FEDPROPHET: MEMORY-EFFICIENT FEDERATED ADVERSARIAL TRAINING VIA ROBUST AND CONSISTENT CASCADE LEARNING

Minxue Tang [*1] **Yitu Wang** [*1] **Jingyang Zhang** [1] **Louis DiValentin** [2] **Aolin Ding** [2] **Amin Hass** [2] **Yiran Chen** [1] **Hai "Helen" Li** [1]

ABSTRACT

Federated Adversarial Training (FAT) can supplement robustness against adversarial examples to Federated Learning (FL), promoting a meaningful step toward trustworthy AI. However, FAT requires large models to preserve high accuracy while achieving strong robustness, incurring high memory-swapping latency when training on memory-constrained edge devices. Existing memory-efficient FL methods suffer from poor accuracy and weak robustness due to inconsistent local and global models. In this paper, we propose FedProphet, a novel FAT framework that can achieve memory efficiency, robustness, and consistency simultaneously. FedProphet reduces the memory requirement in local training while guaranteeing adversarial robustness by adversarial cascade learning with strong convexity regularization, and we show that the strong robustness also implies low inconsistency in FedProphet. We also develop a training coordinator on the server of FL, with Adaptive Perturbation Adjustment for utility-robustness balance and Differentiated Module Assignment for objective inconsistency mitigation. FedProphet significantly outperforms other baselines under different experimental settings, maintaining the accuracy and robustness of end-to-end FAT with 80% memory reduction and up to 10.8× speedup in training time.

1 INTRODUCTION

With the rapid development of data-gluttonous artificial intelligence (AI), concerns about training data privacy also arise. Federated Learning (FL) is proposed as a distributed machine learning paradigm to provide a strong privacy guarantee (Konečnỳ et al., 2015; 2016). FL pushes model training to local edge devices (denoted as clients in FL) and only aggregates locally trained models, which avoids privacy leakage in data gathering and transmission.

While FL can offer privacy benefits, it cannot guarantee robustness against adversarial examples that are deemed as another threat to AI systems. Previous studies have shown that AI models are usually sensitive to small input perturbations, and a manipulated imperceptible noise can cause catastrophic errors in the outputs (Goodfellow et al., 2014). To achieve adversarial robustness, adversarial training is proposed to train the model with adversarially perturbed training data (Madry et al., 2017), and federated adversarial training (FAT) is also proposed to complement FL with adversarial training (Zizzo et al., 2020).

However, adversarial training will sacrifice the model performance as a trade-off between the utility (accuracy) and the robustness (Wang et al., 2021a). Therefore, a larger model with higher capacity is required to achieve high accuracy and strong robustness simultaneously. Since most clients in cross-device FL are resource-constrained edge devices such as IOT devices and mobile phones (Kairouz et al., 2019), not all the clients have sufficient memory to train a large model demanded by FAT.

We are still able to train a model that exceeds the memory capacity by swapping the model parameters and intermediate features between the internal memory (e.g., CPU RAM or GPU memory) and the external storage (e.g., SSD) (Rajbhandari et al., 2020; Wang et al., 2023). When the internal memory is significantly smaller than the memory requirement for training the whole model, frequent memory swapping during each forward and backward propagation can incur high latency. The latency introduced by memory swapping becomes more significant in adversarial training, where additional backward and forward propagations are usually required to generate adversarially perturbed training data (Madry et al., 2017).

Some previous studies explored memory-efficient federated learning frameworks that allow resource-constrained clients to train a smaller local model or a sub-model of the large global model, and aggregate the heterogeneous models with knowledge distillation (Lin et al., 2020; Cho et al., 2022) or partial average (Diao et al., 2020; Alam et al.,

[*]Equal contribution [1]Duke University [2]Accenture Cyber Labs. Correspondence to: Yiran Chen <yiran.chen@duke.edu>, Hai "Helen" Li <hai.li@duke.edu>.

Proceedings of the 8th MLSys Conference, Santa Clara, CA, USA, 2025. Copyright 2025 by the author(s).

2022). However, previous methods cannot achieve high accuracy and strong robustness when being applied in FAT because of the **objective inconsistency**, i.e., clients train different local models from the global model. Specifically, the robustness of the local models does not sufficiently lead to the robustness of the global model with a different architecture. In addition, heterogeneous local models cause higher variance in the local model updates, which can lead to divergence of FAT since FAT is shown to be more unstable than standard FL (Zizzo et al., 2020; Shah et al., 2021).

In this paper, we propose FedProphet, a memory-efficient FAT framework that can achieve strong adversarial robustness and low objective inconsistency simultaneously. Specifically, on the client side, we propose robust and consistent cascade learning which partitions a large global model into cascaded small modules such that the memory-constrained clients can train each module one by one without memory swapping. We incorporate adversarial training and strong convexity regularization into vanilla cascade learning (Belilovsky et al., 2020) to guarantee adversarial robustness, and our theoretical analysis further shows that strong adversarial robustness of the cascaded modules also implies low objective inconsistency in cascade learning. Thus, our adversarial cascade learning can also achieve low inconsistency. On the server side, we develop a training coordinator with two components: (a) Adaptive Perturbation Adjustment automatically controls the intermediate adversarial perturbation magnitude in adversarial cascade learning to attain a better utility-robustness balance and more stable convergence during training; (b) Differentiated Module Assignment further reduces the objective inconsistency by allowing "prophet" clients who have sufficient computational resources to train additional "future" modules together with the current module. In summary:

(1) Framework: We propose FedProphet, a FAT framework with memory efficiency, adversarial robustness, and objective consistency.

(2) Client Side: We develop adversarial cascade learning with strong convexity regularization to guarantee the robustness of the whole model. We theoretically demonstrate that the robustness achieved by our method also implies low objective inconsistency in cascade learning.

(3) Server Side: We develop a training coordinator, with Adaptive Perturbation Adjustment that can attain better accuracy-robustness balance, and Differentiated Module Assignment that can further reduce the objective inconsistency without sacrificing efficiency.

(4) Experiments: Under different settings, FedProphet shows significantly higher accuracy and adversarial robustness than previous memory-efficient federated learning methods, maintaining almost the same utility and robustness as end-to-end FAT while saving 80% memory and achieving up to $10.8\times$ speedup in training time[1].

2 Related Works and Preliminaries

2.1 Federated Learning

Federated Learning (FL) is a distributed learning framework, where different devices (i.e., clients) collaboratively train a model $\boldsymbol{w}$ that can minimize the empirical task loss L (Konečnỳ et al., 2015; McMahan et al., 2017):

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & L(\boldsymbol{w}) = \sum_{k=1}^{N} q_k L_k(\boldsymbol{w}), \\ \text{where} \quad & L_k(\boldsymbol{w}) = \frac{1}{|\mathbb{D}_k|} \sum_{(\boldsymbol{x},y)\in\mathbb{D}_k} l(\boldsymbol{x}, y; \boldsymbol{w}). \end{aligned} \tag{1}$$

$\mathbb{D}_k$ with size $|\mathbb{D}_k| = q_k \sum_i |\mathbb{D}_i|$ is the local dataset of client k. The local dataset is never shared with others such that privacy is preserved in FL. FedAvg is the first and the most popular FL framework, with multi-step local SGD and periodical model average (McMahan et al., 2017).

One challenge in FL is the heterogeneous clients (Kairouz et al., 2019; Li et al., 2020), including statistical heterogeneity (non-I.I.D. and unbalanced local data) (Karimireddy et al., 2019; Wang et al., 2020; Tang et al., 2022; Zhang et al., 2023) and systematic heterogeneity (various computational resources) (Li et al., 2018; Tian et al., 2022; Sun et al., 2022). This paper mainly focuses on the systematic heterogeneity among clients, especially clients with insufficient memory. Some recent studies propose *Knowledge-distillation FL* where clients train heterogeneous models based on their computational resources, and the heterogeneous models are aggregated by knowledge distillation instead of average on the server (Lin et al., 2020; Cho et al., 2022). Some other studies develop *Partial-training FL* where each client trains a sub-model extracted from the global large model, and the server aggregates the sub-models into the global large model by partial average (Caldas et al., 2018; Diao et al., 2020; Alam et al., 2022). Though both knowledge-distillation FL and partial-training FL have been demonstrated effective in standard FL, they fail to tackle the objective inconsistency, leading to poor robustness and convergence in FAT.

2.2 Adversarial Training

It is well known that the performance of deep neural networks can be dramatically undermined by adversarial examples, which are generated by adding small perturbations to the clean data (Goodfellow et al., 2014; Yang et al.,

[1]Our codes are available at `https://github.com/Yoruko-Tang/FedProphet.git`

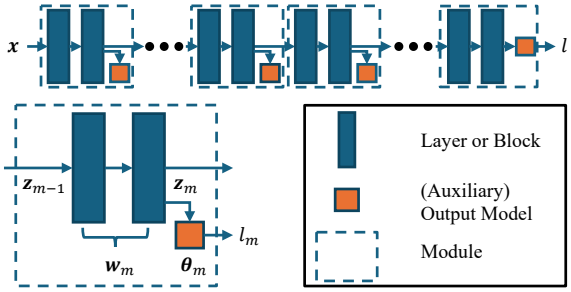

Figure 1. An illustration of Cascade Learning.

2020). To improve the adversarial robustness, Adversarial Training (AT) is proposed. In contrast to standard training (ST) that simply minimizes the empirical task loss, AT solves a min-max problem during training:

$$\min_{\boldsymbol{w}} \max_{\boldsymbol{\delta}:\|\boldsymbol{\delta}\|\leq\epsilon} l(\boldsymbol{x}+\boldsymbol{\delta}, y; \boldsymbol{w}). \tag{2}$$

AT alternatively solves the inner maximization and the outer minimization such that the model becomes insensitive to the small perturbation $\boldsymbol{\delta}$ in the input $\boldsymbol{x}$ (Croce & Hein, 2020; Wong et al., 2020). For example, PGD-n AT (Madry et al., 2017) conducts n steps of projected gradient accent on $\boldsymbol{\delta}$ for the inner maximization, and then the perturbed inputs are used for one-step gradient descent on the model parameter $\boldsymbol{w}$. AT with a larger n usually confers stronger robustness to the model, but also incurs more forward and backward propagations (Wong et al., 2020).

For the analysis purpose, we define (ϵ, c)-robustness:

Definition 1. A model $\boldsymbol{w}$ is (ϵ, c)-robust in a loss function l at input $\boldsymbol{x}$ if $\forall\boldsymbol{\delta}\in\{\boldsymbol{\delta}:\|\boldsymbol{\delta}\|\leq\epsilon\}$,

$$l(\boldsymbol{x}+\boldsymbol{\delta}, y; \boldsymbol{w}) - l(\boldsymbol{x}, y; \boldsymbol{w}) \leq c. \tag{3}$$

2.3 Cascade Learning

Cascade Learning (also known as Decoupled Learning) is proposed to reduce the memory requirement for training a large model (Hettinger et al., 2017; Marquez et al., 2018; Belilovsky et al., 2020). As illustrated in Figure 1, Cascade Learning partitions a large neural network into cascaded small modules and trains the modules one by one in the forward order. Each module $\boldsymbol{w}_m$ is trained with an early exit loss l_m provided by an auxiliary output model $\boldsymbol{\theta}_m$:

$$\min_{\boldsymbol{w}_m,\boldsymbol{\theta}_m} L_m(\boldsymbol{w}_m, \boldsymbol{\theta}_m) = \mathbb{E}\left[l_m(\mathbf{z}_{m-1}, \mathrm{y}; \boldsymbol{w}_m, \boldsymbol{\theta}_m)\right]. \tag{4}$$

After the current module $\boldsymbol{w}_m$ converges, it is fixed as $\boldsymbol{w}_m^*$, and the output features $\boldsymbol{z}_m = \boldsymbol{z}_m(\boldsymbol{z}_{m-1}; \boldsymbol{w}_m^*)$ are used to train the next module $(\boldsymbol{w}_{m+1}, \boldsymbol{\theta}_{m+1})$.

However, vanilla cascade learning is shown to have inferior performance compared to end-to-end training because of the **objective inconsistency**, i.e., each module is independently trained with the early exit loss l_m that is different from the joint loss l of the whole model. Since $\nabla l_m \neq \nabla l$, each module converges to sub-optimum in the independent training (Wang et al., 2021b). In addition, the robustness of each module in the early exit loss does not sufficiently lead to the robustness of the whole model in the joint loss.

Table 1. Results of FAT in CIFAR-10 and Caltech-256 with different sizes of models. We use CNN3/VGG16 as the small model (1× memory)/large model (5× memory) in CIFAR-10, and CNN4/ResNet34 in Caltech-256. "Large-PT" adopts a partial-training FL method FedRolex (Alam et al., 2022).

Model (Mem)	CIFAR-10		Caltech-256	
	Clean Acc.	Adv. Acc.	Clean Acc.	Adv. Acc.
Small (1×)	66.57%	54.33%	25.64%	13.49%
Large (5×)	79.74%	56.76%	46.56%	19.76%
Large-PT (1×)	67.14%	54.13%	30.18%	11.78%

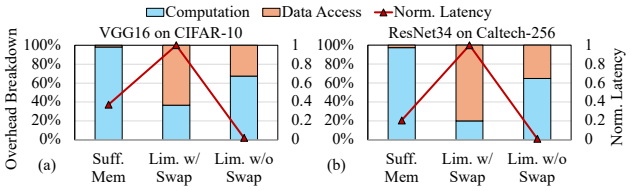

Figure 2. The local training overhead breakdown and latency in two workloads, (a) VGG16 on CIFAR-10 and (b) ResNet34 on Caltech-256. "Suff. Mem" denotes training with sufficient memory resources and "Lim. w/ Swap" denotes training with 20% memory and adopting memory swapping. "Lim. w/o Swap" trains with 20% memory and FedRolex (Alam et al., 2022).

3 Motivations

The resource-constrained edge devices are unable to afford federated adversarial training (FAT). Due to the utility-robustness trade-off, FAT requires large models to achieve high accuracy and strong robustness simultaneously as shown in Table 1 (Wang et al., 2021a). However, the common edge devices in cross-device FL scenarios are IOT devices, mobile phones, and laptops, which may not have sufficient memory capacity to afford training a large model (as shown in Figure 6) (Kairouz et al., 2019).

Memory swapping introduces high data access overhead in FAT. We can train a large model on a memory-constrained device by "memory swapping", which offloads/fetches the model parameters, optimizer states, and activations, to/from the external storage during training (Wang et al., 2023). However, memory swapping leads to high data access latency that can dominate the FAT workload as shown in Figure 2. The high data access latency is usually caused by high software driver management overhead and low storage I/O bandwidth. Furthermore, FAT incurs more forward and backward propagations to solve the min-max problem in Equation (2), which increases the frequency of memory swapping.

Previous memory-efficient FL methods fail to maintain

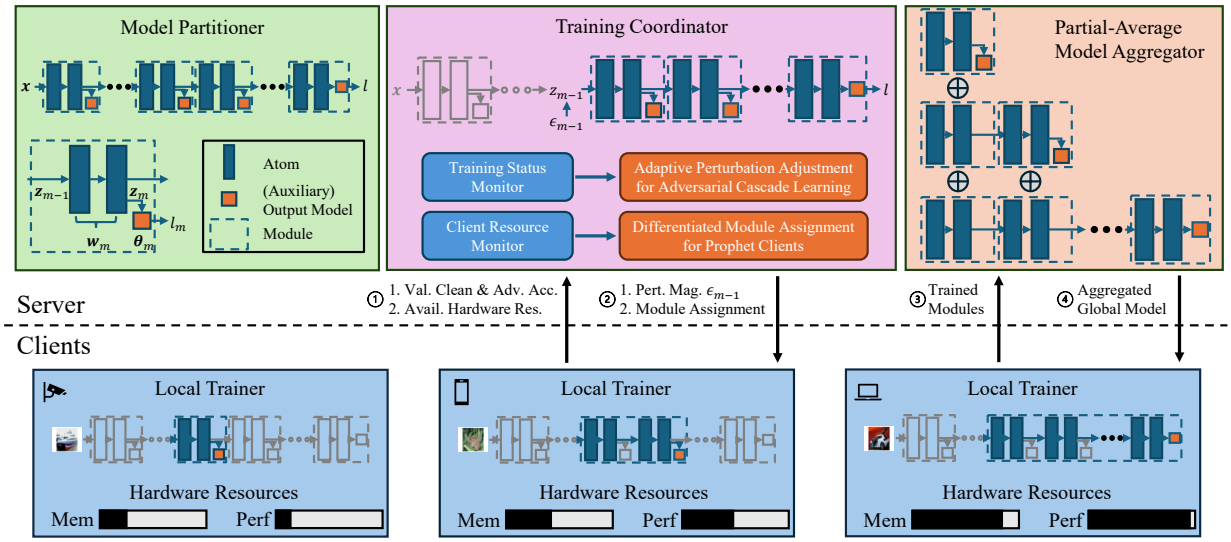

Figure 3. A framework of FedProphet. We formalize the framework in Algorithm 2.

high accuracy and robustness in FAT. Prior memory-efficient FL methods can avoid memory swapping when training a large model (Lin et al., 2020; Diao et al., 2020; Cho et al., 2022; Alam et al., 2022), but they show low clean and adversarial accuracy (exemplified by "Large-PT" in Table 1). This is attributed to **objective inconsistency**, namely, memory-constrained clients locally train different small models from the large global model, leading to sub-optimal local model updates and a gap between the local model robustness and the global model robustness.

4 Overview

Based on the motivations above, we propose FedProphet, a memory-efficient FAT framework that can avoid memory swapping while maintaining utility and robustness when training a large model. As shown in Figure 3, FedProphet consists of client-side local trainers (Section 5) and server-side model partitioner, training coordinator, and model aggregator (Section 6). At the beginning of the FL process, the model partitioner partitions a predefined large global model into small modules that satisfy the memory constraints on clients. After model partitioning, we have four steps in each communication round of FL: ① Clients upload the validation (clean and adversarial) accuracy of and their available hardware resources (memory and performance) to the server; ② The training coordinator on the server adjusts the perturbation magnitude ϵ_{m-1} based on the validation accuracy (Adaptive Perturbation Adjustment), and determines which modules should be trained by each client based on their available hardware resources (Differentiated Module Assignment); ③ Each client conducts adversarial training on the assigned modules with the strong-convexity regularized loss, and uploads the trained modules to the server; ④ The server conducts partial average to aggregate the trained modules, and broadcasts the aggregated global model back to the clients.

5 Local Client Design

In this section, we introduce the local trainer on clients. We first introduce how a client adversarially trains a module with strong convexity regularization to ensure the robustness of the backbone model in Section 5.1. In Section 5.2, we will uncover the relationship between objective inconsistency and robustness, which demonstrates that the adversarial cascade learning we propose can also achieve low objective inconsistency in addition to strong robustness.

5.1 Adversarial Cascade Learning with Strong Convexity Relularization

Sufficient Condition for Backbone Robustness. A client can conduct adversarial training on module m by adding adversarial perturbation to its input $\boldsymbol{z}_{m-1}$, as shown in Figure 4. However, since the module is trained with the early exit loss l_m that is not equivalent to the joint loss l of the backbone model, the robustness of the module in the early exit loss does not sufficiently lead to the robustness of the backbone model in the joint loss. The following proposition gives a sufficient condition for the robustness of the backbone model that consists of M cascaded modules:

Proposition 1. *The backbone model* $(\boldsymbol{w}_1 \circ \cdots \circ \boldsymbol{w}_M)^2$ *have* (ϵ_0, c_M)*-robustness in the joint loss* l*, if for every module*

[2]We use $\boldsymbol{a} \circ \boldsymbol{b}$ to denote a cascade of two modules $\boldsymbol{a}$ and $\boldsymbol{b}$, inputting in $\boldsymbol{a}$ and outputting from $\boldsymbol{b}$.

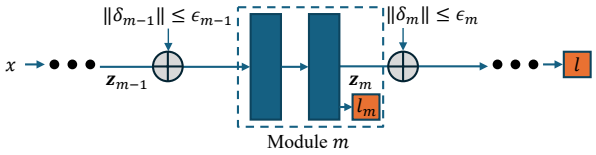

Figure 4. An illustration of Adversarial Cascade Learning.

$m < M$, we have a finite upper bound ϵ_m such that

$$\max_{\|\boldsymbol{\delta}_{m-1}\| \le \epsilon_{m-1}} \|\Delta \boldsymbol{z}_m\| \le \epsilon_m, \tag{5}$$

$$\textit{where} \quad \Delta \boldsymbol{z}_m = \boldsymbol{z}_m(\boldsymbol{z}_{m-1} + \boldsymbol{\delta}_{m-1}) - \boldsymbol{z}_m(\boldsymbol{z}_{m-1}), \tag{6}$$

and the last module has (ϵ_{M-1}, c_M)-robustness in $l_M = l$.

Proposition 1 can be easily proved by induction on the number of modules. It shows that the backbone robustness can be achieved by the module robustness, with an upper bound of the perturbation on the output feature of each module. A straightforward method to ensure this upper bound is conducting adversarial training with the perturbation magnitude as the loss function, namely,

$$\min_{\boldsymbol{w}_m} \max_{\|\boldsymbol{\delta}_{m-1}\| \le \epsilon_{m-1}} \|\Delta \boldsymbol{z}_m\|. \tag{7}$$

However, calculating the gradient $\nabla_{\boldsymbol{w}_m} \|\Delta \boldsymbol{z}_m\|$ by backward propagation requires memory to store additional intermediate results of $\boldsymbol{z}_{m-1} + \boldsymbol{\delta}_{m-1}$, which equivalently doubles the batch size during training. Therefore, this method is infeasible in the memory-constrained scenarios.

Strong Convexity Regularization. The following lemma (proved in Appendix A.1) provides an alternative method to upper bound the perturbation on $\boldsymbol{z}_m$, by making the early exit loss function strongly convex on $\boldsymbol{z}_m$ and conducting adversarial training with it:

Lemma 1. *If l_m is μ_m-strongly convex on $\boldsymbol{z}_m$ and module m is (ϵ_{m-1}, c_m)-robust in l_m, we have*

$$\max_{\|\boldsymbol{\delta}_{m-1}\| \le \epsilon_{m-1}} \|\Delta \boldsymbol{z}_m\|_2 \le \frac{\|\nabla_{\boldsymbol{z}_m} l_m(\boldsymbol{z}_m, y)\|_2}{\mu_m} + \sqrt{\frac{2c_m}{\mu_m} + \frac{\|\nabla_{\boldsymbol{z}_m} l_m(\boldsymbol{z}_m, y)\|_2^2}{\mu_m^2}}. \tag{8}$$

According to Proposition 1 and Lemma 1, we propose the following two designs for each module m to attain robustness of the backbone model in the joint loss l:

(1) Use a linear layer $\boldsymbol{\theta}_m = \{\boldsymbol{W}_m, \boldsymbol{b}_m\}$ (i.e., a fully connected layer) as the auxiliary model.

(2) Conduct adversarial training on the early exit loss with strong convexity regularization:

$$\begin{aligned} &\min_{\boldsymbol{w}_m, \boldsymbol{\theta}_m} \max_{\|\boldsymbol{\delta}_{m-1}\| \le \epsilon_{m-1}} l_m(\boldsymbol{z}_{m-1} + \boldsymbol{\delta}_{m-1}, y; \boldsymbol{w}_m, \boldsymbol{\theta}_m) \\ &= \min_{\boldsymbol{w}_m, \boldsymbol{\theta}_m} \max_{\|\boldsymbol{\delta}_{m-1}\| \le \epsilon_{m-1}} \Big[\frac{\mu}{2} \|\boldsymbol{z}_m(\boldsymbol{z}_{m-1} + \boldsymbol{\delta}_{m-1}; \boldsymbol{w}_m)\|_2^2 \\ &\qquad + l_{\text{CE}}(\boldsymbol{W}_m^T \boldsymbol{z}_m(\boldsymbol{z}_{m-1} + \boldsymbol{\delta}_{m-1}; \boldsymbol{w}_m) + \boldsymbol{b}_m, y) \Big]. \end{aligned} \tag{9}$$

The auxiliary model with a single fully connected layer and the cross-entropy (CE) loss can guarantee convexity but not strong convexity. Thus, we add an ℓ_2 regularizer on $\boldsymbol{z}_m$ to attain strong convexity. By adopting this strong-convexity regularized loss in all the modules, we can achieve upper bounded perturbation on the output feature of each module, thereby ensuring the sufficient condition for backbone robustness in Proposition 1.

Remark 1. Notice that the weak convexity of the linear layer is not sufficient to upper bound the perturbation on $\boldsymbol{z}_m$. The perturbation on $\boldsymbol{z}_m$ that lies in the null space of $\boldsymbol{W}_m$ can be arbitrarily large while not changing the output.

5.2 Robustness-Consistency Relationship in Adversarial Cascade Learning

While we have guaranteed the backbone robustness with adversarial cascade learning, the poor performance caused by the objective inconsistency of cascade learning is not addressed. Since the early exit loss l_m is not equivalent to the joint loss l, the gradient $\nabla_{\boldsymbol{w}_m} l_m$ provided by the early exit loss is not the same as $\nabla_{\boldsymbol{w}_m} l$ either, leading to suboptimal model updates. To mitigate the inconsistency, we need to reduce the gradient difference $\|\nabla_{\boldsymbol{w}_m} l - \nabla_{\boldsymbol{w}_m} l_m\|$.

There is usually no upper bound on the gradient difference since the auxiliary model can be arbitrarily different from the backbone model. However, under the adversarial cascade learning with both module robustness and backbone model robustness, we can derive an upper bound on the gradient difference with the following lemma:

Lemma 2. *If the early exit loss l_m has β_m-smoothness and (ϵ_m, c_m)-robustness on $\boldsymbol{z}_m$, the joint loss l has β'_m-smoothness and (ϵ_m, c_M)-robustness on $\boldsymbol{z}_m$, we have*

$$\|\nabla_{\boldsymbol{w}_m} l - \nabla_{\boldsymbol{w}_m} l_m\|_2 \le \left\| \frac{\partial \boldsymbol{z}_m}{\partial \boldsymbol{w}_m} \right\|_2 \sqrt{2(c_m + c_M)(\beta_m + \beta'_m)}. \tag{10}$$

Lemma 2 is proved in Appendix A.2. An intuitive explanation is that if both l_m and l are not sensitive to the perturbation on $\boldsymbol{z}_m$ (i.e., smooth and robust), their gradients on $\boldsymbol{z}_m$ are close to 0 simultaneously and thus the gradient difference is small. A previous study has shown that the robustness of a deep neural network also implies smoothness (Moosavi-Dezfooli et al., 2019). Thus, we attain small

Algorithm 1: Memory-constrained Model Partition

Require: The "atom" sequence $(a_1 \circ \cdots \circ a_L)$;
Minimal reserved memory R_{min}
Initialize $\mathbb{M} = \emptyset, m = \emptyset$;
for $i \leq L$ **do**
  if *MemReq*$(m \cup \{a_i\}) < R_{min}$ **then**
    Append a_i to m;
  else
    Append m to $\mathbb{M}$;
    $m \leftarrow \{a_i\}$;
Append m to $\mathbb{M}$;
Result: Model partition $\mathbb{M}$

β_m and c_m with the module robustness of module m, and we attain small β'_m, c_M with the backbone robustness of the whole model. In conclusion, our adversarial cascaded learning can also mitigate objective inconsistency.

6 Central Server Design

In this section, we introduce how the server coordinates the training in FedProphet with three components: model partitioner (Section 6.1), training coordinator (Adaptive Perturbation Adjustment in Section 6.2 and Differentiated Module Assignment in Section 6.3), and partial-average model aggregator (Section 6.4).

6.1 Memory-constrained Model Partition

The model partitioner partitions the backbone model into cascaded modules and each module can be independently trained with an auxiliary model. To formalize the model partitioning, we first define the **"atom"** which cannot be further partitioned. An "atom" of a model is a layer or a block such that the backbone model is constructed as a plain cascade of multiple "atoms" $(a_1 \circ \cdots \circ a_L)$. For example, the "atom" of a plain neural network (e.g., VGG (Simonyan & Zisserman, 2014)) is a single layer, while the "atom" of ResNet (He et al., 2016) is a residual block. A module consists of several connected "atoms", with an extra auxiliary model appended to the end.

The key of the model partitioner is to ensure that the memory requirement for training each module does not exceed the minimal reserved memory R_{min} among all the clients, such that the clients can train at least one module without memory swapping in any communication round. We adopt a greedy model partitioning method given in Algorithm 1, which can achieve the least number of modules under the memory constraint. Specifically, we traverse each "atom" in the model and append it into one module until it reaches the memory constraint. The MemReq(m) in Algorithm 1 is a function that returns the memory requirement for training the module m. In this paper, we adopt the methodology proposed by Rajbhandari et al. (2020) to estimate the memory requirement, considering model parameters, gradients, optimizer states, and intermediate activations.

6.2 Adaptive Perturbation Adjustment

When conducting adversarial training on module m, though Proposition 1 requires setting the perturbation constraint ϵ_{m-1} as the upper bound of $\Delta \boldsymbol{z}_{m-1}$ to sufficiently guarantee the backbone robustness, we find that it is not necessary to use such a large perturbation magnitude in practice. On the one hand, a too-large perturbation magnitude on the intermediate features can cause a significant accuracy drop and even lead to divergence. On the other hand, a too-small perturbation magnitude cannot confer strong robustness to the backbone model. Therefore, it is essential to find an appropriate perturbation magnitude for each module to achieve the best utility-robustness trade-off.

Since the optimal perturbation constraint may differ in different modules, we propose **Adaptive Perturbation Adjustment** mechanism to automatically adjust the perturbation magnitudes during training. When module $m-1$ is fixed after convergence, we collect the largest perturbation $\Delta \boldsymbol{z}_{m-1}$ on its output with the adversarial perturbation on its input $\boldsymbol{z}_{m-2}$ from all clients. Then we set the constraint $\epsilon_{m-1}^{(t)}$ for training the next module m based on the averaged perturbation magnitude on $\boldsymbol{z}_{m-1}$ as follows[3]:

$$\epsilon_{m-1}^{(t)} = \alpha_{m-1}^{(t)} \mathbb{E}\left[\max_{\|\boldsymbol{\delta}_{m-2}\| \leq \epsilon_{m-2}^{*}} \|\Delta \mathbf{z}_{m-1}\|\right]. \tag{11}$$

Adaptive Perturbation Adjustment adaptively tunes the scaling factor $\alpha_{m-1}^{(t)}$ at each communication round t to balance the utility and robustness. The foundation of this mechanism is that the ratio between the clean accuracy (accuracy on clean examples) and the adversarial accuracy (accuracy on adversarial examples) reveals the balance between utility and robustness, and this ratio should not change significantly when cascading one more module. Therefore, we monitor the ratio between the clean accuracy and the adversarial accuracy during training, and we adjust $\alpha_{m-1}^{(t)}$ by comparing the accuracy ratio of this module and the previous module as follows:

$$\alpha_{m-1}^{(t)} = \begin{cases} \alpha_{m-1}^{(t-1)} + \Delta\alpha, & \text{if } \frac{C_m^{(t)}}{A_m^{(t)}} > (1+\gamma)\frac{C_{m-1}^{*}}{A_{m-1}^{*}}; \\ \alpha_{m-1}^{(t-1)} - \Delta\alpha, & \text{if } \frac{C_m^{(t)}}{A_m^{(t)}} < (1-\gamma)\frac{C_{m-1}^{*}}{A_{m-1}^{*}}; \\ \alpha_{m-1}^{(t-1)}, & \text{elsewhere.} \end{cases} \tag{12}$$

$C_m^{(t)}$ and $A_m^{(t)}$ are the validation clean accuracy and adversarial accuracy of the cascaded modules $(\boldsymbol{w}_1^* \circ \boldsymbol{w}_2^* \circ \cdots \circ$

[3]Notice that we do not adjust ϵ_0 for the original data and always set it as a predefined value, e.g., $8/255$.

$\boldsymbol{w}_m^{(t)}$) at communication round t. C_{m-1}^* and A_{m-1}^* denote the final clean and adversarial accuracy of $(\boldsymbol{w}_1^* \circ \boldsymbol{w}_2^* \circ \cdots \circ \boldsymbol{w}_{m-1}^*)$ when completing training module $m-1$ and fixing it. γ is a small threshold constant, e.g., 0.05 in our experiments. When the accuracy ratio is too large, which means that the clean accuracy is too high and the adversarial accuracy is too low, we increase the scaling factor $\alpha_{m-1}^{(t)}$ by a small constant $\Delta\alpha$ (e.g., 0.1 in our experiments) to enhance the robustness, and vice versa.

6.3 Differentiated Module Assignment

Since different clients in federated learning may have different available hardware resources, it is possible that some of them can train multiple modules or even the whole backbone model at a time with sufficient memory and performance. Cascading more modules and training them together can equivalently reduce the number of modules and mitigate objective inconsistency in cascade learning (Wang et al., 2021b). Thus, we propose **Differentiated Module Assignment** in the training coordinator to fully utilize the computational resources of resource-sufficient clients.

Differentiated Module Assignment mechanism assigns different numbers of modules to clients in each communication round according to their real-time available resources. A client k who is assigned multiple modules $\{m, m+1, \cdots M_k^{(t)}\}$ in round t trains the cascaded modules jointly with the following loss:

$$\begin{aligned} & l_k^{(t)}\left(\boldsymbol{z}_{m-1}, y; \boldsymbol{w}_m, \cdots, \boldsymbol{w}_{M_k^{(t)}}, \boldsymbol{\theta}_{M_k^{(t)}}\right) \\ =& l_{\text{CE}}\left(\boldsymbol{W}_{M_k^{(t)}}^T \boldsymbol{z}_{M_k^{(t)}}(\boldsymbol{z}_{m-1}; \boldsymbol{w}_m, \cdots, \boldsymbol{w}_{M_k^{(t)}}) + \boldsymbol{b}_{M_k^{(t)}}, y\right) \\ & + \frac{\mu}{2}\|\boldsymbol{z}_{M_k^{(t)}}(\boldsymbol{z}_{m-1}; \boldsymbol{w}_m, \cdots, \boldsymbol{w}_{M_k^{(t)}})\|_2^2, \end{aligned} \tag{13}$$

where $\boldsymbol{z}_{M_k^{(t)}}(\boldsymbol{z}_{m-1})$ calculates the feature $\boldsymbol{z}_{M_k^{(t)}}$ by forward propagating the input $\boldsymbol{z}_{m-1}$ through the cascaded modules $(\boldsymbol{w}_m \circ \boldsymbol{w}_{m+1} \circ \cdots \circ \boldsymbol{w}_{M_k^{(t)}})$. Then the early exit loss provided by the auxiliary model $\boldsymbol{\theta}_{M_k^{(t)}}$ of the last assigned module $M_k^{(t)}$, together with the strong convexity regularization, is used to train the cascaded modules.

Resource-constrained Module Assignment. We now discuss how to assign modules to maximize the utilization of the resources of each client. When determining the module assignment for a client k, we choose the largest $M_k^{(t)}$ that satisfies both of the following two constraints:

(1) The total memory requirement of the assigned modules should not exceed the available memory $R_k^{(t)}$:

$$\text{MemReq}(\boldsymbol{w}_m \circ \cdots \circ \boldsymbol{w}_{M_k^{(t)}} \circ \boldsymbol{\theta}_{M_k^{(t)}}) \leq R_k^{(t)}. \tag{14}$$

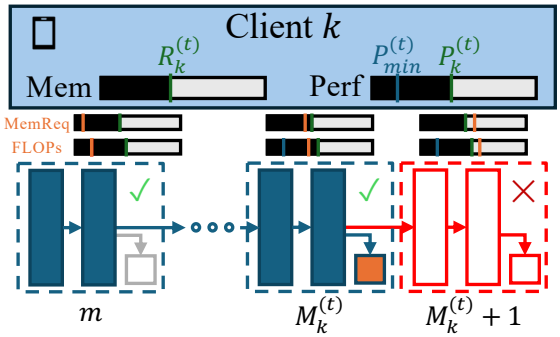

Figure 5. An illustration of resource-constrained module assignment with memory and FLOPs constraints.

(2) Training the assigned modules on client k should not take longer than training a single module m on the slowest client:

$$\text{FLOPs}(\boldsymbol{w}_m \circ \cdots \circ \boldsymbol{w}_{M_k^{(t)}} \circ \boldsymbol{\theta}_{M_k^{(t)}}) \leq \frac{P_k^{(t)}}{P_{\text{min}}^{(t)}} \text{FLOPs}(\boldsymbol{w}_m). \tag{15}$$

MemReq() returns the memory requirement of the given model, and FLOPs() returns the FLOPs for training the model. $P_k^{(t)}$ is the available performance of client k at round t and $P_{\text{min}}^{(t)}$ is the lowest available performance among clients who participate in the training of round t. As shown in Figure 5, we increase the number of modules that are assigned to a "prophet" client when both the memory constraint (Equation (14)) and the FLOPs constraint (Equation (15)) can be satisfied. The memory constraint avoids the data access latency incurred by memory swapping, while the FLOPs constraint minimizes the synchronization time of each communication round in FL by setting a hard limit on the training time of the assigned modules, which should be no more than the time of training only one module m on the slowest client.

6.4 Partial-Average Model Aggregator

With the Differentiated Module Assignment mechanism, the server needs to aggregate the updated local models with different numbers of modules from different clients. Similar to previous partial-training FL algorithms (Caldas et al., 2018; Diao et al., 2020; Alam et al., 2022), we adopt partial average to aggregate local models with different numbers of modules. For each module $m \leq n \leq M$, the module parameter $\boldsymbol{w}_n$ is aggregated as:

$$\boldsymbol{w}_n^{(t+1)} = \frac{\sum_{k \in \mathbb{S}_n^{(t)}} q_k \boldsymbol{w}_{n,k}^{(t,E)}}{\sum_{k \in \mathbb{S}_n^{(t)}} q_k}, \quad \mathbb{S}_n^{(t)} = \{k : M_k^{(t)} \geq n\}, \tag{16}$$

where $\mathbb{S}_n^{(t)}$ is the set of clients who trained module n in communication round t, and $\boldsymbol{w}_{n,k}^{(t,E)}$ is the local module parameters trained by client k for E local iterations in this

Algorithm 2: FedProphet

Require: The initial model $\boldsymbol{w}^{(0)}$; Minimal reserved memory R_{min}; Strong convexity hyperparameter μ.
[**Server**]: partitions the model into M modules $\boldsymbol{w} = \{\boldsymbol{w}_1, \cdots, \boldsymbol{w}_M\}$ according to R_{min} (Sec. 6.1);
[**Server**]: Round $t \leftarrow 0$, broadcasts $\boldsymbol{w}^{(0)}$;
for *Module* $m = 1, \cdots, M$ **do**
  while $\boldsymbol{w}_m$ *does not converge* **do**
    if $m > 1$ **then**
      [**Server**]: adjusts $\epsilon_{m-1}^{(t)}$ (Sec. 6.2);
    for *each Client* k **do**
      [**Server**]: assigns $M_k^{(t)}$ (Sec. 6.3);
      [**Client**]: conducts adversarial training on $\{m, \cdots, M_k^{(t)}\}$ with $\epsilon_{m-1} = \epsilon_{m-1}^{(t)}$ and $l_m = l_k^{(t)}$ in Eq. (9) (Sec. 5.1&6.3);
      [**Client**]: uploads trained modules;
    [**Server**]: aggregates modules (Sec. 6.4);
    [**Server**]: Round $t \leftarrow t+1$, broadcasts $\boldsymbol{w}^{(t)}$;
  [**All Clients**]: Fix $\boldsymbol{w}_m^* \leftarrow \boldsymbol{w}_m^{(t)}, \epsilon_{m-1}^* \leftarrow \epsilon_{m-1}^{(t)}$;
  [**Server**]: Collects $\max_{\boldsymbol{\delta}_{m-1}} \|\Delta \boldsymbol{z}_m\|$ from clients;
Result: Trained model $\boldsymbol{w}^* = \{\boldsymbol{w}_1^*, \cdots, \boldsymbol{w}_M^*\}$.

round. We also aggregate the auxiliary model $\boldsymbol{\theta}_n$ similarly:

$$\boldsymbol{\theta}_n^{(t+1)} = \frac{\sum_{k \in \mathbb{K}_n^{(t)}} q_k \boldsymbol{\theta}_{n,k}^{(t,E)}}{\sum_{k \in \mathbb{K}_n^{(t)}} q_k}, \quad \mathbb{K}_n^{(t)} = \{k : M_k^{(t)} = n\}. \tag{17}$$

We summarize FedProphet in Algorithm 2.

7 Empirical Evaluation

7.1 Experiment Setup

Datasets and Statistical Heterogeneity. We adopt two popular image classification datasets, CIFAR-10 with 10 classes of $3 \times 32 \times 32$ images (Krizhevsky et al., 2009) and Caltech-256 with 256 classes of $3 \times 224 \times 224$ images (Griffin et al., 2007), for empirical evaluation. For both datasets, we partition the whole training set onto $N = 100$ clients and we randomly select $C = 10$ clients to participate in training at each round. We adopt the same statistical heterogeneity as in previous FAT literature (Shah et al., 2021): On each client, 80% training data belongs to around 20% classes (i.e., 2 classes in CIFAR-10 and 46 classes in Caltech-256), and 20% data belongs to the other classes.

Devices and Systematic Heterogeneity. We collect device pools consisting of common edge devices, with details in Appendix B.1. When sampling the devices from the pool, we emulate two levels of systematic heterogeneity: **(a) balanced:** We sample different devices with equal probability; **(b) unbalanced:** We give a higher sampling probability for devices with smaller memory and lower performance. Figure 6 shows the distribution of memory and performance in the real-time device samplings.

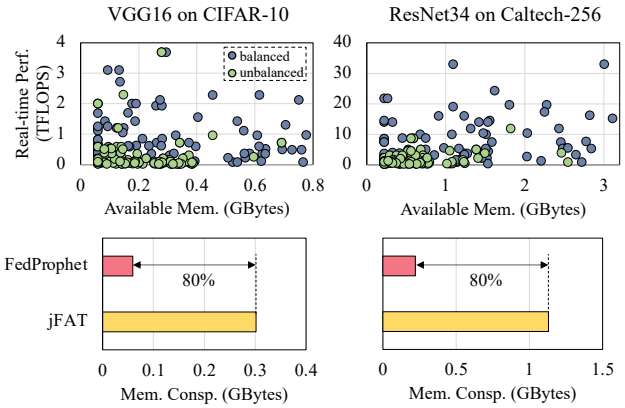

Figure 6. The balanced (blue dots) /unbalanced (green dots) device samplings (upper), and the memory consumption of jFAT and FedProphet (lower) on two workloads.

Models and Evaluation Metrics. We conduct PGD-10 adversarial training (Madry et al., 2017) with VGG16 (Simonyan & Zisserman, 2014) on CIFAR-10, and ResNet34 (He et al., 2016) on Caltech-256. We report the test accuracy on clean data (Clean Acc.), and we conduct PGD-20 attack (PGD Acc. or Adv. Acc.) and Auto Attack (AA Acc.) (Croce & Hein, 2020) to evaluate the robustness of the model. Following previous adversarial training literature (Zhang & Zhu, 2019), the perturbations on both training data and test data are bounded by ℓ_∞ norm with $\epsilon_0 = 8/255$. We report the training time (including computation time and data access time) as the efficiency metric.

Baselines. We compare FedProphet with joint federated adversarial learning (jFAT) (Zizzo et al., 2020), knowledge-distillation federated adversarial training (FedDF-AT (Lin et al., 2020), FedET-AT (Cho et al., 2022)), partial-training federated adversarial training (HeteroFL-AT (Diao et al., 2020), FedDrop-AT (Wen et al., 2022), FedRolex-AT (Alam et al., 2022)), and Federated Robustness Propagation (FedRBN) (Hong et al., 2023). We provide a detailed description of the baselines in Appendix B.2.

7.2 Performance of FedProphet

Memory Requirements. We set the minimal reserved memory R_{min} of memory-constrained clients to be around 20% of the memory required for training the whole model, as shown by Figure 6. Specifically, $R_{\text{min}} = 60$ MBytes when training VGG16 (Requires 302 MBytes) on CIFAR-10, and $R_{\text{min}} = 224$ MBytes when training ResNet34 (Requires 1130 MBytes) on Caltech-256. FedProphet parti-

Table 2. Clean Accuracy (Clean Acc.) and Adversarial Accuracy against PGD (PGD Acc.) and AutoAttack (AA Acc.). We highlight the best results among all methods besides jFAT which requires more memory or memory swapping when training.

Dataset	CIFAR-10 (32×32)						Caltech-256 (224×224)					
Sys. Hetero.	balanced			unbalanced			balanced			unbalanced		
Method	Clean Acc.	PGD Acc.	AA Acc.	Clean Acc.	PGD Acc.	AA Acc.	Clean Acc.	PGD Acc.	AA Acc.	Clean Acc.	PGD Acc.	AA Acc.
jFAT	79.74%	56.76%	55.01%	79.74%	56.76%	55.01%	46.56%	19.76%	18.36%	46.56%	19.76%	18.36%
FedDF-AT	47.77%	24.88%	18.72%	48.16%	25.39%	18.34%	6.74%	4.83%	4.10%	11.78%	0.09%	0%
FedET-AT	40.73%	7.29%	5.12%	34.91%	8.74%	5.54%	11.48%	2.76%	2.44%	16.49%	1.92%	1.73%
HeteroFL-AT	51.63%	39.36%	38.47%	55.25%	43.05%	41.96%	27.80%	8.70%	8.15%	9.43%	3.04%	2.87%
FedDrop-AT	65.92%	54.21%	53.23%	63.26%	53.21%	52.61%	27.10%	11.87%	10.05%	11.68%	6.54%	5.20%
FedRolex-AT	67.14%	54.13%	53.51%	66.44%	53.25%	52.00%	30.18%	11.78%	9.84%	12.51%	5.80%	4.81%
FedRBN	**84.81%**	42.88%	39.82%	**86.70%**	42.99%	39.85%	**78.38%**	3.14%	0%	**78.81%**	1.43%	0%
FedProphet	77.79%	**59.22%**	**57.89%**	76.47%	**59.51%**	**58.64%**	47.07%	**19.10%**	**18.11%**	43.39%	**14.93%**	**14.41%**

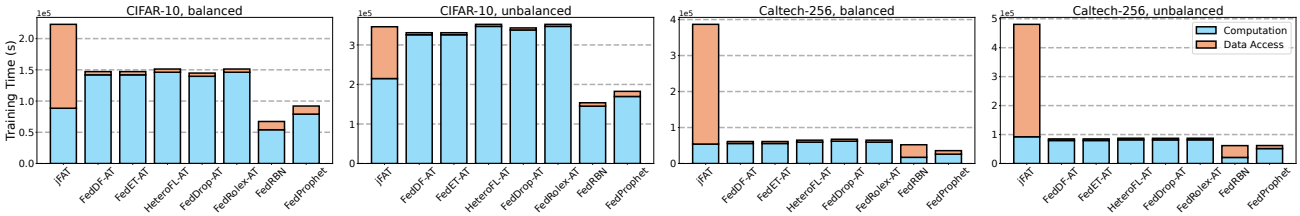

Figure 7. Training time (including computation time and data access time) of baselines and FedProphet.

tions both VGG16 and ResNet34 into 7 modules with Algorithm 1 to avoid memory swapping. In other words, FedProphet reduces the theoretical memory requirement by 80% in comparison to jFAT. Appendix B.3 provides more details of the model partition.

Utility and Robustness. The utility (clean accuracy) and robustness (adversarial accuracy) of different methods are reported in Table 2. Compared to jFAT, only FedProphet can maintain comparable or even higher utility and robustness simultaneously in all our settings. The objective inconsistency between the local models and the global model leads to suboptimal model updates and poor performance of previous memory-efficient baselines. Although FedRBN avoids objective inconsistency with homogeneous models and achieves high clean accuracy, it fails to attain strong robustness under high systematic heterogeneity where most clients cannot afford joint adversarial training (Hong et al., 2023). FedProphet guarantees backbone robustness while overcoming the objective inconsistency, attaining significantly better utility and robustness than these baselines.

Training Efficiency. Figure 7 shows the total training time of different methods, including the computation time and data access time. The high frequency of memory swapping in jFAT when training the large backbone model incurs significant data access time and slows down the training procedure. The other memory-efficient methods avoid training the memory-exceeding large model and thus the data access time is much smaller than that of jFAT. However, since only a small part of the whole model is trained in each round, the memory-efficient methods usually require more communication rounds for convergence (Diao et al., 2020; Wen et al., 2022; Alam et al., 2022), as indicated by the higher computation time than jFAT. FedProphet compensates for the extra communication round by adopting the FLOPs-constrained module assignment in Equation (15), which reduces the synchronization time in each communication round of FL. Thus, FedProphet attains low data access time and low computation time simultaneously, with $2.4\times, 1.9\times, 10.8\times, 7.7\times$ speedup in the total training time compared to jFAT in each setting respectively.

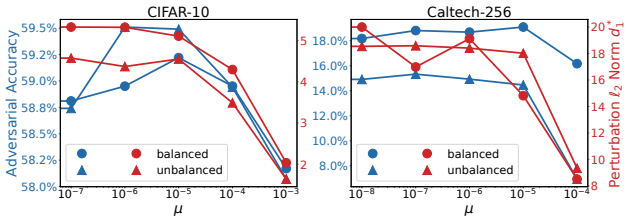

Figure 8. Influence of the strong convexity hyperparameter μ on adversarial accuracy and perturbation magnitude.

7.3 Ablation Study of Components in FedProphet

We conduct ablation studies in this section to evaluate the functionality of each component in FedProphet, including the client-side local trainer, the server-side model partitioner, and the server-side training coordinator.

Local Trainer with Strong Convexity Regularization. We discuss how the strong convexity regularization in the local trainer of each client influences the robustness of FedProphet. Figure 8 shows the adversarial accuracy (in blue color) and the ℓ_2 magnitude of the perturbation $\Delta \boldsymbol{z}_1$, with respect to different strong convexity hyperparameter μ. We can see that increasing μ in the range of $[0, 10^{-5}]$ slightly increases the adversarial accuracy and decreases the per-

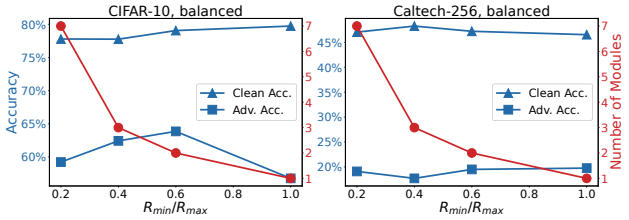

Figure 9. The number of modules and the clean/adversarial accuracy with different R_{min} (given in the ratio of R_{max}).

Table 3. Performance with or without adaptive perturbation adjustment (APA) and differentiated module assignment (DMA).

Dataset		CIFAR-10		Caltech-256	
Sys. Hetero.		balanced	unbalanced	balanced	unbalanced
APA	DMA	Clean / Adv. Acc.	Clean / Adv. Acc.	Clean / Adv. Acc.	Clean / Adv. Acc.
✓	✓	77.79% / 59.22%	76.47% / 59.51%	45.04% / 19.74%	43.39% / 14.93%
✗	✓	79.04% / 56.98%	77.02% / 58.01%	59.99% / 10.80%	53.64% / 8.06%
✓	✗	71.66% / 57.18%	71.66% / 57.18%	14.67% / 7.93%	14.67% / 7.93%
✗	✗	71.68% / 57.34%	71.68% / 57.34%	25.17% / 4.38%	25.17% / 4.38%

turbation magnitude. The insignificant change is attributed to the local strong convexity of the fully connected layer used in the auxiliary model. Though the fully connected layer followed by the cross-entropy loss (or equivalently, multinomial logistic regression) only has convexity instead of strong convexity globally, it can still be locally strongly convex with most inputs (Böhning, 1992). Thus the strong convexity hyperparameter μ does not make a significant difference when it is small. When further increasing μ, the perturbation magnitude begins to drop significantly, which is aligned with the conclusion of Lemma 1. However, using an over-large regularization ($\mu \geq 10^{-4}$) can also distract the training and even lead to divergence.

Model Partitioner. Figure 9 shows the number of modules and the corresponding performance of FedProphet when partitioning the backbone model with different R_{min}. When the clients have more available memory, the number of modules decreases and finally degenerates to jFAT with only one module. However, the performance of FedProphet does not show much difference with different numbers of modules, which also implies the effectiveness of our inconsistency-reduction designs in FedProphet. When training with more than one module in FedProphet, our adversarial cascade learning with strong convexity regularization guarantees the sufficient condition for robustness, thus leading to even higher adversarial accuracy than joint federated adversarial training in some cases.

Training Coordinator. Table 3 shows the performance of FedProphet with/without Adaptive Perturbation Adjustment and Differentiated Module Assignment in the training coordinator. We can get the following two conclusions:

(a) Adaptive Perturbation Adjustment improves robustness and attains better utility-robustness trade-off. When training without Adaptive Perturbation Adjustment

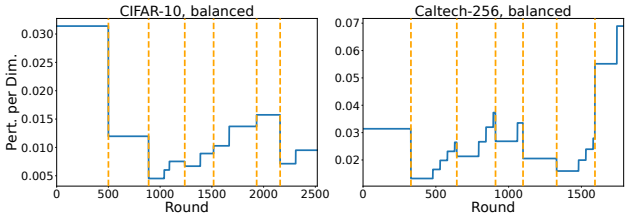

Figure 10. Perturbation magnitude per dimension during the training of FedProphet with adaptive perturbation adjustment in the balanced setting. The orange dash lines divide the training stages of each module $m \in \{1, 2, \cdots, 7\}$.

Table 4. Training time with or without DMA in FedProphet.

Dataset	CIFAR-10		Caltech-256	
Sys. Hetero.	balanced	unbalanced	balanced	unbalanced
w/ DMA	9.2×10^4s	1.8×10^5s	3.6×10^4s	6.2×10^4s
w/o DMA	9.1×10^4s	1.9×10^5s	4.2×10^4s	6.5×10^4s

(APA), FedProphet achieves higher clean accuracy but lower adversarial accuracy. The robustness-utility ratio is lower than FedProphet with APA and jFAT, especially when on Caltech-256. Figure 10 shows that the perturbation magnitude starts from a relatively small value and increases gradually by APA when training each module. We find that initializing α_m with a small value (e.g., 0.3) can stabilize the adversarial training, while APA can adjust the perturbation magnitude to achieve a better balance between the clean accuracy and the adversarial accuracy by comparing the accuracy ratio with the previous module.

(b) Differentiated Module Assignment significantly improves the performance of FedProphet. We can see that both clean and adversarial accuracy drops when removing Differentiated Module Assignment (DMA) from FedProphet, especially on Caltech-256 where the adversarial accuracy is not high enough to mitigate the objective inconsistency with the robustness-consistency relationship in Lemma 2. DMA fully utilizes the resources of each client and assigns "prophet" clients more modules for training, such that the "prophet" clients can train more modules jointly with the "future" loss that is closer to the final loss of the whole backbone model. And Table 4 shows that DMA does not incur extra training latency with the FLOPs constraint in Equation (15), which avoids enlarging the synchronization time in each communication round of FL.

8 Conclusions and Future Works

We propose FedProphet, a memory-efficient federated adversarial training framework with robust and consistent cascade learning. We develop both the client-side local trainer and the server-side training coordinator to achieve high utility and strong robustness simultaneously. On the client side, We theoretically analyze the robustness condition and propose adversarial cascade learning with

strong convexity regularization to guarantee the robustness of the backbone model. We further reveal that the robustness achieved in adversarial cascade learning also implies low objective inconsistency, leading to high utility at the same time. On the server side, we propose the memory-constrained model partitioner to automatically partition a given model to fit into the memory constraints among clients. We derive a training coordinator with Adaptive Perturbation Adjustment and Differentiated Module Assignment on the server to achieve the optimal utility-robustness trade-off and further reduce the objective inconsistency. Our empirical results show that FedProphet consistently outperforms other memory-efficient federated learning methods. Compared with joint training, FedProphet maintains comparable utility and robustness, with significant memory reduction and training speedup.

One future work would be extending FedProphet to NLP tasks. Although the adversarial robustness of NLP tasks is less straightforward since it is difficult to generate adversarial examples with gradient-ascent methods (like PGD and AutoAttack) on the discrete texts, previous studies show that training with adversarially perturbed token embeddings can improve the generalization ability of the language model (Zhu et al., 2019). Therefore, it is meaningful to explore how FedProphet can be applied with language models like Transformers (Vaswani, 2017) to achieve memory efficiency and generalization in NLP tasks.

Another potential direction is to combine FedProphet with other memory-efficient training methods, like low-bit training (Zhong et al., 2022) and LoRA (Hu et al., 2021). Since FedProphet partitions the backbone model with a layer or a block as the "atom", it is complementary to the parameter-level quantization and the layer-level low-rank approximation. Thus, FedProphet can be applied with the other parameter-level or layer-level memory-efficient training methods to further reduce the memory requirement.

ACKNOWLEDGEMENTS

We appreciate the constructive comments of the reviewers. This research is generously supported in part by Gift from Accenture, NSF CNS-2112562, CNS-2233808, CNS-2148253, and ARO W911NF-23-2-0224.

REFERENCES

Alam, S., Liu, L., Yan, M., and Zhang, M. Fedrolex: Model-heterogeneous federated learning with rolling sub-model extraction. *Advances in neural information processing systems*, 35:29677–29690, 2022.

Belilovsky, E., Eickenberg, M., and Oyallon, E. Decoupled greedy learning of cnns. In *International Conference on Machine Learning*, pp. 736–745. PMLR, 2020.

Böhning, D. Multinomial logistic regression algorithm. *Annals of the institute of Statistical Mathematics*, 44(1): 197–200, 1992.

Caldas, S., Konečny, J., McMahan, H. B., and Talwalkar, A. Expanding the reach of federated learning by reducing client resource requirements. *arXiv preprint arXiv:1812.07210*, 2018.

Cho, Y. J., Manoel, A., Joshi, G., Sim, R., and Dimitriadis, D. Heterogeneous ensemble knowledge transfer for training large models in federated learning. *arXiv preprint arXiv:2204.12703*, 2022.

Croce, F. and Hein, M. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *International conference on machine learning*, pp. 2206–2216. PMLR, 2020.

Diao, E., Ding, J., and Tarokh, V. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. *arXiv preprint arXiv:2010.01264*, 2020.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Griffin, G., Holub, A., and Perona, P. Caltech-256 object category dataset. 2007.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

Hettinger, C., Christensen, T., Ehlert, B., Humpherys, J., Jarvis, T., and Wade, S. Forward thinking: Building and training neural networks one layer at a time. *arXiv preprint arXiv:1706.02480*, 2017.

Hong, J., Wang, H., Wang, Z., and Zhou, J. Federated robustness propagation: sharing adversarial robustness in heterogeneous federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 7893–7901, 2023.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

Karimireddy, S. P., Kale, S., Mohri, M., Reddi, S. J., Stich, S. U., and Suresh, A. T. Scaffold: Stochastic controlled averaging for on-device federated learning. *arXiv preprint arXiv:1910.06378*, 2019.

Konečnỳ, J., McMahan, B., and Ramage, D. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

Konečnỳ, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., and Bacon, D. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

Li, T., Sahu, A. K., Talwalkar, A., and Smith, V. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

Lin, T., Kong, L., Stich, S. U., and Jaggi, M. Ensemble distillation for robust model fusion in federated learning. *Advances in Neural Information Processing Systems*, 33: 2351–2363, 2020.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Marquez, E. S., Hare, J. S., and Niranjan, M. Deep cascade learning. *IEEE transactions on neural networks and learning systems*, 29(11):5475–5485, 2018.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pp. 1273–1282. PMLR, 2017.

Moosavi-Dezfooli, S.-M., Fawzi, A., Uesato, J., and Frossard, P. Robustness via curvature regularization, and vice versa. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9078–9086, 2019.

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE, 2020.

Shah, D., Dube, P., Chakraborty, S., and Verma, A. Adversarial training in communication constrained federated learning. *arXiv preprint arXiv:2103.01319*, 2021.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Sun, J., Li, A., Duan, L., Alam, S., Deng, X., Guo, X., Wang, H., Gorlatova, M., Zhang, M., Li, H., et al. Fedsea: A semi-asynchronous federated learning framework for extremely heterogeneous devices. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*, pp. 106–119, 2022.

Tang, M., Ning, X., Wang, Y., Sun, J., Wang, Y., Li, H., and Chen, Y. Fedcor: Correlation-based active client selection strategy for heterogeneous federated learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10102–10111, 2022.

Tian, C., Li, L., Shi, Z., Wang, J., and Xu, C. Harmony: Heterogeneity-aware hierarchical management for federated learning system. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 631–645. IEEE, 2022.

Vaswani, A. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

Wang, G., Qin, H., Jacobs, S. A., Holmes, C., Rajbhandari, S., Ruwase, O., Yan, F., Yang, L., and He, Y. Zero++: Extremely efficient collective communication for giant model training. *arXiv preprint arXiv:2306.10209*, 2023.

Wang, J., Liu, Q., Liang, H., Joshi, G., and Poor, H. V. Tackling the objective inconsistency problem in heterogeneous federated optimization. *arXiv preprint arXiv:2007.07481*, 2020.

Wang, Y., Ma, X., Bailey, J., Yi, J., Zhou, B., and Gu, Q. On the convergence and robustness of adversarial training. *arXiv preprint arXiv:2112.08304*, 2021a.

Wang, Y., Ni, Z., Song, S., Yang, L., and Huang, G. Revisiting locally supervised learning: An alternative to end-to-end training. *arXiv preprint arXiv:2101.10832*, 2021b.

Wen, D., Jeon, K.-J., and Huang, K. Federated dropout—a simple approach for enabling federated learning on resource constrained devices. *IEEE wireless communications letters*, 11(5):923–927, 2022.

Wong, E., Rice, L., and Kolter, J. Z. Fast is better than free: Revisiting adversarial training. *arXiv preprint arXiv:2001.03994*, 2020.

Yang, H., Zhang, J., Dong, H., Inkawhich, N., Gardner, A., Touchet, A., Wilkes, W., Berry, H., and Li, H. Dverge: diversifying vulnerabilities for enhanced robust generation of ensembles. *Advances in Neural Information Processing Systems*, 33:5505–5515, 2020.

Zhang, J., Li, A., Tang, M., Sun, J., Chen, X., Zhang, F., Chen, C., Chen, Y., and Li, H. Fed-cbs: A heterogeneity-aware client sampling mechanism for federated learning via class-imbalance reduction. In *International Conference on Machine Learning*, pp. 41354–41381. PMLR, 2023.

Zhang, T. and Zhu, Z. Interpreting adversarially trained convolutional neural networks. In *International conference on machine learning*, pp. 7502–7511. PMLR, 2019.

Zhong, K., Ning, X., Dai, G., Zhu, Z., Zhao, T., Zeng, S., Wang, Y., and Yang, H. Exploring the potential of low-bit training of convolutional neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(12):5421–5434, 2022.

Zhu, C., Cheng, Y., Gan, Z., Sun, S., Goldstein, T., and Liu, J. Freelb: Enhanced adversarial training for natural language understanding. *arXiv preprint arXiv:1909.11764*, 2019.

Zizzo, G., Rawat, A., Sinn, M., and Buesser, B. Fat: Federated adversarial training. *arXiv preprint arXiv:2012.01791*, 2020.