CLIP: Client-Side Invariant Pruning for Mitigating Stragglers in Secure Federated Learning

Anthony DiMaggio
University of Toronto
Toronto, Canada
dimaggio@cs.toronto.edu

Raghav Sharma
University of Toronto
Toronto, Canada
raghav@cs.toronto.edu

Gururaj Saileshwar
University of Toronto
Toronto, Canada
gururaj@cs.toronto.edu

Abstract

Secure federated learning (FL) preserves data privacy during distributed model training. However, deploying such frameworks across heterogeneous devices results in performance bottlenecks, due to straggler clients with limited computational or network capabilities, slowing training for all participating clients. This paper introduces the first straggler mitigation technique for secure aggregation with deep neural networks. We propose CLIP, a client-side invariant neuron pruning technique coupled with network-aware pruning, that addresses compute and network bottlenecks due to stragglers during training with minimal accuracy loss. Our technique accelerates secure FL training by 13% to 34% across multiple datasets (CIFAR10, Shakespeare, FEMNIST) with an accuracy impact of between 1.3% improvement to 2.6% reduction.

1 Introduction

Federated Learning (FL) enables a decentralized approach to training machine learning models on edge devices, allowing multiple clients to update a global model without sharing local datasets. In each training round, clients compute gradients locally, which are aggregated by a central server and used to update a global model, which is re-distributed to participants for the next training round[19].

Need for Secure Aggregation. While FL improves privacy by keeping client data local, model updates sent to the server can still leak sensitive information through model inversion attacks [31, 12], which show that the server can recover approximate representations of the client's training data by observing the individual client gradients. Secure federated learning mitigates these privacy leaks using secure aggregation (SecAgg) [4, 5, 2]. SecAgg ensures that the server only observes the aggregated update, not individual updates, through secure multi-party computation (MPC) protocols that mask individual contributions, thus protecting against model inversion attacks. While SecAgg enhances privacy, it suffers from performance bottlenecks due to device heterogeneity in the distributed setting.

Straggler Problem. A central challenge in decentralized FL systems is device heterogeneity. Mobile clients vary significantly in their computational power and network bandwidth, creating "stragglers" or slower devices that delay training for all clients. This is because the server waits for all clients to provide their updates, before aggregating them and sending the clients the updated model. Thus, faster clients are held back by stragglers in FL, slowing down training time for all clients [23]. Unfortunately, prior straggler mitigation techniques for FL are incompatible with secure aggregation.

Challenge-1: Lack of Straggler Mitigation in Secure FL. Prior works [26, 23, 22] propose pruning the models for straggler clients, to reduce their computational effort and equalize their training time with non-stragglers. Compared to leaving out the straggler's contributions, this approach achieves higher accuracy and fairness. State-of-the-art techniques, such as FLuID [26], intelligently prune invariant neurons at the server-side to minimize accuracy impact of pruning. Unfortunately, such approaches require the server to access individual client updates from non-stragglers to identify

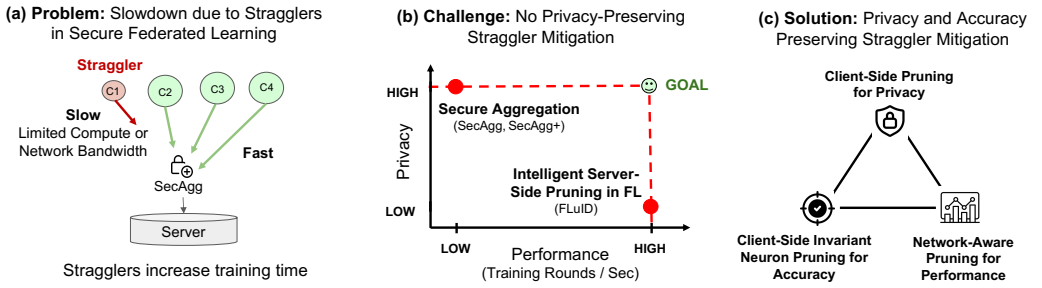


Figure 1: (a) Secure aggregation in federated learning preserves privacy of client training data; however stragglers with slow compute or network bandwidth increase training time. (b) State-of-the-art server-side solutions for straggler mitigation are incompatible with secure aggregation and do not provide privacy. (c) Our solution enables client-side invariant neuron pruning and network-aware pruning that improves training speed while maintaining accuracy and privacy.

invariant neurons in them, which is incompatible with secure aggregation, as the server only has access to the aggregate. Simpler client-side pruning techniques, such as Random Dropout [6, 22], unfortunately impact accuracy significantly. This leaves secure federated learning without effective straggler mitigation techniques that balance training time, accuracy, and privacy.

Challenge-2: Network Stragglers in Secure FL. Pruning in non-secure FL [26, 6, 22] also mitigates network bottlenecks by reducing the size of gradient vectors transferred over the network by straggler clients. However, in secure FL, the secure multi-party computation protocols used in SecAgg require all clients to transmit gradient vectors of identical size to ensure proper cancellation of encryption masks during aggregation. This constraint prevents the naive use of pruning techniques for reducing network overhead. Consequently, while pruning can mitigate compute stragglers, it cannot address the bottlenecks due to network-based stragglers in secure FL.

This paper introduces the first straggler mitigation technique for secure federated learning, addressing both compute and network-related challenges. Our approach extends server-side pruning techniques such as invariant dropout [26] to a client-side solution compatible with secure aggregation (SecAgg). This reduces training time for clients, while preserving model accuracy and client privacy.

To securely address network stragglers, we identify that reducing model update sizes compromises privacy in Secure FL and hence preserve fixed-size updates by padding smaller updates from stragglers. Instead, we propose network-aware pruning—a technique that aggressively prunes models to reduce client training time and compensate for longer communication times. To mitigate accuracy loss from over-pruning, our method ensures stability in pruned neurons across training rounds and dynamically adjusts pruning randomness as training progresses. Together, this approach mitigates both compute and network bottlenecks, with accuracy impact of between 1.3% improvement to 2.6% reduction.

Overall, this paper makes the following contributions:

1. Analysis of the Impact of Stragglers in Secure FL. We provide the first systematic analysis of secure FL training times in the presence of stragglers, quantifying the effects of both compute and network stragglers on the SecAgg protocol [5, 2].

2. Client-Side Invariant Neuron Pruning. We propose the first client-side invariant neuron dropout technique in federated learning that is compatible with Secure Aggregation. This achieves reductions in training time for compute stragglers, reducing training time without sacrificing privacy or accuracy.

3. Network-Aware Pruning for Stragglers. To account for network stragglers, we propose more aggressive pruning to reduce client training time and compensate for the increasing network communication time. This reduces training time for network stragglers, without sacrificing privacy.

We evaluate our solution in a Secure FL setting with up to 50 clients, with the majority of non-straggler clients running at 3GHz CPU frequency with a 5G network connection, and a minority of the clients as stragglers with a 4G connection running at 2GHz CPU frequency. Our technique accelerates secure FL training by 13% to 34% across multiple datasets (CIFAR10, Shakespeare, FEMNIST) addressing straggler overheads securely without impacting privacy, with accuracy impact of no more than 2.6%.

2 Related Work

2.1 Secure Aggregation

Federated learning (FL) enables clients to train models locally and share gradients with a server, which aggregates them using Federated Average [21, 19]. However, keeping data local is not sufficient for privacy; sharing plaintext gradients with the server also exposes client training data to model inversion attacks[31, 30, 12, 29, 14] where an attacker can recover a client's private training data from its gradients.

Secure aggregation protocols address this using secure multi-party computation protocols (MPC) by masking individual updates with random values that cancel during aggregation, ensuring only the aggregated sum is revealed to the server [4, 5, 2].For each training round, the key steps in these protocols apart from the client fit include: (1) securely exchanging keys between clients, (2) generating the random mask vectors using the keys, (3) training the model and masking the gradients and transmitting them to the server, (4) the unmasking step at the server, post aggregation, where it requests the key shares from other clients for any masks that did not cancel out, allowing the server to learn the aggregated model, and (5) the server distributing the aggregated model back to clients.

Secure Aggregation Protocols. While SecAgg [5] first proposed secure aggregation using MPC, several variants subsequently reduce the communication overheads involved with the protocol. SecAgg+[2] replaces the fully connected communication graph among clients with a k-regular graph, where k can be flexibly chosen to produce different performance characteristics. LightSecAgg[25] speeds up the unmask stage by eliminating the typically used secret-share protocol and opting for mask reconstruction via encoding/decoding. FastSecAgg[17] introduces a faster algorithm for secret-sharing, based on the fast fourier transform. While these methods reduce the computation or communication costs of secure aggregation, they do not address the straggler problem or device heterogeneity. While this paper uses SecAgg+ [2], our techniques are as applicable to other protocols.

Straggler-aware Secure Aggregation. CodedSecAgg encodes and strips a client's data across multiple devices to provide resiliency against stragglers [24]. Thus, a dropped out straggler can be compensated by contributions from non-straggler. However, this only addresses network stragglers, not compute stragglers, and it is only applicable to linear regression, not deep neural networks.

2.2 Pruning for Straggler Mitigation in FL

In a decentralized setting, clients with diverse compute or network capabilities [9] can slow down the training times, as the updates sent to the server are aggregated synchronously [1, 27]. Asynchronous processing [8, 7] or the dropping of slower clients [18] can drop accuracy and increase convergence times. Thus, the training round times get bottlenecked by the slowest straggler [23].

PruneFL [15] adjusts sub-model sizes based on varying client resources throughout training. However, it assumes uniform sub-model sizes across clients, limiting its ability to address stragglers. Many recent works [22, 26, 13] mitigate stragglers in FL by pruning DNNs trained by the stragglers. Thus, slower clients contribute to the global model while training and transmitting a smaller model, making their round time comparable to non-stragglers and avoiding slowdown in training times.

Non-intelligent Client-Side Pruning. *Federated Dropout or Random Dropout* [6, 22] randomly choose neurons to drop out until a desired sub-model size is reached. Alternatively, *Ordered Dropout* [13] drops a desired number of neurons from the tail end of the filter for a given layer. These approaches are compatible with secure aggregation as they can be implemented on the client-side. However, the neurons thus pruned non-intelligently can degrade accuracy, making this less desirable.

Intelligent Server-Side Pruning. State-of-the-art pruning techniques intelligently prune neurons to preserve accuracy. *Invariant Dropout* [26] proposes identifying invariant neurons, whose updates from non-stragglers fall within a threshold at the server-side, and pruning these neurons from stragglers, as they have minimal impact on accuracy. As certain neurons train quickly and do not change much subsequently, by pruning them, Invariant Dropout minimizes the impact on accuracy. Unfortunately, this requires inspecting the updates of all the clients at the server-side. Thus, such intelligent straggler mitigation techniques are incompatible with secure aggregation. Our goal is to make intelligent pruning compatible with secure FL, to speedup training times without impacting accuracy or privacy.

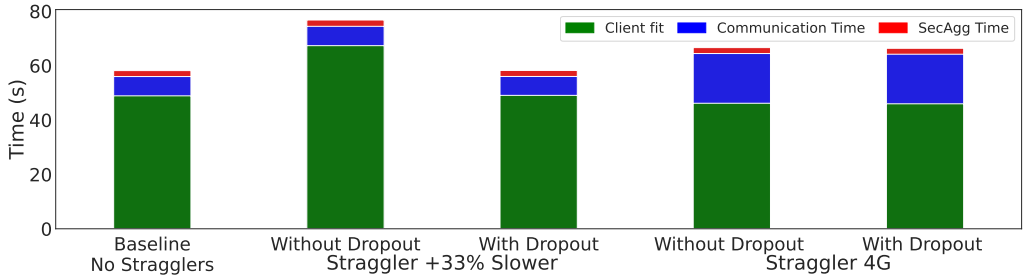


Figure 2: Time per training round for FL with secure aggregation in the presence of stragglers (clients with slower compute and lower network bandwidths). With compute stragglers (with 33% slower compute), training round time increases by 24.2%, but ideally, pruning straggler models has the potential to reduce this to within 1%. For network stragglers, with less bandwidth (4G), the training time increases by 12.7%, but dropout cannot naively address this overhead due to SecAgg constraints.

3 Analysis of Straggler Impact on Training Times in Secure FL

Methodology. We characterize 20 clients training a VGG-9 DNN model on the CIFAR-10 dataset, using the SecAgg+ implementation in the Flower FL framework [3]. By default, our clients have a CPU frequency of 3GHz, and a network connection equivalent to a 5G network (155 Mbps download/17 Mbps upload). We model 4 of the 20 clients as stragglers. We characterize two kinds of stragglers: compute stragglers – clients with a 33% slower compute (2GHz CPU frequency), and network stragglers – clients with a 4G connection (27 Mbps download/7 Mbps upload). Figure 2 shows the impact of stragglers with slower compute and network connection on training time, breaking it down into its different components: client fit (training), communication (sending and receiving gradients), and SecAgg (setup keys, mask, unmask). It demonstrates the potential to address this slowdown with an ideal dropout scheme that prunes the model trained by stragglers.

Impact of Compute and Network Stragglers. Compared to a baseline without stragglers, a compute straggler (33% slower) increases training round time by 24.2%, due to a slower client fit time. Similarly, the configuration with a network straggler (client with a 4G connection) causes the overall time to increase by 12.7% compared to baseline (all 5G connections), due to an increase in the time it takes for the straggler to send and receive the gradients and update the global to and from the server.

Observation 1: Compute stragglers increase the overall training round time by up to 24.2% (due to slower client fit), while network stragglers increase it by 12.7% (due to slower communication).

Interestingly, the secure aggregation steps (SecAgg+ setup, mask, unmask) do not incur considerable overheads, indicating that straggler mitigation methods in FL using pruning should be highly effective.

Potential for Ideal Dropout. To evaluate the potential benefits of dropout, we implement an idealized dropout technique on the client-side, similar to prior approaches [6, 13], making the idealized assumption of no accuracy impact.

As shown in Figure 2, dropout can reduce the overhead of compute stragglers completely, making training round time equivalent to the baseline (no stragglers), as the straggler's client fit time no longer bottlenecks the FL rounds. However, for network stragglers, dropout cannot provide any benefit as even after training a smaller model, the straggler has to transmit a full gradient vector, padded with 0s for the pruned layers, due to the constraints of secure aggregation. Thus, the slowdown due to network stragglers is difficult to mitigate with existing dropout techniques.

Observation 2: Client-side pruning can mitigate compute stragglers if they are made compatible with secure aggregation with minimal accuracy impact, but it cannot mitigate network stragglers.

These observations motivate us to extend invariant dropout to make it compatible with secure aggregation and mitigate compute stragglers, and explore new techniques to mitigate network stragglers, as we describe next.

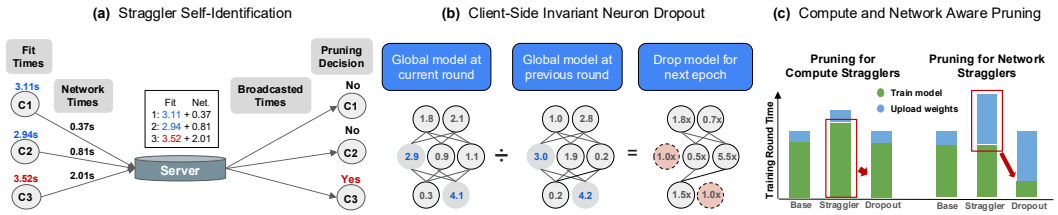


Figure 3: Overview of CLIP. It consists of (a) a straggler-self identification protocol on the client-side, (b) client-side invariant neuron selection algorithm for minimal accuracy impact and maintaining privacy (compatibility with SecAgg), and (c) compute and network aware pruning for performance.

4 Design of CLIP: Client-Side Invariant Pruning for Stragglers in Secure FL

To address the straggler problem in Secure FL without compromising accuracy or privacy, we extend prior server-side invariant pruning [26] fully to the client-side, ensuring compatibility with the privacy constraints of SecAgg. Our proposal, CLIP, Client-side Invariant Pruning, consists of the following:

1. **Straggler-Self Identification for Privacy.** We design a distributed protocol for clients to self-identify as stragglers. This decentralized approach ensures the dropout mechanism operates exclusively on the client-side, maintaining privacy by avoiding server involvement.
2. **Client-Side Invariant Neuron Dropout for Accuracy.** We propose a method to identify globally invariant neurons, to minimize the accuracy impact of pruning, with only locally available information. This reduces training time for compute stragglers while preserving model accuracy.
3. **Network-Aware Pruning for Performance.** To address network bottlenecks, we over-prune models on straggler clients to reduce client fit times further, compensating for increased communication times. Despite pruning, full model updates are transmitted to ensure compatibility with SecAgg, maintaining privacy while accelerating training with network stragglers.

4.1 Client-Side Straggler Self-Identification.

To conceal which clients prune their models and prevent the server from forcing clients to perform arbitrary amount of pruning, the clients must self-identify as a straggler. Fortunately, secure aggregation enables encrypted all-to-all communication using keys shared during setup. In CLIP, clients reuse these keys to broadcast their fit times to others, with the server forwarding encrypted packets, as shown in Figure 3(a). By comparing received times, a client determines if it is a straggler (e.g., bottom 20th percentile) and decides on pruning. Additionally, the server also creates an additional packet per client that contains the network times to send/receive every client's updates, and forwards these along with the compute time packets. These are used for network-aware pruning in Section 4.3.

4.2 Client-Side Invariant Neuron Selection and Pruning

Intelligent pruning identifies and removes neurons that minimally contribute to model accuracy. Unlike server-side methods relying on non-straggler gradients to identify invariant neurons [26] to be pruned in stragglers, CLIP operates within secure FL constraints where neither server nor client has a network-wide gradient view. Instead, it leverages temporal information, comparing global model weights from consecutive epochs to identify invariant neurons with minimal relative weight change.

Figure 3(b) shows how our technique identifies invariant neurons at the client-side. In each training round, the client buffers the previous epoch's global model weights and calculates relative changes for each neuron. Neurons with changes below a threshold (e.g., $0.8\times$ to $1.2\times$) are deemed invariant and fit for pruning for the straggler client. This computation by the straggler is entirely server-independent.

Invariant Neuron Selection. First, the straggler determines the sub-model size after pruning, p, to achieve the desired speedup. Suppose there are M stragglers, t_k is the round time (computation and network times together) for the slowest non-straggler, and T for the slowest non-straggler. The desired sub-model size for the kth slowest straggler is $p_k = \frac{t_k}{T}$.

Algorithm 1 Invariant Neuron Selection

Input: round i, current weights $w^{(i)}$, last weights $w^{(i-1)}$, model size N, sub-model size p
if $i = 2$ **then**
  Initialize $threshold = avg(\frac{w_j^{(i)} - w_j^{(i-1)}}{w_j^{(i-1)}})$.
if $i > 2$ **then**
  Initialize $toDrop = []$
  for $j = 1$ **to** N **do**
    if $\frac{w_j^{(i)} - w_j^{(i-1)}}{w_j^{(i-1)}} \leq threshold$ **then**
      $toDrop.push(j)$
  if $size(toDrop) \geq round(N * p)$ **then**
    $toDrop = toDrop.sample(round(N * p))$
  else
    $alt = round(N * p) - size(toDrop)$
    $slack = slackNeurons(i, N, alt, prevDrop)$
    $toDrop.extend(slack)$
$prevDrop = toDrop$
Output: $toDrop$

Algorithm 2 Slack Neuron Selection

Input: round i, model size N, slack size S, previously dropped neurons $prevDrop$, global accuracy history $accs$
$S = round(S)$
if $i > 5$ **then**
  $bench = avg(accs[0, 5])$
  $cur = avg(accs[i - 5, i])$
  $detPct = min(max(cur/bench, 1), 0)$
  $prev = prevDrop.sample(S * detPct)$
  $rand = seq(1..N).sample(S - size(prev))$
  $slack = prev.concat(rand)$
else
  $slack = seq(1..N).sample(S)$
Output: $slack$

For a one-layer model with N neurons, such that $w_j^{(i)}$ is the j^{th} neuron of the global model at the i^{th} round, invariance is quantified as:

$$Invariance(j) = \frac{|w_j^{(i)} - w_j^{(i-1)}|}{w_j^{(i-1)}} \tag{1}$$

After allowing two warm-up rounds, we establish an invariance $threshold$ by calculating the average invariance across all neurons and neurons meeting the invariance threshold are collected into a set D, $\mathcal{D} = \{j \mid Invariance(j) \leq threshold\}$. If $|\mathcal{D}| \geq round(N * p_k)$, we have sufficient invariant neurons that can be pruned to achieve our desired sub-model size. If $|\mathcal{D}| < round(N * p_k)$, we need to prune additional neurons to achieve the desired sub-model size. In this case, we augment $\mathcal{D}$ by selecting $|\mathcal{D}| - round(N * p_k)$ additional "slack" neurons to be pruned from the remaining neurons.

Adaptive Slack Neuron Selection. Slack neuron selection is critical in minimizing accuracy impact, particularly in scenarios where we need to over-prune stragglers (as described in Section 4.3). We select slack neurons adaptively, partially from previously dropped neurons to ensure stability, and partially at random to enable the model to learn more effectively. We observe that varying the ratio between the two based on the relative accuracy gains in successive rounds is effective as stability is preferred early on and randomness is preferred in later rounds. Let $accs$ represent the array of accuracy gains per round ($accs[i]$ for round i). We compute the average of $accs[0, 5]$ and $accs[i - 5, i]$, using their ratio to estimate how much training has progressed – a higher ratio indicates more advanced training, prompting a higher ratio of random selection in our slack neurons. This approach stabilizes the model composition in early rounds and increases randomness in later rounds. Algorithm 1 and 2 formalize how CLIP selects invariant and slack neurons.

4.3 Pruning for Network Stragglers

Problem. Thus far we only discussed pruning to address increased training times caused by compute stragglers (increase in client-fit time), that were also addressed in prior works [26]. However, our analysis in Section 3 suggests that network stragglers, clients with slower network connections, also bottleneck training due to larger delays in communicating model weight updates. Unfortunately, training a smaller, pruned model naively cannot address communication delays because full model updates must be sent to the server, padded with zeros for the pruned neurons to remain compatible with the secure aggregation protocols.

Our solution. To mitigate the impact of increased communication times, CLIP enables network stragglers to start weight uploads earlier by aggressively pruning their models beyond what is needed to equalize client fit times. This approach ensures network stragglers complete their training epochs first, freeing time to initiate communication while other clients are still training. The resulting overlap hides longer communication times. Figure 3(c) illustrates how our method addresses both compute and network stragglers by pruning models even when client fit times are equal. In comparison to prior

works [26] that only prune to equalize client fit times, we focus on shortening the overall training epochs for network stragglers, equalizing their overall round times with non-stragglers. The accuracy impact of over-pruning is minimized by our adaptive slack neuron selection (Section 4.2), which balances the impact of random neurons ("slack" neurons) selected during over-pruning by ensuring stability of pruned neurons across epochs, especially at the early stage of training.

5 Privacy Arguments

CLIP performs straggler-aware pruning fully on the client-side, and thus is fully compatible with secure aggregation, unlike prior server-side pruning [26]. Dropped layers are padded with zeros and masked using random masks compliant with the SecAgg protocol, preventing the server from inferring which layers were pruned based on network communication. The resulting update vectors maintain identical size and entropy for both stragglers and non-stragglers.

Under an honest-but-curious threat model, the server cannot identify pruned neurons or exploit pruning for targeted model inversion attacks. Randomized neuron selection at the client-side, which increases over training epochs, further enhances privacy. Together, the design of CLIP ensures that the server cannot carry out any worse model inversion attacks by specifically targeting layers of the model that have experienced dropout by the stragglers, compared to secure aggregation itself.

As secure aggregation relies on statistical privacy guarantees, where the server observes the aggregate from N clients, the privacy of secure aggregation with pruning for K straggler clients is lower-bounded by the privacy of secure aggregation with $N - K$ clients. As long as stragglers are a small fraction and the set of total clients is reasonably large ($\sim$10 or more), the privacy loss is negligible. Techniques using additive noise for differential privacy [10] can be used with secure aggregation for stronger privacy; CLIP is orthogonal and poses no limitations on adoption of these techniques.

6 Evaluation

6.1 Methodology

Model and datasets. Like prior FL works [26, 16], we evaluate CLIP with the CIFAR10 dataset using VGG-9, FEMNIST dataset using ResNet-18, and Shakespeare dataset using LSTM. These meet the resource constraints of edge devices (e.g., mobile phones) where FL may be deployed.

Secure Federated Learning Setup. Our FL client and server applications are built on top of the Flower [3] Python library using the implementation of SecAgg+ [2] in Flower. In our experiments, we use 20 clients communicating with a single server, among which 4 clients are randomly chosen as stragglers. We divide our dataset equally into non-overlapping partitions, and each client trains on a distinct partition. We also perform a scalability study with 5 and 50 clients, with 20% as stragglers.

Evaluation Setup. We run our experiments on an AMD EPYC 7713 64-core system. Each client and the server process are pinned to a separate core. The client and server interact using gRPC streams established by Flower's [3] networking stack. We control the network throughput for each client separately using the iproute2 [11] tool in Linux, which can throttle TCP/IP network traffic for the client to an arbitrary speed. By default, non-stragglers are configured to have a CPU frequency of 3GHz (similar to a high-end Apple iPhone 15) and a network speed of 5G (155 Mbps download/17 Mbps upload). Stragglers are configured with a +33% slower CPU frequency of 2GHz (similar to a medium-end Samsung Galaxy A16) and a network speed of 4G (27 Mbps download/7 Mbps upload). The network speeds for 5G and 4G are based on the average speeds reported by OpenSignal [28, 20].

Evaluation metrics. We compare CLIP against a baseline, SecAgg implementation with no dropout, and two prior client-side dropout techniques: 1) random dropout [6] and 2) ordered dropout [13]. We compare these based on test accuracy and overall training times (wall-clock).

6.2 Accuracy evaluations

Accuracy vs Sub-Model Size. Fig. 4 compares the accuracy versus submodel size trade-off of our client-side invariant dropout versus ordered dropout, random dropout, and the baseline (SecAgg). Across a range of sub-model sizes, CLIP achieves as good or better accuracy compared to prior dropout techniques.

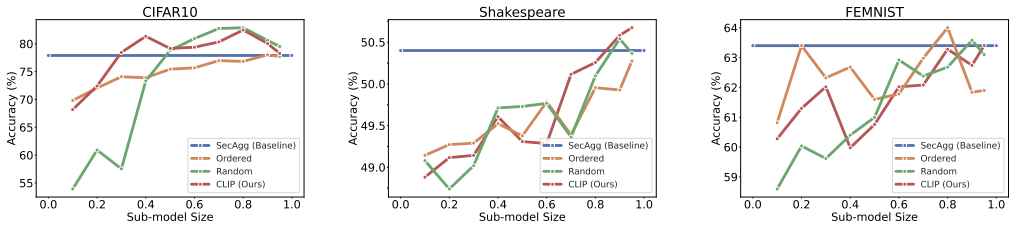


Figure 4: Accuracy versus sub-model size of CLIP compared to Ordered and Random Dropout and SecAgg (baseline). Averaged over 3 runs, where each run is trained for 100 rounds with 20 clients.

For CIFAR10, CLIP outperforms the baseline for submodel sizes of 0.3× to 0.9×, and is better than all other dropout techniques for 0.2× to 0.5×, while being comparable to Random Dropout from 0.5× to 0.9×. At 0.8× sub-model size, it achieves its best model accuracy of 82.4% accuracy, beating out baseline by 4.5%. Even at sub-model size of 0.5 (the maximum pruning we permit in practice), CLIP has an accuracy improvement of more than 1% compared to baseline.

For Shakespeare, CLIP outperforms baseline and the other dropout techniques for sub-model sizes above 0.8× and is better than the other dropout techniques until the sub-model size of 0.6×. For smaller sub-model sizes, it is comparable to the other dropout techniques or has slightly lower accuracy.

For FEMNIST, CLIP suffers an accuracy drop compared to baseline and the other dropout techniques for all sub-model sizes below 0.95×. For the sub-model size of 0.5, which is the maximum pruning we perform in practice, CLIP suffers an accuracy drop of almost 2%, whereas other techniques perform slightly better – Random is almost 1% better at sub-model size of 0.6×, while Ordered is better by almost 0.6% at sub-model sizes of 0.7-0.8. However, CLIP still provides overall speedups in training, since it trains to similar levels of accuracies much faster, as we show in Section 6.3.

Overall Accuracy Impact. To measure the overall accuracy impact of pruning in CLIP, we train 100 rounds with 20 clients and 4 stragglers, and record the maximum training accuracy with Baseline, CLIP with only pruning for compute stragglers (CLIP-C) and CLIP for both compute and network stragglers. The difference between CLIP-C and CLIP shows the impact of security on accuracy, i.e., due to the constraint of secure aggregation resulting in over-pruning for network stragglers. As shown in Table 2 in Section A, CLIP incurs an accuracy improvement of 1.3% for CIFAR10, and a drop of 1.1% for Shakespeare and 2.6% for FEMNIST. CLIP-C alone has a benefit of 3.5% for CIFAR10 and 0.4% for FEMNIST and a drop of 0.5% for Shakespeare. Thus, CLIP has an accuracy impact of no more than 2.6%, with almost 3% attributed to security, i.e., compatibility with secure aggregation.

6.3 Performance Evaluations

Training Round Times. We measure speedup in round times with CLIP across datasets in Figure 6. We observe that depending on the straggler configuration, training round time is dominated by either the client fit and/or the network time. CLIP is able to reduce the round time by 10.9% - 31.6% using compute-aware dropout and by an additional 2.2% - 12.7% using compute + network-aware dropout.

Accuracy vs Wall-Clock Times. While dropout improves training round times, it can impact accuracy per training round compared to the baseline, making it unclear whether pruning is in fact always beneficial. To measure the effectiveness of CLIP, we measure the accuracy versus elapsed wall clock time and measure the speedup in overall training times to reach the same level of accuracy, across all configurations. Figure 5 shows the accuracy versus wall clock time for 20 clients across all datasets. CLIP performs the best on CIFAR10, by training more quickly and achieving a maximum accuracy of approximately 82%, 4% higher than the baseline. Random Dropout is a bit slower than CLIP, although it still achieves speedups over baseline. Ordered Dropout, however, suffers from a slowdown compared to the baseline, as its negative impact on accuracy per training round outweighs its round time improvements, leading to an overall slowdown compared to baseline. On Shakespeare, all dropout techniques perform significantly better than baseline, whereas on FEMNIST all dropout techniques perform similarly, narrowly outperforming the baseline.

End-to-End Training Time Speedup. Table 1 provides the end-to-end speedup in training time for each technique compared to the SecAgg baseline. We calculate the average speedup by averaging

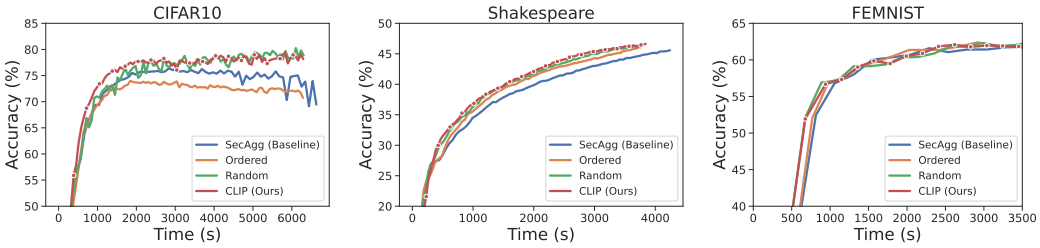


Figure 5: Accuracy versus training time for CLIP compared to Ordered dropout, Random dropout and SecAgg (baseline) after 100 rounds, with 20 clients of which 4 are stragglers.

the speedup in training time at three accuracy levels: the maximum accuracy (m), $m - 2.5\%$, and the $m - 5\%$. Across the datasets, CLIP achieves the highest speedup for 20 clients, achieving 30% for CIFAR10, 27.5% for Shakespeare, and 3% for FEMNIST. For CIFAR10, the speedups for CLIP are considerably higher than Random and Ordered, which have 12.6% speedup and 6.7% slowdown respectively. In both Shakespeare and FEMNIST, Random has a slightly lower speedup of 23.5% and 2.1% respectively, while Ordered has the lowest speedups with 17.1% and 0% respectively. All three dropout techniques have diminished benefits with FEMNIST as the reduced model accuracy (as shown in section 6.2) counteracts the gains in round time.

Table 1: Average Speedup as Number of Clients Vary from 5 to 50 (20% stragglers).

	CIFAR10			Shakespeare			FEMNIST	
Clients	50	20	5	50	20	5	20	5
CLIP	**21.1%**	**30.2%**	**33.0%**	**14.3%**	**27.5%**	**17.4%**	**3.0%**	**24.1%**
Random	12.6%	18.4%	11.7%	11.0%	23.5%	11.7%	2.1%	15.4%
Ordered	-6.7%	-6.7%	-4.1%	14.0%	17.1%	13.7%	0.0%	13.2%

Scalability. We also repeat the end-to-end training time speedup while varying the number of clients from 5 to 50 (maintaining 20% of the clients as stragglers). CLIP continues to outperform other techniques, achieving 21.1% to 33.0% speedup for CIFAR10 with 50 and 5 clients respectively, whereas Random has a 12.6% and 11.7% speedup respectively, while Ordered has slowdowns of 6.7% to 4.1%. These trends are similar to 20 clients for CIFAR10. In Shakespeare, CLIP outperforms other techniques with 14% to 17% speedups for 50 and 5 clients, while Random and Ordered are approximately at 11% and 14% speedup respectively for both number of clients. For FEMINST with 5 clients, CLIP has 24% speedup compared to other techniques, higher than 20 clients, since the communication bottleneck is more limited with fewer clients, allowing pruning to provide more pronounced benefits. We find that running 50 clients is infeasible for FEMNIST with Flower since the total message size per round exceeds Flower's maximum with ResNet-18 and $>$20 clients.

7 Limitations

CLIP's network-aware pruning addresses network stragglers using over-pruning while padding gradients for stragglers to maintain compatibility with secure aggregation. Future work could explore advanced protocols for securely aggregating non-uniform gradient sizes to improve efficiency. Additionally, CLIP's invariant neuron identification is designed for consistent client participation across training rounds. In dynamic environments with variable connectivity, adaptive pruning strategies could enhance robustness to intermittent participation and fluctuating bandwidth.

8 Conclusion

In this work, we propose CLIP, the first straggler mitigation strategy using pruning for secure federated learning. CLIP is a client-side invariant neuron pruning method, which preserves SecAgg's privacy, while reducing training times at negligible accuracy cost. With network-aware pruning, we also compensate for the slower communication of network stragglers. In evaluations with stragglers, CLIP speeds up secure FL by up to 34% while incurring negligible accuracy loss of up to 2.6%.

References

[1] Saar Barkai, Ido Hakimi, and Assaf Schuster. Gap aware mitigation of gradient staleness. In *International Conference on Learning Representations (ICLR)*, 2020.

[2] James Bell, K. A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly)logarithmic overhead. Cryptology ePrint Archive, Paper 2020/704, 2020.

[3] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Hei Li Kwing, Titouan Parcollet, Pedro PB de Gusmão, and Nicholas D Lane. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

[4] K. A. 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. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[5] 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 *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1175–1191, New York, NY, USA, 2017. Association for Computing Machinery.

[6] Sebastian Caldas, Jakub Konečny, H. Brendan McMahan, and Ameet Talwalkar. Expanding the reach of federated learning by reducing client resource requirements, 2019.

[7] Zheng Chai, Yujing Chen, Ali Anwar, Liang Zhao, Yue Cheng, and Huzefa Rangwala. Fedat: A high-performance and communication-efficient federated learning system with asynchronous tiers. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 1–16, 2021.

[8] Yujing Chen, Yue Ning, Martin Slawski, and Huzefa Rangwala. Asynchronous online federated learning for edge devices with non-iid data. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 15–24. IEEE, 2020.

[9] Enmao Diao, Jie Ding, and Vahid Tarokh. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. In *International Conference on Learning Representations (ICLR)*, 2021.

[10] Ahmed Roushdy Elkordy, Jiang Zhang, Yahya H Ezzeldin, Konstantinos Psounis, and Salman Avestimehr. How much privacy does federated learning with secure aggregation guarantee? *arXiv preprint arXiv:2208.02304*, 2022.

[11] Linux Foundation. Iproute2. `https://wiki.linuxfoundation.org/networking/iproute2`, 2023. Accessed: 2024-10-27.

[12] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients-how easy is it to break privacy in federated learning? *Advances in neural information processing systems*, 33:16937–16947, 2020.

[13] Samuel Horvath, Stefanos Laskaridis, Mario Almeida, Ilias Leontiadis, Stylianos I. Venieris, and Nicholas D. Lane. Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout, 2022.

[14] Yangsibo Huang, Samyak Gupta, Zhao Song, Kai Li, and Sanjeev Arora. Evaluating gradient inversion attacks and defenses in federated learning. *Advances in neural information processing systems*, 34:7232–7241, 2021.

[15] Yuang Jiang, Shiqiang Wang, Victor Valls, Bong Jun Ko, Wei-Han Lee, Kin K. Leung, and Leandros Tassiulas. Model pruning enables efficient federated learning on edge devices, 2022.

[16] Yuang Jiang, Shiqiang Wang, Víctor Valls, Bong Jun Ko, Wei-Han Lee, Kin K. Leung, and Leandros Tassiulas. Model pruning enables efficient federated learning on edge devices. *IEEE Transactions on Neural Networks and Learning Systems*, 34(12):10374–10386, 2023.

[17] Swanand Kadhe, Nived Rajaraman, O. Ozan Koyluoglu, and Kannan Ramchandran. Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning, 2020.

[18] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning, 2021.

[19] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine*, 37(3):50–60, 2020.

[20] Sue Marek. Mobile network experience report january 2020. Technical report, Opensignal, jan 2020.

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

[22] Martin Rapp, Ramin Khalili, Kilian Pfeiffer, and Jörg Henkel. Distreal: Distributed resource-aware learning in heterogeneous systems, 2022.

[23] Amirhossein Reisizadeh, Isidoros Tziotis, Hamed Hassani, Aryan Mokhtari, and Ramtin Pedarsani. Straggler-resilient federated learning: Leveraging the interplay between statistical accuracy and system heterogeneity, 2020.

[24] Reent Schlegel, Siddhartha Kumar, Eirik Rosnes, and Alexandre Graell i Amat. Codedpaddedfl and codedsecagg: Straggler mitigation and secure aggregation in federated learning, 2022.

[25] Jinhyun So, Chaoyang He, Chien-Sheng Yang, Songze Li, Qian Yu, Ramy E. Ali, Basak Guler, and Salman Avestimehr. Lightsecagg: a lightweight and versatile design for secure aggregation in federated learning, 2022.

[26] Irene Wang, Prashant J. Nair, and Divya Mahajan. Fluid: Mitigating stragglers in federated learning using invariant dropout, 2023.

[27] Di Wu, Rehmat Ullah, Paul Harvey, Peter Kilpatrick, Ivor Spence, and Blesson Varghese. Fedadapt: Adaptive offloading for iot devices in federated learning. *IEEE Internet of Things Journal*, 9(21):20889–20901, 2022.

[28] Robert Wyrzykowski. Mobile network experience report january 2024. Technical report, Opensignal, jan 2024.

[29] Hongxu Yin, Arun Mallya, Arash Vahdat, Jose M Alvarez, Jan Kautz, and Pavlo Molchanov. See through gradients: Image batch recovery via gradinversion. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16337–16346, 2021.

[30] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. idlg: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*, 2020.

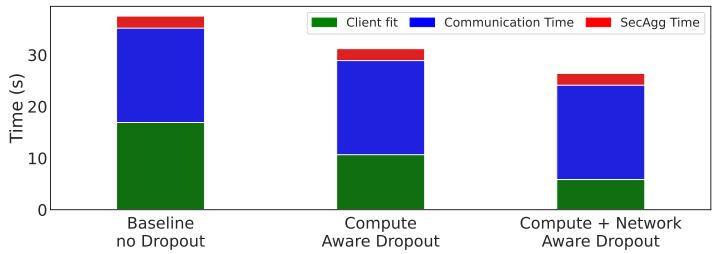
[31] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. *Advances in neural information processing systems*, 32, 2019.

A Accuracy Impact of CLIP

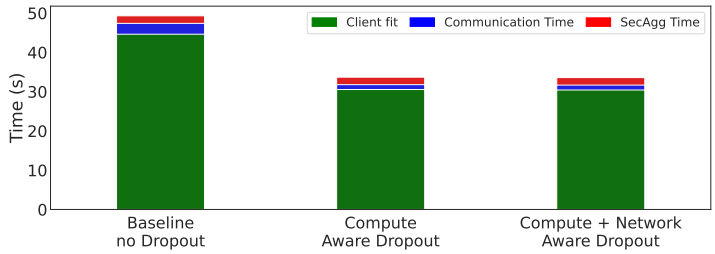
Table 2: Accuracy for CLIP (Pruning for Compute + Network Stragglers) with 20 clients (20% stragglers) after 100 training rounds, compared to Baseline and CLIP-C (Pruning for Compute Stragglers Only).

	CIFAR10	Shakespeare	FEMNIST
Baseline (No Pruning)	77.9%	50.4%	63.4%
CLIP-C (Pruning for Only Compute Stragglers)	81.4%	49.9%	63.8%
CLIP (Pruning for Compute + Network Stragglers)	79.2%	49.3%	60.8%

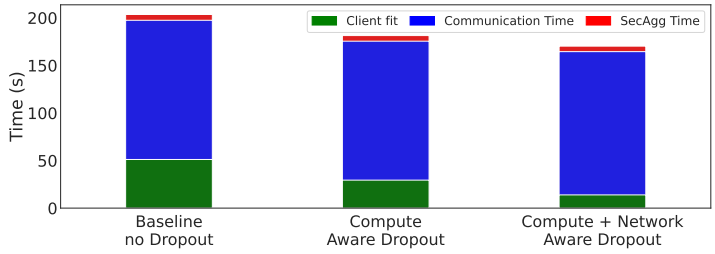
B Analysis of Training Round Times with CLIP



(a) CIFAR10 dataset with VGG-9 model



(b) Shakespeare dataset with LSTM model



(c) FEMNIST dataset with ResNet-18 model

Figure 6: Time per training round for FL with secure aggregation with and without CLIP for configurations (a), (b), and (c), with 20 clients of which 4 are stragglers. Compute aware dropout can reduce the round time by at least 10.9% (for FEMINIST) to at most 31.7% (Shakespeare). With network aware dropout, the round time reduces further by at least an additional 2.2% (Shakespeare) to at most an additional 12.7% (CIFAR10).