DOES FEDERATED LEARNING REALLY NEED BACKPROPAGATION?

Anonymous authors
Paper under double-blind review

ABSTRACT

Federated learning (FL) provides general principles for decentralized clients to train a server model collectively without sharing local data. FL is a promising framework with practical applications, but its standard training paradigm requires the clients to backpropagate through the model to compute gradients. Since these clients are typically edge devices and not fully trusted, executing backpropagation on them incurs computational and storage overhead as well as white-box vulnerability. In light of this, we develop backpropagation-free federated learning, dubbed BAFFLE, in which backpropagation is replaced by multiple forward processes to estimate gradients. BAFFLE is 1) memory-efficient and easily fits uploading bandwidth; 2) compatible with inference-only hardware optimization and model quantization or pruning; and 3) well-suited to trusted execution environments, because the clients in BAFFLE only execute forward propagation and return a set of scalars to the server. In experiments, we use BAFFLE to train models from scratch or to finetune pretrained models, achieving empirically acceptable results.

1 INTRODUCTION

Federated learning (FL) allows decentralized clients to collaboratively train a server model (Konečný et al., 2016; McMahan et al., 2017). In each training round, the selected clients compute model gradients or updates on their local private datasets, without explicitly exchanging sample points to the server. While FL describes a promising blueprint and has several applications (Yang et al., 2018; Hard et al., 2018; Li et al., 2020b), the mainstream training paradigm of FL is still gradient-based that requires the clients to locally execute backpropagation, which leads to two practical limitations:

(i) Overhead for edge devices. The clients in FL are usually edge devices, such as mobile phones and IoT sensors, whose hardware is primarily optimized for inference-only purposes (Sharma et al., 2018; Umuroglu et al., 2018), rather than for backpropagation. Due to the limited resources, computationally affordable models running on edge devices are typically quantized and pruned (Wang et al., 2019a), making exact backpropagation difficult. In addition, standard implementations of backpropagation rely on either forward-mode or reverse-mode auto-differentiation in contemporary machine learning packages (Bradbury et al., 2018; Paszke et al., 2019b), which increases storage requirements.

(ii) White-box vulnerability. To facilitate gradient computing, the server regularly distributes its model status to the clients, but this white-box exposure of the model renders the server vulnerable to, e.g., poisoning or inversion attacks from malicious clients (Shokri et al., 2017; Xie et al., 2020; Zhang et al., 2020; Geiping et al., 2020). With that, recent attempts are made to exploit trusted execution environments (TEEs) in FL, which can isolate the model status within a black-box secure area and significantly reduce the success rate of malicious evasion (Chen et al., 2020; Mo et al., 2021; Zhang et al., 2021; Mondal et al., 2021). However, TEEs are highly memory-constrained (Truong et al., 2021), while backpropagation is memory-consuming to restore intermediate states.

While numerous solutions have been proposed to alleviate these limitations (discussed in Appendix B), in this paper, we raise an essential question: *does FL really need backpropagation?* Inspired by the literature on zero-order optimization (Stein, 1981), we intend to substitute backpropagation with multiple forward or inference processes to estimate the gradients. Technically speaking, we propose the framework of **BA**ckpropagation-**F**ree **F**ederated **LE**arning (**BAFFLE**). As illustrated in Figure 1, BAFFLE consists of three conceptual steps: (1) each client locally perturbs the model parameters $2K$ times as $\mathbf{W} \pm \boldsymbol{\delta}_k$ (the server sends the random seed to clients for generating $\{\boldsymbol{\delta}_k\}_{k=1}^{K}$); (2) each client

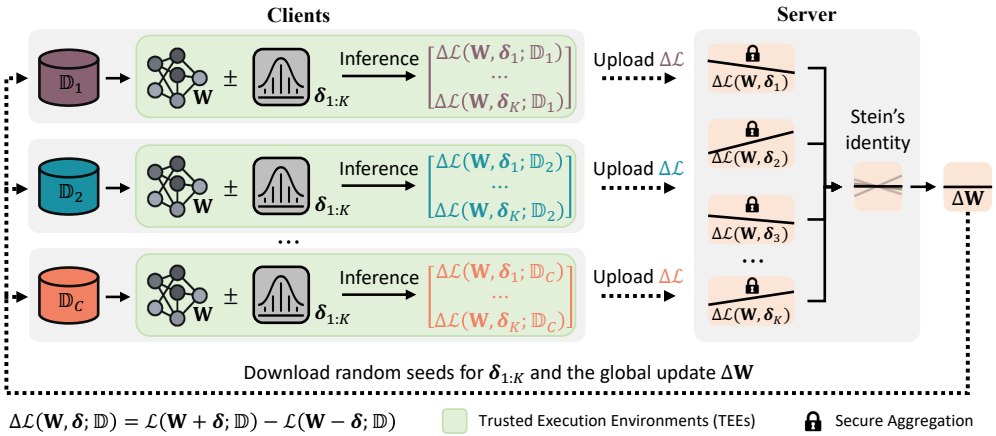


Figure 1: A sketch map of BAFFLE. In addition to the global parameters update $\Delta\mathbf{W}$, each client downloads random seeds to locally generate perturbations $\pm\boldsymbol{\delta}_{1:K}$ and perform $2K$ times of forward propagation (i.e., inference) to compute loss differences. The server can recover these perturbations using the same random seeds and obtain $\Delta\mathcal{L}(\mathbf{W},\boldsymbol{\delta}_k)$ by secure aggregation. Each loss difference $\Delta\mathcal{L}(\mathbf{W},\boldsymbol{\delta}_k;\mathbb{D}_c)$ is a floating-point number, so K can be easily adjusted to fit the uploading bandwidth.

executes forward processes on the perturbed models using its private dataset $\mathbb{D}_c$ and obtains K loss differences $\{\Delta\mathcal{L}(\mathbf{W},\boldsymbol{\delta}_k;\mathbb{D}_c)\}_{k=1}^K$; (3) the server aggregates loss differences to estimate gradients.

BAFFLE's defining characteristic is that it only utilizes forward propagation, which is memory-efficient and does not require auto-differentiation. It is well-adapted to model quantization and pruning as well as inference-only hardware optimization on edge devices. Compared to backpropagation, the computation graph of forward propagation in BAFFLE may be more easily optimized, such as by slicing it into per-layer calculation (Kim et al., 2020). Since each loss difference $\Delta\mathcal{L}(\mathbf{W},\boldsymbol{\delta}_k;\mathbb{D}_c)$ is a scalar, BAFFLE can easily accommodate the uploading bandwidth of clients by adjusting the value of K as opposed to using, e.g., gradient compression (Suresh et al., 2017). BAFFLE is also compatible with recent advances in inference approaches for TEE (Tramer & Boneh, 2019; Truong et al., 2021), providing an efficient solution for combining TEE into FL and preventing white-box evasion.

Base on our convergence analyses, we adapt secure aggregation (Bonawitz et al., 2017a) to zero-order optimization and investigate ways to improve gradient estimation in BAFFLE. In our experiments, BAFFLE is used to train models from scratch on MNIST (LeCun et al., 1998) and CIFAR-10/100 (Krizhevsky & Hinton, 2009), and finetune ImageNet-pretrained models to transfer to OfficeHome (Venkateswara et al., 2017). Compared to conventional FL, BAFFLE achieves sub-optimal but acceptable performance. These results shed light on the potential of BAFFLE and the effectiveness of backpropagation-free methods in FL.

2 Preliminaries

In this section, we introduce the basic concepts of federated learning (FL) (Kairouz et al., 2021) and the finite difference formulas that will serve as the foundation for our methods.

2.1 Federated Learning

Suppose we have C clients, and the c-th client's private dataset is defined as $\mathbb{D}_c := \{(\mathbf{X}_i^c, \mathbf{y}_i^c)\}_{i=1}^{N_c}$ with N_c input-label pairs. Let $\mathcal{L}(\mathbf{W};\mathbb{D}_c)$ represent the loss function calculated on the dataset $\mathbb{D}_c$, where $\mathbf{W} \in \mathbb{R}^n$ denotes the server model's global parameters. The training objective of FL is to find $\mathbf{W}$ that minimize the total loss function as

$$\mathcal{L}(\mathbf{W}) := \sum_{c=1}^{C} \frac{N_c}{N} \mathcal{L}(\mathbf{W};\mathbb{D}_c), \text{ where } N = \sum_{c=1}^{C} N_c. \tag{1}$$

In the conventional FL framework, clients locally compute gradients $\{\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W};\mathbb{D}_c)\}_{c=1}^C$ or model updates through backpropagation and then upload them to the server. Federated average (McMahan et al., 2017) performs global aggregation using $\Delta\mathbf{W} := \sum_{i=1}^C \frac{N_c}{N}\Delta\mathbf{W}_c$, where $\Delta\mathbf{W}_c$ is the local update obtained via executing $\mathbf{W}_c \leftarrow \mathbf{W}_c - \eta\nabla_{\mathbf{W}_c}\mathcal{L}(\mathbf{W}_c;\mathbb{D}_c)$ multiple times and η is learning rate.

2.2 FINITE DIFFERENCE

Gradient-based optimization techniques (either first-order or higher-order) are the most frequently used tools to train deep networks (Goodfellow et al., 2016). Nevertheless, recent progress demonstrates promising applications of zero-order optimization methods for training, particularly when exact derivatives cannot be obtained (Flaxman et al., 2004; Nesterov & Spokoiny, 2017; Liu et al., 2020a) or backward processes are computationally prohibitive (Pang et al., 2020; He et al., 2022). Zero-order approaches require only multiple forward processes that may be executed in parallel. Along this routine, finite difference stems from the definition of derivatives and can be generalized to higher-order and multivariate cases by Taylor's expansion. For any differentiable loss function $\mathcal{L}(\mathbf{W};\mathbb{D})$ and a small perturbation $\boldsymbol{\delta} \in \mathbb{R}^n$, finite difference employs the *forward difference scheme*

$$\mathcal{L}(\mathbf{W}+\boldsymbol{\delta};\mathbb{D}) - \mathcal{L}(\mathbf{W};\mathbb{D}) = \boldsymbol{\delta}^\top \nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W};\mathbb{D}) + o(\|\boldsymbol{\delta}\|_2), \tag{2}$$

where $\boldsymbol{\delta}^\top \nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W};\mathbb{D})$ is a scaled directional derivative along $\boldsymbol{\delta}$. Furthermore, we can use the *central difference scheme* to obtain higher-order residuals as

$$\mathcal{L}(\mathbf{W}+\boldsymbol{\delta};\mathbb{D}) - \mathcal{L}(\mathbf{W}-\boldsymbol{\delta};\mathbb{D}) = 2\boldsymbol{\delta}^\top \nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W};\mathbb{D}) + o(\|\boldsymbol{\delta}\|_2^2). \tag{3}$$

Finite difference formulas are typically used to estimate quantities such as gradient norm or Hessian trace, where $\boldsymbol{\delta}$ is sampled from random projection vectors (Pang et al., 2020).

3 BAFFLE: BACKPROPAGATION-FREE FEDERATED LEARNING

In this section, we introduce zero-order optimization techniques into FL and develop BAFFLE, a backpropagation-free federated learning framework that uses multiple forward processes in place of backpropagation. An initial attempt is to apply finite difference as the gradient estimator. To estimate the full gradients, we need to perturb each parameter $w \in \mathbf{W}$ once to approximate the partial derivative $\frac{\partial \mathcal{L}(\mathbf{W};\mathbb{D})}{\partial w}$, causing the forward computations to grow with n (recall that $\mathbf{W} \in \mathbb{R}^n$) and making it difficult to scale to large models. In light of this, we resort to Stein's identity (Stein, 1981) to obtain an unbiased estimation of gradients from loss differences calculated on various perturbations. As depicted in Figure 1, BAFFLE clients need only download random seeds and global parameters update, generate perturbations locally, execute multiple forward propagations and upload loss differences back to the server. Furthermore, we also present convergence analyses of BAFFLE, which provides guidelines for model design and acceleration of training progress.

3.1 UNBIASED GRADIENT ESTIMATION WITH STEIN'S IDENTITY

Previous work on sign-based optimization (Moulay et al., 2019) demonstrates that deep networks can be effectively trained if the majority of gradients have proper signs. Thus, we propose performing forward propagation multiple times on perturbed parameters, in order to obtain a stochastic estimation of gradients without backpropagation. Specifically, assuming that the loss function $\mathcal{L}(\mathbf{W};\mathbb{D})$ is continuously differentiable w.r.t. $\mathbf{W}$ given any dataset $\mathbb{D}$, which is true (almost everywhere) for deep networks using non-linear activation functions, we define a smoothed loss function

$$\mathcal{L}_\sigma(\mathbf{W};\mathbb{D}) := \mathbb{E}_{\boldsymbol{\delta}\sim\mathcal{N}(0,\sigma^2\mathbf{I})} \mathcal{L}(\mathbf{W}+\boldsymbol{\delta};\mathbb{D}), \tag{4}$$

where the perturbation $\boldsymbol{\delta}$ follows a Gaussian distribution with zero mean and covariance $\sigma^2\mathbf{I}$. Given this, Stein (1981) proves the *Stein's identity* (we recap the proof in Appendix A.1), formulated as

$$\nabla_{\mathbf{W}} \mathcal{L}_\sigma(\mathbf{W};\mathbb{D}) = \mathbb{E}_{\boldsymbol{\delta}\sim\mathcal{N}(0,\sigma^2\mathbf{I})} \left[\frac{\boldsymbol{\delta}}{2\sigma^2} \Delta\mathcal{L}(\mathbf{W},\boldsymbol{\delta};\mathbb{D}) \right], \tag{5}$$

where $\Delta\mathcal{L}(\mathbf{W},\boldsymbol{\delta};\mathbb{D}) := \mathcal{L}(\mathbf{W}+\boldsymbol{\delta};\mathbb{D}) - \mathcal{L}(\mathbf{W}-\boldsymbol{\delta};\mathbb{D})$ is the loss difference. Note that computing a loss difference only requires the execution of two forward processes $\mathcal{L}(\mathbf{W}+\boldsymbol{\delta};\mathbb{D})$ and $\mathcal{L}(\mathbf{W}-\boldsymbol{\delta};\mathbb{D})$ without backpropagation. It is straightforward to show that $\mathcal{L}_\sigma(\mathbf{W};\mathbb{D})$ is continuously differentiable for any $\sigma \geq 0$ and $\nabla_{\mathbf{W}} \mathcal{L}_\sigma(\mathbf{W};\mathbb{D})$ converges uniformly as $\sigma \to 0$; hence, it follows that $\nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W};\mathbb{D}) = \lim_{\sigma\to 0} \nabla_{\mathbf{W}} \mathcal{L}_\sigma(\mathbf{W};\mathbb{D})$. Therefore, we can obtain a stochastic estimation of gradients using Monte Carlo approximation by 1) selecting a small value of σ; 2) randomly sampling K perturbations from $\mathcal{N}(0,\sigma^2\mathbf{I})$ as $\{\boldsymbol{\delta}_k\}_{k=1}^K$; and 3) utilizing the Stein's identity in Eq. (5) to calculate

$$\widehat{\nabla_{\mathbf{W}}} \mathcal{L}(\mathbf{W};\mathbb{D}) := \frac{1}{K} \sum_{k=1}^{K} \left[\frac{\boldsymbol{\delta}_k}{2\sigma^2} \Delta\mathcal{L}(\mathbf{W},\boldsymbol{\delta}_k;\mathbb{D}) \right]. \tag{6}$$

Algorithm 1 Backpropagation-free federated learning (BAFFLE)

1: **Notations:** Se denotes the operations done on servers; Cl denotes the operations done on clients; TEE for the TEE module; and $\Rightarrow$ denotes the communication process.
2: **Inputs:** C clients with local dataset $\{\mathbb{D}_c\}_{c=1}^{C}$ containing N_c input-label pairs, $N = \sum_{c=1}^{C} N_c$; learning rate η, training iterations T, perturbation number K, noise scale σ.
3: Se: initializing model parameters $\mathbf{W} \leftarrow \mathbf{W}_0$;
4: Se: encoding the computing paradigm into TEE as TEE $\circ\, \Delta\mathcal{L}(\mathbf{W}, \boldsymbol{\delta}; \mathbb{D})$; *# optional*
5: **for** $t = 0$ **to** $T-1$ **do**
6: Se $\Rightarrow$ all Cl: downloading model parameters $\mathbf{W}_t$ and the computing paradigm;
7: Se $\Rightarrow$ all Cl: downloading the random seed s_t; *# 4 Bytes*
8: Se: sampling K perturbations $\{\boldsymbol{\delta}_k\}_{k=1}^{K}$ from $\mathcal{N}(0, \sigma^2\mathbf{I})$ using the random seed s_t;
9: all Cl: negotiating a group of zero-sum noises $\{\boldsymbol{\epsilon}_c\}_{c=1}^{C}$ for secure aggregation;
10: **for** $c = 1$ **to** C **do**
11: Cl: sampling K perturbations $\{\boldsymbol{\delta}_k\}_{k=1}^{K}$ from $\mathcal{N}(0, \sigma^2\mathbf{I})$ using the random seed s_t;
12: Cl: computing TEE $\circ\, \Delta\mathcal{L}(\mathbf{W}_t, \boldsymbol{\delta}_k; \mathbb{D}_c)$ via forward propagation for each k;
13: Cl $\Rightarrow$ Se: uploading K outputs $\left\{\text{TEE} \circ \Delta\mathcal{L}(\mathbf{W}_t, \boldsymbol{\delta}_k; \mathbb{D}_c) + \frac{N}{N_c}\boldsymbol{\epsilon}_c\right\}_{k=1}^{K}$; *# $4\times K$ Bytes*
14: **end for**
15: Se: aggregating $\Delta\mathcal{L}(\mathbf{W}_t, \boldsymbol{\delta}_k) \leftarrow \sum_{c=1}^{C} \frac{N_c}{N}\left[\text{TEE} \circ \Delta\mathcal{L}(\mathbf{W}_t, \boldsymbol{\delta}_k; \mathbb{D}_c) + \frac{N}{N_c}\boldsymbol{\epsilon}_c\right]$ for each k;
16: Se: computing $\widehat{\nabla_{\mathbf{W}_t}}\mathcal{L}(\mathbf{W}_t) \leftarrow \frac{1}{K}\sum_{k=1}^{K} \frac{\boldsymbol{\delta}_k}{2\sigma^2}\Delta\mathcal{L}(\mathbf{W}_t, \boldsymbol{\delta}_k)$;
17: Se: $\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t - \eta\widehat{\nabla_{\mathbf{W}_t}}\mathcal{L}(\mathbf{W}_t)$;
18: **end for**
19: **Return:** final model parameters $\mathbf{W}_T$.

3.2 Operating flow of BAFFLE

Based on the forward-only gradient estimator $\widehat{\nabla_{\mathbf{W}}}\mathcal{L}(\mathbf{W}; \mathbb{D})$ derived in Eq. (6), we outline the basic operating flow of our BAFFLE system as in Algorithm 1, which consists of the following:

Model initialization. (Lines 3∼4, done by server) The server initializes the model parameters to $\mathbf{W}_0$ and optionally encodes the computing paradigm of loss differences $\Delta\mathcal{L}(\mathbf{W}, \boldsymbol{\delta}; \mathbb{D})$ into the TEE module (see Appendix C for more information on TEE);

Downloading paradigms. (Lines 6∼7, server $\Rightarrow$ all clients) In round t, the server distributes the most recent model parameters $\mathbf{W}_t$ (or the model update $\Delta\mathbf{W}_t = \mathbf{W}_t - \mathbf{W}_{t-1}$) and the computing paradigm to all the C clients. In addition, in BAFFLE, the server sends a random seed s_t (rather than directly sending the perturbations to reduce communication burden);

Local computation. (Lines 11∼12, done by clients) Each client generates K perturbations $\{\boldsymbol{\delta}_k\}_{k=1}^{K}$ locally from $\mathcal{N}(0, \sigma^2\mathbf{I})$ using random seed s_t, and executes the computing paradigm to obtain loss differences. K is chosen adaptively based on clients' upload bandwidth and computation capability;

Uploading loss differences. (Line 13, all clients $\Rightarrow$ server) Each client uploads K noisy outputs $\{\Delta\mathcal{L}(\mathbf{W}_t, \boldsymbol{\delta}_k; \mathbb{D}_c) + \frac{N}{N_c}\boldsymbol{\epsilon}_c\}_{k=1}^{K}$ to the server, where each output is a floating-point number and the noise $\boldsymbol{\epsilon}_c$ is negotiated by all clients to be zero-sum. The Bytes uploaded for K noisy outputs is $4\times K$;

Secure aggregation. (Lines 15∼16, done by server) In order to prevent the server from recovering the exact loss differences and causing privacy leakage (Geiping et al., 2020), we adopt the secure aggregation method (Bonawitz et al., 2017a) that was originally proposed for conventional FL and apply it to BAFFLE. Specifically, all clients negotiate a group of noises $\{\boldsymbol{\epsilon}_c\}_{c=1}^{C}$ satisfying $\sum_{c=1}^{C}\boldsymbol{\epsilon}_c = 0$. Then we can reorganize our gradient estimator as

$$\widehat{\nabla_{\mathbf{W}_t}}\mathcal{L}(\mathbf{W}_t) = \frac{1}{K}\sum_{c=1}^{C}\frac{N_c}{N}\sum_{k=1}^{K}\left[\frac{\boldsymbol{\delta}_k}{2\sigma^2}\Delta\mathcal{L}(\mathbf{W}_t, \boldsymbol{\delta}_k; \mathbb{D}_c)\right] = \frac{1}{K}\sum_{k=1}^{K}\frac{\boldsymbol{\delta}_k}{2\sigma^2}\Delta\mathcal{L}(\mathbf{W}_t, \boldsymbol{\delta}_k), \tag{7}$$

where $\Delta\mathcal{L}(\mathbf{W}_t, \boldsymbol{\delta}_k) = \sum_{c=1}^{C}\frac{N_c}{N}[\Delta\mathcal{L}(\mathbf{W}_t, \boldsymbol{\delta}_k; \mathbb{D}_c) + \frac{N}{N_c}\boldsymbol{\epsilon}_c]$. Since $\{\boldsymbol{\epsilon}_c\}_{c=1}^{C}$ are zero-sum, there is $\Delta\mathcal{L}(\mathbf{W}_t, \boldsymbol{\delta}_k) = \sum_{c=1}^{C}\frac{N_c}{N}\Delta\mathcal{L}(\mathbf{W}_t, \boldsymbol{\delta}_k; \mathbb{D}_c)$ and Eq. (7) holds. Thus, the server can correctly aggregate $\Delta\mathcal{L}(\mathbf{W}_t, \boldsymbol{\delta}_k)$ and protect client privacy without recovering individual $\Delta\mathcal{L}(\mathbf{W}_t, \boldsymbol{\delta}_k; \mathbb{D}_c)$.

Remark. After calculating the gradient estimation $\widehat{\nabla_{\mathbf{W}_t}}\mathcal{L}(\mathbf{W}_t)$, the server updates the parameters to $\mathbf{W}_{t+1}$ using techniques such as gradient descent with learning rate η. Similar to the discussion in McMahan et al. (2017), the BAFFLE form presented in Algorithm 1 corresponds to FedSGD where Lines 11∼12 execute once for each round t. We can generalize BAFFLE to an analog of FedAvg, in which each client updates its local parameters multiple steps using the gradient estimator $\widehat{\nabla_{\mathbf{W}_t}}\mathcal{L}(\mathbf{W}_t, \mathbb{D}_c)$ derived from $\Delta\mathcal{L}(\mathbf{W}_t, \boldsymbol{\delta}_k; \mathbb{D}_c)$ via Eq. (6), and upload model updates to the server.

3.3 Convergence analyses

Now we analyze the convergence rate of our gradient estimation method. For continuously differentiable loss functions, we have $\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W};\mathbb{D}) = \lim_{\sigma\to 0}\nabla_{\mathbf{W}}\mathcal{L}_\sigma(\mathbf{W};\mathbb{D})$, so we choose a relatively small value for σ. The convergence guarantee can be derived as follows:

Theorem 1. *(Proof in Appendix A.2) For perturbations $\{\boldsymbol{\delta}_k\}_{k=1}^K \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2\mathbf{I})$, the empirical covariance matrix is $\widehat{\boldsymbol{\Sigma}} := \frac{1}{K\sigma^2}\sum_{k=1}^K \boldsymbol{\delta}_k\boldsymbol{\delta}_k^T$ and mean is $\widehat{\boldsymbol{\delta}} := \frac{1}{K}\sum_{k=1}^K \boldsymbol{\delta}_k$. Then for any $\mathbf{W} \in \mathbb{R}^n$, the relation between $\widehat{\nabla_{\mathbf{W}}}\mathcal{L}(\mathbf{W};\mathbb{D})$ and the true gradient $\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W};\mathbb{D})$ can be written as*

$$\widehat{\nabla_{\mathbf{W}}}\mathcal{L}(\mathbf{W};\mathbb{D}) = \widehat{\boldsymbol{\Sigma}}\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W};\mathbb{D}) + o(\widehat{\boldsymbol{\delta}}); \text{ s.t. } \mathbb{E}[\widehat{\boldsymbol{\Sigma}}] = \mathbf{I}, \mathbb{E}[\widehat{\boldsymbol{\delta}}] = \mathbf{0}, \tag{8}$$

where σ is a small value and the central difference scheme in Eq. (3) holds.

When expectation is applied to both sides of Eq. (8), we obtain $\mathbb{E}[\widehat{\nabla_{\mathbf{W}}}\mathcal{L}(\mathbf{W};\mathbb{D})] = \nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W};\mathbb{D})$, which degrades to Stein's identity. To determine the convergence rate w.r.t. the value of K, we have

Theorem 2. *(Adamczak et al. (2011)) With overwhelming probability, the empirical covariance matrix satisfies the inequality $\|\widehat{\boldsymbol{\Sigma}} - \mathbf{I}\|_2 \leq C_0\sqrt{\frac{n}{K}}$, where $\|\cdot\|_2$ denotes the operator 2-norm for matrix and C_0 is an absolute positive constant.*

Note that in the finetuning setting, n represents the number of *trainable* parameters, excluding frozen parameters. As concluded, $\widehat{\nabla_{\mathbf{W}}}\mathcal{L}(\mathbf{W};\mathbb{D})$ provides an unbiased estimation for the true gradients with convergence rate of $\mathcal{O}\left(\sqrt{\frac{n}{K}}\right)$. Empirically, $\widehat{\nabla_{\mathbf{W}}}\mathcal{L}(\mathbf{W};\mathbb{D})$ is used as a noisy gradient to train models, the generalization of which has been analyzed in previous work (Zhu et al., 2019; Li et al., 2020a).

4 Experiments

We evaluate the performance of BAFFLE on four benchmark datasets: MNIST (LeCun et al., 1998), CIFAR-10/100 (Krizhevsky & Hinton, 2009) and OfficeHome (Venkateswara et al., 2017). We consider three models: 1) LeNet (LeCun et al., 1998) with two convolutional layers as the shallow model (2.7×10^4 parameters); 2) WideResNet (Zagoruyko & Komodakis, 2016) with depth $= 10$ and width $= 2$ (WRN-10-2) as the light weight deep model (3.0×10^5 parameters) and 3) MobileNet (Howard et al., 2017) as the deep neural networks (1.3×10^7 parameters) that works on ImageNet. To perform a comprehensive evaluation of BAFFLE, we simulate three popular FL scenarios (Caldas et al., 2018b) with the participation tools from FedLab (Zeng et al., 2021): *iid participations*, *label non-iid participations* and *feature non-iid participations*. For iid participations, we set the client number $C = 10$ and use uniform distribution to build local datasets. Then we evaluate our BAFFLE on MNIST and CIFAR-10/100 under both batch-level (FedSGD) and epoch-level (FedAvg) communication settings. For label non-iid participations, we set client number $C = 100$, use Dirichlet distribution to build clients. For feature non-iid participations, we build clients from the prevailing domain adaptation dataset OfficeHome, which contains 65 categories from 4 different domains, i.e. Art, Clipart, Product and Real-world. We set the total client number to $C = 40$ and generate 10 clients from each domain. As results, we report Top-1 accuracy for MNIST, CIFAR-10 and OfficeHome and Top-5 accuracy for OfficeHome and CIFAR-100.

4.1 Experimental settings

Following the settings in Section 2.1, we use FedAVG to aggregate gradients from multiple clients and use SGD-based optimizer to update global parameters. Specifically, we use Adam (Kingma & Ba, 2015) to train a random initialized model with $\beta = (0.9, 0.99)$, learning rate 0.01 and epochs

Table 1: The classification accuracy (%) of BAFFLE in **iid scenarios** ($C = 10$) and epoch-level communication settings with different K values (K_1/K_2 annotations mean using K_1 for MNIST and K_2 for CIFAR-10/100). In this configuration, each client updates its local model based on BAFFLE estimated gradients and uploads model updates to the server after an entire epoch on the local dataset. The four guidelines work well under epoch-level communication settings.

Settings		**LeNet**			**WRN-10-2**		
		MNIST	CIFAR-10	CIFAR-100	MNIST	CIFAR-10	CIFAR-100
K	100/200	87.27	48.78	41.54	88.35	52.27	46.61
	200/500	89.48	51.82	45.68	89.57	55.59	51.65
	500/1000	**92.18**	**53.62**	**48.72**	**95.17**	**58.63**	**53.15**
Ablation Study (100/200)	w/o EMA	85.06	47.97	36.81	85.89	50.01	45.86
	ReLU	81.55	44.99	39.49	79.08	49.76	44.44
	SELU	86.18	48.65	37.34	76.44	43.37	41.79
	Central	76.02	45.97	36.53	77.45	42.89	39.62
Ground truth		94.31	58.75	54.67	97.11	62.29	60.08

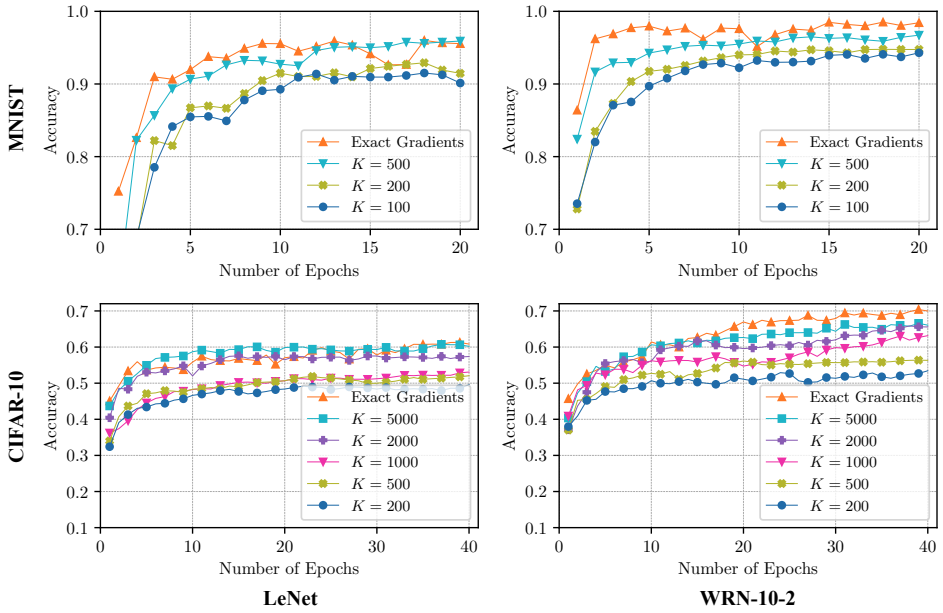


Figure 2: The classification accuracy (%) of BAFFLE in **iid scenarios** ($C = 10$) and batch-level communication settings with various K values. We treat the models trained by exact gradients on conventional FL systems as the ground truth. On different datasets and architectures, our BAFFLE achieves comparable performance to the exact gradient results with a reasonable K.

20/40 for MNIST and CIFAR-10/100. For OfficeHome, we adapt the transfer learning strategy (Huh et al., 2016) by loading the pretrained model on ImageNet and finetuning the final layers with Adam, but setting learning rate 0.005 and epochs 40. In BAFFLE, the perturbation scale σ and number K are the most important hyperparameters. As shown in Theorem 1, with less noise and more samples, the BAFFLE will obtain more accurate gradients, leading to improved performance. However, there exists a trade-off between accuracy and computational efficiency: an extremely small σ will cause the underflow problem (Goodfellow et al., 2016) and a large K will increase computational cost. In practice, we empirically set $\sigma = 10^{-4}$ because it is the smallest value that does not cause numerical problems in all experiments, and works well on edge devices with half-precision floating-point numbers. We also evaluate the impact of K across a broad range from 100 to 5000.

For a general family of continuously differentiable models, we analyze their convergence rate of BAFFLE in Section 3.3. Since deep networks are usually stacked with multiple linear layers and non-linear activation, this layer linearity can be utilized to improve the accuracy-efficiency trade-off. Combining the linearity property and the unique conditions in edge devices (e.g., small data size and

Table 2: The classification accuracy (%) of BAFFLE in **label non-iid scenarios** ($C = 100$) and epoch-level communication settings with different K values. We employ Dirichlet distribution to ensure that each client has a unique label distribution.

Settings		LeNet		WRN-10-2	
		CIFAR-10	CIFAR-100	CIFAR-10	CIFAR-100
K	200	35.21	28.12	39.53	30.44
	500	38.14	30.92	41.69	32.89
	1000	**39.71**	**33.35**	**43.42**	**34.08**
Ground truth		44.41	38.43	51.18	40.85

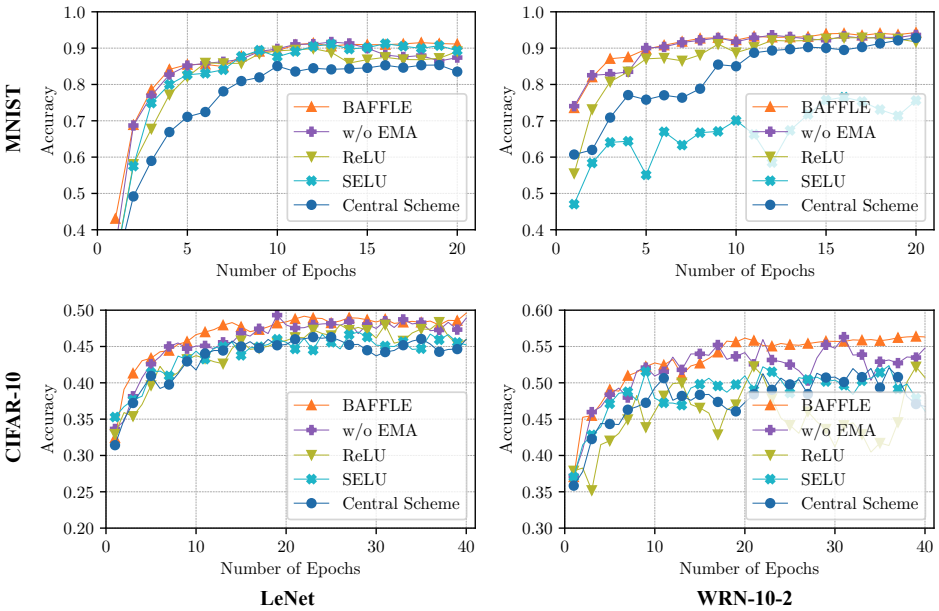


Figure 3: The ablation study of our BAFFLE guidelines, with $K = 100$ on MNIST and $K = 500$ on CIFAR-10. As seen, twice-FD, Hardswish, and EMA all improve performance with the same computational cost. EMA reduces oscillations in training by lessening the effect of white noise.

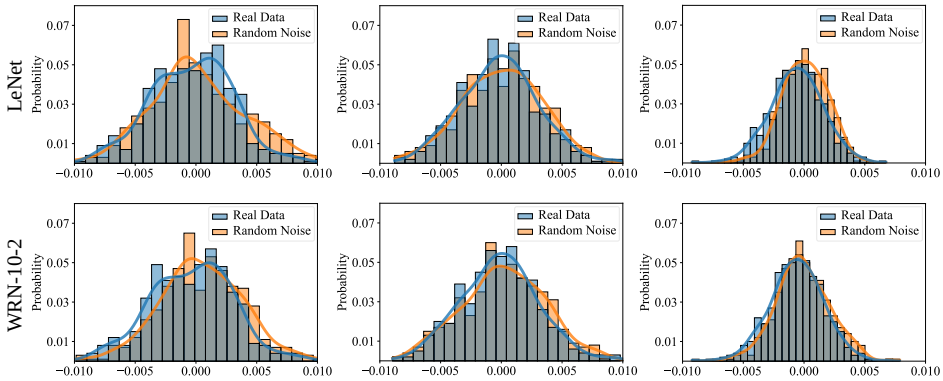
half-precision format), we present four guidelines for model design and training that can increase accuracy without introducing extra computation (for the details of linearity analysis, see Appendix D):

Using twice forward difference (twice-FD) scheme rather than central scheme. Combining difference scheme Eq. (2) and Eq. (3), we find that by executing twice as many forward inferences (i.e.$\mathbf{W} \pm \boldsymbol{\delta}$), the central scheme achieves lower residuals than twice-FD, despite the fact that twice-FD can benefit from additional sample times. With the same forward times (e.g., $2K$), determining which scheme performs better is a practical issue. As shown in Appendix D, we find that twice-FD performs better in all experiments, in part because the linearity reduces the benefit from second-order residuals.

Using Hardswish in BAFFLE. ReLU is effective when the middle features ($h(\cdot)$ denotes the feature mapping) have the same sign before and after perturbations, i.e. $h(\mathbf{W} + \boldsymbol{\delta}) \cdot h(\mathbf{W}) > 0$. Since ReLU is not differentiable at zero, the value jump occurs when the sign of features changes after perturbations, i.e. $h(\mathbf{W} + \boldsymbol{\delta}) \cdot h(\mathbf{W}) < 0$. We use Hardswish (Howard et al., 2019) to overcome this problem as it is continuously differentiable at zero and easy to implement on edge devices.

Using exponential moving average (EMA) to reduce oscillations. As shown in Theorem 1, there exists an zero-mean white-noise $\widehat{\boldsymbol{\delta}}$ between the true gradient and our estimation. To smooth out the oscillations caused by white noise, we apply EMA strategies from BYOL (Grill et al., 2020) to the global parameters, with a smoothing coefficient of 0.995.

Using GroupNorm as opposed to BatchNorm. On edge devices, the dataset size is typically small, which leads to inaccurate batch statistics estimation and degrades performance when using BatchNorm. Thus we employ GroupNorm (Wu & He, 2020) to solve this issue.



(a) MNIST (b) CIFAR-10 (c) CIFAR-100

Figure 4: The robustness of BAFFLE to inference attacks. For real data, we randomly sample some input-label pairs from the validation dataset. For random noise, we generate input-label pairs from standard normal distribution. We sample 500 perturbations $\boldsymbol{\delta}$ from $\mathcal{N}(0, \sigma^2\mathbf{I})$, collect the values of $\Delta\mathcal{L}(\mathbf{W}, \boldsymbol{\delta}; \mathbb{D})$ for real data and random noise separately, and compare their distributions.

4.2 Performance on iid clients

Following the settings in Section 4.1, we evaluate the performance of BAFFLE in the iid scenarios. We reproduce all experiments on the backpropagation-based FL systems with the same settings and use them as the ground truth. We refer to the ground truth results as *exact gradients* and report the training process of BAFFLE in Figure 2. The value of K (e.g., 200 for LeNet and 500 for WRN-10-2) is significantly less than the dimensions of parameter space (e.g., 2.7×10^4 for LeNet and 3×10^5 for WRN-10-2). Since the convergence rate to the exact gradient is $\mathcal{O}\left(\sqrt{\frac{n}{K}}\right)$, the marginal benefit of increasing K decreases. For instance, increasing K from 2000 to 5000 on CIFAR-10 with WRN-10-2 barely improves accuracy by 2%. Given that the convergence rate of normal distribution is $\mathcal{O}\left(\sqrt{\frac{n}{K}}\right)$, the sampling efficiency may be improved by choosing an alternative distribution for perturbations.

Ablation studies. As depicted in Figure 3, we conduct ablation studies for BAFFLE to evaluate the aforementioned guidelines. In general, twice-FD, Hardswish and EMA can all improve the accuracy. For two difference schemes, we compare the twice-FD to central scheme with the same computation cost and show that the former outperforms the later, demonstrating that linearity reduces the gain from second-order residuals. As to activation functions, Hardswish is superior to ReLU and SELU because it is differentiable at zero and vanishes to zero in the negative part. Moreover, EMA enhances the performance of training strategies by reducing the effect of white noise.

Communication efficiency. Each client uploads a K-dimensional vector to the server and downloads the updated global parameters during the communication round of BAFFLE. Since K is significantly less than the parameter amounts (e.g., 500 versus 0.3 million), BAFFLE reduces data transfer by approximately half when compared to the batch-level communication settings (FedSGD) in a backpropagation-based FL system. In order to reduce communication costs, the prevalent FL system requires each client to perform model optimization on the local training dataset and upload the model updates to the server after a specified number of local epochs. BAFFLE can also communicate at the epoch level by employing an $\mathcal{O}(n)$ additional memory to store the perturbation in each forward process and estimate the local gradient using Eq. (6). Each client optimizes the local model with SGD and uploads local updates after a number of epochs. As shown in Table 1, we evaluate the performance of BAFFLE under one-epoch communication settings. As epoch-level communication is more prevalent in the real-world FL, all the following experiments will be conducted in this context.

4.3 Performance on non-iid clients

Following the settings in Section 4.1, we evaluate the performance of BAFFLE in both label non-iid and feature non-iid scenarios. **For label non-iid scenarios**, we use the CIFAR-10/100 datasets and employ Dirichlet distribution to ensure that each client has a unique label distribution. We evaluate the performance of BAFFLE with 100 clients and various K values. As seen in Table 2, the model suffers a significant drop in accuracy (e.g., 14% in CIFAR-10 and 16% in CIFAR-100) due to the label non-iid effect. **For feature non-iid scenarios**, we construct clients using the OfficeHome dataset and use MobileNet as the deep model. As seen in Table 3, we use the transfer learning strategy to train MobileNet, i.e., we load the parameters pretrained on ImageNet, freeze the backbone parameters, and retrain the classification layers. The accuracy decrease is approximately $3\% \sim 5\%$.

Table 3: The Top-1 | Top-5 classification accuracy (%) of BAFFLE on OfficeHome with **feature non-iid participations** ($C = 40$) and epoch-level communication settings. We utilize the pretrained MobileNet, freeze the backbone parameters, and retrain the final classification layers.

Settings		Domains: Art	Clipart	Product	Real World	Avg.
K	20	44.75|69.46	52.48|73.88	66.63|89.77	63.78|87.83	56.91|80.24
	50	47.87|71.32	53.43|76.83	71.28|91.74	67.02|89.95	59.90|82.46
	100	50.32|74.42	57.43|80.73	74.19|93.02	69.53|90.43	62.87|84.65
	200	51.42|76.64	60.98|86.41	76.05|94.42	71.51|93.14	64.98|87.65
	500	**53.33|77.85**	**62.58|86.64**	**78.84|95.23**	**73.17|93.85**	**66.98|88.40**
Ground truth		55.71|80.43	65.13|88.65	82.44|96.05	77.19|95.04	70.12|90.04

4.4 Computation efficiency, memory efficiency, and robustness

BAFFLE uses K times forward passes instead of backward. Since the backward pass is about as expensive as two normal forward passes (Hinton & Srivastava, 2010) and five single-precision accelerated forward passes Nakandala et al. (2020), BAFFLE results in approximately $\frac{K}{5}$ times the computation expense of BP-based FL. Although BAFFLE results in $\frac{K}{5} - 1$ times extra computation cost, we show the cost can be reduced with proper training strategies, e.g., the transfer learning in Table 3 can reduce K to 20 on the MobileNet and the 224×224 sized OfficeHome dataset.

Moreover, BAFFLE can reduce huge memory cost on edge devices. Backpropagation consumes large memory due to the requirement on restoring intermediate states, e.g., $\geq$ 5GB for VGG-16 (Gao et al., 2020). Since BAFFLE only requires forward, we can slice the computation graph and execute the per-layer forward calculation, which reduces the memory cost to 64MB (Kim et al., 2020; Tramer & Boneh, 2019). In addition, backpropagation requires the auto-differential framework, leading to extra memory cost (e.g., 200MB for Caffe (Jia et al., 2014) and 1GB for Pytorch (Paszke et al., 2019a)), while BAFFLE does not require auto-differential tools. To summarize the above, BAFFLE can reduce the memory cost to approximately 1% compared with backpropagation.

Recent works exploit TEE to protect models from white-box attacks (Kim et al., 2020), which can defend against white-box attacks by preventing model exposure. However, due to the security guarantee, the usable memory of TEE is usually small (Truong et al., 2021) (e.g., 90MB on Intel SGX for Skylake CPU (McKeen et al., 2016)), which is typically far less than what a backpropagation-based FL system requires. In contrast, BAFFLE can execute in TEE due to its little memory cost (more details are in Appendix C). Membership inference attack and model inversion attack need to repeatedly perform model inference and obtain confidence values or classification scores (Shokri et al., 2017; Zhang et al., 2020). Given that BAFFLE provides stochastic loss differences $\Delta\mathcal{L}(\mathbf{W}, \boldsymbol{\delta}; \mathbb{D})$ associated with the random perturbation $\boldsymbol{\delta}$, the off-the-shelf inference attacks may not perform on BAFFLE directly (while adaptively designed attacking strategies are possible to evade BAFFLE). We further select random samples from the validation dataset and generate random input pairs as $(\tilde{\mathbf{X}}, \tilde{y})$. As shown in Figure 4, it is difficult to distinguish between real data and random noise, indicating that it is difficult for attackers to obtain useful information from BAFFLE's outputs.

5 Conclusion and discussion

Backpropagation is the gold standard for training deep networks, and it is also utilized by traditional FL systems. However, backpropagation is unsuited for edge devices due to their limited resources and possible lack of reliability. Using zero-order optimization techniques, we explore the possibility of backpropagation-free FL in this paper. We need to specify that there are scenarios in which clients are fully trusted and have sufficient computing and storage resources. In these situations, traditional FL with backpropagation is preferred over BAFFLE.

While our preliminary studies on BAFFLE have generated encouraging results, there are still a number of tough topics to investigate: **(i)** Compared to the models trained using exact gradients, the accuracy of models trained using BAFFLE is inferior. One reason is that we select small values of K (e.g., 500) relative to the number of model parameters (e.g., 3.0×10^5); another reason is that gradient descent is designed for exact gradients, whereas our noisy gradient estimation may require advanced learning algorithms. **(ii)** The empirical variance of zero-order gradient estimators affects training convergence in BAFFLE. It is crucial to research variance reduction approaches, such as control variates and non-Gaussian sampling distributions. **(iii)** Stein's identity is proposed for loss functions with Gaussian noises imposed on model parameters. Intuitively, this smoothness is related to differential privacy in FL, but determining their relationship requires theoretical derivations.

REFERENCES

Jayadev Acharya, Clément L Canonne, and Himanshu Tyagi. Inference under information constraints i: Lower bounds from chi-square contraction. *IEEE Transactions on Information Theory*, 2020.

Radosław Adamczak, Alexander E Litvak, Alain Pajor, and Nicole Tomczak-Jaegermann. Sharp bounds on the rate of convergence of the empirical covariance matrix. *Comptes Rendus Mathematique*, 349(3-4):195–200, 2011.

Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

Ravikumar Balakrishnan, Tian Li, Tianyi Zhou, Nageen Himayat, Virginia Smith, and Jeff Bilmes. Diverse client selection for federated learning via submodular maximization. In *International Conference on Learning Representations (ICLR)*, 2022.

Leighton Pate Barnes, Yanjun Han, and Ayfer Özgür. Lower bounds for learning distributions under communication constraints via fisher information. *Journal of Machine Learning Research (JMLR)*, 2020.

Debraj Basu, Deepesh Data, Can Karakus, and Suhas Diggavi. Qsparse-local-sgd: Distributed sgd with quantization, sparsification and local computations. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning (ICML)*, 2019.

Kallista A. Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *ACM Conference on Computer and Communications Security (CCS)*, 2017a.

Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. *arXiv preprint arXiv:1611.04482*, 2016.

Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *ACM SIGSAC Conference on Computer and Communications Security*, 2017b.

Eitan Borgnia, Jonas Geiping, Valeriia Cherepanova, Liam Fowl, Arjun Gupta, Amin Ghiasi, Furong Huang, Micah Goldblum, and Tom Goldstein. Dp-instahide: Provably defusing poisoning and backdoor attacks with differentially private data augmentations. *arXiv preprint arXiv:2103.02079*, 2021.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Mark Braverman, Ankit Garg, Tengyu Ma, Huy L Nguyen, and David P Woodruff. Communication lower bounds for statistical estimation problems via a distributed data processing inequality. In *ACM Symposium on Theory of Computing*, 2016.

Sebastian Caldas, Jakub Konečny, H Brendan McMahan, and Ameet Talwalkar. Expanding the reach of federated learning by reducing client resource requirements. *arXiv preprint arXiv:1812.07210*, 2018a.

Sebastian Caldas, Peter Wu, Tian Li, Jakub Konečný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. LEAF: A benchmark for federated settings. *CoRR*, 2018b.

Mingzhe Chen, Nir Shlezinger, H Vincent Poor, Yonina C Eldar, and Shuguang Cui. Communication-efficient federated learning. *Proceedings of the National Academy of Sciences*, 2021.

Yu Chen, Fang Luo, Tong Li, Tao Xiang, Zheli Liu, and Jin Li. A training-integrity privacy-preserving federated learning scheme with trusted execution environment. *Information Sciences*, 2020.

Yinpeng Dong, Xiao Yang, Zhijie Deng, Tianyu Pang, Zihao Xiao, Hang Su, and Jun Zhu. Black-box detection of backdoor attacks with limited information and data. In *IEEE International Conference on Computer Vision (ICCV)*, 2021.

Hubert Eichner, Tomer Koren, Brendan McMahan, Nathan Srebro, and Kunal Talwar. Semi-cyclic stochastic gradient descent. In *International Conference on Machine Learning (ICML)*, 2019.

Abraham D Flaxman, Adam Tauman Kalai, and H Brendan McMahan. Online convex optimization in the bandit setting: gradient descent without a gradient. *arXiv preprint cs/0408007*, 2004.

Yanjie Gao, Yu Liu, Hongyu Zhang, Zhengxian Li, Yonghao Zhu, Haoxiang Lin, and Mao Yang. Estimating gpu memory consumption of deep learning models. In *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020.

Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients-how easy is it to break privacy in federated learning? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Badih Ghazi, Ravi Kumar, Pasin Manurangsi, and Rasmus Pagh. Private counting from anonymous messages: Near-optimal accuracy with vanishing communication overhead. In *International Conference on Machine Learning (ICML)*, 2020.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Ávila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent - A new approach to self-supervised learning. In *NeurIPS 2020*, 2020.

Jenny Hamer, Mehryar Mohri, and Ananda Theertha Suresh. Fedboost: A communication-efficient algorithm for federated learning. In *International Conference on Machine Learning (ICML)*, 2020.

Yanjun Han, Ayfer Özgür, and Tsachy Weissman. Geometric lower bounds for distributed parameter estimation under communication constraints. In *Conference On Learning Theory (COLT)*, 2018.

Meng Hao, Hongwei Li, Xizhao Luo, Guowen Xu, Haomiao Yang, and Sen Liu. Efficient and privacy-enhanced federated learning for industrial artificial intelligence. *IEEE Transactions on Industrial Informatics*, 2019.

Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

Chaoyang He, Murali Annavaram, and Salman Avestimehr. Group knowledge transfer: Federated learning of large cnns at the edge. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Di He, Wenlei Shi, Shanda Li, Xiaotian Gao, Jia Zhang, Jiang Bian, Liwei Wang, and Tie-Yan Liu. Learning physics-informed neural networks without stacked back-propagation. *arXiv preprint arXiv:2202.09340*, 2022.

Geoffrey Hinton and Nitish Srivastava. Csc321: Introduction to neural networks and machine learning. *Lecture*, 10, 2010.

Samuel Horváth, Chen-Yu Ho, Ludovit Horvath, Atal Narayan Sahu, Marco Canini, and Peter Richtárik. Natural compression for distributed deep learning. *arXiv preprint arXiv:1905.10988*, 2019.

Andrew Howard, Ruoming Pang, Hartwig Adam, Quoc V. Le, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, and Yukun Zhu. Searching for mobilenetv3. In *IEEE International Conference on Computer Vision (ICCV)*, 2019.

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. *CoRR*, 2017.

Yangsibo Huang, Samyak Gupta, Zhao Song, Kai Li, and Sanjeev Arora. Evaluating gradient inversion attacks and defenses in federated learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Mi-Young Huh, Pulkit Agrawal, and Alexei A. Efros. What makes imagenet good for transfer learning? *CoRR*, 2016.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In Kien A. Hua, Yong Rui, Ralf Steinmetz, Alan Hanjalic, Apostol Natsev, and Wenwu Zhu (eds.), *Proceedings of the ACM International Conference on Multimedia*, 2014.

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. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

Jiawen Kang, Zehui Xiong, Dusit Niyato, Yuze Zou, Yang Zhang, and Mohsen Guizani. Reliable federated learning for mobile networks. *IEEE Wireless Communications*, 2020.

Kyungtae Kim, Chung Hwan Kim, Junghwan" John" Rhee, Xiao Yu, Haifeng Chen, Dave Tian, and Byoungyoung Lee. Vessels: Efficient and scalable deep learning prediction on trusted processors. In *ACM Symposium on Cloud Computing*, 2020.

Yejin Kim, Jimeng Sun, Hwanjo Yu, and Xiaoqian Jiang. Federated tensor factorization for computational phenotyping. In *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

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

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of IEEE*, 1998.

Jian Li, Xuanyuan Luo, and Mingda Qiao. On generalization error bounds of noisy gradient methods for non-convex learning. In *International Conference on Learning Representations (ICLR)*, 2020a.

Li Li, Yuxi Fan, Mike Tse, and Kuo-Yi Lin. A review of applications in federated learning. *Computers & Industrial Engineering*, 2020b.

Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization. In *International Conference on Machine Learning (ICML)*, 2021.

Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. In *International Conference on Learning Representations (ICLR)*, 2020c.

Sijia Liu, Pin-Yu Chen, Bhavya Kailkhura, Gaoyuan Zhang, Alfred O Hero III, and Pramod K Varshney. A primer on zeroth-order optimization in signal processing and machine learning: Principals, recent advances, and applications. *IEEE Signal Processing Magazine*, 2020a.

Yuhan Liu, Ananda Theertha Suresh, Felix Xinnan X Yu, Sanjiv Kumar, and Michael Riley. Learning discrete distributions: user vs item-level privacy. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020b.

Xinjian Luo, Yuncheng Wu, Xiaokui Xiao, and Beng Chin Ooi. Feature inference attack on model predictions in vertical federated learning. In *IEEE International Conference on Data Engineering (ICDE)*, 2021.

Lingjuan Lyu, Han Yu, Xingjun Ma, Lichao Sun, Jun Zhao, Qiang Yang, and Philip S Yu. Privacy and robustness in federated learning: Attacks and defenses. *arXiv preprint arXiv:2012.06337*, 2020.

Jing Ma, Qiuchen Zhang, Jian Lou, Joyce C Ho, Li Xiong, and Xiaoqian Jiang. Privacy-preserving tensor factorization for collaborative health data analysis. In *ACM International Conference on Information and Knowledge Management (CIKM)*, 2019.

Othmane Marfoq, Giovanni Neglia, Aurélien Bellet, Laetitia Kameni, and Richard Vidal. Federated multi-task learning under a mixture of distributions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Frank McKeen, Ilya Alexandrovich, Ittai Anati, Dror Caspi, Simon Johnson, Rebekah Leslie-Hurd, and Carlos Rozas. Intel® software guard extensions (intel® sgx) support for dynamic memory management inside an enclave. In *Proceedings of the Hardware and Architectural Support for Security and Privacy*, 2016.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *International Conference on Learning Representations (ICLR)*, 2018.

Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. Ppfl: privacy-preserving federated learning with trusted execution environments. In *Annual International Conference on Mobile Systems, Applications, and Services*, 2021.

Arup Mondal, Yash More, Ruthu Hulikal Rooparaghunath, and Debayan Gupta. Flatee: Federated learning across trusted execution environments. *arXiv preprint arXiv:2111.06867*, 2021.

Emmanuel Moulay, Vincent Léchappé, and Franck Plestan. Properties of the sign gradient descent algorithms. *Inf. Sci.*, 492:29–39, 2019.

Supun Nakandala, Kabir Nagrecha, Arun Kumar, and Yannis Papakonstantinou. Incremental and approximate computations for accelerating deep CNN inference. *ACM Trans. Database Syst.*, 45, 2020.

Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *IEEE symposium on security and privacy (S&P)*. IEEE, 2019.

Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 2017.

Tianyu Pang, Kun Xu, Chongxuan Li, Yang Song, Stefano Ermon, and Jun Zhu. Efficient learning of generative models via finite-difference score matching. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Tianyu Pang, Xiao Yang, Yinpeng Dong, Hang Su, and Jun Zhu. Accumulative poisoning attacks on real-time data. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, 2019a.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019b.

Daniel Rothchild, Ashwinee Panda, Enayat Ullah, Nikita Ivkin, Ion Stoica, Vladimir Braverman, Joseph Gonzalez, and Raman Arora. Fetchsgd: Communication-efficient federated learning with sketching. In *International Conference on Machine Learning (ICML)*, 2020.

Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted execution environment: what it is, and what it is not. In *IEEE Trustcom/BigDataSE/ISPA*, 2015.

Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Robust and communication-efficient federated learning from non-iid data. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, 2019.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Sanjay Seetharaman, Shubham Malaviya, Rosni KV, Manish Shukla, and Sachin Lodha. Influence based defense against data poisoning attacks in online learning. *arXiv preprint arXiv:2104.13230*, 2021.

Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi Esmaeilzadeh. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In *IEEE Annual International Symposium on Computer Architecture (ISCA)*, 2018.

Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *IEEE symposium on security and privacy (S&P)*. IEEE, 2017.

Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated multi-task learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Charles M Stein. Estimation of the mean of a multivariate normal distribution. *The annals of Statistics*, 1981.

Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H Brendan McMahan. Can you really backdoor federated learning? *arXiv preprint arXiv:1911.07963*, 2019.

Ananda Theertha Suresh, X Yu Felix, Sanjiv Kumar, and H Brendan McMahan. Distributed mean estimation with limited communication. In *International Conference on Machine Learning (ICML)*, 2017.

Florian Tramer and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *International Conference on Learning Representations (ICLR)*, 2019.

Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. A hybrid approach to privacy-preserving federated learning. In *ACM workshop on Artificial Intelligence and Security*, 2019.

Jean-Baptiste Truong, William Gallagher, Tian Guo, and Robert J Walls. Memory-efficient deep learning inference in trusted execution environments. In *IEEE International Conference on Cloud Engineering (IC2E)*, 2021.

Yaman Umuroglu, Lahiru Rasnayake, and Magnus Själander. Bismo: A scalable bit-serial matrix multiplication overlay for reconfigurable computing. In *International Conference on Field Programmable Logic and Applications (FPL)*, 2018.

Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. Deep hashing network for unsupervised domain adaptation. In *CVPR*, 2017.

Hongyi Wang, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris Papailiopoulos. Attack of the tails: Yes, you really can backdoor federated learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020a.

Hui-Po Wang, Sebastian Stich, Yang He, and Mario Fritz. Progfed: effective, communication, and computation efficient federated learning by progressive training. In *International Conference on Machine Learning (ICML)*, 2022.

Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020b.

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019a.

Luping Wang, Wei Wang, and Bo Li. Cmfl: Mitigating communication overhead for federated learning. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2019b.

Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 2020.

Yuxin Wu and Kaiming He. Group normalization. *Int. J. Comput. Vis.*, 128(3):742–755, 2020.

Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *International Conference on Learning Representations (ICLR)*, 2020.

Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018.

Xue Yang, Yan Feng, Weijun Fang, Jun Shao, Xiaohu Tang, Shu-Tao Xia, and Rongxing Lu. An accuracy-lossless perturbation method for defending privacy attacks in federated learning. In *ACM Web Conference (WWW)*, 2022.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference 2016*, 2016.

Dun Zeng, Siqi Liang, Xiangjing Hu, Hui Wang, and Zenglin Xu. Fedlab: A flexible federated learning framework. *arXiv preprint arXiv:2107.11621*, 2021.

Yuchen Zhang, John Duchi, Michael I Jordan, and Martin J Wainwright. Information-theoretic lower bounds for distributed statistical estimation with communication constraints. *Advances in Neural Information Processing Systems (NeurIPS)*, 2013.

Yuheng Zhang, Ruoxi Jia, Hengzhi Pei, Wenxiao Wang, Bo Li, and Dawn Song. The secret revealer: Generative model-inversion attacks against deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Yuhui Zhang, Zhiwei Wang, Jiangfeng Cao, Rui Hou, and Dan Meng. Shufflefl: gradient-preserving federated learning using trusted execution environment. In *ACM International Conference on Computing Frontiers*, 2021.

Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

Zhanxing Zhu, Jingfeng Wu, Bing Yu, Lei Wu, and Jinwen Ma. The anisotropic noise in stochastic gradient descent: Its behavior of escaping from sharp minima and regularization effects. In *International Conference on Machine Learning (ICML)*, 2019.

A PROOFS

A.1 PROOF OF STEIN'S IDENTITY

We recap the proof of Stein's identity following He et al. (2022), where

$$\begin{aligned}
\nabla_{\mathbf{W}}\mathcal{L}_\sigma(\mathbf{W};\mathbb{D}) &= \nabla_{\mathbf{W}}\mathbb{E}_{\boldsymbol{\delta}\sim\mathcal{N}(0,\sigma^2 I)}\mathcal{L}(\mathbf{W}+\boldsymbol{\delta};\mathbb{D}) \\
&= (2\pi)^{-\frac{n}{2}}\cdot\nabla_{\mathbf{W}}\int\mathcal{L}(\mathbf{W}+\boldsymbol{\delta};\mathbb{D})\cdot\exp\left(-\frac{\|\boldsymbol{\delta}\|_2^2}{2\sigma^2}\right)d\boldsymbol{\delta} \\
&= (2\pi)^{-\frac{n}{2}}\cdot\int\mathcal{L}(\widetilde{\mathbf{W}};\mathbb{D})\cdot\nabla_{\mathbf{W}}\exp\left(-\frac{\|\widetilde{\mathbf{W}}-\mathbf{W}\|_2^2}{2\sigma^2}\right)d\widetilde{\mathbf{W}} \\
&= (2\pi)^{-\frac{n}{2}}\cdot\int\mathcal{L}(\widetilde{\mathbf{W}};\mathbb{D})\cdot\frac{\widetilde{\mathbf{W}}-\mathbf{W}}{\sigma^2}\cdot\exp\left(-\frac{\|\widetilde{\mathbf{W}}-\mathbf{W}\|_2^2}{2\sigma^2}\right)d\widetilde{\mathbf{W}} \\
&= (2\pi)^{-\frac{n}{2}}\cdot\int\mathcal{L}(\mathbf{W}+\boldsymbol{\delta};\mathbb{D})\cdot\frac{\boldsymbol{\delta}}{\sigma^2}\cdot\exp\left(-\frac{\|\boldsymbol{\delta}\|_2^2}{2\sigma^2}\right)d\boldsymbol{\delta} \\
&= \mathbb{E}_{\boldsymbol{\delta}\sim\mathcal{N}(0,\sigma^2\mathbf{I})}\left[\frac{\boldsymbol{\delta}}{\sigma^2}\mathcal{L}(\mathbf{W}+\boldsymbol{\delta};\mathbb{D})\right].
\end{aligned} \tag{9}$$

By symmetry, we change $\boldsymbol{\delta}$ to $-\boldsymbol{\delta}$ and obtain

$$\nabla_{\mathbf{W}}\mathcal{L}_\sigma(\mathbf{W};\mathbb{D}) = -\mathbb{E}_{\boldsymbol{\delta}\sim\mathcal{N}(0,\sigma^2\mathbf{I})}\left[\frac{\boldsymbol{\delta}}{\sigma^2}\mathcal{L}(\mathbf{W}-\boldsymbol{\delta};\mathbb{D})\right], \tag{10}$$

and further we prove that

$$\begin{aligned}
\nabla_{\mathbf{W}}\mathcal{L}_\sigma(\mathbf{W};\mathbb{D}) &= \frac{1}{2}\left(\mathbb{E}_{\boldsymbol{\delta}\sim\mathcal{N}(0,\sigma^2\mathbf{I})}\left[\frac{\boldsymbol{\delta}}{\sigma^2}\mathcal{L}(\mathbf{W}+\boldsymbol{\delta};\mathbb{D})\right]-\mathbb{E}_{\boldsymbol{\delta}\sim\mathcal{N}(0,\sigma^2\mathbf{I})}\left[\frac{\boldsymbol{\delta}}{\sigma^2}\mathcal{L}(\mathbf{W}-\boldsymbol{\delta};\mathbb{D})\right]\right) \\
&= \mathbb{E}_{\boldsymbol{\delta}\sim\mathcal{N}(0,\sigma^2\mathbf{I})}\left[\frac{\boldsymbol{\delta}}{2\sigma^2}\Delta\mathcal{L}(\mathbf{W},\boldsymbol{\delta};\mathbb{D})\right].
\end{aligned}$$

□

A.2 PROOF OF THEOREM 1

We rewrite the format of $\widehat{\nabla_{\mathbf{W}}}\mathcal{L}(\mathbf{W};\mathbb{D})$ as follows:

$$\begin{aligned}
\widehat{\nabla_{\mathbf{W}}}\mathcal{L}(\mathbf{W};\mathbb{D}) &= \frac{1}{K}\sum_{k=1}^{K}[\frac{\boldsymbol{\delta}_k}{2\sigma^2}\Delta\mathcal{L}(\mathbf{W},\boldsymbol{\delta}_k;\mathbb{D})] \\
&= \frac{1}{K}\sum_{k=1}^{K}[\frac{\boldsymbol{\delta}_k}{2\sigma^2}\left(2\boldsymbol{\delta}_k^\top\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W};\mathbb{D})+o(\|\boldsymbol{\delta}_k\|_2^2)\right)] \qquad \text{(using central scheme in Eq. (3))} \\
&= \frac{1}{K\sigma^2}\sum_{k=1}^{K}[\boldsymbol{\delta}_k\boldsymbol{\delta}_k^\top]\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W};\mathbb{D})+\frac{1}{K}\sum_{k=1}^{K}\frac{\boldsymbol{\delta}_k}{2\sigma^2}o(\|\boldsymbol{\delta}_k\|_2^2) \\
&= \widehat{\boldsymbol{\Sigma}}\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W};\mathbb{D})+\frac{1}{K}\sum_{k=1}^{K}\frac{\boldsymbol{\delta}_k}{2\sigma^2}o(\|\boldsymbol{\delta}_k\|_2^2).
\end{aligned} \tag{11}$$

Then we prove $\frac{1}{K}\sum_{k=1}^{K}\frac{\boldsymbol{\delta}_k}{2\sigma^2}o(\|\boldsymbol{\delta}_k\|_2^2) = o(\widehat{\boldsymbol{\delta}})$. Suppose $\boldsymbol{\delta}_k = (\delta_{k,1},\cdots,\delta_{k,n})$, then we have $\frac{\|\boldsymbol{\delta}_k\|_2^2}{\sigma^2} = \sum_{i=1}^{n}(\frac{\delta_{k,i}}{\sigma})^2$. Since $\forall i$, $\frac{\delta_{k,i}}{\sigma}\sim\mathcal{N}(0,1)$, we have $\frac{\|\boldsymbol{\delta}_k\|_2^2}{\sigma^2}\sim\chi^2(n)$ and $\mathbb{E}(\frac{\|\boldsymbol{\delta}_k\|_2^2}{\sigma^2}) = n$. So with high probability, $\frac{o(\|\boldsymbol{\delta}_k\|_2^2)}{\sigma^2} = o(n)$. Substituting it into Eq. (11), we have with high probability,

$$\frac{1}{K}\sum_{k=1}^{K}\frac{\boldsymbol{\delta}_k}{2\sigma^2}o(\|\boldsymbol{\delta}_k\|_2^2) = \widehat{\boldsymbol{\delta}}\cdot o(n) = o(\widehat{\boldsymbol{\delta}}),$$

where we regard n as a constant for a given model architecture. Finally, we prove $\mathbb{E}[\widehat{\boldsymbol{\delta}}] = \mathbf{0}$ and $\mathbb{E}[\widehat{\boldsymbol{\Sigma}}] = \mathbf{I}$. It is trivial that $\mathbb{E}[\widehat{\boldsymbol{\delta}}] = \mathbf{0}$ since $\widehat{\boldsymbol{\delta}} \sim \mathcal{N}(0, \frac{1}{K\sigma^2}\mathbf{I})$. For $\mathbb{E}[\widehat{\boldsymbol{\Sigma}}] = \mathbf{I}$, we can observe by examining each of its entries

$$\widehat{\boldsymbol{\Sigma}}_{[ij]} = \frac{1}{K\sigma^2}\sum_{k=1}^{K}\delta_{k[i]}\delta_{k[j]} = \frac{1}{K}\sum_{k=1}^{K}\frac{\delta_{k[i]}}{\sigma}\frac{\delta_{k[j]}}{\sigma}, \tag{12}$$

where we have used subscripts $[ij]$ and $[i]$ to denote the usual indexing of matrices and vectors. Specifically, for diagonal entries (i.e., $i = j$), we observe $K \cdot \widehat{\boldsymbol{\Sigma}}_{[ii]} = \sum_{k=1}^{K}\left(\frac{\delta_{k[i]}}{\sigma}\right)^2$ distributes as $\chi^2(K)$, which means $\mathbb{E}[\widehat{\boldsymbol{\Sigma}}_{[ii]}] = 1 = \mathbf{I}_{[ii]}$ and $\mathrm{Var}[\widehat{\boldsymbol{\Sigma}}_{[ii]}] = \frac{2}{K}$; for non-diagonal entries (i.e., $i \neq j$), we have $\mathbb{E}[\widehat{\boldsymbol{\Sigma}}_{[ij]}] = \frac{1}{K}\sum_{k=1}^{K}\mathbb{E}\left[\frac{\delta_{k[i]}}{\sigma}\frac{\delta_{k[j]}}{\sigma}\right] = \frac{1}{K}\sum_{k=1}^{K}\frac{\mathbb{E}[\delta_{k[i]}]}{\sigma}\frac{\mathbb{E}[\delta_{k[j]}]}{\sigma} = 0 = \mathbf{I}_{[ij]}$, due to the independence between different dimensions in $\boldsymbol{\delta}_k$. □

B Related work

Along the research routine of FL, many efforts have been devoted to, e.g., dealing with non-IID distributions (Zhao et al., 2018; Sattler et al., 2019; Eichner et al., 2019; Wang et al., 2020b; Li et al., 2020c), multi-task learning (Smith et al., 2017; Marfoq et al., 2021), and preserving privacy of clients (Bonawitz et al., 2016; 2017b; McMahan et al., 2018; Truex et al., 2019; Hao et al., 2019; Lyu et al., 2020; Ghazi et al., 2020; Liu et al., 2020b). Below we introduce the work on efficiency and vulnerability in FL following the survey of Kairouz et al. (2021), which is more related to this paper.

Efficiency in FL. It is widely understood that the communication and computational efficiency is a primary bottleneck for deploying FL in practice (Wang et al., 2019b; Rothchild et al., 2020; Chen et al., 2021; Balakrishnan et al., 2022; Wang et al., 2022). Specifically, communicating between the server and clients could be potentially expensive and unreliable. The seminal work of Konečnỳ et al. (2016) introduces sparsification and quantization to reduce the communication cost, where several theoretical works investigate the optimal trade-off between the communication cost and model accuracy (Zhang et al., 2013; Braverman et al., 2016; Han et al., 2018; Acharya et al., 2020; Barnes et al., 2020). Since practical clients usually have slower upload than download bandwidth, much research interest focuses on gradient compression (Suresh et al., 2017; Alistarh et al., 2017; Horváth et al., 2019; Basu et al., 2019). On the other hand, different methods have been proposed to reduce the computational burden of local clients (Caldas et al., 2018a; Hamer et al., 2020; He et al., 2020), since these clients are usually edge devices with limited resources. Training paradigms exploiting tensor factorization in FL can also achieve promising performance (Kim et al., 2017; Ma et al., 2019).

Vulnerability in FL. The characteristic of decentralization in FL is beneficial to protecting data privacy of clients, but in the meanwhile, providing white-box accessibility of model status leaves flexibility for malicious clients to perform poisoning/backdoor attacks (Bhagoji et al., 2019; Bagdasaryan et al., 2020; Wang et al., 2020a; Xie et al., 2020; Pang et al., 2021), model/gradient inversion attacks (Zhang et al., 2020; Geiping et al., 2020; Huang et al., 2021), and membership inference attacks (Shokri et al., 2017; Nasr et al., 2019; Luo et al., 2021). To alleviate the vulnerability in FL, several defense strategies have been proposed via selecting reliable clients (Kang et al., 2020), data augmentation (Borgnia et al., 2021), update clipping (Sun et al., 2019), robust training (Li et al., 2021), model perturbation (Yang et al., 2022), detection methods (Seetharaman et al., 2021; Dong et al., 2021), and methods based on differential privacy (Wei et al., 2020), just to name a few.

C Trusted execution environments

A trusted execution environment (TEE) (Sabt et al., 2015) is regarded as the ultimate solution for defending against all white-box attacks by preventing any model exposure. TEE protects both data and model security with three components: physical secure storage to ensure the confidentiality, integrity, and tamper-resistance of stored data; a root of trust to load trusted code; and a separate kernel to execute code in an isolated environment, as illustrated in Figure 5. Using TEE, the FL system is able to train deep models without revealing any model specifics. However, due to the security guarantee, the usable memory of TEE is typically small (Truong et al., 2021) (e.g., 90MB

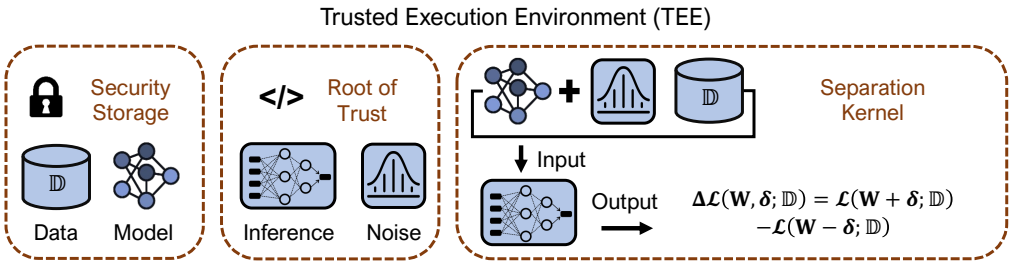


Figure 5: A sketch map of trusted execution environments.

on Intel SGX for Skylake CPU (McKeen et al., 2016)), which is considerably less than what deep models require for backpropagation (e.g., $\geq$ 5GB for VGG-16 (Gao et al., 2020)).

D Convergence analyses of deep linear networks in BAFFLE

We analyze the convergence of BAFFLE in Section 3.3 using a general technique applicable to any continuously differentiable models corresponding to the loss function $\mathcal{L}(\mathbf{W};\mathbb{D})$. Since deep networks are the most prevalent models in FL, which has strong linearity, it is simpler to investigate the convergence of deep linear networks (Saxe et al., 2013).

Consider a two-layer deep linear network in a classification task with L categories. We denote the model parameters as $\{\mathbf{W}_1, \mathbf{W}_2\}$, where in the first layer $\mathbf{W}_1 \in \mathbb{R}^{n\times m}$, in the second layer $\mathbf{W}_2 \in \mathbb{R}^{L\times n}$ consists of L vectors related to the L categories as $\{\mathbf{w}_2^l\}_{l=1}^L$ and $\mathbf{w}_2^c \in \mathbb{R}^{1\times n}$. For the input data $\mathbf{X} \in \mathbb{R}^{m\times 1}$ with label y, we train the deep linear network by maximizing the classification score on the y-th class. Since there is no non-linear activation in deep linear networks, the forward inference can be represented as $h = \mathbf{w}_2^y \mathbf{W}_1 \mathbf{X}$, and the loss is $-h$. It is easy to show that $\frac{\partial h}{\partial \mathbf{w}_2^y} = (\mathbf{W}_1\mathbf{X})^\top$ and $\frac{\partial h}{\partial \mathbf{W}_1} = (\mathbf{X}\mathbf{w}_2^y)^\top$. We sample $\boldsymbol{\delta}_1, \boldsymbol{\delta}_2$ from noise generator $\mathcal{N}(0, \sigma^2\mathbf{I})$, where $\boldsymbol{\delta}_1 \in \mathbb{R}^{n\times m}$ and $\boldsymbol{\delta}_2 \in \mathbb{R}^{1\times n}$. Let $h(\boldsymbol{\delta_1}, \boldsymbol{\delta_2}) := (\mathbf{w}_2^y + \boldsymbol{\delta}_2)(\mathbf{W}_1 + \boldsymbol{\delta}_1)\mathbf{X}$, we discover that the BAFFLE estimation in Eq. (6) follows the same pattern for both forward (2) and central schemes (3):

$$\begin{aligned}
&\Delta_{\text{for}} h(\boldsymbol{\delta}_1, \boldsymbol{\delta}_2) := h(\boldsymbol{\delta_1}, \boldsymbol{\delta}_2) - h(\mathbf{0}, \mathbf{0});\\
&\Delta_{\text{ctr}} h(\boldsymbol{\delta}_1, \boldsymbol{\delta}_2) := h(\boldsymbol{\delta}_1, \boldsymbol{\delta}_2) - h(-\boldsymbol{\delta}_1, -\boldsymbol{\delta}_2);\\
&\frac{\Delta_{\text{for}} h(\boldsymbol{\delta}_1, \boldsymbol{\delta}_2)}{\sigma^2} = \frac{\Delta_{\text{ctr}} h(\boldsymbol{\delta}_1, \boldsymbol{\delta}_2)}{2\sigma^2} = \frac{1}{\sigma^2}(\mathbf{w}_2^c \boldsymbol{\delta}_1 \mathbf{X} + \boldsymbol{\delta}_2 \mathbf{W}_1 \mathbf{X}).
\end{aligned} \tag{13}$$

This equivalent form in deep linear networks illustrates that the residual benefit from the central scheme is reduced by the linearity, hence the performance of the two finite difference schemes described above is same in deep linear networks. We refer to this characteristic as FD scheme independence. We also find the property of σ independence, that is, the choice of σ does not effect the results of finite difference, due to the fact that $\frac{\boldsymbol{\delta}_1}{\sigma}$ and $\frac{\boldsymbol{\delta}_2}{\sigma}$ follow the standard normal distribution.

Based on the findings from Eq. (13), we propose the following useful guideline that improves accuracy under the same computation cost: *Using twice forward difference (twice-FD) scheme rather than central scheme.* Combining the forward scheme (2) and central scheme (3), we find that the central scheme produces smaller residuals than the forward scheme by executing twice as many forward inferences, i.e. $\mathbf{W} \pm \boldsymbol{\delta}$. With the same forward inference times (e.g., $2K$), one practical difficulty is to identify which scheme performs better. We find that the forward scheme performs better in all experiments, in part because the linearity reduces the benefit from second-order residuals, as demonstrated by Eq. (13).