FEDGEMS: FEDERATED LEARNING OF LARGER SERVER MODELS VIA SELECTIVE KNOWLEDGE FUSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Today data is often scattered among billions of resource-constrained edge devices with security and privacy constraints. Federated Learning (FL) has emerged as a viable solution to learn a global model while keeping data private, but the model complexity of FL is impeded by the computation resources of edge nodes. In this work, we investigate a novel paradigm to take advantage of a powerful server model to break through model capacity in FL. By selectively learning from multiple teacher clients and itself, a server model develops in-depth knowledge and transfers its knowledge back to clients in return to boost their respective performance. Our proposed framework achieves superior performance on both server and client models and provides several advantages in a unified framework, including flexibility for heterogeneous client architectures, robustness to poisoning attacks, and communication efficiency between clients and server. By bridging FL effectively with larger server model training, our proposed paradigm paves ways for robust and continual knowledge accumulation from distributed and private data.

1 INTRODUCTION

Nowadays, large models with tremendous parameters trained with sufficient computation power are indispensable in Artificial Intelligence (AI), such as AlphaGo (Silver et al., 2016), Alphafold (Senior et al., 2020) and GPT-3 (Brown et al., 2020). However, billions of resource-constrained mobile and IoT devices have become the primary data source to empower the intelligence of many applications (Bonawitz et al., 2019; Brisimi et al., 2018; Li et al., 2019a). Due to privacy, security, regulatory and economic considerations (Voigt & Von dem Bussche, 2017; Li et al., 2018), it is increasingly difficult and undesirable to pool data together for centralized training. Therefore, federated learning approaches (McMahan et al., 2017; Smith et al., 2017; Caldas et al., 2018; Kairouz et al., 2019; Yang et al., 2019) which allow all the participants to reap the benefits of shared models without sharing private data have become increasingly attractive.

In a typical Federated Learning (FL) training process using FedAvg (McMahan et al., 2017), each client sends its model parameters or gradients to a central server, which aggregates all clients' updates and sends the aggregated parameters back to the clients to update their local model. Because FL places computation burden on edge devices, its learnability is largely limited by the edge resources, on which training large models is often impossible. Making FL trainable with larger models is desirable to break through model capacity and enable continual collaborative knowledge fusion and accumulation.

One feasible approach to bridge FL with larger server models is through knowledge distillation (KD) (Hinton et al., 2015), where clients and the server transfer knowledge through logits. For example, FedGKT (He et al., 2020a) adopts a server model as a downstream sub-model and transfers knowledge directly from smaller edge models. In FedGKT the large server model essentially learns from one small teacher at a time and doesn't learn consensus knowledge from multiple teachers. In FL, KD has also been applied (Li & Wang, 2019; Lin et al., 2020) to transfer ensemble knowledge of clients through consensus of output logits rather than parameters. These works either assume no

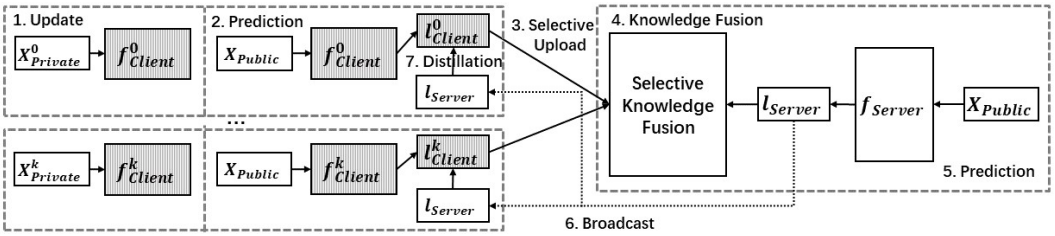

Figure 1: The framework of FedGEMS.

model on the server (Li & Wang, 2019) or a prototype server model of the same architecture as the client model (Lin et al., 2020).

In this paper, we first propose a new paradigm to bridge FL with a lar**g**er s**e**rver **m**odel, termed **FedGEM**, which can learn effectively and efficiently from fused knowledge by resource-constrained clients, and is also able to transfer knowledge back to clients with heterogeneous architectures. To further prevent negative and malicious knowledge transfer, we carefully design a **s**election and weighting criterion to enhance our knowledge transfer protocol, termed **FedGEMS**. We demonstrate with extensive experiments that our results significantly surpass the previous state-of-the-art baselines in both homogeneous and heterogeneous settings. Furthermore, thanks to our effective and selective protocol, our framework improves the robustness of FL on various malicious attacks and significantly reduces the overall communication. In summary, we propose a new framework to bridge FL with larger server models and simultaneously consolidate several benefits altogether, including superior performance, robustness of the whole system, and lower communication cost.

2 Related Work

Federated Learning with larger server models. FL is a collaborating learning framework without sharing private data among the clients. The classical method FedAvg (McMahan et al., 2017) and its recent variations (Mohri et al., 2019; Lin et al., 2018; Li et al., 2019b) directly transfer the clients' parameters or gradients to the server nodes. To tackle the performance bottleneck by training resource-constrained clients in FL, there are two lines of work to bridge FL with large powerful server models. The first line of studies adopts model compression (Han et al., 2015; He et al., 2018; Yang et al., 2018), manually designed architectures (Howard et al., 2017; Zhang et al., 2018; Iandola et al., 2016) or even efficient neural architecture search (Tan & Le, 2019; Wu et al., 2019) to adapt a large server model to on-device learning. Another line is adopting knowledge distillation (Hinton et al., 2015) to transfer knowledge through output logits rather than parameters between a client model and a larger server model (He et al., 2020a). However FedGKT (He et al., 2020a)'s focus is to transfer knowledge directly from clients to server without considering the consensus knowledge fused from clients. Therefore its performance is limited.

Federated Learning with Knowledge Distillation. In fact, ensemble knowledge distillation has been shown to boost collaborative performance in FL. Specifically, FedMD (Li & Wang, 2019) adopts a labeled public dataset and averaged logits to transfer knowledge. FedDF (Lin et al., 2020) proposes ensemble distillation for model fusion by aggregating both logits and models from clients. In addition, KD is used to enhance robustness in FL. Cronus (Chang et al., 2019) and DS-FL (Itahara et al., 2020) utilizes a public dataset with soft labels jointly with local private dataset for local training, and combines with Cronus or entropy reduction aggregation, respectively, to defend against poisoning attacks in FL. In this work, we propose to exploit the benefit of both a large server model and client knowledge fusion. Our work is also motivated by the recent studies in KD (Qin et al., 2021; You et al., 2017; Li et al., 2021; Yuan et al., 2021; Wang et al., 2021), which show that student models can have larger capacities by learning from multiple teacher models.

Method	Public Data	Client Model Heterogeneity	Aggregation	Server model Size
FedAvg	-	No	Average	-
FedMD	Labeled	Yes	Average	-
Cronus	Unlabeled	Yes	Cronus	-
FedDF	Unlabeled	No	Average	$=$ Client
FedGKT	-	Yes	-	$>$ Client
DS-FL	Unlabeled	Yes	Entropy-reduction	-
FedGEM	Labeled	Yes	Average	$>$ Client
FedGEMS	Labeled	Yes	Selective	$>$ Client

Table 1: Comparison of FedGEMS with related works.

3 METHODOLOGY

3.1 PRELIMINARIES

We assume that there are K clients in federated learning process. The kth client has its own private labeled dataset $\boldsymbol{X}^k := \{(\boldsymbol{x}_i^k, \boldsymbol{y}_i^k)\}_{i=1}^{N^k}$ that can be drawn from the same or different distribution, where $\boldsymbol{x}_i^k$ is the ith training sample in the kth client model, $\boldsymbol{y}_i$ is its corresponding ground truth label, and N^k denotes the total number of samples. Each client also trains its own model $\boldsymbol{f}^k$ with parameters $\boldsymbol{W}^k$ which can be of the same architecture (**homogeneous**) or different architecture (**heterogeneous**). There is also a public dataset $\boldsymbol{X}^0 := \{(\boldsymbol{x}_i^0, \boldsymbol{y}_i^0)\}_{i=1}^{N^0}$ which is accessible to both server and clients. On the server side, we assume that there is a larger server model to be trained, denoted as $\boldsymbol{f}^0$ with parameter $\boldsymbol{W}^0$.

3.2 FEDGEMS FRAMEWORK

We illustrate our overall framework in Fig. 1 and summarize our training algorithm in Algorithm 1.

FedGEM. During each communication round, all client models first use private datasets to train several epochs, then transfer the predicted logits on public dataset as knowledge to the server model. The server model aggregates the clients' logits and then trains its server model with the guidance of fused knowledge. After training, the server model then transfers its logits back to all client models. Finally, each client model distills knowledge from received logits and continues their training on private datasets. Continuously iterating over multiple rounds, both the server and client models mutually learn knowledge from each other. Through this alternating training processing, we can obtain a large server model with accumulated knowledge and an ensemble of high-performance client models.

FedGEMS: In FedGEMS, the server adopts a selection and weighting criterion to select knowledgeable clients for aggregation, which is detailed in the next section 3.3.

To illustrate the features of our framework, we compare it with the related studies of KD-based methods in federated learning in Table 1 on the following aspects: whether they use a labeled, unlabeled or no public dataset, whether client models can have heterogeneous architectures, the aggregation strategy on the server, and whether the server has a larger model. Note FedGEM can be regarded as placing a larger server model on top of the FedMD framework, while keeping other settings the same.

3.3 SELECTIVE KNOWLEDGE FUSION IN SERVER MODEL

Since clients' knowledge may negatively impact the server model in the heterogeneous or malicious setting, and vice versa, we further propose selective strategies on both server and client sides to enforce positive knowledge fusion into the large server model as shown in Fig. 2.

Algorithm 1 Illustration of the Framework of FedGEMS. T is the number of communication rounds; $\boldsymbol{X}^0$ and $\boldsymbol{Y}^0$ denotes the images and their corresponding labels in public dataset; $\boldsymbol{X}^k$ and $\boldsymbol{Y}^k$ denotes the private dataset in the kth client model; $\boldsymbol{f}_s$ and $\boldsymbol{f}_c^k$ are the server and the kth client model; $\boldsymbol{L}_{\text{Global}}$ indicates the global logits to save correct logits; $\boldsymbol{L}_s$ and $\boldsymbol{L}_c^k$ are the logit tensors from the server and the kth client model.

ServerExecute():
for each round $t = 1, 2, ..., T$ **do**
  // Selective Knowldge Fusion
  for idx, $\boldsymbol{x}^0, \boldsymbol{y}^0 \in \{\boldsymbol{X}^0, \boldsymbol{Y}^0\}$ **do**
    if $\boldsymbol{f}_s(\boldsymbol{W}_s; \boldsymbol{x}^0) == y^0$ **then**
      $\mathcal{L}_s \leftarrow \mathcal{L}_{S_1}(\boldsymbol{x}^0, \boldsymbol{y}^0)$ ▷ in Eq. 1
      $\boldsymbol{L}_{\text{Global}}[idx] \leftarrow \boldsymbol{L}_s[idx]$
    else if idx in $\boldsymbol{L}_{\text{Global}}$ **then**
      $\mathcal{L}_s \leftarrow \mathcal{L}_{S_2}(\boldsymbol{x}^0, \boldsymbol{y}^0, \boldsymbol{L}_{\text{Global}})$ ▷ in Eq. 2
    else
      $\boldsymbol{L}_c \leftarrow$ **ClientSelect**(idx)
      $\mathcal{L}_s \leftarrow \mathcal{L}_{S_3}(\boldsymbol{x}^0, \boldsymbol{y}^0, \boldsymbol{L}_c)$ ▷ in Eq. 3, 4, 5
    $\boldsymbol{W}_s \leftarrow \boldsymbol{W}_s - \eta_k \nabla \mathcal{L}_s$
    $\boldsymbol{L}_s[idx] \leftarrow \boldsymbol{f}_s(\boldsymbol{W}_s; \boldsymbol{x}^0)$
  // Transfer Knowledge to Client Models
  ClientTrain($\boldsymbol{L}_s$)

ClientTrain($\boldsymbol{L}_s$):
for each client kth **in parallel do**
  // Knowledge Distillation on Clients
  for $\boldsymbol{x}^0, \boldsymbol{y}^0 \in \{\boldsymbol{X}^0, \boldsymbol{L}_s, \boldsymbol{Y}^0\}$ **do**
    $\mathcal{L}_c \leftarrow \mathcal{L}_C(\boldsymbol{x}^0, \boldsymbol{y}^0, \boldsymbol{L}_s)$ ▷ in Eq. 6
    $\boldsymbol{W}_c^k \leftarrow \boldsymbol{W}_c^k - \eta_k \nabla \mathcal{L}_c$
  // Local Training on Clients
  for $\boldsymbol{x}^k, \boldsymbol{y}^k \in \{\boldsymbol{X}^k, \boldsymbol{Y}^k\}$ **do**
    $\mathcal{L}_c \leftarrow \mathcal{L}_{CE}(\boldsymbol{x}^k, \boldsymbol{y}^k)$
    $\boldsymbol{W}_c^k \leftarrow \boldsymbol{W}_c^k - \eta_k \nabla \mathcal{L}_c$

ClientSelect(idx):
// Selective Transfer to Server
for each client kth **in parallel do**
  $\boldsymbol{L}_c^0[idx] \leftarrow \boldsymbol{f}_c^k(\boldsymbol{W}_c^k; \boldsymbol{x}^0[idx])$
Return $\boldsymbol{L}_c$ to server

3.3.1 SERVER-SIDE SELF-DISTILLATION

At each iteration, the server model first performs self evaluation on the public dataset and split the samples into two classes, those it can predict correctly, S_{Correct}, and those it predicts wrongly, $S_{\text{Incorrect}}$. For each sample $\boldsymbol{x}^i$ in S_{Correct} where the model prediction matches the ground truth label, we simply adopt the cross-entropy loss between the predicted values and the ground truth labels to train the server model.

$$\mathcal{L}_{S_1} = \mathcal{L}_{CE} = -\boldsymbol{y}^i \log(\boldsymbol{f}_{Server}(\boldsymbol{x}^i)) \tag{1}$$

At the same time, we save the correct logit $\boldsymbol{l}_s^i$ into the global pool of logits as $\boldsymbol{l}_{Global}^i$, which can be further used as memory to recover its reserved knowledge from self-distillation.

For each sample $\boldsymbol{x}^i$ that the server predicts wrongly, we first check whether its corresponding global logit $\boldsymbol{l}_{Global}^i$ exists or not. We denote all samples that do not exist in $\boldsymbol{l}_{Global}$ as $S_{\text{Incorrect}}^*$, for which we believe the server model can not learn the sample entirely by itself and propose to use clients' collective knowledge as its teacher, which will be explained in the next section. If $\boldsymbol{l}_{Global}^i$ exists, it means that the knowledge was reserved by the server before, the server model performs self-distillation to recover this part of knowledge. The final training objective of self-distillation can be formulated as follows, where $\mathcal{D}_{KL}$ is the Kullback Leibler (KL) Divergence function.

$$\begin{aligned} \mathcal{L}_{S_2} = \mathcal{L}_{SS} &= \epsilon \mathcal{L}_{CE} + (1-\epsilon)\mathcal{L}_{DL} \\ &= \epsilon \mathcal{L}_{CE} + (1-\epsilon)\mathcal{D}_{KL}\left(\boldsymbol{l}_{Global}^i \| \boldsymbol{l}_{Server}^i\right) \end{aligned} \tag{2}$$

Our self-distillation strategy has at least two advantages. On the one hand, distilling knowledge from itself is better than from other models which have different architectures. On the other hand, learning from its own stored logits can greatly reduce the communication cost as compared to transferring all redundant logits to client models.

3.3.2 CLIENT-SIDE SELECTIVE ENSEMBLE DISTILLATION

For those samples in $S_{\text{Incorrect}}^*$ that the server model fails to predict correctly, we try to distill knowledge from the ensemble of clients. Furthermore, considering the correctness and relative importance of client models, we propose a weighted selective strategy on client side.

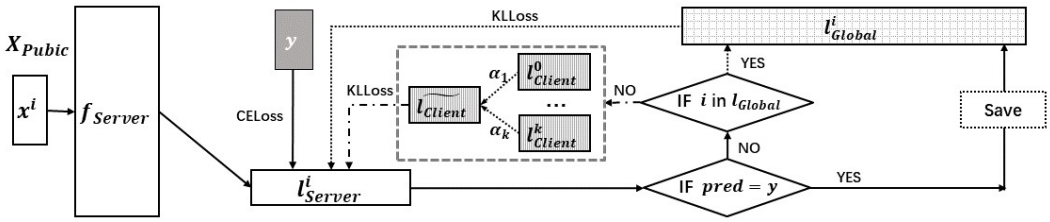

Figure 2: Selective knowledge fusion module.

Given an instance $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ in $S^*_{\text{Incorrect}}$, we first split all clients $\{C_1, C_2, ..., C_K\}$ into the reliable and unreliable clients according to their predictions $\boldsymbol{p}_{C_j}$. For those clients who predict the wrong labels, we consider them as unreliable and discard their knowledge by setting their weights equal to 0. As for the rest of clients who predict the labels correctly, we consider them as reliable and use their entropy $H(\boldsymbol{p}_{C_j})$ as a measure of the confidence.

$$H(\boldsymbol{p}_{C_j}) = -\sum_{i=1}^{N} \boldsymbol{p}(\boldsymbol{x}_i) log \boldsymbol{p}(\boldsymbol{x}_i) \tag{3}$$

Following previous work (Li et al., 2021; Pereyra et al., 2017; Szegedy et al., 2016), low entropy indicates high confidence, and vice versa. Specifically, given an instance $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ in $S^*_{\text{Incorrect}}$, we design its corresponding weights to aggregate output logits from different clients as below.

$$\boldsymbol{\alpha}_{C_j} = \begin{cases} 0, & C_j \in C_{\text{Unreliable}} \\ \text{softmax}(\frac{1}{H(\boldsymbol{p}_{C_j})}), & C_j \in C_{\text{Reliable}} \end{cases} \tag{4}$$

Therefore, for samples in $S^*_{\text{Incorrect}}$, the knowledge transferred from ensemble clients to server model can be formulated as the following.

$$\begin{aligned} \mathcal{L}_{S_3} = \mathcal{L}_{CS} &= \epsilon \mathcal{L}_{CE} + (1-\epsilon)\mathcal{L}_{DL} \\ &= \epsilon \mathcal{L}_{CE} + (1-\epsilon)\mathcal{D}_{KL}\left(\sum_{j=1}^{K} \boldsymbol{\alpha}_{C_j} \boldsymbol{l}_{C_j} \middle|\middle| \boldsymbol{l}_{Server}\right) \end{aligned} \tag{5}$$

Up to now, we have covered the selective knowledge fusion strategy for the server, next we will discuss the knowledge distillation on clients.

3.4 Training in Client Models

Each client model first receives the logits $\boldsymbol{l}^i_{Server}$ of public dataset from the server model. Then it distills knowledge according to the logits as well as computes the cross-entropy loss to train on public dataset.

$$\begin{aligned} \mathcal{L}_C &= \epsilon \mathcal{L}_{CE} + (1-\epsilon)\mathcal{L}_{DL} \\ &= \epsilon \mathcal{L}_{CE} + (1-\epsilon)\mathcal{D}_{KL}\left(\boldsymbol{l}^i_{Server} \| \boldsymbol{l}^i_{Client}\right) \end{aligned} \tag{6}$$

After knowledge distillation, each client model further adopts the cross-entropy loss to train on its local private dataset to better fit into its target distribution.

4 EXPERIMENTAL EVALUATIONS

4.1 EXPERIMENT SETTINGS

Task and Dataset. For fair comparison, we use the same training tasks as He et al. (2020a), which include image classifications on the CIFAR-10 (Krizhevsky et al., 2009), CIFAR-100 (Krizhevsky et al., 2009) and CINIC-10 (Darlow et al., 2018) datasets. More details about these three dataset can be found in Appendix C.1. For each dataset, we randomly split data into two subsets: 30,000 samples in public and 30,000 samples in private. The public dataset is used for transferring knowledge between server node and client nodes, while the private dataset is for client training. Both of them split into 25,000 samples for training and 5,000 samples for testing.

Homogeneous Setting. We randomly shuffle and partition private dataset into 8 clients. As the model architectures shown in Appendix C.4, our client models all adopt a tiny CNN architecture called ResNet-11, while the server model architecture is ResNet-20. The parameters of the server model are as twice large as the client model.

Heterogeneous Setting. We adopt the Dirichlet distribution (Yurochkin et al., 2019; Hsu et al., 2019) to control the degree of heterogeneity in our non-iid setting. In this experiment, our α is 0.5. We also introduce model heterogeneity in our experiments. The client models vary from ResNet-11 to Resnet-17 as shown in Appendix C.5, while the server model is ResNet-20.

Baselines. We compare our framework FedGEM/FedGEMS with the following baselines. (1) **Stand-Alone**: the server and client models train on their local datasets $\boldsymbol{X}^0$ and $\boldsymbol{X}^k$, respectively; (2) **Centralized**: the client model trains on the whole private dataset $\{\boldsymbol{X}^1 + \cdots + \boldsymbol{X}^k\}$; (3) **Centralized-All**: the server and clients perform centralized training on the whole dataset $\{\boldsymbol{X}^0 + \cdots + \boldsymbol{X}^k\}$, respectively. Note in reality no party has this complete dataset so this can be viewed as an upper limit of model performance. In addition, we also compare our performance with several existing related works, including **FedAvg** (McMahan et al., 2017), **FedMD** (Li & Wang, 2019), **Cronus** (Chang et al., 2019), **DS-FL** (Itahara et al., 2020), **FedDF** (Lin et al., 2020) and **FedGKT** (He et al., 2020a). For fair comparison, the settings of client models and the split of public and private dataset in all approaches are kept the same. The server models in FedDF and FedGKT are ResNet-11 (same as client models) and ResNet-56 respectively, following their own designs. More details of different baselines and their important hyper-parameters in our experiments are listed in Appendixes A and C.7, respectively.

4.2 PERFORMANCE EVALUATIONS

Method	Homo				Hetero			
	CIFAR-10		CIFAR-100		CIFAR-10		CIFAR-100	
	Server	Clients	Server	Clients	Server	Clients	Server	Clients
Stand-Alone	81.60	57.38	61.90	33.78	81.60	44.55	61.90	27.26
Centralized	-	74.81	-	56.42	-	79.95	-	60.52
Centralized-All	88.97	79.77	73.51	65.07	88.97	87.96	73.51	71.75
FedAvg	-	63.64	-	39.16	-	-	-	-
FedMD	-	67.53	-	46.26	-	43.87	-	34.11
Cronus	-	66.03	-	45.00	-	39.61	-	38.38
DS-FL	-	60.36	-	47.63	-	36.46	-	31.05
FedDF	65.18	64.78	42.11	40.87	-	-	-	-
FedGKT	70.80	44.75	31.18	27.72	35.47	12.78	35.53	21.05
FedGEM	83.53	79.70	66.45	61.28	83.47	78.89	65.77	57.21
FedGEMS	**85.65**	**81.10**	**67.31**	**61.70**	**85.32**	**80.38**	**65.91**	**58.61**

Table 2: Model performance in homogeneous and heterogeneous settings.

The experimental results for both homogeneous and heterogeneous settings are shown in Table 2. The results of CINIC-10 in both homogeneous and heterogeneous settings can be found in Appendix C.2. As for FedDF and FedGKT which also have a server model, we report their performance on both server and clients.

First of all, our approach FedGEM outperforms all the above KD-based federated learning baselines significantly on both server and client side, demonstrating the effectiveness of knowledge transfer with a larger server model. Compared with FedGKT which also adopts a larger server model, our performance is significantly improved by knowledge fusion from multiple clients. Comparing to the performance of stand-alone server and clients, our framework simultaneously improves both the sever model and client model, indicating that server and clients mutually benefit from knowledge transfer. Using public labels for supervision and selection, our FedGEMS framework is able to enhance positive knowledge fusion and further improve performance on both server and clients.

4.3 ROBUSTNESS

Poisoning Attacks. We employ three different types of model poisoning attacks to evaluate the robustness of our methods, including Naive Poisioning (PAF), Little Is Enough Attack (LIE) and One Far One Mean (OFOM). In PAF and LIE, the attacker poisons one client at each round via disturbing the logits or parameters which transfer from clients to server in different methods. For example, FedDF sends both logits and parameters to the server so that our attacker poisons both of them. In OFOM, two clients' benign predictions or parameters are poisoned at each round. We do not compare with FedGKT because it has a different setting which not only transfers the logits but also extra feature maps to the server model. The detailed algorithms about different poisoning attacks can be found in Appendix B.

Method	**PAF**		**LIE**		**OFOM**	
	Server	**Clients**	**Server**	**Clients**	**Server**	**Clients**
FedAvg	-	14.42/-49.22	-	09.50/-54.14	-	13.82/-49.82
FedDF	13.12/-52.06	13.97/-50.81	15.89/-49.29	25.82/-38.96	12.64/-52.54	14.40/-50.38
FedMD	-	33.27/-34.26	-	43.97/-23.56	-	32.74/-34.79
DS-FL	-	46.62/-13.74	-	44.74/-15.62	-	45.20/-15.16
Cronus	-	**66.29/+00.26**	-	**66.08/+00.05**	-	**65.98/-00.05**
FedGEM	60.01/-23.52	74.79/-04.91	43.62/-39.91	69.26/-10.44	83.71/+00.18	79.36/-00.34
FedGEMS	**86.86/+01.21**	**80.85/-00.25**	**84.64/-01.01**	**80.48/-00.62**	**86.33/+00.68**	**80.78/-00.32**

Table 3: The A in "A/B" denotes the model performance after attacks, while B denotes the percentage changes compared to the original accuracy ("+" denotes increasing, and "-"denotes dropping). We remark both the best performance and the least changed ratio (%) to its original accuracy.

The results of poisoning attacks in homogeneous setting on CIFAR-10 dataset are shown in Table 3. By adopting a selective strategy, our proposed FedGEMS provides consistent robustness across various attacks on both server and client models. FedAvg and FedDF can not defend against model poisoning attacks since they directly average the model parameters including the malicious ones. FedMD is relatively more robust than FedAvg via transferring only logits rather than parameters. By adopting a larger server model whose architecture is stable and independent from clients, FedGEM demonstrates superior robustness to FedMD on various model poisoning attacks. Since DS-FL and Cronus are designed for robust aggregation in a FedMD framework, their robustness to the attacks are strong, as expected. Our selective knowledge transfer strategy shows comparably strong robustness on both clients and server side without additional computation cost for robust aggregation algorithms.

4.4 COMMUNICATION COST

We further evaluate the communication costs per epoch by varying public dataset sizes for various approaches on CIFAR-10, and the results are shown in Fig. 3. The formulations to calculate communication cost for different methods are shown in Appendix C.3.

Due to the compact models deployed in client nodes, the bits of model parameters are limited, thus the communication overhead of FedAvg is not always higher than typical KD-based methods

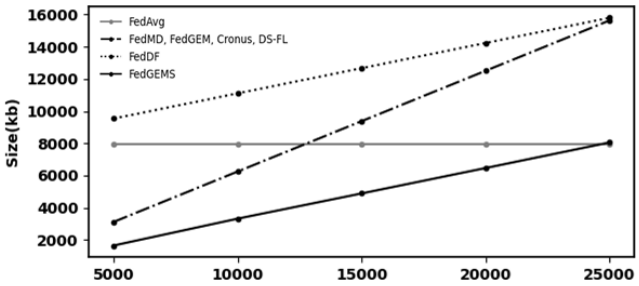

Figure 3: Communication cost per epoch of different public dataset sizes on CIFAR-10. Due to the enormous communication overhead of feature maps, we do not figure the flat line (irrelevance to the size of public dataset) of FedGKT which is always 1.6M (kb).

when a large public dataset is used. FedDF's communication cost is heavier than other KD-based FL approaches because it sends both logits and model parameters from client models to the server. Thanks to our selection strategy, the communication cost of our proposed framework FedGEMS is the lowest among all methods. Since FedGKT also communicates feature maps of 32,768 bits according to its implementation, its communication overhead is much higher than other approaches and is not included in our results. In summary, the results demonstrate that our selection strategy can greatly save the communication cost with a reasonable public data size.

5 UNDERSTANDING FEDGEMS

5.1 KNOWLEDGE ACCUMULATION AT SERVER

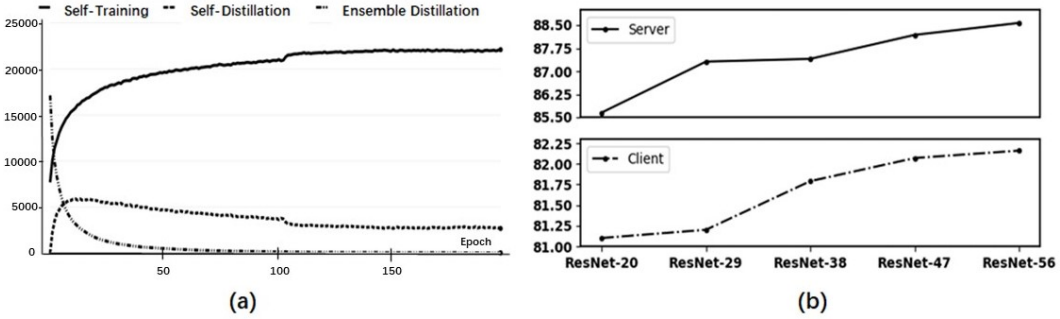

Figure 4: (a) Total number of samples in different selective decisions; (b) Model performances of different server sizes.

To analyze the selective knowledge fusion process of the server model in our framework, we report the number of samples associated with each step in our decision-making process in our experiment on CIFAR-10 with 25,000 samples as public dataset. Each sample will choose one of three strategies according to our selective knowledge fusion module. Specifically, in Fig. 4 (a), the line of self-training indicates the total number of samples the server model predicts correctly, while the lines of self-distillation and ensemble-distillation indicate the number of samples the server learns via self-distillation and client-side ensemble knowledge, respectively. In the initial stage, since both the server and client models learn from scratch, the number of samples that the server model can predict correctly is limited, and most of the knowledge is accumulated by distillation from the client models' fusion. As the training progresses, the server model needs to restore some of its knowledge from a self-distillation strategy. With the server model continually fusing sufficient knowledge, the model performance of server model eventually exceeds client models and the number of transferred samples from clients dropped to 0. Notice that in the final stage, there still remains some stubborn

samples which are hard for the server model while most samples can be solved by the large server model with accumulated knowledge.

5.2 Effect of Server Model Size

We further investigate the influence of the server model size on the overall performance by gradually changing the server model from ResNet-20 to larger models, while fixing the client models as ResNet-11. The details of the model parameters can be found in Appendix C.6. The results are shown in Fig. 4 (b). It can be witnessed that the performance of both the server and client models improves as the server model becomes larger and deeper. This phenomenon verifies the importance of placing a larger deeper server model to boost the performance of both server and the resource-constrained client models.

5.3 Effect of Public Dataset Size

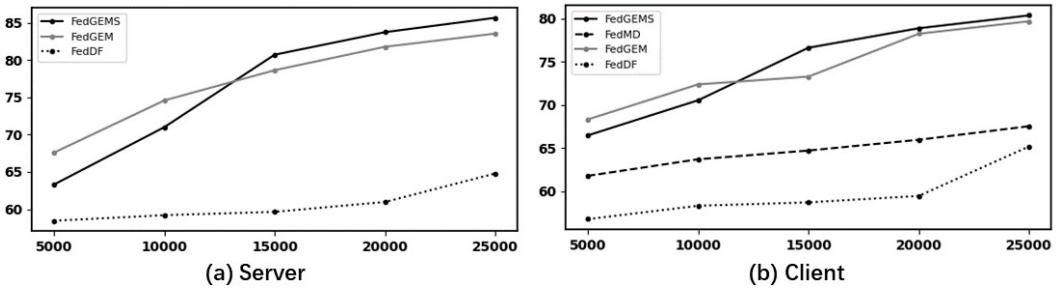

Figure 5: (a) Model performance of server on public dataset of different sizes; (b) Model performance of clients on public dataset of different sizes.

We further evaluate the effect of the size of the public dataset on model performance. The size of public dataset still varies from 5,000 to 25,000, and the results are shown in Fig. 5. It can be seen that both the client and server model continuously improve as the size of public data increases, indicating that their performances are highly correlated. Our performance consistently outperforms FedMD, and the performance gap over FedMD continuously grows as more public dataset is available. Note again FedGEM is essentially FedMD with a larger server model so the performance gain is due to the accumulated knowledge on the server side. As for FedDF which owns an ensemble server model as same architecture as clients, the performance in both server and client models remains a huge gap to our performance. This phenomenon further demonstrates the importance of the large capacity of server model. Interestingly, when only a relatively small public data size is available, FedGEM can outperform FedGEMS, possibly because the models can not afford to be overly selective when insufficient data is available and the collective knowledge from all clients outperforms the superior knowledge from strictly selected clients.

6 Conclusion

In this work, we first propose a new paradigm to apply a large deeper server model to effectively and efficiently fuse and accumulate knowledge, which can enhance the model performances on both server and client sides. Furthermore, we design a selection and weighted criterion on both sides to distill only positive knowledge into the server. We conduct a series of experiments on three distinct datasets (CIFAR-10, CIFAR-100 and CINIC-10) to evaluate our proposed framework FedGEMS. Our results show that FedGEMS can significantly surpass all baselines in both homogeneous and heterogeneous settings. Meanwhile, FedGEMS can further improve the robustness of FL on poisoning attacks as well as reduce the communication costs between server and client sides. In future work, we will study the effectiveness of our work in other critical tasks, such as NLP and knowledge graph-based tasks.

REFERENCES

Moran Baruch, Gilad Baruch, and Yoav Goldberg. A little is enough: Circumventing defenses for distributed learning. *arXiv preprint arXiv:1902.06156*, 2019.

Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

Theodora S Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Ch Paschalidis, and Wei Shi. Federated learning of predictive models from federated electronic health records. *International journal of medical informatics*, 112:59–67, 2018.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

Hongyan Chang, Virat Shejwalkar, Reza Shokri, and Amir Houmansadr. Cronus: Robust and heterogeneous collaborative learning with black-box knowledge transfer. *arXiv preprint arXiv:1912.11279*, 2019.

Luke N Darlow, Elliot J Crowley, Antreas Antoniou, and Amos J Storkey. Cinic-10 is not imagenet or cifar-10. *arXiv preprint arXiv:1810.03505*, 2018.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Chaoyang He, Murali Annavaram, and Salman Avestimehr. Group knowledge transfer: Federated learning of large cnns at the edge. *arXiv preprint arXiv:2007.14513*, 2020a.

Chaoyang He, Songze Li, Jinhyun So, Xiao Zeng, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, et al. Fedml: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518*, 2020b.

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 784–800, 2018.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.

Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Sohei Itahara, Takayuki Nishio, Yusuke Koda, Masahiro Morikura, and Koji Yamamoto. Distillation-based semi-supervised federated learning for communication-efficient collaborative training with non-iid private data. *arXiv preprint arXiv:2008.06180*, 2020.

Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Daliang Li and Junpu Wang. Fedmd: Heterogenous federated learning via model distillation. *arXiv preprint arXiv:1910.03581*, 2019.

JiLiang Li, Kim-Kwang Raymond Choo, WeiGuo Zhang, Saru Kumari, Joel JPC Rodrigues, Muhammad Khurram Khan, and Dieter Hogrefe. Epa-cppa: An efficient, provably-secure and anonymous conditional privacy-preserving authentication scheme for vehicular ad hoc networks. *Vehicular Communications*, 13:104–113, 2018.

Suyi Li, Yong Cheng, Yang Liu, Wei Wang, and Tianjian Chen. Abnormal client behavior detection in federated learning. *arXiv preprint arXiv:1910.09933*, 2019a.

Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. Fair resource allocation in federated learning. *arXiv preprint arXiv:1905.10497*, 2019b.

Ziyun Li, Xinshao Wang, Haojin Yang, Di Hu, Neil Martin Robertson, David A. Clifton, and Christoph Meinel. Not all knowledge is created equal. *CoRR*, abs/2106.01489, 2021. URL https://arxiv.org/abs/2106.01489.

Tao Lin, Sebastian U Stich, Kumar Kshitij Patel, and Martin Jaggi. Don't use large mini-batches, use local sgd. *arXiv preprint arXiv:1808.07217*, 2018.

Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi. Ensemble distillation for robust model fusion in federated learning. *arXiv preprint arXiv:2006.07242*, 2020.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. PMLR, 2017.

Mehryar Mohri, Gary Sivek, and Ananda Theertha Suresh. Agnostic federated learning. In *International Conference on Machine Learning*, pp. 4615–4625. PMLR, 2019.

Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*, 2017.

Yujia Qin, Yankai Lin, Jing Yi, Jiajie Zhang, Xu Han, Zhengyan Zhang, Yusheng Su, Zhiyuan Liu, Peng Li, Maosong Sun, et al. Knowledge inheritance for pre-trained language models. *arXiv preprint arXiv:2105.13880*, 2021.

Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, Alex Bridgland, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, 2020.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet Talwalkar. Federated multi-task learning. *arXiv preprint arXiv:1705.10467*, 2017.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pp. 6105–6114. PMLR, 2019.

Paul Voigt and Axel Von dem Bussche. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10:3152676, 2017.

Fusheng Wang, Jianhao Yan, Fandong Meng, and Jie Zhou. Selective knowledge distillation for neural machine translation. *arXiv preprint arXiv:2105.12967*, 2021.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10734–10742, 2019.

Qiang Yang, Yang Liu, Yong Cheng, Yan Kang, Tianjian Chen, and Han Yu. Federated learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 13(3):1–207, 2019.

Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 285–300, 2018.

Shan You, Chang Xu, Chao Xu, and Dacheng Tao. Learning from multiple teacher networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1285–1294, 2017.

Fei Yuan, Linjun Shou, Jian Pei, Wutao Lin, Ming Gong, Yan Fu, and Daxin Jiang. Reinforced multi-teacher selection for knowledge distillation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 14284–14291, 2021.

Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Nghia Hoang, and Yasaman Khazaeni. Bayesian nonparametric federated learning of neural networks. In *International Conference on Machine Learning*, pp. 7252–7261. PMLR, 2019.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6848–6856, 2018.

A Related Work

We illustrate the most related works FedMD, FedGKT and FedDF in Fig. 6.

B Poisoning Attacks in Federated Learning

In this section, we introduce a set of poisoning attacks from the literature which used to evaluate the robustness in federated learning.

B.1 Naive poisoning (PAF)

Naive poisoning is a type of model poisoning attack. The adversary is able to poison ϵ fraction of total client. With the knowledge of the distribution of benign updates, the opponent can introduce malicious update θ_m. This attack is far from the mean of the benign updates, obtained by adding a significantly large vector θ' to it:

$$\theta_m = \frac{\sum_{i=1}^{n} \theta_i}{(1-\epsilon)n} + \theta' \tag{7}$$

Then, the server updates with the malicious update θ_m and transfers it to all the clients.

B.2 Little is enough attack (LIE)

LIE is a type of model poisoning attack proposed by Baruch et al. (2019). The detailed implementation of LIE is summarized in Algorithm 2. The malicious update θ_m is achieved by making small changes in many dimensions to a benign update. The malicious update is trained in server and then shared by each client.

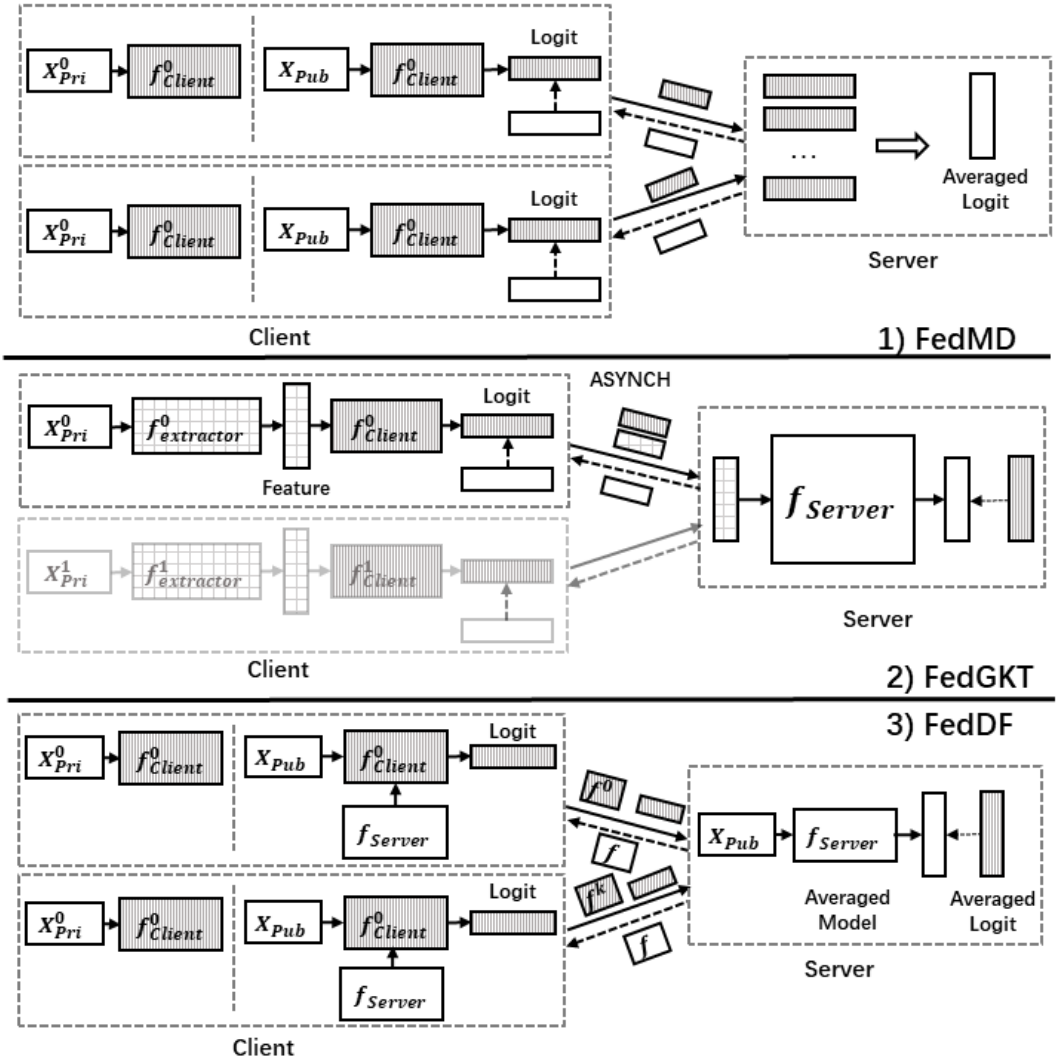

Figure 6: Baselines of most related work.

Algorithm 2 Little is enough attack (LIE) Baruch et al. (2019)

1: **Input:** The number of benign updates n, the fraction of malicious updates ϵ

2: The number of majority benign clients:

$$s = \left\lfloor \frac{n}{2} + 1 \right\rfloor - \epsilon n \tag{8}$$

3: Using z-table, set:

$$z^{\max} = \max_{z} \left(\phi(z) < \frac{n-s}{n} \right) \tag{9}$$

4: **for** $j \in [d]$ **do**

5: compute mean (μ_j) and standard deviation (σ_j) of benign updates.

$$\theta_m^j \leftarrow \mu_j + z^{\max} \cdot \sigma_j \tag{10}$$

6: **Output:** θ_m

B.3 ONE FAR ONE MEAN (OFOM)

This model poisoning attack, OFOM, is proposed by Chang et al. (2019). In this attack, two malicious updates will be added to the benign updates. First, the opponent adds a significantly large vector θ' to the mean of the benign updates to obtain θ_m^1. Then, the opponent crafts θ_m^2, which is the mean of the benign updates and θ_m^1.

$$\theta_1^m = \frac{\sum_{i=1}^n \theta_i}{n} + \theta', \quad \theta_2^m = \frac{\sum_{i=1}^n \theta_i + \theta_1^m}{n+1} \tag{11}$$

C EXTRA EXPERIMENTAL RESULTS AND DETAILS

C.1 A SUMMARY OF DATASET

Following He et al. (2020a), our experiments utilize CIFAR-10 (Krizhevsky et al., 2009), CIFAR-100 (Krizhevsky et al., 2009) and CINIC-10 (Darlow et al., 2018) as our datasets. **CIFAR-10** consists 10 classes colour images, with 6000 images per class. **CIFAR-100** is a more challenging dataset with 100 subclasses that falls under 20 superclasses, e.g. *baby, boy, girl, man* and *woman* belong to *people*. **CINIC-10** (Darlow et al., 2018) confuses two different sources CIFAR-10 and ImageNet drawn from similar but not identical distribution.

C.2 MODEL ACCURACY ON CINIC-10

In this section, we compare our framework FedGEMS with several baselines in both homogeneous and heterogeneous settings on CINIC-10. The results are shown in Table 4 which is a supplement to the main pages.

Method	**Homo**		**Hetero**	
	CINIC-10		**CINIC-10**	
	Server	**Clients**	**Server**	**Clients**
Stand-Alone	66.34	45.52	66.34	37.34
Centralized	-	62.44	-	65.93
Centralized-All	72.32	68.61	72.32	71.95
FedAvg	-	49.10	-	-
FedMD	-	54.41	-	42.26
Cronus	-	55.77	-	41.60
DS-FL	-	47.97	-	34.43
FedDF	51.64	49.61	-	-
FedGKT	48.34	38.59	28.02	22.31
FedGEMS	**68.77**	**67.80**	**68.55**	**62.97**

Table 4: Model accuracy in both homogeneous and heterogeneous settings on CINIC-10.

C.3 COMMUNICATION COST

In the Fig. 7, we list the formulations to compute communication costs between server and client models in different methods.

C.4 MODEL ARCHITECTURES OF HOMOGENEOUS SETTING

The model architectures of client and server models in homogeneous setting are shown in Table 5.

C.5 MODEL ARCHITECTURE OF HETEROGENEOUS SETTING

The different model architectures of client models in heterogeneous setting are shown in Table 6. The server model in heterogeneous setting is the same as homogeneous setting in Table 5.

Method	Formulation
FedAvg	$(B_{paras}) \times (\#C + \#S)$
FedMD **DS-FL** **Cronus** **FedLM**	$(B_{logits}) \times (\#C + \#S)$
FedDF	$(B_{logits}) \times (\#C) + (B_{paras}) \times (\#C + \#S)$
FedGKT	$(B_{logits}) \times (\#C + \#S) + (B_{Features}) \times (\#C)$
FedLMS	$(B_{logits}) \times (\#C_{Selective} + \#S)$

Figure 7: Communication Cost for CIFAR-10. B indicates the corresponding bits of logits or parameters. S and C denotes Server and Client, while $\#S$ and $\#C$ denotes the total number of Server and Clients.

Model	Conv1	Conv2_x	Conv3_x	Conv4_x		Parameters
ResNet-11 (Client)	3×3 16 Stride 1	$\begin{bmatrix} 1 \times 1, 16 \\ 3 \times 3, 16 \\ 1 \times 1, 64 \end{bmatrix} \times 1$	$\begin{bmatrix} 1 \times 1, 32 \\ 3 \times 3, 32 \\ 1 \times 1, 128 \end{bmatrix} \times 1$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 1$	average pool 10-d fc	127642
ResNet-20 (Server)	3×3 16 Stride 1	$\begin{bmatrix} 1 \times 1, 16 \\ 3 \times 3, 16 \\ 1 \times 1, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 32 \\ 3 \times 3, 32 \\ 1 \times 1, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 2$	average pool 10-d fc	220378

Table 5: Details of Model Architectures in Homogeneous Setting

Model	Conv1	Conv2_x	Conv3_x	Conv4_x	
ResNet-11(1)	3×3, 16, stride 1	$\begin{bmatrix} 1 \times 1, 16 \\ 3 \times 3, 16 \\ 1 \times 1, 64 \end{bmatrix} \times 1$	$\begin{bmatrix} 1 \times 1, 32 \\ 3 \times 3, 32 \\ 1 \times 1, 128 \end{bmatrix} \times 1$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 1$	average pool, 10-d fc
ResNet-14(2)	3×3, 16, stride 1	$\begin{bmatrix} 1 \times 1, 16 \\ 3 \times 3, 16 \\ 1 \times 1, 64 \end{bmatrix} \times 1$	$\begin{bmatrix} 1 \times 1, 32 \\ 3 \times 3, 32 \\ 1 \times 1, 128 \end{bmatrix} \times 1$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 2$	average pool, 10-d fc
ResNet-14(3)	3×3, 16, stride 1	$\begin{bmatrix} 1 \times 1, 16 \\ 3 \times 3, 16 \\ 1 \times 1, 64 \end{bmatrix} \times 1$	$\begin{bmatrix} 1 \times 1, 32 \\ 3 \times 3, 32 \\ 1 \times 1, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 1$	average pool, 10-d fc
ResNet-14(4)	3×3, 16, stride 1	$\begin{bmatrix} 1 \times 1, 16 \\ 3 \times 3, 16 \\ 1 \times 1, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 32 \\ 3 \times 3, 32 \\ 1 \times 1, 128 \end{bmatrix} \times 1$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 1$	average pool, 10-d fc
ResNet-17(5)	3×3, 16, stride 1	$\begin{bmatrix} 1 \times 1, 16 \\ 3 \times 3, 16 \\ 1 \times 1, 64 \end{bmatrix} \times 1$	$\begin{bmatrix} 1 \times 1, 32 \\ 3 \times 3, 32 \\ 1 \times 1, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 2$	average pool, 10-d fc
ResNet-17(6)	3×3, 16, stride 1	$\begin{bmatrix} 1 \times 1, 16 \\ 3 \times 3, 16 \\ 1 \times 1, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 32 \\ 3 \times 3, 32 \\ 1 \times 1, 128 \end{bmatrix} \times 1$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 2$	average pool, 10-d fc
ResNet-17(7)	3×3, 16, stride 1	$\begin{bmatrix} 1 \times 1, 16 \\ 3 \times 3, 16 \\ 1 \times 1, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 32 \\ 3 \times 3, 32 \\ 1 \times 1, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 1$	average pool, 10-d fc
ResNet-17(8)	3×3, 16, stride 1	$\begin{bmatrix} 1 \times 1, 16 \\ 3 \times 3, 16 \\ 1 \times 1, 64 \end{bmatrix} \times 1$	$\begin{bmatrix} 1 \times 1, 32 \\ 3 \times 3, 32 \\ 1 \times 1, 128 \end{bmatrix} \times 1$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 2$	average pool, 10-d fc

Table 6: Details of the 8 client models architecture used in our experiment

C.6 Model Architectures of Larger Server Models

The detailed model architectures of large server models are shown in Table 7.

C.7 Hyper-parameters

In table 8, we sum up the hyper-parameter settings for all the methods in our experiments. We directly run FedAvg, FedGKT and re-implement all the other methods base on an open-source federated learning research library FedML (He et al., 2020b) in a distributed computing environment.

Model	Conv1	Conv2_x	Conv3_x	Conv4_x	
ResNet-20	3×3, 16, stride 1	$\begin{bmatrix} 1 \times 1, 16 \\ 3 \times 3, 16 \\ 1 \times 1, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 32 \\ 3 \times 3, 32 \\ 1 \times 1, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 2$	average pool, 10-d fc
ResNet-29	3×3, 16, stride 1	$\begin{bmatrix} 1 \times 1, 16 \\ 3 \times 3, 16 \\ 1 \times 1, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 32 \\ 3 \times 3, 32 \\ 1 \times 1, 128 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	average pool, 10-d fc
ResNet-38	3×3, 16, stride 1	$\begin{bmatrix} 1 \times 1, 16 \\ 3 \times 3, 16 \\ 1 \times 1, 64 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 32 \\ 3 \times 3, 32 \\ 1 \times 1, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 4$	average pool, 10-d fc
ResNet-47	3×3, 16, stride 1	$\begin{bmatrix} 1 \times 1, 16 \\ 3 \times 3, 16 \\ 1 \times 1, 64 \end{bmatrix} \times 5$	$\begin{bmatrix} 1 \times 1, 32 \\ 3 \times 3, 32 \\ 1 \times 1, 128 \end{bmatrix} \times 5$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 5$	average pool, 10-d fc
ResNet-56	3×3, 16, stride 1	$\begin{bmatrix} 1 \times 1, 16 \\ 3 \times 3, 16 \\ 1 \times 1, 64 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 32 \\ 3 \times 3, 32 \\ 1 \times 1, 128 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 6$	average pool, 10-d fc

Table 7: Details of the large server models architecture used in our experiment

Methods	Hyperparameters	Homogeneous	Heterogeneous
FedGEMS	optimizer	Adam, lr=0.001, wd=0.0001	Adam, lr=0.001, wd=0.0001
	batch size	256	256
	client epochs of public data	1	1
	client epochs of private data	1	1
	server epochs	20	20
	communication rounds	200	200
FedAvg	optimizer	Adam, lr=0.001, wd=0.0001	-
	batch size	64	
	client epochs	20	
	communication rounds	200	
FedMD	optimizer	Adam, lr=0.001, wd=0.0001	Adam, lr=0.001, wd=0.0001
	batch size	64	64
	client epochs of public data	1	1
	client epochs of private data	2	2
	communication rounds	200	200
FedDF	optimizer	Adam, lr=0.001, wd=0.0001	-
	batch size	64	
	client epochs of private data	40	
	server epochs	5	
	communication rounds	100	
FedGKT	optimizer	Adam, lr=0.001, wd=0.0001	SGD, lr=0.005, wd=0.0001
	batch size	256	256
	client epochs of public data	1	1
	client epochs	20	40
	communication rounds	200	200
Cronus	optimizer	Adam, lr=0.001, wd=0.0001	Adam, lr=0.001, wd=0.0001
	batch size	64	64
	client epochs of public data	1	1
	client epochs of private data	1	1
	communication rounds	300	300
DS-FL	optimizer	Adam, lr=0.001, wd=0.0001	Adam, lr=0.001, wd=0.0001
	batch size	64	64
	client epochs of public data	1	1
	client epochs of private data	1	1
	server epochs	2	2
	communication rounds	300	300
Standalone	optimizer	Adam, lr=0.001, wd=0.0001	Adam, lr=0.001, wd=0.0001
	batch size	256	256
	epochs	200	200
Centralized	optimizer	Adam, lr=0.001, wd=0.0001	Adam, lr=0.001, wd=0.0001
	batch size	256	256
	epochs	200	200

Table 8: Hyper-parameters used in Experiments on dataset CIFAR-10, CIFAR-100 and CINIC-10