Federated Learning with Differential Privacy and an Untrusted Aggregator

Kunlong Liu and Trinabh Gupta
University of California, Santa Barbara, U.S.A.

Keywords: Federated Learning, Secure Multi-Party Computation, Homomorphic Encryption.

Abstract: Federated learning for training models over mobile devices is gaining popularity. Current systems for this task exhibit significant trade-offs between model accuracy, privacy guarantee, and device efficiency. For instance, Oort (OSDI 2021) provides excellent accuracy and efficiency but requires a trusted central server. On the other hand, Orchard (OSDI 2020) provides good accuracy and the differential privacy guarantee without a trusted server, but creates high overhead for the devices. This paper describes Aero, a new federated learning system that significantly improves this trade-off. Aero guarantees good accuracy, differential privacy without a trusted server, and low device overhead. The key idea of Aero is to tune system architecture and design to a specific federated learning algorithm. This tuning requires novel optimizations and techniques, including a new protocol to securely aggregate gradient updates from devices. An evaluation of Aero demonstrates that it provides comparable accuracy to plain federated learning (without differential privacy), and it improves efficiency (CPU and network) over Orchard by a factor of 10^5.

1 INTRODUCTION

Federated learning (FL) is a recent paradigm in machine learning that embraces a decentralized training architecture (McMahan et al., 2017). In contrast to the central model of learning where users *ship* their training data to a central server, users in FL download the latest model parameters from a server, perform *local* training to generate *updates* to the parameters, and send these updates to the server. Federated learning has gained popularity for mobiles as it can save network bandwidth and it is privacy-friendly—raw data stays at the devices.

Current systems for federated learning exhibit significant trade-offs between model accuracy, privacy, and device efficiency. For instance, one class of systems that includes FedScale (Lai et al., 2022) provides excellent accuracy (comparable to centralized learning) and device efficiency. But these systems provide only a weak notion of privacy. Specifically, even though devices ship updates rather than the raw training data (user images, text messages, search queries, etc.) to a central server, these updates can be reverse-engineered to reveal the raw data (Zhu et al., 2019; Melis et al., 2019). Thus, the compromise of a single server can reveal users' data.

On the other hand, systems such as Orchard (Roth et al., 2020) offer good accuracy and the rigorous differential privacy guarantee (Dwork, 2011; Dwork et al., 2006) for users' data. In fact, Orchard guarantees differential privacy while assuming that the server is byzantine, using techniques from cryptography. But the downside is the high overhead for the devices. For example, to train a convolutional neural network (CNN) with 1.2 million parameters (Reddi et al., 2020), Orchard requires from each device $\approx$14 minutes of training time on a six-core processor and $\approx$840 MiB in network transfers *per round of training* (§6.2). The full training requires a few hundred rounds. Furthermore, for a few randomly chosen devices, this per-round cost spikes to $\approx$214 hours of CPU time and $\approx$11 TiB of network transfers.

This paper describes a new federated learning system, Aero, that significantly improves the tradeoff between accuracy, privacy, and device overhead. Aero provides good accuracy, the differential privacy guarantee in the same byzantine threat model as Orchard, and low device overhead. For instance, most of the time Aero's devices incur overhead in milliseconds of CPU time and KiBs of network transfers.

The key idea in Aero is that it does not aim to be a general-purpose federated learning system, rather focuses on a particular algorithm called DP-FedAvg (McMahan et al., 2018) (§3.1). This algorithm samples devices that contribute updates in a round using a simple probability parameter (e.g., a device is selected with a probability of 10^{-5}), then aggregates updates across devices by averaging them,

Liu, K. and Gupta, T.
Federated Learning with Differential Privacy and an Untrusted Aggregator.
DOI: 10.5220/0012322100003648

In *Proceedings of the 10th International Conference on Information Systems Security and Privacy (ICISSP 2024)*, pages 379-389
ISBN: 978-989-758-683-5; ISSN: 2184-4356

and generates noise needed for differential privacy from a Gaussian distribution. Aero tunes system architecture and design to this algorithm, thereby gaining on performance by orders of magnitude.

We implemented Aero by extending the FedScale FL system (Lai et al., 2022) (§5). FedScale supports plain federated learning without differential privacy or protection against a byzantine server. However, we choose it as it is flexible, allows a programmer to specify models in the PyTorch framework, and includes a variety of models and datasets with a varying number of parameters for performance evaluation. Our evaluation of Aero's prototype (§6) shows that Aero trains models with comparable accuracy to FedScale, in particular, the plain FedAvg algorithm in FedScale (§6.1). Aero also improves overhead relative to Orchard by up to five orders of magnitude. For instance, for a 1.2M parameter CNN on the FEMNIST dataset (Reddi et al., 2020), and for a total population of 10^9 devices where 10^4 contribute updates per round, an Aero device requires 15 ms of CPU time and 3.12 KiB of network transfers per round. Occasionally (with a probability of 10^{-5} in a round) this overhead increases (e.g., when a device contributes updates) to 13.4 minutes of latency on a six-core processor and 234 MiB in network transfers.

2 PROBLEM AND BACKGROUND

This section outlines the problem and gives a short background on Orchard (Roth et al., 2020) that forms both a baseline and an inspiration for Aero.

2.1 Scenario and Threat Model

We consider a scenario consisting of a data analyst and a large number of mobile devices, e.g., hundreds of million. The analyst is interested in learning a machine learning model over the data on the devices. For instance, the analyst may want to train a recurrent neural network (RNN) to provide auto-completion suggestions for the android keyboard (Hard et al., 2018).

One restriction we place on this scenario is that the training must be done in a federated manner. As noted earlier (§1), federated learning proceeds in rounds, where in each round devices download the latest model parameters from a server, generate updates to these parameters locally, and send the updates to the server. The server aggregates the model updates. This repeats until the model achieves a target accuracy.

In this scenario, a malicious server, or even a malicious device, can execute many attacks. For instance, a malicious server can infer the training data of a device from the updates contributed by the device (Zhu et al., 2019; Melis et al., 2019). Similarly, a malicious device that receives model parameters from the server can reverse-engineer the model parameters, and learn another device's training input.

We assume the same threat model as Orchard: *OB+MC*. The server is honest-but-curious most of the time but occasionally byzantine (OB), while the devices are mostly correct (MC), but a small fraction can be malicious. The rationale behind this model is that the server's operator, e.g., Google, is reputed and subject to significant scrutiny from the press and the users, and thus unlikely to be byzantine for long. However, it may occasionally come under attack, e.g., from a rogue employee. Meanwhile, with billions of devices, even a small fraction is significant, and unlikely to be under adversary's control. For instance, 3% of a billion devices is already significantly larger than a large botnet.

2.2 Goals

Under the OB+MC threat model, we want our system to meet the following goals.

Privacy (Always). It must guarantee the gold standard definition of privacy, i.e., differential privacy (DP) (Dwork, 2011; Dwork et al., 2006; Dwork et al., 2014; Abadi et al., 2016), even when the server is byzantine. Informally, a system offers DP for model training if the probability of learning a particular set of model parameters is (approximately) independent of whether a device's training data is included. This means that DP prevents inference attacks where a particular device's input is revealed, as models are (approximately) independent of a device's input.

Accuracy (Conditional). During periods when the server or the devices that contribute in a round are not byzantine, the system must produce models with accuracy comparable to models trained via plain federated learning. That is, we want the impact of differential privacy to be low.

Efficiency and Scalability (Always). We want the system to support models with a large number of parameters while imposing a low to moderate device-side overhead. For the former, a reference point is the android keyboard auto-completion model (an RNN) with 1.4M parameters (Hard et al., 2018). For the device overhead, if a device participates regularly in training, e.g., in every round, then it should incur no more than a few seconds of CPU and a few MiBs in network transfers per round. However, we assume that devices can tolerate occasional amounts of addi-

```
    MAIN:
 1:   parameters
 2:     device selection probability $q \in (0,1]$
 3:     DP noise scale $z$
 4:     total # of devices $W$
 5:     clipping bound on device updates $S$
 6:   Initialize model $\theta^0$, DP budget accountant $\mathcal{M}$
 7:   for each round $t = 0, 1, 2, \ldots$ do
 8:     $C^t \leftarrow$ (sample users with probability $q$)
 9:     for each user $k \in C^t$ do
10:       $\Delta_k^t \leftarrow \textsc{UserUpdate}(k, \theta^t, S)$
11:     // Add updates and Gaussian DP noise
12:     $\Delta^t \leftarrow \sum_k \Delta_k^t + \mathcal{N}(0, I\sigma^2)$
13:     $\theta^{t+1} \leftarrow \theta^t + (\Delta^t/(qW))$        // Update model
14:     Update $\mathcal{M}$ based on $z$ and $q$

    USERUPDATE($k$, $\theta^0$, $S$)
15:   parameters $B, E, \eta$              // $\eta$ is learning rate
16:   $\theta \leftarrow \theta^0$
17:   for each local epoch $i$ in 1 to $E$ do
18:     $\mathcal{B} \leftarrow$ ($k$'s data split into size $B$ batches)
19:     for batch $b \in \mathcal{B}$ do
20:       $\theta \leftarrow \theta - \eta\nabla\ell(\theta\,;\,b)$          // $\ell$ is loss fn.
21:       $\theta \leftarrow \theta^0 + Clip(\theta - \theta^0, S)$
22:   return $\Delta_k = \theta - \theta^0$               // Already clipped
```

Figure 1: Pseudocode for the DP-FedAvg algorithm. $Clip(\cdot, S)$ scales its input vector such that its norm (Euclidean distance from the origin) is less than S. $\mathcal{M}$ is the privacy budget accountant of Abadi et al. (Abadi et al., 2016) that tracks the values of the DP parameters ε and δ.

tional work, contributing tens of minutes of CPU and a few hundred MiBs in network transfers.

3 OVERVIEW OF AERO

Aero focuses on a specific federated learning algorithm called DP-FedAvg (McMahan et al., 2018). We briefly explain how DP-FedAvg works and then give an overview of Aero.

3.1 DP-FedAvg Without Amplification

DP-FedAvg proceeds in discrete rounds (Figure 1). In each round t, it samples a small subset of user devices using a probability parameter q (line 8), and asks the sampled devices to provide updates to the global model parameters (line 10). The devices locally generate the updates before clipping them by a value S and uploading them (line 21); DP-FedAvg then aggregates these updates (line 12) and (separately) adds noise sampled from a Gaussian distribution. Finally, DP-FedAvg updates a privacy accountant $\mathcal{M}$ that computes, based on the noise scale z and sampling probability q, two parameters ε and δ associated with differential privacy (line 14). These parameters capture the strength of the guarantee: how much the model parameters learned after a round varies depending on a device's input.

Aero is tailored to three characteristics of DP-FedAvg. The first is the sampling of devices. The second characteristic is that the noise is sampled from a Gaussian distribution whose standard deviation σ is predetermined (set before the algorithm is run). The third characteristic is averaging of updates: DP-FedAvg sums updates with noise rather than combining them using a more complex function.

Finally, we remark that Aero can support DP-FedAvg only without the amplification assumption for DP. This is because the adversary (the byzantine server) can observe all traffic and knows which devices contribute updates for training. In contrast, the amplification assumption requires the server to be oblivious to the contributors, which in turn improves the privacy budget. We leave the addition of expensive oblivious approaches (which hide who is contributing updates besides hiding the updates themselves) to future work.

3.2 Architecture of Aero

Aero borrows two system components from Orchard: an *aggregator* and a public *bulletin board* (Figure 2). The aggregator runs server-side inside a data center and therefore consists of one or more powerful machines. Its main role is to combine updates from user devices without learning their content. The bulletin board is an immutable append-only log. The aggregator (which is potentially malicious) and the devices use the bulletin board to reliably broadcast messages and store state across rounds, e.g., the latest values of DP parameters ε and δ. Like Orchard, Aero assumes that free web services such as Wikipedia, or a public block-chain could serve as the bulletin board.

Like Orchard, Aero also consists of *committees* of devices, that is, randomly sampled subsets of devices. But instead of a single committee in Orchard, Aero has three types of committees tailored to the needs of DP-FedAvg. A *master committee* handles system setup, including key generation for cryptographic primitives. A *DP-noise committee* handles Gaussian noise generation. And multiple *decryption committees* perform decryption operations to release aggregated updates to the global model parameters at the end of a training round. Aero samples each committee afresh each round, dividing the committee workload across the large population of devices.

An architecture with separate committee types is deliberate and crucial. It helps tailor a committee's

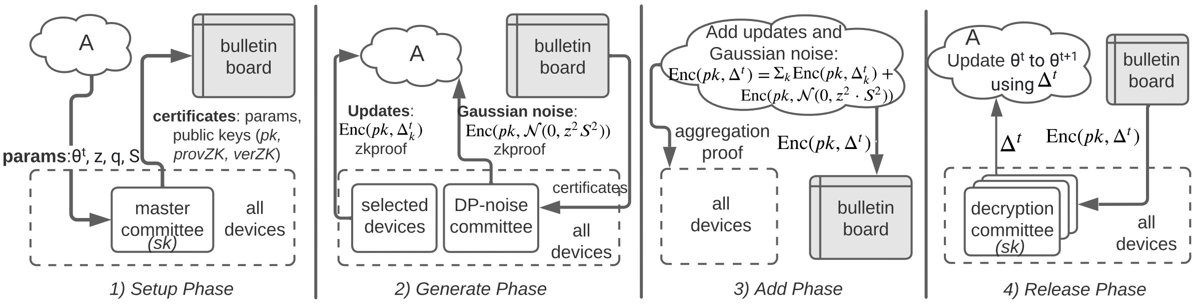


Figure 2: An overview of Aero's architecture and the four phases of its protocol.

protocol to its tasks to significantly improve efficiency. Besides, the use of multiple committees of the same type, i.e., multiple decryption committees, helps Aero scale with model size as each committee works on a subset of model parameters.

3.3 Protocol Overview of Aero

To begin training a model, a data analyst supplies input parameters (the model architecture, the initial model parameters, and parameters for DP-FedAvg) to the aggregator. The aggregator then initiates a *round-based protocol* consisting of discrete rounds. In each round, it executes one iteration of the for loop in the MAIN procedure of DP-FedAvg (line 7 in Figure 1). Each round further consists of the four phases of *setup*, *generate*, *add*, and *release* (Figure 2).

In the setup phase, the aggregator samples the various committees for the round. The master committee then receives and validates the input parameters, and generates keys for an additively homomorphic encryption (AHE) and a zero-knowledge (ZK) proof scheme. Aero's setup phase is similar to Orchard with the difference that Aero's master committee uses techniques to reuse keys across rounds rather than generating them fresh for each round.

Next, in the generate phase, (i) devices select themselves to generate updates for the round, and (ii) the DP-noise committee generates the Gaussian noise for DP. Both types of devices use novel techniques to perform their work efficiently. For instance, the DP-noise committee generates noise in an efficient, distributed manner.

Next, in the add phase, the aggregator adds the model updates to the Gaussian noise without learning the plaintext content of either of them. This is done through the use of the AHE scheme. Since the aggregator can be malicious, the entire population of devices collectively verifies the aggregator's work. The key point again is the efficiency for the devices, for which the aggregator and the devices use a new verifiable, secure aggregation protocol.

Finally, in the release phase, each decryption committee receives the secret key for the AHE scheme from the master committee and decrypts a few AHE ciphertexts from the add phase. The key point here is that a decryption committee avoids expensive, general-purpose cryptographic protocols to do the decryption.

4 DESIGN OF AERO

We now go over the design details of Aero phase-by-phase and we refer the readers to a technical report[1] for the detailed protocol description. The main challenge in each phase is keeping the device overhead low while protecting against the malicious aggregator and the malicious fraction of devices. We highlight these challenges, and Aero's key design choices and techniques.

But before proceeding, we briefly discuss committee formation, which is common to multiple phases. To form committees of randomly sampled devices, Aero uses the sortition protocol from Orchard. An important aspect of committee formation is committee size and the number of malicious devices in a committee: provision of a larger number of malicious devices A relative to the committee size C increases costs but ensures higher resiliency. Like Orchard, Aero makes a probabilistic argument (Roth et al., 2019) to select C and A such that the probability of the number of malicious devices exceeding A is small.

4.1 Setup Phase

Much of Aero's setup phase is similar to Orchard. During this phase, (i) the aggregator samples the master committee, which then (ii) receives inputs for the round (i.e., receives model parameters θ^t for the current round t, the device selection probability q, noise scale z, and clipping bound S), (iii) generates new values of the DP parameters ε, δ, and (iv) generates keys for cryptographic primitives (§3.3). We do not

[1] https://arxiv.org/abs/2312.10789

focus on the first three pieces as they are the same as in Orchard. Instead, the key challenge in Aero is the overhead of key generation.

Orchard uses secure multi-party computation (MPC) among the master committee members to correctly run the key generation function and ensure that even if the malicious members of the committee collude, they cannot recover the AHE secret key. The overhead of this MPC is high: $\approx$1 GiB of network transfers and 180 seconds of CPU time per committee device.

To reduce the overhead, one idea (Roth et al., 2021) is to reuse keys across rounds rather than generate them afresh for each round. But one has to be careful.

Consider the following attack. Say that the malicious aggregator receives a victim device $k's$ update $Enc(pk, \Delta_k^t)$ in round t; here Enc is the AHE scheme. Then, in the next round $t+1$, the aggregator colludes with a malicious device in the overall population to use $Enc(pk, \Delta_k^t)$ as the malicious device's update. This attack enables the aggregator to violate differential privacy as the victim device's input does not satisfy the required clipping bound S in round $t+1$ due to its multiple copies (§3.1). Orchard does not suffer from this attack as it generates fresh keys in each round: the ciphertext for round t decrypts to a random message with round $t+1$'s key. However, prior work that reuses keys in this manner (Roth et al., 2021) has this vulnerability.

Thus, Aero must apply the reuse-of-keys idea with care. Aero adjusts the generate and add phases of its protocol (§3.3) to prevent the aforementioned attack. We are not in a position yet to describe these changes, but we will detail them shortly when we describe these other phases (§4.2, §4.3).

Meanwhile, the changes in the setup phase relative to Orchard are the following: for the AHE secret key sk, Aero implements an efficient verifiable secret redistribution scheme (Gupta and Gopinath, 2006; Roth et al., 2021) such that committee members at round $t+1$ securely obtain the relevant secret (re)-shares of the key from the committee at round t. For the public keys (AHE public key pk, and both the ZK-proof public proving and verification keys), the committee for round t signs a certificate containing these keys and uploads it to the bulletin board, and the committee for round $t+1$ downloads it from the board.

4.2 Generate Phase

Recall from §3.3 that during this phase (i) Aero must pick a subset of devices to generate updates to the model parameters, (ii) the DP-noise committee must generate Gaussian noise for differential privacy, and (iii) both types of devices must encrypt their generated data (updates and noise) using AHE.

Device Sampling for Updates. Aero adopts a hybrid and efficient design in which devices sample themselves but the aggregator verifies the sampling. Let B^t be a publicly verifiable source of randomness for round t; this is the same randomness that is used in the sortition protocol to sample committees for the round. Then, each device k with public key π_k computes $PRG(\pi_k || B^t)$, where PRG is a pseudorandom generator. Next, the device scales the PRG output to a value between 0 and 1, and checks if the result is less than q. If selected, the device runs the USERUPDATE procedure (line 10 in Figure 1) to generate updates for the round. This approach of sampling is efficient as devices perform only local computations.

Gaussian Noise Generation. The default option is to make the DP-noise committee generate the noise using a secure computation MPC protocol, but as noted earlier in this paper, this option is expensive. Instead, Aero adapts prior work (Truex et al., 2019) on distributed Gaussian noise generation. Aero asks each honest device to sample its noise share from the distribution $\mathcal{N}(0, I\frac{\sigma^2}{C-A})$. Since there will be at least $C-A$ honest devices in the committee, the sum of noise shares is at least $\mathcal{N}(0, I\sigma^2)$.

Encryption and ZK-Proofs. Once the devices generate their updates or shares of the Gaussian noise, they encrypt the content using the public key of the AHE scheme to prevent the aggregator from learning the content. Further, they certify using a ZK-proof scheme that the encryption is done correctly and the data being encrypted is bounded by the clipping value S (so that malicious devices may not supply arbitrary updates). This encryption and ZK-proof generation is same as in Orchard, but Aero requires additional changes. Recall from the setup phase that Aero must ensure a ciphertext generated in a round is used only in that round, to prevent complications due to reuse of keys (§4.1). To do this, each device concatenates the round number t (as a timestamp) to the plaintext message before encrypting it. Further, the ZK-proof includes a check that proves that a prefix of the plaintext message equals the current round number.

4.3 Add Phase

Recall that during the add phase (i) the aggregator adds ciphertexts containing device updates to those containing shares of Gaussian noise, (ii) the devices collectively verify the aggregator's addition (§3.3).

This work during the add phase has subtle requirements. So first, we expand on these requirements while considering a toy example with two honest and a malicious device. The first honest device's input is $\textsc{Enc}(pk,\Delta)$, where Δ is its update, while the second honest device's input is $\textsc{Enc}(pk,n)$, where n is the Gaussian noise. For this toy example, first (**R1**), the aggregator must not omit $\textsc{Enc}(pk,n)$ from the aggregate as the added noise would then be insufficient to protect Δ and guarantee DP. Second (**R2**), the aggregator must not let the malicious device use $\textsc{Enc}(pk,\Delta)$ as its input. Relatedly, the aggregator itself must not modify $\textsc{Enc}(pk,\Delta)$ to $\textsc{Enc}(pk,k\cdot\Delta)$, where k is a scalar, using the homomorphic properties of the encryption scheme. The reason is that these changes can violate the clipping requirement that a device's input is bounded by S (e.g., $2\cdot\Delta$ may be larger than S). And, third (**R3**), the aggregator must ensure that the above (the malicious device or the aggregator copying a device's input) does not happen across rounds, as Aero uses the same encryption key in multiple rounds (§4.1).

One option to satisfy these requirements is to use the verifiable aggregation protocol of Orchard (Roth et al., 2019) that is based on summation trees. The main challenge is resource costs. Briefly, in this protocol, the aggregator arranges the ciphertexts to be aggregated as leaf nodes of a tree, and publishes the nodes of the tree leading to the root node. For example, the leaf nodes will be $\textsc{Enc}(pk,\Delta)$ and $\textsc{Enc}(pk,n)$, and the root node will be $\textsc{Enc}(pk,\Delta)+\textsc{Enc}(pk,n)$, for the toy example above. Then, devices in the entire population inspect parts of this tree: download a few children and their parents and check that the addition is done correctly, that the leaf nodes haven't been modified by the aggregator, and the leaf nodes that should be included are indeed included. The problem is that Orchard requires a device to download and check about $3\cdot s$ nodes of the tree, where s is a configurable parameter whose default value is six. For large models, each node is made of many ciphertexts (e.g., the 1.2M parameter CNN model requires $\ell=293$ ciphertexts), and 18 such nodes add to 738 MiB. Thus, Aero improves this protocol using two ideas.

Incorporating Finer-Grained Summation Trees. Aero observes that the entire population of devices that must collectively check the tree is massive (e.g., 10^9). Besides, although the tree has bulky nodes with many ciphertexts, the total number of nodes is not high due to sampling (e.g., only 10,000 devices contribute updates in a round). Thus, Aero moves away from one summation tree with "bulky" nodes, to ℓ summation trees with "small" nodes, where ℓ is the number of ciphertexts comprising a device's update (e.g., $\ell=293$ for the 1.2M parameter model). Then, each device probabilistically selects a handful of trees, and checks a few nodes within each selected tree. With this optimization, the verification work can be divided among more devices compared with Orchard.

Incorporating PIT. Checking the non-leaf vertices is a main source of overhead for the protocol above. The reason is that even though each non-leaf is a single ciphertext, this ciphertext is large: for the quantum-secure AHE scheme Aero uses (§5), a ciphertext is 131 KiB.

Aero reduces this overhead by using polynomial identity testing (PIT) (Schwartz, 1980; Zippel, 1979). This test says that given a d-degree polynomial $g(x)$ whose coefficients are in a field $\mathbb{F}$, one can test whether $g(x)$ is a zero polynomial by picking a number $r\in\mathbb{F}$ uniformly and testing whether $g(r)==0$. This works because a d-degree polynomial has at most d solutions to $g(x)==0$ and d is much less than $|\mathbb{F}|$.

Using PIT, Aero replaces the ciphertexts at the non-leafs with their evaluations at a random point r. Then, during the "Verify" step, a device checks whether these evaluations (rather than ciphertexts) add up. Thus, instead of downloading three quantum-secure ciphertexts with $2\cdot 2^{12}$ field elements each, a device downloads 2 elements of $\mathbb{F}$ per ciphertext.

A requirement for PIT is generation of r, which must be sampled uniformly from the coefficient field. For this task, Aero extends the master committee to publish an r to the bulletin board in the add step, using a known protocol to securely and efficiently generate a random number (Damgård et al., 2012).

4.4 Release Phase

During the release phase, Aero must decrypt the ℓ ciphertexts from the add phase, i.e., the ℓ root nodes of the ℓ summation trees. The default, but expensive, option is to use MPC among the members of the decryption committees. Aero addresses this efficiency challenge using known ideas and applying them. Aero uses multiple decryption committees (§3.2) and reduces each committee's work relative to the MPC baseline, using a fast distributed decryption protocol to decrypt the ciphertexts (Chen et al., 2019). The use of this protocol is possible as a decryption committee's task is only decryption given how Aero assigns work to different types of committees (§3.2).

Dataset	Model	Params.	FedScale	Aero
FEMNIST	LeNet	49K	75%	74%
	CNND	1.2M	78%	68%
	CNNF	1.7M	79%	68%
	AlexNet	3.9M	78%	40%
CIFAR10	LeNet	62K	48%	48%
	ResNet20	272K	59%	48%
	ResNet56	855K	54%	35%
Speech	MobileNetV2	2M	57%	4%

Figure 3: Test accuracy for different models after 480 rounds of training and differential privacy parameters (ε, δ) set to (5.04, $W^{-1.1}$). As shown later, increasing ε can recover the accuracy loss.

5 PROTOTYPE IMPLEMENTATION

We implemented a prototype of Aero[2] atop FedScale (Lai et al., 2022), which is a scalable system for federated learning capable of handling a large number of devices. By default, FedScale supports algorithms such as FedAvg (without differential privacy). Besides, it allows a data analyst to specify the model using the popular PyTorch framework.

Our Aero prototype first extends FedScale by extending the programming layer of FedScale with Opacus (Yousefpour et al., 2021), which is a library that adjusts a PyTorch model to make it suitable for differential privacy. Our prototype then extends the device-side and server-side code with Aero's corresponding components. Our prototype configures the cryptographic primitives for 128-bit security. For AHE, we use the BFV encryption scheme (Fan and Vercauteren, 2012). For ZK-proofs, we use Groth16 (Groth, 2016).

6 EVALUATION

We evaluate Aero in two parts. First, we compare it with plain federated learning, specifically, the FedScale system. This comparison sheds light on the cost of privacy in terms of model accuracy. Second, we compare Aero to Orchard, which is the state-of-the-art system for training models in a federated manner in the same threat model as Aero. This comparison helps understand the effectiveness of Aero's techniques in reducing overhead.

Testbed. Our testbed has machines of type `c5.24xlarge` on Amazon EC2. Each machine has 96vCPUs, 192 GiB RAM, and 25 Gbps network bandwidth. We use a single machine for running Aero's server. We use multiple machines to run each committee, where each device is assigned six CPUs given that modern mobiles have processors with four to eight CPUs. We also run a few "generator" and "verifier" devices that generate parameter updates and verify aggregation, to measure these devices' overhead.

Default System Configuration. Unless specified otherwise, we configure the systems to assume $W = 10^9$ total devices. For Aero, we set the default device sampling probability q in DP-FedAvg to 10^{-5}; i.e., the expected number of devices that contribute updates in a round is 10^4. We also configure Aero to use ten decryption committees, where each committee has a total of $C = 45$ devices of which $A = 18$ may be malicious. The first decryption committee also serves as the master committee. We configure the DP-noise committee with $(A, C) = (40, 280)$. For Orchard, we configure its committee to have 40 devices of which 16 may be malicious.[3]

6.1 Comparison with FedScale

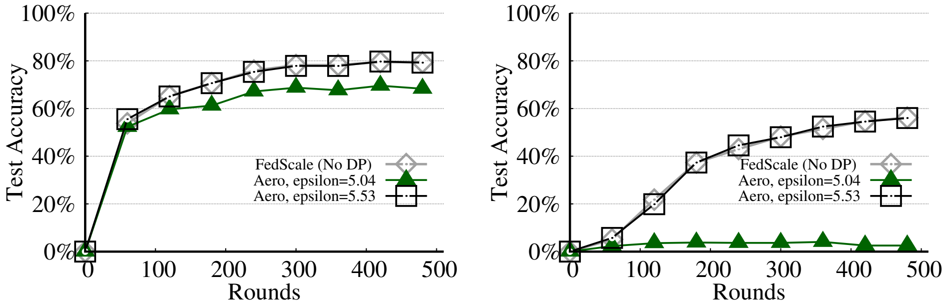
We evaluate several datasets and models to compare Aero with FedScale (Figure 3). We use CNND and CNNF for two different CNN models: one dropout model (Reddi et al., 2020) and the other from the FedAvg paper (McMahan et al., 2017). Aero's accuracy depends on the DP parameters ε and δ. For Figure 3 experiments, we set $\varepsilon = 5.04$ and $\delta = 1/W^{1.1}$. For both systems, we set all other training parameters (batch size, the number of device-side training epochs, etc.) per the examples provided by FedScale for each dataset.

Figure 3 compares the accuracies after 480 rounds of training (these models converge in roughly 400-500 rounds). Generally, Aero's accuracy loss grows with the number of model parameters. The reason is that DP-FedAvg adds noise for every parameter and thus the norm of the noise increases with the number of parameters.

Although Aero's accuracy loss is (very) high for a larger number of parameters, this loss is recoverable by increasing ε. Figure 4 shows accuracy for two values of ε for two example models. Increasing ε from 5.04 to 5.53 recovers the accuracy loss. For instance, for the CNNF model, FedScale's accuracy is 79.3% after 480 rounds, while Aero's is 79.2%. The reason is that as ε increases, more devices can contribute updates (q increases), which increases the signal relative to the differential privacy noise. Overall, Aero can give competitive accuracy as plain federated learning

[2] https://github.com/lonhuen/aero

[3] Aero's committees are larger because it must ensure that the chance of sampling more than A malicious devices across *any* of its committees is the same as in Orchard.



(a) CNNF

(b) MobileNetV2

Figure 4: Test accuracy versus rounds for Aero and FedScale for the 1.7M parameter CNNF model and the 2M parameter MobileNetV2 model.

for models with parameters ranging from tens of thousand to a few million.

6.2 Comparison to Orchard

Both Aero and Orchard have multiple types of devices. Aero has devices that participate in the master committee, generate updates (or Gaussian noise), verify the aggregator's work, and participate in the decryption committee. Similarly, Orchard has generator, verifier, and committee devices (Orchard has a single committee). We compare overhead for these devices separately.

Generator Device Overhead. The overhead for the generators changes only with the model size (after excluding the training time to generate the plain updates). Thus, we vary the number of model parameters and report overhead.

Figure 5a shows the CPU time and Figure 6a shows the network transfers with a varying number of model parameters. These overheads grow linearly with the number of model parameters (the network overhead is not a straight line as it includes a fixed cost of 60 MiB to download ZK-proof proving keys). The reason is that the dominant operations for a generator device are generating ZK-proofs and shipping ciphertexts to the aggregator. The number of both operations is proportional to the number of parameters (§4.2).

In terms of absolute overhead, a specific data point is a million-parameter model, e.g., the CNND model with 1.2M parameters. For this size, a generator device spends 1.01 hours in CPU time, or equivalently 13.4 minutes of latency (wall-clock time) over six cores. The generator also sends 101 MiB of data over the network. These overheads are moderate, considering the fact that the probability that a device will be a generator in a round is small: 10^{-5}.

Finally, the CPU and network overhead for Aero and Orchard is roughly the same. The reason is that the dominant operations for the two systems are common: ZK-proofs and upload of ciphertexts.

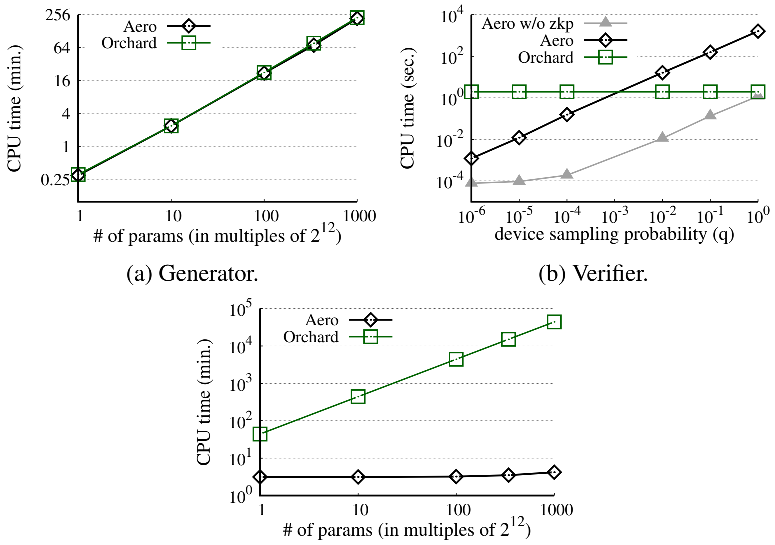
Verifier Device Overhead. Figure 5b shows CPU and Figure 6b shows network overhead for the verifier devices that participate in the verifiable aggregation protocol (§4.3). These experiments fix the number of model parameters to 1.2M and vary the probability q with which a verifier device samples summation trees to inspect (a verifier device in Aero checks $q \cdot \ell$ summation trees). For Orchard, overhead does not change with q (q is 1 as all devices are sampled).

Overall, Aero's verifier devices, which are the bulk of the devices in the system, are efficient consuming a few milliseconds of CPU and a few KiBs of network transfers. For instance, for $q = 10^{-5}$, Aero incurs 3.12 KiB in network and 15 ms in CPU time, while Orchard takes 1.96 seconds (130×) and 738 MiB ($2.36 \cdot 10^5 \times$).

Comparing Aero with Orchard, a verifier in Aero consumes lower CPU than Orchard for smaller values of q but a higher CPU for larger q. This trend is due to constants: even though an Aero device checks $q \cdot \ell$ summation trees and $3s$ ciphertexts in each tree versus $\ell \cdot 3s$ ciphertexts in Orchard, Aero devices verify the ZK-proofs to address the reuse-of-keys issue, while Orchard does not have such a requirement (§4.3). Each proof check takes ≈700 ms on a single CPU of `c5.24xlarge`. Indeed, Aero w/o ZK-proof check (another line in the plot) is strictly better than Orchard.

Aero's network overhead increases linearly with q, while Orchard's stays constant as it does not do sampling (Figure 6b). Notably, when $q = 1$, i.e., when Aero and Orchard check the same number of ciphertexts, a Aero verifier consumes 251 MB, which is $\approx 1/3$rd of Orchard. This is because polynomial identity testing allows a Aero verifier to download evaluations of ciphertext polynomials rather than full polynomials from non-leaf vertices (§4.3).

Committee Device Overhead. Figure 5c and Fig-



(a) Generator.

(b) Verifier.

(c) Decryption committee.

Figure 5: CPU time per device per round of training for different device roles in Aero and Orchard.

ure 6c show the CPU and network overhead of decryption committee devices as a function of the model size. (In Aero, the first decryption committee also serves as the master committee.)

Aero's overheads are much lower than Orchard's—for 1.2M parameters, CPU time is 206 s in Aero versus 214 hours in Orchard (i.e., 3751× lower), and network is 234 MiB in Aero versus 11 TiB in Orchard (i.e., $4.8 \cdot 10^4\times$ lower). This improvement is for two reasons. First, Aero divides the decryption of multiple ciphertexts across committees, and thus each performs less work. Second, Aero uses the distributed decryption protocol (§4.4), while Orchard uses the general-purpose MPC.

7 RELATED WORK

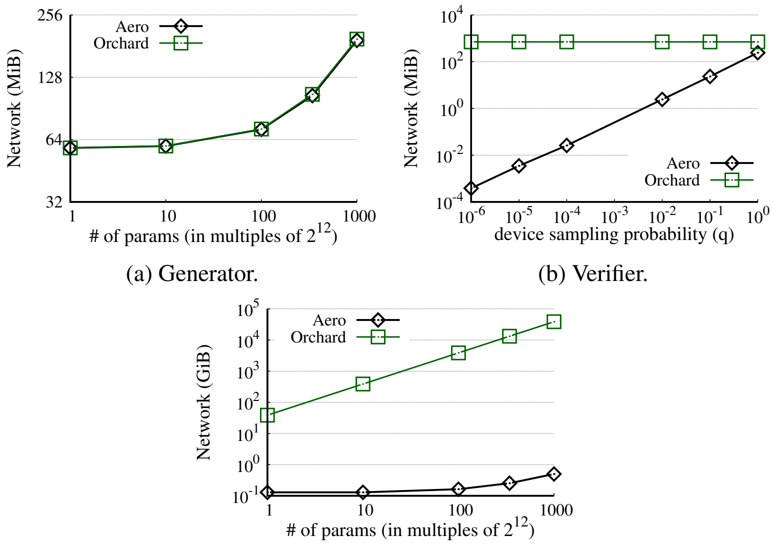
Aero's goal is to add the rigorous guarantee of differential privacy to federated learning—at low device overhead. This section compares Aero to prior work with similar goals.

Local Differential Privacy (LDP). In LDP, devices *locally* add noise to their updates before submitting them for aggregation (Truex et al., 2020; Ding et al., 2021). On the plus side, the privacy guarantee in LDP does not depend on the behavior of the aggregator, as devices add noises locally. Further, LDP is scalable as it adds small device-side overhead relative to plain federated learning. However, since each device perturbs its update, the trained model can have a large error.

Central Differential Privacy (CDP). Given the accuracy loss in LDP, many systems target CDP. For example, Oort (Lai et al., 2021) provides CDP but assumes a trusted aggregator. The core challenge is hiding sensitive device updates from the aggregator. Honeycrisp (Roth et al., 2019), Orchard (Roth et al., 2020), and Mycelium (Roth et al., 2021) target a setting of a billion devices. One of their key insights is to run expensive cryptographic protocols among a small, randomly-sampled committee, while leveraging an untrusted resourceful aggregator to help with the aggregation. Among the three systems, Orchard supports learning tasks, while Honeycrisp supports aggregate statistics and Mycelium supports graph analytics. The limitation of Orchard is that it imposes a large overhead on the devices and Aero improves over Orchard by several orders of magnitude (§6).

8 SUMMARY AND FUTURE WORK

Federated learning over a large number of mobile devices is getting significant attention both in industry and academia. One big challenge of current practical systems, those that provide good accuracy and efficiency, is the trust they require: devices must trust that the server will not be compromised. Aero extends these systems by showing that one can perform FL with good accuracy, moderate overhead, and the rigorous guarantee of differential privacy without trusting a central server. Aero improves the trade-off by focusing on a specific learning algorithm and tuning



(a) Generator.

(b) Verifier.

(c) Decryption committee.

Figure 6: Network transfers per device per round of training for different device roles in Aero and Orchard.

system architecture and design to this algorithm (§4). The main evaluation highlight is that Aero has comparable accuracy to plain federated learning, and improves over prior work Orchard that has strong guarantees by five orders of magnitude (§6).

Aero has some limitations. For example, Aero is designed for synchronized FL that proceeds in rounds and Aero uses the synchronization assumption for its privacy guarantee. For example, Aero requires enough verifier devices to be online during a round to check the aggregator's work. Thus Aero doesn't support asynchronous federated learning. In addition, Aero doesn't support filtering of device updates, which can improve model accuracy. We leave these improvements as future work.

ACKNOWLEDGMENTS

Richa Wadaskar helped conduct preliminary experiments for Aero. Natacha Crooks helped write an earlier draft of Aero. The presentation of this paper was greatly improved by the careful comments of anonymous reviewers of OSDI, SIGCOMM, NSDI, and ICISSP. This work is supported in part by a 2021 UCSB Academic Senate Grant, a UCSB Early Career Faculty Acceleration Award, DARPA under agreement number HR001118C0060, and National Science Foundation under Grant No. 2126327.

REFERENCES

Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. (2016). Deep learning with differential privacy. In *ACM Conference on Computer and Communications Security (CCS)*.

Chen, H., Dai, W., Kim, M., and Song, Y. (2019). Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In *ACM Conference on Computer and Communications Security (CCS)*.

Damgård, I., Pastro, V., Smart, N., and Zakarias, S. (2012). Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology—CRYPTO*.

Ding, J., Liang, G., Bi, J., and Pan, M. (2021). Differentially private and communication efficient collaborative learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

Dwork, C. (2011). A firm foundation for private data analysis. *Communications of the ACM*.

Dwork, C., McSherry, F., Nissim, K., and Smith, A. (2006). Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference (TCC)*.

Dwork, C., Roth, A., et al. (2014). The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*.

Fan, J. and Vercauteren, F. (2012). Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*

Groth, J. (2016). On the size of pairing-based non-interactive arguments. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*.

Gupta, V. and Gopinath, K. (2006). An extended verifiable secret redistribution protocol for archival systems. In

International Conference on Availability, Reliability and Security (ARES).

Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., and Ramage, D. (2018). Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*.

Lai, F., Dai, Y., Singapuram, S. S., Liu, J., Zhu, X., Madhyastha, H. V., and Chowdhury, M. (2022). FedScale: Benchmarking model and system performance of federated learning at scale. In *International Conference on Machine Learning (ICML)*.

Lai, F., Zhu, X., Madhyastha, H. V., and Chowdhury, M. (2021). Oort: Efficient federated learning via guided participant selection. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. (2017). Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*.

McMahan, H. B., Ramage, D., Talwar, K., and Zhang, L. (2018). Learning differentially private recurrent language models. In *International Conference on Learning Representations*.

Melis, L., Song, C., Cristofaro, E. D., and Shmatikov, V. (2019). Exploiting unintended feature leakage in collaborative learning. In *IEEE Symposium on Security and Privacy (S&P)*.

Reddi, S. J., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečnỳ, J., Kumar, S., and McMahan, H. B. (2020). Adaptive federated optimization. In *International Conference on Learning Representations*.

Roth, E., Newatia, K., Ma, Y., Zhong, K., Angel, S., and Haeberlen, A. (2021). Mycelium: Large-scale distributed graph queries with differential privacy. In *ACM Symposium on Operating Systems Principles (SOSP)*.

Roth, E., Noble, D., Falk, B. H., and Haeberlen, A. (2019). Honeycrisp: Large-scale differentially private aggregation without a trusted core. In *ACM Symposium on Operating Systems Principles (SOSP)*.

Roth, E., Zhang, H., Haeberlen, A., and Pierce, B. C. (2020). Orchard: Differentially private analytics at scale. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

Schwartz, J. T. (1980). Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*.

Truex, S., Baracaldo, N., Anwar, A., Steinke, T., Ludwig, H., Zhang, R., and Zhou, Y. (2019). A hybrid approach to privacy-preserving federated learning. In *ACM Workshop on Artificial Intelligence and Security*.

Truex, S., Liu, L., Chow, K.-H., Gursoy, M. E., and Wei, W. (2020). LDP-Fed: Federated learning with local differential privacy. In *Proceedings of the ACM International Workshop on Edge Systems, Analytics and Networking*.

Yousefpour, A., Shilov, I., Sablayrolles, A., Testuggine, D., Prasad, K., Malek, M., Nguyen, J., Ghosh, S., Bharadwaj, A., Zhao, J., Cormode, G., and Mironov, I. (2021). Opacus: User-friendly differential privacy library in PyTorch. *arXiv preprint arXiv:2109.12298*.

Zhu, L., Liu, Z., and Han, S. (2019). Deep leakage from gradients. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Zippel, R. (1979). Probabilistic algorithms for sparse polynomials. In *International Symposium on Symbolic and Algebraic Manipulation*.