SCIENCE CHINA
Information Sciences

• **Supplementary File** •

FUSE: A Federated Learning and U-shape Split Learning-based Electricity Theft Detection Framework

Xuan LI[1], Naiyu WANG[1], Liehuang ZHU[2], Shuai YUAN[3] & Zhitao GUAN[1*]

[1]*School of Control and Computer Engineering, North China Electric Power University, Beijing* 102206*, China;*
[2]*School af Cyberspace Science and Technology, Beijing Institute of Technology, Beijing* 100081*, China;*
[3]*Department of Finance, Operations, and Information Systems (FOIS), Brock University, St.Catharines* L2S 3A1*, Canada*

Appendix A Preliminary

Appendix A.1 Federated Learning

Federated learning is a method of distributed machine learning where a global model is trained collaboratively across multiple clients, coordinated by a central server [1]. A vanilla federated learning typically consists of four steps: **Initialization**, **Local Training**, **Model Aggregation**, and **Model Update**.

• **Initialization:** During this phase, participating clients (e.g., user devices) reach a consensus on training a high-performing model using data distributed across the clients. An initial global model is created on the server, which serves as the beginning of the federated learning cycle. Each client then downloads the initial global model as their local model for the first round.

• **Local Training:** Each client trains their local model using their own data following the predefined hyperparameters and training procedures. Then after a set number of iterations, either the local model parameters or the model updates are sent to the server by the clients.

• **Model Aggregation:** Upon receiving all local model parameters or updates, the server aggregates them to generate an updated global model for the next round of global communication.

• **Model Update:** All clients download the newly aggregated global model to update their local models. The cycle of Local Training, Model Aggregation, and Model Update repeats iteratively until the global model achieves convergence or meets the predefined number of communication rounds.

Based upon the aggregation strategy on the server, federated learning can be categorized into synchronous, asynchronous, and semi-asynchronous federated learning [2]. In synchronous federated learning, the server waits to aggregate the global model until it has received all local models from the clients, as discussed above [1]. To improve efficiency, asynchronous federated learning enables the clients to submit their local models asynchronously, with the server updating the global model upon each receipt [3]. However, the high communication overhead in the asynchronous aggregation strategy and the extended waiting periods in the synchronous aggregation strategy have led to the development of semi-asynchronous federated learning [4–6]. This approach stores early-arriving local models and aggregates them within a specific timeframe. Furthermore, local models collected within this timeframe may come from different communication rounds and are aggregated with variable weights. Thus it effectively combines the advantages of both synchronous and asynchronous aggregation strategies.

Appendix A.2 Split Learning

Split learning is designed to minimize the substantial computational overhead while preserving the privacy of the raw data [7–9]. In this context, two distinct roles are involved: the client, typically with limited computational capability, and the server, introduced to alleviate the computational burden of the client. Specifically, in a basic split learning configuration, a neural network with a large number of parameters is partitioned into two sections. The lower layers, closer to input, are assigned to the client, while the remaining layers are managed by the server. This division occurs at a designated "cut layer". The client trains the lower layers using its data and submits the output from the cut layer to the server. The server then completes the forward propagation process without needing to access the raw data of the client. Backpropagation begins at the server, with the gradients of the cut layer transmitted back to the client for subsequent steps. This process repeats until the model achieves the desired performance level. Through split learning, the bulk of the model training computation is performed by the server with sufficient computational resources. Meanwhile, the client retains the raw data locally, ensuring its privacy throughout the entire training process.

Appendix B Problem Formulation & Threat Model

Appendix B.1 Problem Formulation

Consider M transformer districts $\mathrm{TD} = \{\mathrm{TD}_i | i \in [1, M]\}$ in different energy service providers (ESP) are involved in the federated learning process to train a theft detection model jointly. For each TD, there are N smart meters SM_i^j, where j is from 1 to N. SM_i^j maintains the daily consumption data x_i^j of a given consumer. y_i^j represents the consumer label of x_i^j, denoting whether or not the theft behavior exists on SM_i^j. Thus, it is assigned and managed by TD_i. The local dataset of TD_i, therefore, can be represented

* Corresponding author (email: guan@ncepu.edu.cn)

as $D_i = \left\{ \left(x_i^j, y_i^j \right) \mid i \in [1, M], j \in [1, N] \right\}$ with the size of $|D_i|$. The theft detection model is then defined as $\hat{y} = f(w, x)$, where $\hat{y}$ denotes the prediction of the model, w denotes the global model parameter, and x denotes the model input, i.e. electricity consumption data. Accordingly, the loss function of TD_i is represented as $\mathcal{L}_i = \frac{1}{|D_i|} \sum_{j \in [1,N]} \ell \left(\hat{y}_i^j, y_i^j \right)$, where $\ell \left(\hat{y}_i^j, y_i^j \right)$ is the loss function for data point $\left(x_i^j, y_i^j \right)$ and $\ell(\cdot) \in (0, +\infty)$. As a result, the global loss function is defined as:

$$F(w) = \sum_{i=1}^{M} \frac{|D_i|}{S} \mathcal{L}_i \tag{B1}$$

where $S = \sum |D_i|$.

Therefore, the final objective of the framework is to solve the optimization problem as follows:

$$w^* = \arg\min F(w). \tag{B2}$$

where w^* denotes the optima of the global parameter.

Meanwhile, the straggler issue arises due to variations in computation and communication resources among TDs from different ESPs. Consequently, TDs send their local models asynchronously, resulting in the global model being updated based on the aggregation of local models trained in different rounds. Note that w_i^k denotes the local model trained by TD_i based on the global model in round k, w^r denotes the global model in round r, and $\mathcal{A}$ denotes an aggregation algorithm. Thus the update of the global model in round r can be expressed as $w^r = w^{r-1} + \mathcal{A}(w_i^k | i \in [1, M], k \in [0, r-1])$. We also define the staleness of w_i^k as $\tau = r - k$.

Appendix B.2 Threat Model and Objectives

The threat model considered in this paper addresses both internal and external issues as follows:

Internal Threats: We assume that the assisted cloud server and federated learning aggregator are honest-but-curious, i.e., they will perform their tasks honestly but may attempt to infer sensitive information from the raw data through model updates.

External Threats: The adversary may attempt to obtain sensitive data, such as the consumer consumption data or information regarding electricity theft, either directly or indirectly by eavesdropping on the communication link between the entities in our framework.

Therefore, our framework is designed with the following objectives in mind:

(1) Lightweight model training on TDs and SMs.

(2) Accommodation of asynchronous model submissions from heterogeneous clients without compromising model performance.

(3) Preservation of training data, including the consumption data as input in SMs and the consumer labels in TDs, at the location where it is generated to the greatest extent possible.

(4) Prevention of the assisted cloud, aggregator, and eavesdropper from inferring sensitive information throughout the entire process.

Appendix C The Proposed Framework

Appendix C.1 Entities in FUSE

Four entities participate in the FUSE framework:

• **Smart Meter (SM):** The SM collects and stores the electricity consumption data, i.e., the input of the model. Due to the constrained computational capacity and communication bandwidth, only the Feature Extractor is allocated to a given SM and the forward propagation of Feature Extractor is executed by SM. In each local iteration, the output of Feature Extractor is sent to the associated TD.

• **Transformer District (TD):** The TD stores the label on whether or not the theft behavior exists on a given SM in the region. Thus, the Feature Classifier is allocated to each TD in the training process. Meanwhile, to reduce the computational requirements of SM, Feature Extractor is also assigned to each TD such that backpropagation and parameter updates can also be performed.

• **Cloud Server (CS):** The CS is assumed to have sufficient computational capacity and communication bandwidth. Note that there is a 1:1 mapping of each TD to a CS logically, where the main component of the detection framework, i.e., Feature Learner, is allocated to perform local model training. As a result, a local theft detection model in federated learning consists of the Feature Extractor in the TD (or SM), the Feature Classifier in the TD, and the Feature Learner in the TD's corresponding CS.

• **Aggregator:** To address additional privacy concerns, the ciphertext of the serialized model parameters is submitted such that the global model is securely aggregated by the Aggregator. This aggregation process can be performed by the parent company of ESPs.

Appendix C.2 Table C1: Key Notations

Table C1 Key Notations

Notation	Definition
TD_i	The ith Transformer District
SM_i^j	The jth Smart Meter in the ith Transformer District
CS_i	The Cloud Server which completes the local model training for TD_i
x_i^j	The electric consumption data collected by SM_i^j
y_i^j	The consumer label of x_i^j which is assigned and managed by TD_i
D_i	The dataset on TD_i
w_{FE},w_{FL},w_{FC}	The parameter of the Feature Extractor, the Feature Learner, and the Feature Classifier
$v_{\mathrm{TD}_i}^k$, $v_{\mathrm{CS}_i}^k$	The serialized local model parameter of TD_i and CS_i in round k

v_{TD}^{k} , v_{CS}^{k}	The serialized global model parameter in round k
pk_{TD} , sk_{TD}	The public key and the private key of all TDs
pk_{CS} , sk_{CS}	The public key and the private key of all CSs
$\mathit{Buff}_{\mathrm{TD}}$	The buffer of serialized Feature Extractors and Feature Classifiers
$\mathit{Buff}_{\mathrm{CS}}$	The buffer of serialized Feature Learners
T	The maximum waiting time for a communication round
E	The number of local iterations

Appendix C.3 Algorithm C1: FUSE

Algorithm C1 FUSE

Input: $\mathrm{TD}_i \in \mathrm{TD}$, $\mathrm{CS}_i \in \mathrm{CS}$, global communication round R, max waiting time T

Output: $[\![v_{\mathrm{TD}}]\!]_{pk_{\mathrm{TD}}}$, $[\![v_{\mathrm{CS}}]\!]_{pk_{\mathrm{CS}}}$

▷ initialize the global model

Select TD_i from $\mathrm{TD} = \{\mathrm{TD}_1, \mathrm{TD}_2, \dots\}$ at random.

TD_i generates $(pk_{\mathrm{TD}}, sk_{\mathrm{TD}})$ and broadcasts to $\mathrm{TD}_i \in \mathrm{TD}$.

TD_i initializes w_{FE}^{0}, w_{FC}^{0} and sends $[\![v_{\mathrm{TD}}^{0}]\!]_{pk_{\mathrm{TD}}} = encrypt(Serialized(w_{\mathrm{FE}}, w_{\mathrm{FC}}), pk_{\mathrm{TD}})$ to the Aggregator.

Select CS_i from $\mathrm{CS} = \{\mathrm{CS}_1, \mathrm{CS}_2, \dots\}$ at random.

CS_i generates $(pk_{\mathrm{CS}}, sk_{\mathrm{CS}})$ and broadcasts to $\mathrm{CS}_i \in \mathrm{CS}$.

CS_i initializes w_{FL}^{0} and sends $[\![v_{\mathrm{CS}}^{0}]\!]_{pk_{\mathrm{CS}}} = encrypt(Serialized(w_{\mathrm{FL}}), pk_{\mathrm{CS}})$ to the Aggregator.

▷ global training process

for round $r = 1, 2, 3, \dots, R$ **do**

  The Aggregator broadcasts $[\![v_{\mathrm{TD}}^{r-1}]\!]_{pk_{\mathrm{TD}}}$ to $\mathrm{TD}_i \in \mathrm{TD}$, $[\![v_{\mathrm{CS}}^{r-1}]\!]_{pk_{\mathrm{CS}}}$ to $\mathrm{CS}_i \in \mathrm{CS}$.

  ▷ local training process

  for each $(\mathrm{TD}_i, \mathrm{CS}_i)$ in parallel **do**

    LocalSplitTraining($[\![v_{\mathrm{TD}}^{r-1}]\!]_{pk_{\mathrm{TD}}}$,$[\![v_{\mathrm{CS}}^{r-1}]\!]_{pk_{\mathrm{CS}}}$ ***)***

  end for

  ▷ initialize the local model buffer

  $\mathit{Buff}_{\mathrm{TD}} = \emptyset$, $\mathit{Buff}_{\mathrm{CS}} = \emptyset$, $j = 0$

  while not reaching the max waiting time T **do**

    The Aggregator retrieves $[\![v_{\mathrm{TD}_i}^{k}]\!]_{pk_{\mathrm{TD}}}$ and $[\![v_{\mathrm{CS}_i}^{k'}]\!]_{pk_{\mathrm{CS}}}$

    $cos_{\mathrm{TD}}^{j} = \dfrac{[\![v_{\mathrm{TD}_i}^{k}]\!]_{pk_{\mathrm{TD}}} \cdot [\![v_{\mathrm{TD}}^{r-1}]\!]_{pk_{\mathrm{TD}}}}{\left\|[\![v_{\mathrm{TD}_i}^{k}]\!]_{pk_{\mathrm{TD}}}\right\| \left\|[\![v_{\mathrm{TD}}^{r-1}]\!]_{pk_{\mathrm{TD}}}\right\|}$ ▷ calculate the cosine similarity of the model from TD

    $cos_{\mathrm{CS}}^{j} = \dfrac{[\![v_{\mathrm{CS}_i}^{k'}]\!]_{pk_{\mathrm{CS}}} \cdot [\![v_{\mathrm{CS}}^{r-1}]\!]_{pk_{\mathrm{CS}}}}{\left\|[\![v_{\mathrm{CS}_i}^{k'}]\!]_{pk_{\mathrm{CS}}}\right\| \left\|[\![v_{\mathrm{CS}}^{r-1}]\!]_{pk_{\mathrm{CS}}}\right\|}$ ▷ calculate the cosine similarity of the model from CS

    $\mathit{Buff}_{\mathrm{TD}} = \mathit{Buff}_{\mathrm{TD}} \cup ([\![v_{\mathrm{TD}_i}^{k}]\!]_{pk_{\mathrm{TD}}}, cos_{\mathrm{TD}}^{j}, i, k, j)$ ▷ add the model from TD to the buffer

    $\mathit{Buff}_{\mathrm{CS}} = \mathit{Buff}_{\mathrm{CS}} \cup ([\![v_{\mathrm{CS}_i}^{k'}]\!]_{pk_{\mathrm{CS}}}, cos_{\mathrm{CS}}^{j}, i, k', j)$ ▷ add the model from CS to the buffer

    $j = j + 1$

  end while

  ▷ perform aggregation process to obtain the new global model

  $[\![v_{\mathrm{CS}}^{r}]\!]_{pk_{\mathrm{CS}}} =$ ***Two-StageAggregation***$(\mathit{Buff}_{\mathrm{CS}}, [\![v_{\mathrm{CS}}^{r-1}]\!]_{pk_{\mathrm{CS}}}, r)$

  $[\![v_{\mathrm{TD}}^{r}]\!]_{pk_{\mathrm{TD}}} =$ ***Two-StageAggregation***$(\mathit{Buff}_{\mathrm{TD}}, [\![v_{\mathrm{TD}}^{r-1}]\!]_{pk_{\mathrm{TD}}}, r)$

end for

$[\![v_{\mathrm{TD}}]\!]_{pk_{\mathrm{TD}}} = [\![v_{\mathrm{TD}}^{r}]\!]_{pk_{\mathrm{TD}}}$, $[\![v_{\mathrm{CS}}]\!]_{pk_{\mathrm{CS}}} = [\![v_{\mathrm{CS}}^{r}]\!]_{pk_{\mathrm{CS}}}$

return $[\![v_{\mathrm{TD}}]\!]_{pk_{\mathrm{TD}}}$, $[\![v_{\mathrm{CS}}]\!]_{pk_{\mathrm{CS}}}$

Appendix C.4 Initialization

At the commencement of the training, an arbitrary $\mathrm{TD}_i \in \mathrm{TD}$ is selected to initialize the global parameter of the Feature Extractor w_{FE}^{0}, the global parameter of the Feature Classifier w_{FC}^{0}, and the public-private key pair $(pk_{\mathrm{TD}}, sk_{\mathrm{TD}})$ for homomorphic encryption. Then, $(pk_{\mathrm{TD}}, sk_{\mathrm{TD}})$ is broadcasted to all $\mathrm{TD}_i \in \mathrm{TD}$ through secure channels. Meanwhile, an arbitrary $\mathrm{CS}_i \in \mathrm{CS}$ is also selected to initialize the global parameter of the Feature Learner w_{FL}^{0} and the public-private key pair $(pk_{\mathrm{CS}}, sk_{\mathrm{CS}})$. Then $(pk_{\mathrm{CS}}, sk_{\mathrm{CS}})$ is

broadcasted to all $\mathrm{CS}_i \in \mathrm{CS}$ through secure channels. To ensure that the model parameters and structure are not disclosed to the Aggregator, each local update is submitted after the encryption by the public key of TD or CS and serialized. The serialization operation refers to the transformation from the origin parameter $w \in \mathbb{R}^{M\times N}$ to $v \in \mathbb{R}^{1\times MN}$, such that the structure of the model remains unknown. Furthermore, at the beginning of each communication round r, the serialized and encrypted model parameters $[\![v_{\mathrm{TD}}^{r-1}]\!]_{pk_{\mathrm{TD}}}$ and $[\![v_{\mathrm{CS}}^{r-1}]\!]_{pk_{\mathrm{CS}}}$ are broadcasted to all TD_i and CS_i respectively.

Appendix C.5 Three-tier U-shape Split Learning-based Local Training

Each local model is spilt and trained by different entities in our framework following Algorithm C2. In this section, we elaborate on the three-tier U-shape split learning-based local model training from the perspective of a given TD_i and its corresponding CS_i as follows.

Algorithm C2 LocalSplitTraining

Input: $[\![v_{\mathrm{TD}}]\!]_{pk_{\mathrm{TD}}}$,$[\![v_{\mathrm{CS}}]\!]_{pk_{\mathrm{CS}}}$

Output: $[\![v_{\mathrm{TD}_i}]\!]_{pk_{\mathrm{TD}}}$,$[\![v_{\mathrm{CS}_i}]\!]_{pk_{\mathrm{CS}}}$

▷ initialize the local model

TD_i: $v_{\mathrm{TD}} = decrypt([\![v_{\mathrm{TD}}]\!]_{pk_{\mathrm{TD}}}, sk_{\mathrm{TD}})$, $w_{\mathrm{FE}}, w_{\mathrm{FC}} = Deserialize(v_{\mathrm{TD}})$

CS_i: $v_{\mathrm{CS}} = decrypt([\![v_{\mathrm{CS}}]\!]_{pk_{\mathrm{CS}}}, sk_{\mathrm{CS}})$, $w_{\mathrm{FL}} = Deserialize(v_{\mathrm{CS}})$

$w_{\mathrm{FE}}^0 \leftarrow w_{\mathrm{FE}}$, $w_{\mathrm{FL}}^0 \leftarrow w_{\mathrm{FL}}$ $w_{\mathrm{FC}}^0 \leftarrow w_{\mathrm{FC}}$

▷ three-tier U-shape split learning process

for $e \in [1, 2, \ldots, E]$ **do**

  TD_i broadcasts w_{FE}^{e-1} to $\mathrm{SM}_i^j \in \mathrm{SM}_i$

  ▷ forward propagation process

  for $\mathrm{SM}_i^j \in \mathrm{SM}_i$ in parallel **do**

    $output_{\mathrm{FE}_j}^{e-1} \leftarrow ForwardPropagation(w_{\mathrm{FE}}^{e-1}, x_i^j)$

    send $output_{\mathrm{FE}_j}^{e-1}$ to TD_i

  end for

  TD_i: $output_{\mathrm{FE}}^{e-1} = Concatenate(output_{\mathrm{FE}_j}^{e-1}), j \in [1, \ldots, n]$, send $output_{\mathrm{FE}}^{e-1}$ to CS_i.

  CS_i: $output_{\mathrm{FL}}^{e-1} \leftarrow ForwardPropagation(w_{\mathrm{FL}}^{e-1}, output_{\mathrm{FE}}^{e-1})$

  CS_i: send $output_{\mathrm{FL}}^{e-1}$ to TD_i

  TD_i: $\hat{y} \leftarrow ForwardPropagation(w_{\mathrm{FC}}^{e-1}, output_{\mathrm{FL}}^{e-1})$, $L = \ell(\hat{y}, y)$ ▷ calculate the loss of the data

  ▷ backpropagation and local model update process

  TD_i: $g_{\mathrm{FC}} \leftarrow BackPropagation(\bigtriangledown L, w_{\mathrm{FC}}^{e-1})$ ▷ calculate the gradient of the FC's parameters

  TD_i: update Feature Classifier by $w_{\mathrm{FC}}^{e} \leftarrow w_{\mathrm{FC}}^{e-1} - \eta g_{\mathrm{FC}}$, send g_{FC}^*, the gradient of the first layer of w_{FC}^{e}, to CS_i

  CS_i: $g_{\mathrm{FL}} \leftarrow BackPropagation(g_{\mathrm{FC}}^*, w_{\mathrm{FL}}^{e-1})$ ▷ calculate the gradient of the FL's parameters

  CS_i: update Feature Learner by $w_{\mathrm{FL}}^{e} \leftarrow w_{\mathrm{FL}}^{e-1} - \eta g_{\mathrm{FL}}$, send g_{FL}^*, the gradient of the first layer of w_{FL}^{e}, to TD_i

  TD_i: $g_{\mathrm{FE}} \leftarrow BackPropagation(g_{\mathrm{FL}}^*, w_{\mathrm{FE}}^{e-1})$ ▷ calculate the gradient of the FE's parameters

  TD_i: update Feature Extractor by $w_{\mathrm{FE}}^{e} \leftarrow w_{\mathrm{FE}}^{e-1} - \eta g_{\mathrm{FE}}$

end for

TD_i: send $[\![v_{\mathrm{TD}_i}]\!]_{pk_{\mathrm{TD}}} = encrypt(Serialize(w_{\mathrm{FE}}, w_{\mathrm{FC}}), pk_{\mathrm{TD}})$ to the Aggregator

CS_i: send $[\![v_{\mathrm{CS}_i}]\!]_{pk_{\mathrm{CS}}} = encrypt(Serialize(w_{\mathrm{FL}}), pk_{\mathrm{CS}})$ to the Aggregator

Once $[\![v_{\mathrm{TD}}]\!]_{pk_{\mathrm{TD}}}$ is received, TD_i decrypts it by the private key sk_{TD}. And $v_{\mathrm{TD}} \in \mathbb{R}^{1\times MN}$ is deserialized to obtain parameters w_{FE} and w_{FC}. Similarly, $[\![v_{\mathrm{CS}}]\!]_{pk_{\mathrm{CS}}}$ is decrypted by sk_{CS} in CS_i and w_{FL} is obtained through deserialization on the vector.

In a given local iteration, TD_i first distributes w_{FE} to all SM_i^j in its region. Then for each SM_i^j, the consumption data is fed into the Feature Extractor where the forward propagation is carried out. Accordingly, the outputs of all Feature Extractors from the last layer are collected by TD_i and concatenated as the input to the Feature Learner in CS_i such that the forward propagation can be performed. The output of the Feature Learner is then fed into the Feature Classifier in TD_i where the predicted label is derived. In addition, the backpropagation is performed by feeding the gradient of the first layer of the partitioned model into the prior one. For instance, the gradient of the first layer of the Feature Classifier is sent from TD_i to CS_i. Hence, the parameters are updated once the gradients are calculated as follows:

$$w^e \leftarrow w^{e-1} - \eta g \tag{C1}$$

where w^e denotes the updated parameter in local round e, w^{e-1} denotes the parameter in last round, g denotes the gradient of weights, and η denotes the learning rate. It is worth noting that the backward propagation and the update of Feature Extractor are performed by TD_i, rather than SM_i^j, such that the risk of incurring high computational overhead on SMs is mitigated.

After a pre-determined number of local iteration rounds, the serialized split model in each entity is encrypted and sent to the Aggregator. Since parameters are updated ahead, the model can be submitted by CS_i without the necessity of waiting for TD_i such that the asynchronous problem between TD_i and its corresponding CS_i is addressed.

Appendix C.6 Two-stage Semi-asynchronous Aggregation

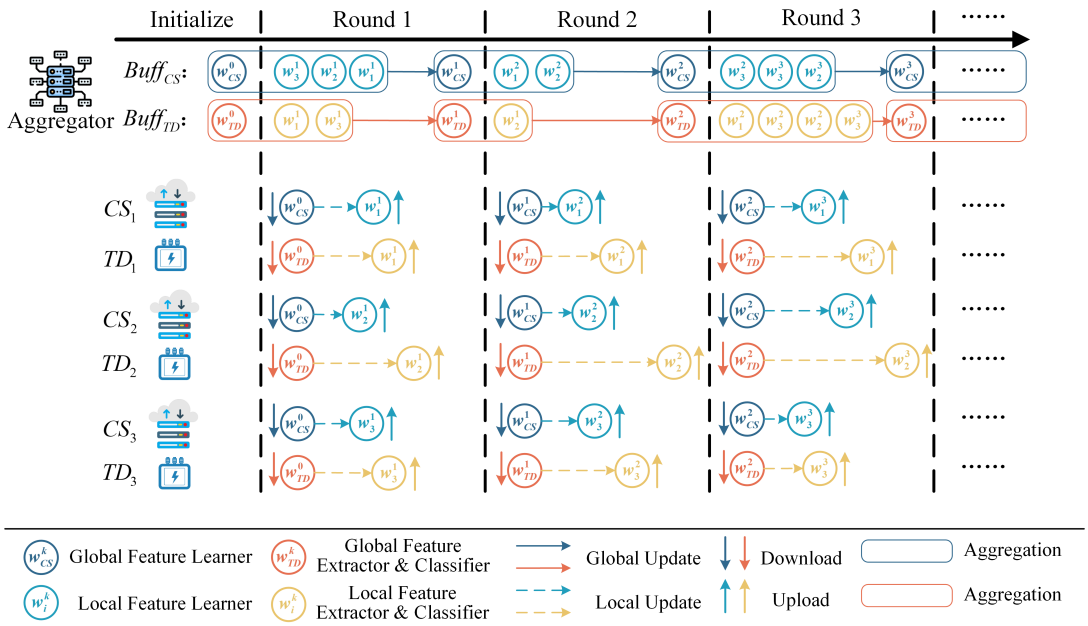


Figure C1 Semi-asynchronous aggregation, k denotes the number of communication round, i denotes index of TD or CS.

The updated model from different TDs and CSs may be received by the Aggregator asynchronously, as demonstrated in Figure C1. In classical federated learning models, either longer training time occurs in waiting for the stragglers, or the model performance is reduced because of the removals of such stragglers. On the other hand, while computation time can be reduced by asynchronous aggregation during the training process, extensively higher communication overhead is incurred on TD since each submission of a new local model necessitates the update of the global model. Therefore, a two-stage semi-asynchronous aggregation mechanism is proposed in our work to address these issues. Meanwhile, to guarantee that both the structure of the model and the plaintext of the parameters are not disclosed to the Aggregator, homomorphic encryption is implemented. Furthermore, considering the computation involved in the subsequent aggregation process, fully homomorphic encryption is employed. Hence the serialized and encrypted model parameters $[\![v^{r-1}]\!]_{pk}$ will be aggregated in each communication round. The two-stage semi-asynchronous aggregation is shown in Algorithm C3.

Algorithm C3 Two-StageAggregation

Input: $Buff$, $[\![v^{r-1}]\!]_{pk}$, r

Output: $[\![v^{r}]\!]_{pk}$

1: Sort $Buff$ in descending order of cos^j

2: $Buff \leftarrow Buff[0:m]$ ▷ only top-m models are retained in $Buff$

▷ pre-aggregate the models in the buffer

3: $[\![v^{new}]\!]_{pk} = \sum_{i,k,j \in Buff} \frac{cos^j}{\sum_{j \in Buff} cos^j} [\![v_i^k]\!]_{pk}$

4: $|\overline{D}| = \sum_{i,j \in Buff} \frac{cos^j}{\sum_{j \in Buff} cos^j} |D_i|$

5: $\overline{\tau} = \sum_{k,j \in Buff} \frac{cos^j}{\sum_{j \in Buff} cos^j} (r-k)$

▷ aggregate the local models to obtain the new global model

6: $[\![v^{r}]\!]_{pk} \leftarrow (1-\theta_r)[\![v^{r-1}]\!]_{pk} + \theta_r [\![v^{new}]\!]_{pk}$, where $\theta_r = \theta g(\overline{\tau}, |\overline{D}|)$, $g(\cdot)$ is a function of staleness and dataset size, respectively. $g(\overline{\tau}, |\overline{D}|) = \frac{1}{(1+\overline{\tau}+\frac{\sum |D_i|}{M|\overline{D}|})^{\frac{1}{e}}}$

7: return $[\![v^{r}]\!]_{pk}$

Once a local update is received, the model parameters are stored in the corresponding buffer, i.e., the models from TDs are stored in $Buff_{\mathrm{TD}}$ while those from CSs are stored in $Buff_{\mathrm{CS}}$. Taking the varying communication capabilities of different devices into account, models in the buffers with lower quality or higher variance from the global optima have a direct impact on the model performance. The cosine similarity between current local model and previous global model can effectively indicate the discrepancy in the convergence direction [10]. This similarity measure serves as a valuable tool to identify and filter out malicious models. Building upon this conclusion, we employ cosine similarity to evaluate the quality of the model, which provides the basis for obtaining the aggregation weights. Thus, before the aggregation with the global model from prior round, we pre-aggregate the

models in each buffer in terms of cosine similarity as follows:

$$cos = \frac{[\![v_i^k]\!]_{pk} \cdot [\![v^{r-1}]\!]_{pk}}{\left\|[\![v_i^k]\!]_{pk}\right\| \left\|[\![v^{r-1}]\!]_{pk}\right\|} \tag{C2}$$

such that the quality can be measured, where k represents the round from which the basic global model derived, r represents the current round, and the staleness of the model can be derived as $\tau = r - k$. To further improve the efficiency, only top-m models with the highest cos(the model with the $cos < 0$ will be dropped directly) will be pre-aggregated, thereby filtering out the models with low quality. This pre-aggregation is comprised of deriving the weighted average parameters, dataset size and the staleness regarding, as follows:

$$[\![v^{new}]\!]_{pk} = \sum_{i,k,j \in Buff} \frac{cos^j}{\sum_{j \in Buff} cos^j} [\![v_i^k]\!]_{pk} \tag{C3}$$

$$|\overline{D}| = \sum_{i,j \in Buff} \frac{cos^j}{\sum_{j \in Buff} cos^j} |D_i| \tag{C4}$$

$$\overline{\tau} = \sum_{k,j \in Buff} \frac{cos^j}{\sum_{j \in Buff} cos^j} (r - k) \tag{C5}$$

For the update of the global model, we define a hyper-parameter $\theta \in [0, 1]$. Therefore the final weight of the updated model θ_r is determined by the product of θ and $g(\cdot)$, which is a function of staleness and dataset size, i.e., $\theta_r = \theta g(\overline{\tau}, |\overline{D}|)$. $g(\cdot)$ is defined as follows where a smaller weight will be assigned to the model with larger staleness and smaller dataset size:

$$g(\overline{\tau}, |\overline{D}|) = \frac{1}{(1 + \overline{\tau} + \frac{\sum |D_i|}{M|\overline{D}|})^{\frac{1}{e}}} \tag{C6}$$

Hence, the global model is updated as:

$$[\![v^r]\!]_{pk} \leftarrow (1 - \theta_r)[\![v^{r-1}]\!]_{pk} + \theta_r [\![v^{new}]\!]_{pk} \tag{C7}$$

Note that all the algorithms including cosine similarity calculations and model aggregations, are performed in ciphertext because of the fully homomorphic encryption applied. In addition, since the comparisons in ciphertext can be approximated by addition and multiplication according to [11], the detailed operations on the ciphertext will not be disclosed in our framework.

Appendix D Experimental Evaluation

Appendix D.1 Experimental Setups

Implementation. All the experiments are carried out on Windows 10 operating system equipped with AMD Ryzen 3600 CPU, 16 GB RAM, and NVIDIA GeForce GTX 1660 SUPER. The federated learning algorithm is conducted on top of Pytorch. And the implementation of the homomorphic encryption for secure aggregation is based upon the library of TenSEAL.

Dataset. To evaluate the performance of FUSE on electricity theft detection tasks, we conduct experiments on a real-world dataset published by State Grid Cooperation of China (SGCC). The dataset consists of daily electricity consumption data collected by the SMs deployed in 42,372 electricity consumers' homes. For each consumer, it contains the data for 1,035 days from 2014 to 2016 where a regular consumer is labeled as 0 while the abnormal one is labeled as 1. There are 38,757 regular consumers and 3,615 abnormal consumers. Accordingly, the entire dataset is partitioned into a training set and a test set with a ratio of 9:1.

Theft Detection Model. In our work, to simulate the scenario wherein large-scale models are trained on those resource-constrained devices, we choose an attention-based theft detection model with over 50,000,000 parameters [12]. As mentioned in Appendix C, the theft detection model is split into the Feature Extractor, the Feature Learner, and the Feature Classifier, with 106,785 parameters, 50,808,706 parameters, and 2,050 parameters respectively. The local training process follows Algorithm C2.

Federated Setting. Three TDs and corresponding CSs are considered in our experiments. In addition, the number of SMs differs over these TDs, thereby resulting in different local models with varying sample sizes. Therefore, the data follows the quantity-based skew situation of non-IID [13].

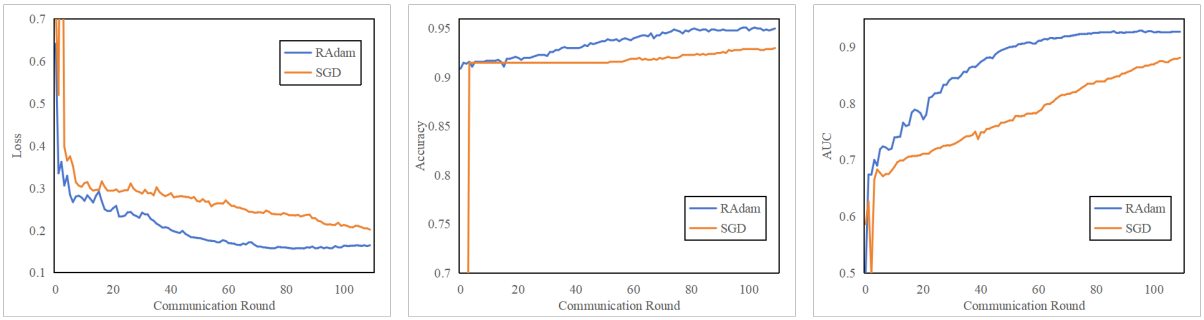
Evaluation Metrics. Due to the sample imbalance phenomenon in the dataset, the loss and accuracy (ACC) as the only evaluation criteria are not sufficient. Thus, Area under the ROC Curve (AUC), recall rate, F1 score, and the Matthews correlation coefficient (MCC) are also incorporated in our analysis.

Appendix D.2 Hyperparamater Learning

In this subsection, we explore the hyperparameters from local training to global aggregation, such that the optimized hyperparameters can be applied in following experiments on the performance evaluation of FUSE. For the purpose of simulating the asynchronous communications between TD, CS, and the Aggregator, the maximum staleness of model submission for TD is set to 6, and for CS is set to 4.

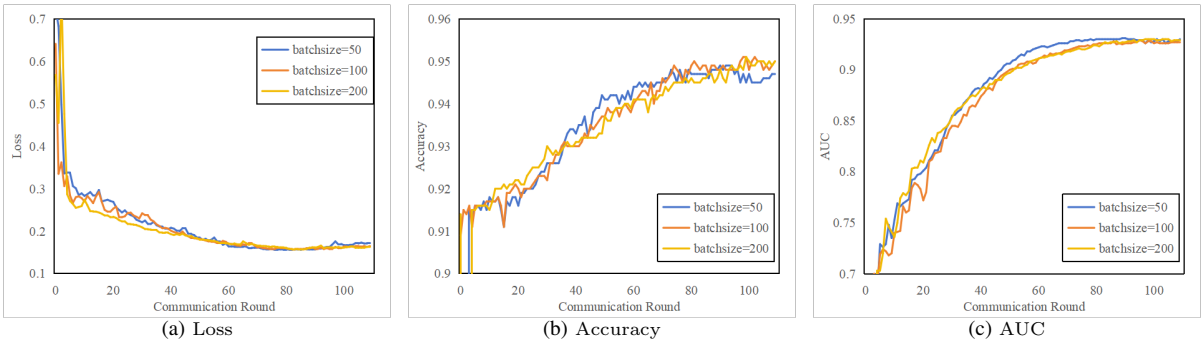
Appendix D.2.1 *Impact of Hyperparameter on Local Training*

1) Impact of local optimizer: In classical federated learning, Stochastic Gradient Descent (SGD) is the most commonly used local optimization algorithm [1]. However, RAdam is deployed in [12] instead of SGD in a centralized learning setting. We thereby first conduct an experiment to explore whether RAdam performs better than SGD in a federated learning setting. We set $batchsize = 100$, $m = 3$,and $\theta = 0.6$, where m denotes the maximum number of models in buffers that will be aggregated and θ denotes the basic aggregation weight of the new models in each round. The results are presented in both Figure D1 and Table D1. We verify that RAdam with a learning rate $\eta = 0.001$ outperforms SGD with a learning rate $\eta = 0.1$. Furthermore, RAdam has a faster convergence rate and better performance in terms of the criteria used in our experiments. Therefore, RAdam is chosen for subsequent experiments in the federated learning context.



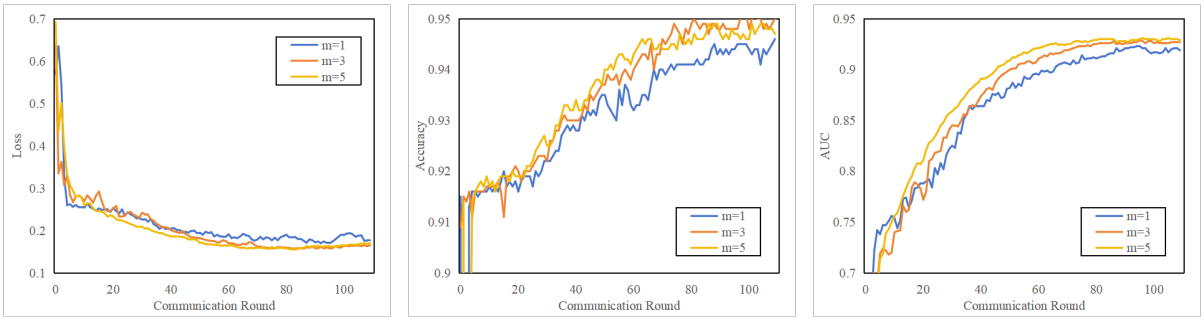
(a) Loss (b) Accuracy (c) AUC

Figure D1 Model comparisons with different local optimizers



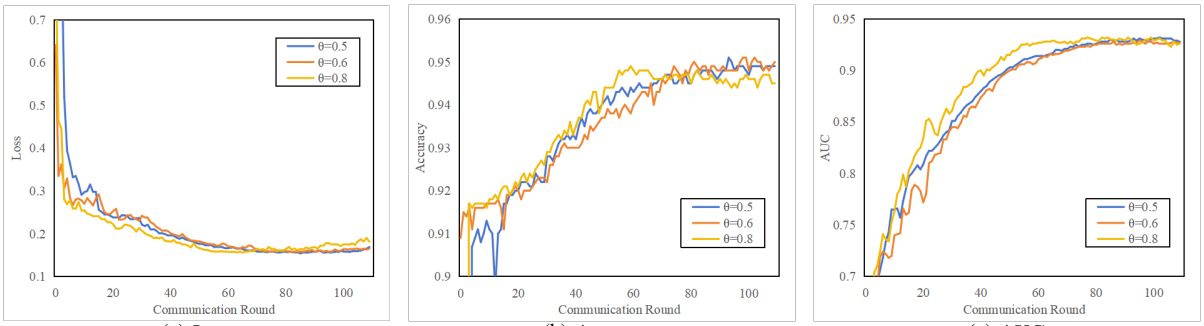
(a) Loss (b) Accuracy (c) AUC

Figure D2 Model comparisons with different batch sizes



(a) Loss (b) Accuracy (c) AUC

Figure D3 Model comparisons by varying m



(a) Loss (b) Accuracy (c) AUC

Figure D4 Model comparisons by varying θ

Table D1 Comparisons on the Performance between Models with Different Settings

Item	Setting	ACC	Recall	F1	AUC	MCC
Optimizer	SGD	0.930	0.255	0.379	0.881	0.408
	RAdam	**0.951**	**0.582**	**0.670**	**0.929**	**0.653**
Batchsize	50	0.949	0.557	0.645	**0.930**	0.627
	100	**0.951**	**0.582**	**0.670**	0.929	**0.653**
	200	0.950	0.548	0.653	**0.930**	0.642
n	1	0.946	0.521	0.617	0.921	0.600
	3	**0.951**	**0.582**	**0.670**	0.929	**0.653**
	5	0.949	0.562	0.651	**0.931**	0.633
θ	0.5	0.949	0.548	0.657	**0.932**	0.646
	0.6	**0.951**	**0.582**	**0.670**	0.929	**0.653**
	0.8	0.948	0.557	0.640	**0.932**	0.620

2) Impact of batch size: Batch size is another significant factor affecting the performance of the model. Thus, an appropriate batch size should be determined. We set $m = 3$ and $\theta = 0.6$, and batch size is set three levels: 50, 100, and 200 respectively. The result are shown in Figure D2 and Table D1. As Figure D2 illustrates, the batch size has a negligible impact on the convergence rate, since the number of rounds to reach the convergence remains almost unchanged under each setting. However, best performance regarding the recall rate, F1 score, and MCC can be observed at the model trained with $batchsize = 100$. Therefore we set $batchsize = 100$ for the rest of the experiments.

Appendix D.2.2 *Impact of Hyperparameters during Model Aggregation*

1) Impact of m in top-m: Since m determines the number of the models in the buffers to be aggregated in a new round, we set $\theta = 0.6$ and set $m = 1$, 3, and 5 respectively in this experiment. Figure D3 and Table D1 compare the performance across the models where the worst convergence occurs when $m = 1$. Though a slightly faster convergence can be achieved when $m = 5$, it does not work better than $m = 3$ in terms of accuracy, recall rate, F1 score, and MCC. This is because the update of the global model is negligible during each aggregation when n is relatively small. On the other hand, the accumulated variance from the global optimum becomes larger as m increases, thereby leading to negative impact on the performance. Consequently, we set $m = 3$ for the following experiments.

2) Impact of θ: θ has a direct impact on the basic weight of local models for aggregation in each communication round. To explore such influence, we compare the model convergence and the other metrics over $\theta = 0.5, 0.6, 0.8$, respectively. The results are presented in Figure D4 and Table D1. As illustrated, the models with different θ are on par in convergence. Meanwhile, $\theta = 0.6$ performs better on accuracy, recall rate, F1 score, and MCC. Hence we set θ to 0.6 in the later experiments.

Appendix D.3 Result and Performance Analysis

In this section, we carry out experimental studies to compare how FUSE performs vis-à-vis a set of benchmark works. All the hyperparameters are obtained from the discussions in Apprendix D.2.

Appendix D.3.1 *Comparisons on Performance*

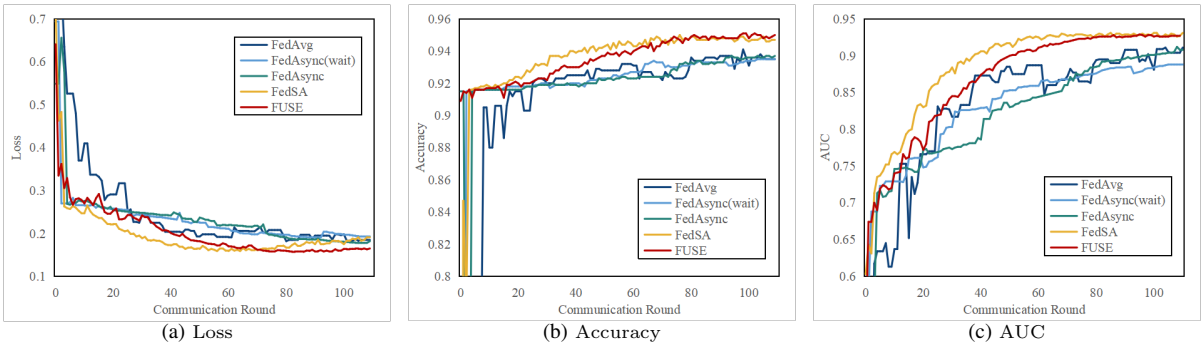
We first compare our framework with a synchronous aggregation algorithm FedAvg [1], an asynchronous aggregation algorithm FedAsync [3], and a semi-asynchronous aggregation algorithm FedSA [14]. The local training process for all models follows Algorithm C2 and the maximum staleness of model submission is set to 6 for TD, 4 for CS. Furthermore, we assume FedAvg removes the model when a straggler issue occurs [15]. As in FedAsync, we consider two strategies to address the asynchronous problem between TD and its corresponding CS during aggregation for the purpose of exploring the effects of aggregating different components of the full model asynchronously: FedAsync (wait) is to wait until all components of the model have arrived, whereas FedAsync is to perform aggregation asynchronously based upon the arrivals of components. In addition, FedSA filters the models according to a predefined threshold on staleness such that the models are aggregated based on the number of samples.

Figure D5 and Table D2 compare the model performance between our framework and the benchmarks. As expected, a faster convergence is achieved by FedAsync than FedAsync(wait). We also find that better performance and faster convergence are fulfilled through FUSE in comparison with FedAvg and FedAsync. This is because FedAvg removes models with high level of latency, thereby resulting in loss of features from the stragglers. Whereas only one model is aggregated in each communication round in FedAsync, leading to a lower rate of convergence. Meanwhile, better performance in terms of accuracy, recall rate, F1 score, and MCC is obtained by FUSE in contrast to FedSA, since the models with latencies are pre-aggregated in FUSE based upon cosine similarity, instead of imposing strong assumptions on threshold. On the other hand, not only the number of samples but the degree of model latency is taken into account in our framework, thus incentivizing a significantly better performance.

Appendix D.3.2 *Overhead Analysis*

Homomorphic Overhead: In this section, we evaluate the additional overhead introduced by the homomorphic encryption for secure aggregation in our framework. More specifically, Cheon-Kim-Kim-Song (CKKS) scheme is used in our experiments to encrypt the parameters [16]. The overhead brought on by the homomorphic encryption includes parameter encryption and decryption on TDs and CSs, and ciphertext computing of aggregation on the Aggregator.

Both encryption and decryption overheads are presented in Table D3, where the full model denotes the solution without U-shape split learning for local training. We find that our framework is more cost-effective, while the overhead on CS is significantly greater



(a) Loss (b) Accuracy (c) AUC

Figure D5 Comparison on model performance

Table D2 Comparison on Model Performance between FUSE and the Benchmarks

Methods	Accuracy	Recall	F1	AUC	MCC
FedAvg	0.938	0.332	0.476	0.911	0.505
FedAsync(wait)	0.935	0.288	0.432	0.888	0.476
FedAsync	0.937	0.357	0.490	0.912	0.500
FedSA	0.950	0.524	0.636	**0.931**	0.627
FUSE	**0.951**	**0.582**	**0.670**	0.929	**0.653**

Table D3 Computation Time for Encryption & Decryption

	Encryption	Decryption
v_{TD}	129.12ms	31.03ms
v_{CS}	60740.20ms	13759.50ms
Full model	64819.94ms	14637.31ms

than TD. This is not surprising because the majority of the overhead is transferred from TD to CS such that large scale of models are processed on CS with higher capacities. To further explore the overhead of ciphertext aggregation in depth, we decouple the operations into atomic ones based on addition and multiplication before the testing. The results are shown in Table D4, where a denotes a scalar, v_{TD} and v_{CS} denote the serialized model parameters of TD and CS, respectively. Considering the enhancement on privacy-preserving and the scenario wherein such operations are performed by the Aggregator with sufficient capacities instead of TD, it would be worthwhile to compromise on the overhead.

Table D4 Computation Time for Operations in Ciphertext

Operation	Time Cost	Operation	Time Cost
$[\![a]\!] \cdot [\![a]\!]$	3.00ms	$[\![v_{\mathrm{TD}}]\!] \cdot [\![v_{\mathrm{TD}}]\!]$	2313.10ms
$[\![v_{\mathrm{CS}}]\!] \cdot [\![v_{\mathrm{CS}}]\!]$	1060607.01ms	$a \cdot [\![v_{TD}]\!]$	22.02ms
$a \cdot [\![v_{\mathrm{CS}}]\!]$	9393.54ms	$[\![a]\!] + [\![a]\!]$	$\approx$ 0.00ms
$[\![v_{\mathrm{TD}}]\!] + [\![v_{\mathrm{TD}}]\!]$	4.00ms	$[\![v_{\mathrm{CS}}]\!] + [\![v_{\mathrm{CS}}]\!]$	933.85ms
$[\![a]\!] \cdot [\![v_{\mathrm{TD}}]\!]$	588.54ms	$[\![a]\!] \cdot [\![v_{\mathrm{CS}}]\!]$	263427.03ms

Communication and Computing Overhead: We evaluate the communication and computational overhead from the perspective of the entities in FUSE. Since both of them are positively related to the number of model parameters, such overhead can be closely approximated by the model size. Assuming the number of local iterations is E we present the results in Table D5. To demonstrate that FUSE enables reducing the computational overheads on resource-constrained devices, we compare our framework with a vanilla federated learning-based framework, i.e., the whole local training process executed by SM or TD. Given a traditional approach without U-shape split learning, the computational overhead can be represented as $E \cdot [FP(|w|) + BP(|w|)]$, where $|w| = |w_{\mathrm{FE}}| + |w_{\mathrm{FL}}| + |w_{\mathrm{FC}}|$. Since $|w_{\mathrm{FL}}| \gg |w_{\mathrm{FE}}| + |w_{\mathrm{FC}}|$, the computational overheads of TD or SM are significantly lower than $E \cdot [FP(|w|) + BP(|w|)]$. As a result, the analysis highlights that the computational overheads on SM and TD are migrated to assisted cloud servers, in contrast to the solution without U-shape split learning for local training. However, this trade-off of reducing computational overhead on resource-constrained devices comes at the cost of increased overall communication overhead. The communication overhead of TD can be represented as $|w| = |w_{\mathrm{FE}}| + |w_{\mathrm{FL}}| + |w_{\mathrm{FC}}|$ while the whole local training process is executed completely by TD. In our scheme, the communication overhead of TD can be further represented as $(E \cdot |\mathrm{SM}| + 1) \cdot |w_{\mathrm{FE}}| + E \cdot (|output_{\mathrm{FE}}| \cdot |\mathrm{SM}| + |output_{\mathrm{FL}}|) + |w_{\mathrm{FC}}|$ according to Table D5, since $|g^*_{\mathrm{FC}}| = |output_{\mathrm{FL}}|$, where $|g^*_{\mathrm{FL}}|$

denotes the gradient of the first layer of the Feature Learner. Therefore, the communication overhead of TD is reduced if and only if $|w_{\text{FL}}| > E \cdot |\text{SM}| \cdot |w_{\text{FE}}| + E \cdot (|output_{\text{FE}}| \cdot |\text{SM}| + |output_{\text{FL}}|)$. This inequality holds true in most cases, as the parameters of the Feature Learner constitute the majority of the overall model parameters. Therefore, our approach can significantly reduce communication overhead for TD in most scenarios.

Table D5 Communication & Computing Overhead In One Communication Round

	Computing	**Communication**
SM	$E \cdot FP(\|w_{\text{FE}}\|)$	$\underbrace{E \cdot \|output_{\text{FE}}\|}_{\text{to TD}}$
TD	$E \cdot [FP(\|w_{\text{FC}}\|) + BP(\|w_{\text{FC}}\|) + BP(\|w_{\text{FE}}\|)]$	$\underbrace{E \cdot \|w_{\text{FE}}\| \cdot \|\text{SM}\|}_{\text{to SM}} + \underbrace{E \cdot \|output_{\text{FE}}\| \cdot \|\text{SM}\| + E \cdot \|g^*_{\text{FC}}\|}_{\text{to CS}} + \underbrace{\|w_{\text{FE}}\| + \|w_{\text{FC}}\|}_{\text{to Aggregator}}$
CS	$E \cdot [FP(\|w_{\text{FL}}\|) + BP(\|w_{\text{FL}}\|)]$	$\underbrace{E \cdot \|output_{\text{FL}}\| + E \cdot \|g^*_{\text{FL}}\|}_{\text{to TD}} + \underbrace{\|w_{\text{FL}}\|}_{\text{to Aggregator}}$

* $FP(\cdot)$ and $BP(\cdot)$ denotes the function of computing overhead of forward propagation and back-propagation respectively.

Appendix E Related Work

Appendix E.1 Distributed Theft Detection Method

Typically, the increasing demands on privacy-preserving may not be fulfilled by the centralized methods satisfactorily, and even though studies are conducted on enhancing the security through cryptography techniques, they are not suitable for the scenario of Advanced Metering Infrastructure(AMI) wherein computational resources are limited due to the extensive overhead required for decryption and encryption. Thus, federated learning and split learning have emerged as major offerings for theft detection models by providing commitment on protecting sensitive data. FedDetect is the first federated learning-based schema [17] where data privacy is guaranteed by homomorphic encryption and local differential privacy. However, the work has not considered the substantial computational overhead for the clients and asynchronous model updates. Ashraf et al. [18] subsequently introduce FedDP, which presents a novel federated voting classifier specifically for theft detection. In contrast, a decentralized federated learning-based theft detector is proposed in [19]. This approach designs a secure and communication-efficient functional encryption-based aggregation scheme, eliminating the need for a centralized key distribution center to generate security parameters. However, all these schemes involve collecting the electricity consumption data from SMs to detection stations, which exposes them to the risk of raw data leakage. Alromih et al. [20] employ split learning in the smart grid to ensure better privacy protection. However, they model the detection as an unsupervised task, which is not appropriate for the scenarios where labeled data is available.

Appendix E.2 Federated Learning

Federated learning is proposed by Google which allows the data owners to train a global model collaboratively without exposing the raw data [1]. Due to the ability to preserve the privacy of raw data, federated learning is increasingly applied in various fields, including industry [21, 22], healthcare [23], and traffic [24]. Particularly, FedAvg [1] is the first algorithm for model aggregation and its performance and effectiveness has been verified in many studies. However, the straggler issue due to heterogeneity of bandwidths among devices is a bigger challenge in realistic environment. Li et al. [15] propose a synchronous aggregation algorithm where such issue is addressed by adjusting the local epoch of the straggler, while the performance has not been improved. On the other hand, a faster convergence can be achieved in asynchronous federated learning schema since the aggregation occurs when the update is received by the aggregator. Xie et al. [3] design FedAsync to improve both the flexibility and scalability of federated learning on massive devices with varying resources. Besides, asynchronous aggregation algorithms such as ASO-Fed and PAFLM are proposed to accommodate the model submissions at different rates [25, 26]. However, the extensive communication overhead brought on by the asynchronous aggregation strategy is placing additional hurdles on resource-constrained devices.

Balancing the trade-offs between the model performance, convergence rate, and communication overhead, semi-asynchronous aggregation mechanisms are developed by combining the advantages of both synchronous and asynchronous aggregations. A first-come-first-merge client selection mechanism is incorporated in SAFA to improve the efficiency of the training process and the quality of the global model [5]. FedSA is another semi-asynchronous framework proposed by Ma et al., where the number of local models for aggregation is optimized in order to minimize the training time [14]. To our knowledge, metrics that are critical to assuring model quality other than latency have not been considered in this study.

References

1 McMahan B, Moore E, Ramage D, et al. Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics, 2017. 1273-1282.
2 Xu C H, Qu Y Y, Xiang Y, et al. Asynchronous federated learning on heterogeneous devices: A survey. Computer Science Review, 2023, 50: 100595.
3 Xie C, Koyejo S, Gupta I. Asynchronous federated optimization. arXiv preprint, 2019, arXiv:1903.03934.
4 Zhang Y, Duan M, Liu D, et al. Csafl: A clustered semi-asynchronous federated learning framework. In: 2021 International Joint Conference on Neural Networks (IJCNN), 2021. 1-10.
5 Wu W T, He L A, Lin W W, et al. Safa: A semi-asynchronous protocol for fast federated learning with low overhead. IEEE Transactions on Computers, 2020, 70(5):655-668.
6 Hao J S, Zhao Y C, Zhang J L. Time efficient federated learning with semi-asynchronous communication. In: 2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS), 2020. 156-163.

7 Vepakomma P, Gupta O, Swedish T et al. Split learning for health: Distributed deep learning without sharing raw patient data. arXiv preprint arXiv:1812.00564, 2018.
8 Thapa C, Arachchige P C M, Camtepe S, et al. Splitfed: When federated learning meets split learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, 2022. volume 36, pages 8485-8493.
9 Alromih A, Clark J A, Gope P. Privacy-aware split learning based energy theft detection for smart grids. In: Information and Communications Security: 24th International Conference, ICICS 2022, Canterbury, UK, 2022. 281-300.
10 Wang T, Liu Y, Zheng X, et al. Edge-based communication optimization for distributed federated learning. IEEE Transactions on Network Science and Engineering, 2021, 9(4):2015-2024.
11 Cheon J H, Kim D, Kim D, et al. Numerical method for comparison on homomorphically encrypted numbers. In: Advances in Cryptology-ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, 2019. Part II, pages 415-445.
12 Finardi P, Campiotti I, Plensack G, et al. Electricity theft detection with self-attention. arXiv preprint, 2020, arXiv:2002.06219.
13 Li Q B, Diao Y Q, Chen Q, et al. Federated learning on non-iid data silos: An experimental study. In: 2022 IEEE 38th International Conference on Data Engineering (ICDE), 2022. 965-978.
14 Ma Q P, Xu Y, Xu H L, et al. Fedsa: A semi-asynchronous federated learning mechanism in heterogeneous edge computing. IEEE Journal on Selected Areas in Communications, 2021, 39(12):3654-3672.
15 Li T, Sahu A K, Zaheer M, et al. Federated optimization in heterogeneous networks. Proceedings of Machine learning and systems, 2020, 2:429-450.
16 Cheon J H, Kim A, Kim M, et al. Homomorphic encryption for arithmetic of approximate numbers. In: Advances in Cryptology-ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, 2017. Part I 23, pages 409-437.
17 Wen M, Xie R, Lu K J, et al. Feddetect: a novel privacy-preserving federated learning framework for energy theft detection in smart grid. IEEE Internet of Things Journal, 2021, 9(8):6069-6080.
18 Ashraf M M, Waqas M, Abbas G, et al. Feddp: A privacy-protecting theft detection scheme in smart grids using federated learning. Energies, 2022, 15(17):6241.
19 Ibrahem M I, Mahmoud M, Fouda M M, et al. Privacy preserving and efficient decentralized federated learning-based energy theft detector. In: GLOBECOM 2022-2022 IEEE Global Communications Conference, 2022. 287-292.
20 Alromih A, Clark J A, Gope P. Privacy-aware split learning based energy theft detection for smart grids. In: Information and Communications Security: 24th International Conference, ICICS 2022, Canterbury, UK, 2022. 281-300.
21 Chen J B, Xue J F, Wang Y, et al. Privacy-Preserving and Traceable Federated Learning for data sharing in industrial IoT applications. Expert Systems with Applications, 2023, 213: 119036.
22 Wang N Y, Li X, Guan Z T, et al. FedStream: A Federated Learning Framework on Heterogeneous Streaming Data for Next-Generation Traffic Analysis. IEEE Transactions on Network Science and Engineering, 2023.
23 Bashir A K, Victor N, Bhattacharya S, et al. Federated learning for the healthcare metaverse: Concepts, applications, challenges, and future directions. IEEE Internet of Things Journal, 2023.
24 Liu L, Tian Y X, Chakraborty C, et al. Multilevel Federated Learning based Intelligent Traffic Flow Forecasting for Transportation Network Management. IEEE Transactions on Network and Service Management, 2023.
25 Chen Y J, Ning Y, Slawski M, et al. Asynchronous online federated learning for edge devices with non-iid data. In: 2020 IEEE International Conference on Big Data (Big Data), 2020. 15-24.
26 Lu X F, Liao Y Y, Lio P, et al. Privacy-preserving asynchronous federated learning mechanism for edge network computing. IEEE Access, 2020, 8:48970-48981.