution statement

Dan Wang: Conceptualization, Methodology, Software, Writing – original draft. **Ju Ren:** Supervision, Writing – review & editing, Funding acquisition. **Zhibo Wang:** Writing – review & editing. **Yaoxue Zhang:** Supervision, Funding acquisition. **Xuemin (Sherman) Shen:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was supported in part by the National Natural Science Foundation of China under Grant No. 62122095, 62072472 and U19A2067, Natural Science Foundation of Hunan Province, China under Grant No. 2020JJ2050, 111 Project under Grant No. B18059, and the Young Talents Plan of Hunan Province of China under Grant No. 2019RS2001.

References

[1] J. Ren, H. Guo, C. Xu, Y. Zhang, Serving at the edge: A scalable iot architecture based on transparent computing, IEEE Netw. 31 (5) (2017) 96–105.

[2] F. Lyu, J. Ren, N. Cheng, P. Yang, M. Li, Y. Zhang, X. Shen, LEAD: Large-scale edge cache deployment based on spatio-temporal wifi traffic statistics, IEEE Trans. Mob. Comput. (2020) 1–16.

[3] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, J. Zhang, Edge intelligence: Paving the last mile of artificial intelligence with edge computing, 2019, arXiv preprint arXiv:1905.10083.

[4] M. Malekzadeh, R.G. Clegg, A. Cavallaro, H. Haddadi, Protecting sensory data against sensitive inferences, 2018, arXiv preprint arXiv:1802.07802.

[5] J. Zhou, Z. Cao, X. Dong, X. Lin, Security and privacy in cloud-assisted wireless wearable communications: Challenges, solutions, and future directions, IEEE Wirel. Commun. 22 (2) (2015) 136–144.

[6] S.K. Pasupuleti, S. Ramalingam, R. Buyya, An efficient and secure privacy-preserving approach for outsourced data of resource constrained mobile devices in cloud computing, J. Netw. Comput. Appl. 64 (2016) 12–22.

[7] J. Wang, J. Zhang, W. Bao, X. Zhu, B. Cao, P.S. Yu, Not just privacy: Improving performance of private deep learning in mobile cloud, in: Proc. of KDD, ACM, 2018, pp. 2407–2416.

[8] N. Apthorpe, D. Reisman, N. Feamster, A smart home is no castle: Privacy vulnerabilities of encrypted iot traffic, 2017, arXiv preprint arXiv:1705.06805.

[9] J. Hamm, Preserving privacy of continuous high-dimensional data with minimax filters, in: Artificial Intelligence and Statistics, 2015, pp. 324–332.

[10] C. Liu, S. Chakraborty, P. Mittal, Deeprotect: Enabling inference-based access control on mobile sensing applications, 2017, arXiv preprint arXiv:1702.06159.

[11] J. Hamm, Minimax filter: learning to preserve privacy from inference attacks, J. Mach. Learn. Res. 18 (1) (2017) 4704–4734.

[12] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.

[13] C. Dwork, Differential privacy: A survey of results, in: International Conference on Theory and Applications of Models of Computation, Springer, 2008, pp. 1–19.

[14] Y. Chen, A. Machanavajjhala, M. Hay, G. Miklau, Pegasus: Data-adaptive differentially private stream processing, in: Proc. of CCS, ACM, 2017, pp. 1375–1388.

[15] Q. Wang, Y. Zhang, X. Lu, Z. Wang, Z. Qin, K. Ren, RescueDP: Real-time spatio-temporal crowd-sourced data publishing with differential privacy, in: Proc. of INFOCOM, 2016, pp. 1–9.

[16] L. Fan, L. Xiong, Real-time aggregate monitoring with differential privacy, in: Proc. of CIKM, ACM, 2012, pp. 2169–2173.

[17] W.-H. Lee, R.B. Lee, Multi-sensor authentication to improve smartphone security, in: Proc. of ICISSP, IEEE, 2015, pp. 1–11.

[18] S. Li, L. Da Xu, X. Wang, Compressed sensing signal and data acquisition in wireless sensor networks and internet of things, IEEE Trans. Ind. Inf. 9 (4) (2013) 2177–2186.

[19] G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.

[20] D. Wang, J. Ren, C. Xu, J. Liu, Z. Wang, Y. Zhang, X. Shen, Privstream: Enabling privacy-preserving inferences on IoT data stream at the edge, in: Proc. of HPCC, IEEE, 2019, pp. 1290–1297.

[21] M. Abadi, A. Chu, I. Goodfellow, H.B. McMahan, I. Mironov, K. Talwar, L. Zhang, Deep learning with differential privacy, in: Proc. of CCS, ACM, 2016, pp. 308–318.

[22] Spark Streaming http://spark.apache.org/streaming/.

[23] Apache Storm https://storm.apache.org/.

[24] Apache Kafka https://kafka.apache.org/.

[25] G. Kellaris, S. Papadopoulos, X. Xiao, D. Papadias, Differentially private event sequences over infinite streams, Proc. VLDB Endow. 7 (12) (2014) 1155–1166.

[26] N. Phan, Y. Wang, X. Wu, D. Dou, Differential privacy preservation for deep auto-encoders: an application of human behavior prediction, in: AAAI, Vol. 16, 2016, pp. 1309–1316.

[27] Z. Bu, J. Dong, Q. Long, W.J. Su, Deep learning with Gaussian differential privacy, 2019, arXiv preprint arXiv:1911.11607.

[28] R. Shokri, V. Shmatikov, Privacy-preserving deep learning, in: Proc. of CCS, ACM, 2015, pp. 1310–1321.

[29] N. Phan, X. Wu, H. Hu, D. Dou, Adaptive laplace mechanism: Differential privacy preservation in deep learning, in: Proc. of ICDM, IEEE, 2017, pp. 385–394.

[30] M.A.P. Chamikara, P. Bertok, I. Khalil, D. Liu, S. Camtepe, M. Atiquzzaman, Local differential privacy for deep learning, 2019, arXiv preprint arXiv:1908.02997.

[31] B. Jayaraman, L. Wang, D. Evans, Q. Gu, Distributed learning without distress: Privacy-preserving empirical risk minimization, in: Proc. of NeurIPS, 2018, pp. 6343–6354.

[32] U. Weinsberg, S. Bhagat, S. Ioannidis, N. Taft, Blurme: Inferring and obfuscating user gender based on ratings, in: Proc. of RecSys, ACM, 2012, pp. 195–202.

[33] N.Z. Gong, B. Liu, You are who you know and how you behave: Attribute inference attacks via users' social friends and behaviors, in: Proc. of USENIX, 2016, pp. 979–995.

[34] J. Jia, B. Wang, L. Zhang, N.Z. Gong, AttriInfer: Inferring user attributes in online social networks using markov random fields, in: Proc. of WWW, 2017, pp. 1561–1569.

[35] N.Z. Gong, B. Liu, Attribute inference attacks in online social networks, ACM Trans. Privacy Secur. 21 (1) (2018) 1–30.

[36] L. Lerman, G. Bontempi, O. Markowitch, Side channel attack: an approach based on machine learning, Center for Advanced Security Research Darmstadt (2011) 29–41.

[37] S. Narain, T.D. Vo-Huu, K. Block, G. Noubir, Inferring user routes and locations using zero-permission mobile sensors, in: Proc. of S&P, IEEE, 2016, pp. 397–413.

[38] T. Chen, R. Boreli, M.-A. Kaafar, A. Friedman, On the effectiveness of obfuscation techniques in online social networks, in: Proc. of PETS, Springer, 2014, pp. 42–62.

[39] J. Jia, N.Z. Gong, Attriguard: A practical defense against attribute inference attacks via adversarial machine learning, in: Proc. of USENIX, 2018, pp. 513–529.

[40] Z. Wang, X. Pang, Y. Chen, H. Shao, Q. Wang, L. Wu, H. Chen, H. Qi, Privacy-preserving crowd-sourced statistical data publishing with an untrusted server, IEEE Trans. Mob. Comput. (2018).