Every Vote Counts: Ranking-Based Training of Federated Learning to Resist Poisoning Attacks
Hamid Mozaffari, Virat Shejwalkar, and Amir Houmansadr, University of Massachusetts Amherst
https://www.usenix.org/conference/usenixsecurity23/presentation/mozaffari

This paper is included in the Proceedings of the 32nd USENIX Security Symposium.
August 9–11, 2023 • Anaheim, CA, USA
978-1-939133-37-3

Open access to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.
Every Vote Counts:
Ranking-Based Training of Federated Learning to Resist Poisoning Attacks

Hamid Mozaffari, Virat Shejwalkar, and Amir Houmansadr
University of Massachusetts Amherst
{hamid, vshejwalkar, amir}@cs.umass.edu

Abstract
Federated learning (FL) allows untrusted clients to collaboratively train a common machine learning model, called global model, without sharing their private/proprietary training data. However, FL is susceptible to poisoning by malicious clients who aim to hamper the accuracy of the global model by contributing malicious updates during FL’s training process.

We argue that the key factor to the success of poisoning attacks against existing FL systems is the large space of model updates available to the clients to choose from. To address this, we propose Federated Rank Learning (FRL). FRL reduces the space of client updates from model parameter updates (a continuous space of float numbers) in standard FL to the space of parameter rankings (a discrete space of integer values). To be able to train the global model using parameter ranks (instead of parameter weights), FRL leverage ideas from recent supermasks training mechanisms. Specifically, FRL clients rank the parameters of a randomly initialized neural network (provided by the server) based on their local training data, and the FRL server uses a voting mechanism to aggregate the parameter rankings submitted by the clients.

Intuitively, our voting-based aggregation mechanism prevents poisoning clients from making significant adversarial modifications to the global model, as each client will have a single vote! We demonstrate the robustness of FRL to poisoning through analytical proofs and experimentation, and we show its high communication efficiency.¹

1 Introduction
Federated Learning (FL) allows mutually untrusted clients (e.g., Android devices) to collaborate and train a common model, called global model, without sharing their private data. In a single FL round, a server (e.g., a Google server) broadcasts the current global model to a random subset of clients, the clients compute model updates using the global model and their private data, and share them with the server, server aggregates the updates using an aggregation rule (AGR) and updates the global model using the aggregate.

Robustness to poisoning attacks: Most of the distributed learning algorithms, including FedAvg [31] and FedProx [29], operate on mutually untrusted clients and server. This makes distributed learning susceptible to the threat of poisoning [10, 24, 39]. A poisoning adversary can either own or control a few of FL clients, called malicious clients, and instruct them to share malicious updates with the central server in order to reduce the performance of the global model. There are three approaches to poisoning FL: targeted [8, 41] attacks aim to reduce the utility of the global FL model on specific test inputs of adversary’s choice; untargeted [5, 18, 38] attacks aim to reduce the utility of global model on arbitrary test inputs; and backdoor [4, 42, 44] attacks aim to reduce the utility on test inputs that contain a specific signal called the trigger. In our work, we focus on the more severe threat of untargeted poisoning [39], which, unlike targeted and backdoor poisoning, affects the majority FL clients.

¹An extended version available at https://arxiv.org/abs/2110.04350
**High-level intuition of FL untargeted poisoning:*** Figure 1 shows how the poisoning adversary searches for malicious updates in the space of possible updates to maximize the distance between benign and malicious aggregates. When the server’s AGR is not robust, e.g., dimension-wise average \([31]\), there is no limitation on the adversary’s choices, so they can maximize their goal using a malicious update arbitrarily far from benign updates; (Figure 1-a)). Therefore, even a single malicious client can jeopardize the accuracy of the global model trained using FedAvg \([9]\). Current robust AGRs, such as Multi-krum \([9]\) or Trimmed-mean \([45]\) limit the space of acceptable updates, i.e., the safe zone shown in Figure 1-b). These robust AGRs only consider the updates that are in the safe zone and thereby reduce the adversary’s choices.

**Continuous versus discrete space of updates:** Figure 1-c) shows how our proposed defense (FRL, which is introduced next) limits the poisoning adversary’s choices of malicious updates by making the space of acceptable updates discrete. To the best of our knowledge, most of previous Byzantine robust FL algorithms use a continuous space of updates \((\in \mathbb{R}^d)\), as their frameworks are built on exchanging trained (32-bit) weight parameters. On the other hand, in our approach, the clients send their updates in the form of edge rankings, i.e., a permutation of integers \((\in [1, d])\) where \(d\) is the size of the network layer; more useful edges have higher ranks. In Figure 1-c), the black dots show the discrete space of acceptable client updates. For example, a network with 4 edges can have \(4!\) possible permutations of edge rankings starting from \([1,2,3,4]\) to \([4,3,2,1]\). On the other hand, in FL algorithms with a continuous space of updates (with or without a safe zone), the adversary’s choices are 4 weight parameters (each of 32 bits). Note that, sparsification of the space of acceptable updates is different from sparsification of model updates used in compression methods, e.g., TopK \([3]\), RandomK \([40]\) and Sketched-SGD \([23]\). In these methods, the FL client sends only a fraction of model updates instead of all of them, but each parameter still has a continuous space.

**Federated Rank Learning (FRL):** We present FRL, a novel FL algorithm that concurrently achieves the two goals of robustness against poisoning attacks and communication efficiency. FRL uses a novel learning paradigm called super-masks training \([35, 46]\) to create edge rankings, which, as we will show, allows FRL to reduce communication costs while achieving significantly stronger robustness. Specifically, in FRL, clients collaborate to find a subnetwork within a randomly initialized neural network which we call the supernetwork (this is in contrast to conventional FL where clients collaborate to train a neural network). The goal of training in FRL is to collaboratively rank the supernetwork’s edges based on the importance of each edge and find a global ranking. The global ranking can be converted to a supermask, which is a binary mask of 1’s and 0’s, that is superimposed on the random neural network (the supernetwork) to obtain the final subnetwork. For example, in our experiments, the final subnetwork is constructed using the top 50% of all edges. The subnetwork is then used for downstream tasks, e.g., image classification, hence it is equivalent to the global model in conventional FL. Note that in entire FRL training, weights of the supernetwork do not change.

More specifically, each FRL client computes the importance of the edges of the supernetwork based on their local data. The importance of the edges is represented as a ranking vector. Each FRL client will use the edge popup algorithm \([35]\) and their data to compute their local rankings (the edge popup algorithm aims at learning which edges in a supernetwork are more important over the other edges by minimizing the loss of the subnetwork on their local data). Each client then will send their local edge ranking to the server. Finally, the FRL server uses a novel voting mechanism to aggregate client rankings into a global ranking vector, which represents which edges of the random neural network (the supernetwork) will form the global subnetwork.

**Intuitions on FRL’s robustness:** In traditional FL algorithms, clients send large-dimension model updates \((\in \mathbb{R}^d)\) (real numbers) to the server, providing malicious clients significant flexibility in fabricating malicious updates. By contrast, FRL clients merely share the rankings of the edges of the supernetwork, i.e., integers \((\in [1, d])\), where \(d\) is the size of the supernetwork. This allows the FRL server to use a voting mechanism to aggregate client updates (i.e., ranks), therefore, providing high resistance to adversarial ranks submitted by poisoning clients, since each client can only cast a single vote! Therefore, as we will show both theoretically and empirically, FRL provides robustness by design and reduces the impact of untargeted poisoning attacks. Furthermore, unlike most existing robust FL frameworks, FRL does not require any knowledge about the percentages of malicious clients.

**Intuitions on FRL’s communication efficiency:** In FRL, the clients and the server communicate just the rankings of the edges in the supernetwork, i.e., a permutation of indices in \([1, d]\). Ranking vectors are generally significantly smaller than the global model. This, as we will show, significantly reduces the upload and download communication in FRL compared to Federated Averaging (FedAvg) \([31]\), where clients communicate model parameters, each of 32/64 bits.

**Evaluation results:** We experiment with three datasets in real-world heterogeneous FL settings and show that: (1) FRL achieves similar performance (e.g., model accuracy) as state-of-the-art FedAvg but with significantly reduced communication costs: for CIFAR10, the accuracy and communication cost per client are 85.4% and 40.2MB for FedAvg, while 85.3% and 26.2MB for FRL. (2) FRL is highly robust to poisoning attacks as compared to state-of-the-art robust aggregation algorithms: from 85.4% in the benign setting, 10% malicious clients reduce the accuracy of FL to 56.3% and 58.8% with Trimmed-Mean \([45]\) and Multi-Krum \([9]\), respectively, while FRL’s performance only decreases to 79.0%.

We also compare FRL with two communication reduc-
tion methods, SignSGD [7] and TopK [3] and show that FRL produces comparable communication costs and model accuracies. For instance, on CIFAR10, FRL, SignSGD, and TopK achieve 85.3%, 79.1%, and 82.1% test accuracy, respectively, when the corresponding communication costs (download and upload) are 26.2MB, 20.73MB, and 30.79MB. On the other hand, FRL offers significantly superior robustness. For instance, on CIFAR10, 10% (20%) malicious clients reduce the accuracy of SignSGD to 39.7% (10.0%), but FRL’s accuracy decreases to only 79.0% (69.5%). TopK is incompatible with existing robust aggregation algorithms, hence uses Average aggregation and is as vulnerable as FedAvg, especially in the real-world heterogeneous settings.

Although our primary focus is on defending against untargeted attacks, we also evaluate the FRL’s robustness against targeted poisoning for completeness. We consider state-of-the-art backdoor attacks [4, 42, 44], and we empirically show that FRL is more robust against these backdoor attacks compared to standard weight training FL. For example on CIFAR10, with 2% of malicious clients, semantic backdoor attacks [4] achieve 82.7% (84.4%) average backdoor (clean) task accuracy against FedAvg, while they achieve 49.2% (84.1%) backdoor (clean) task accuracy against FRL.

In summary, we propose a federated learning approach that is built on exchanging rankings instead of parameter weights, and we show a ranking-based FL is more robust to both untargeted and targeted poisoning attacks. Our key contributions are as follows:

• We show that sparsifying the space of FL clients’ updates can improve robustness of FL to poisoning. Because, it significantly reduces the space for attackers to search for malicious updates and enables more robust voting-based aggregations. Building on this hypothesis, we design Federated Rank Learning (FRL), a novel FL system in which clients collaboratively train a global model by ranking the importance of the edges of a random network based on their local data.

• We evaluate FRL on three benchmark datasets including MNIST, CIFAR10, FEMNIST where we split them in heterogeneous fashion among a large number of users, i.e., among 1000, 1000, 3400 users respectively. We show that FRL provides more robustness and competitive communication efficiency compared to state-of-the-art AGRs and compression techniques.

• We obtain theoretical robustness bounds for FRL, showing its strong robustness without any knowledge of the number of malicious clients.

2 Related works

Supermask learning: Modern neural networks have a very large number of parameters. These networks are generally overparameterized [16, 17, 27, 28], i.e., they have more parameters than they need to perform a particular task, e.g., classification. The lottery ticket hypothesis [19] states that a fully-trained neural network, i.e., supernetwork, contains sparse subnetworks, i.e., subsets of all neurons in supernetwork, which can be trained from scratch (i.e., by training same initialized weights of the subnetwork) and achieve performances close to the fully trained supernetwork. The lottery ticket hypothesis allows for massive reductions in the sizes of neural networks. Ramanujan et al. [35] offer a complementary conjecture that an overparameterized neural network with randomly initialized weights contains subnetworks which perform as good as the fully trained network.

Poisoning attacks and defenses for federated learning (FL): FL involves mutually untrusting clients. Hence, a poisoning adversary may own or compromise some of the FL clients, called malicious clients, with the goal of mounting a targeted or untargeted poisoning attack. In a targeted attack [8, 41], the goal is to reduce the utility of the model on specific test inputs, while in the untargeted attack [5, 18, 33, 38], the goal is to reduce the utility for all (or most) test inputs. It is shown [9] that even a single malicious client can mount an effective untargeted attack on FedAvg.

In order to make FL robust to the presence of such malicious clients, the literature has designed various robust aggregation rules (AGR) [9, 13, 32, 45], which aim to remove or attenuate the updates that are more likely to be malicious according to some criterion. For instance, Multi-krum [9] repeatedly removes updates that are far from the geometric median of all the updates, and Trimmed-mean [45] removes the largest and smallest values of each update dimension and calculates the mean of the remaining values. Unfortunately, these robust AGRs are not very effective in non-convex FL settings and multiple works have demonstrated strong targeted [8, 42] and untargeted attacks [18, 38] on them.

Communication cost of FL: In many real-world FL applications, it is essential to minimize the communication between FL server and clients. Especially in cross-device FL, the clients (e.g., mobile phones and wearable devices) have limited resources and communication can be a major bottleneck. There are two major types of communication reduction methods: (1) Quantization methods reduce the resolution of (i.e., number of bits used to represent) each dimension of a client update. For instance, SignSGD [7] uses the sign (1 bit) of each dimension of model updates. (2) Sparsification methods propose to use only a subset of all the update dimensions. For instance, in TopK [3], only the largest K% update dimensions are sent to the server in each FL round. We note that, communication reduction methods primarily focus on and succeed at reducing upload communication (client → server), but they use the entire model in download communication (server → client).
3 Preliminaries

3.1 Federated learning

In FL [24, 25, 31], N clients collaborate to train a global model without directly sharing their data. In round \( t \), the service provider (server) selects \( n \) out of \( N \) total clients and sends them the most recent global model \( \theta^{\text{v}} \). Each client trains a local model for \( E \) local epochs on their data starting from the \( \theta^{v} \) using stochastic gradient descent (SGD). Then the client sends back the calculated gradients (\( \nabla L \)) to the server. The server then aggregates the collected gradients and decides the global model for the next round. FL can be either cross-device or cross-silo [24]. In cross-device FL, \( N \) is large (from few thousands to billions) and only a small fraction of clients is chosen in each FL training round, i.e., \( n \ll N \). By contrast, in cross-silo FL, \( N \) is moderate (up to 100) and all clients are chosen in each round, i.e., \( n = N \). In this work, we evaluate the performance of FRL and other FL baselines for cross-device FL under realistic production FL settings.

3.2 Edge-popup algorithm

The edge-popup (EP) algorithm [35] is an optimization to find supermasks within a large, randomly initialized neural network, i.e., called supernetwork, with performances close to the fully trained supernet. EP algorithm does not train the weights (\( \theta^{v} \)) of the network, instead only decides the set of edges to keep and removes (pops) the rest of the edges. Specifically, EP algorithm assigns a positive score to each of the edges in the supernet (\( \theta^{v} \)). On forward pass, it selects top \( k\% \) edges with highest scores, where \( k \) is the percentage of the total number of edges in the supernet that will remain in the final subnetwork. On the backward pass, it updates the scores with the straight-through gradient estimator [6].

Algorithm 1 presents EP algorithm. Suppose in a fully connected neural network, there are \( L \) layers and layer \( \ell \in [1, L] \) has \( n_{\ell} \) neurons, denoted by \( V_{\ell} = \{V_{\ell}^{1}, \ldots, V_{\ell}^{n_{\ell}}\} \). If \( I_{\ell} \) and \( Z_{\ell} \) denote the input and output for neuron \( v \) respectively, then the input of the node \( v \) is the weighted sum of all nodes in previous layer, i.e., \( I_{v} = \sum_{\ell \in V^{\ell-1}} W_{\ell v} Z_{\ell} \). Here, \( W_{\ell v} \) is the weight of the edge connecting \( u \) to \( v \). Edge-popup algorithm tries to find subnetwork \( E \), so the input for neuron \( v \) would be: \( I_{v} = \sum_{(u,v) \in E} W_{uv} Z_{u} \).

**Updating scores.** Consider an edge \( W_{uv} \) that connects two neurons \( u \) and \( v \), \( W_{uv} \), be the weight of \( E_{uv} \), and \( s_{uv} \) be the score assigned to the edge \( E_{uv} \) by Edge-popup algorithm. Then the edge-popup algorithm removes edge \( E_{uv} \) from the supernet if its score \( s_{uv} \) is not high enough. Each iteration of supermask training updates the scores of all edges such that, if having an edge \( E_{uv} \) in subnetwork reduces loss (e.g., cross-entropy loss) over training data, the score \( s_{uv} \) increases.

**Algorithm 1.** Edge-popup (EP) algorithm

\[
\begin{align*}
1: & \text{Input: number of local epochs } E, \text{ training data } D, \text{ initial weights } \theta^{v} \text{ and scores } \theta^{s}, \text{ subnetwork size } k\%, \text{ learning rate } \eta \\
2: & \text{for } e \in [E] \text{ do} \\
3: & \ B \leftarrow \text{Split } D \text{ in } B \text{ batches} \\
4: & \text{for batch } b \in [B] \text{ do} \\
5: & \theta^{s} = \text{EP FORWARD}(\theta^{v}, \theta^{s}, k, b) \\
6: & \theta^{v} = \theta^{v} - \eta \nabla L(\theta^{v}, b) \\
7: & \end{for} \\
8: & \text{end for} \\
9: & \text{return } \theta^{v} \\
10: & \text{function } \text{EP FORWARD}(\theta^{v}, \theta^{s}, k, b) \\
11: & \text{Initialize random scores and weights using } SEED \\
12: & \text{return } \theta^{v} \\
\end{align*}
\]

The algorithm selects top \( k\% \) edges (i.e., finds a subnetwork with sparsity of \( k\% \)) with highest scores, so \( I_{v} \) reduces to \( I_{v} = \sum_{\ell \in V^{\ell-1}} W_{\ell v} h_{\ell}(s_{uv}) \) where \( h_{\ell}(.) \) returns 1 if the edge exists in top-\( k\% \) highest score edges and 0 otherwise. Because of existence of \( h_{\ell}(.) \), which is not differentiable, it is impossible to compute the gradient of loss with respect to \( s_{uv} \). Recall that, the Edge-popup algorithm use straight-through gradient estimator [6] to compute gradients. In this approach, \( h(\cdot) \) will be treated as the identity in the backward pass meaning that the upstream gradient (i.e., \( \frac{dL}{dz_{u}} \)) goes straight through \( h(\cdot) \).

Now using chain rule, we can derive \( \frac{dL}{dz_{u}} = \frac{dL}{ds_{uv}} \frac{ds_{uv}}{dz_{u}} \) where \( L \) is the loss to minimize. Then we can SGD with step size \( \eta \) to update scores as \( s_{uv} \leftarrow s_{uv} - \eta \frac{dL}{ds_{uv}} \).

4 Our proposal: Federated Rank Learning

**Algorithm 2.** Federated Ranking Learning (FRL)

\[
\begin{align*}
1: & \text{Input: number of rounds } T, \text{ number of local epochs } E, \text{ number of users per round } n, \text{ seed SEED, learning rate } \eta, \text{ subnetwork size } k\% \\
2: & \text{Server: Initialization} \\
3: & \theta^{v}, \theta^{s} \leftarrow \text{Initialize random scores and weights using SEED} \\
4: & R_{u}^{E} \leftarrow \text{ArgSort}(\theta^{v}) \triangleright \text{Sort the initial scores and obtain initial rankings} \\
5: & \text{for } t \in [1, T] \text{ do} \\
6: & \text{U} \leftarrow \text{set of } n \text{ randomly selected clients out of } N \text{ total clients} \\
7: & \text{for } u \in \text{U} \text{ do} \\
8: & \text{Clients: Calculating the ranks} \\
9: & \theta^{v}, \theta^{s} \leftarrow \text{Initialize scores and weights using SEED} \\
10: & \text{set } \text{score threshold } \theta^{s}(R_{u}^{E}) \triangleright \text{sort the scores based on the global ranking} \\
11: & S \leftarrow \text{Edge-PopUp}(E, D, R_{u}^{E}, \theta^{v}, \theta^{s}, \theta^{v}, k, \eta, \text{ Client u uses Algorithm1 to train a supermask on its local training data)} \\
12: & \text{end for} \\
13: & R_{u}^{E} \leftarrow \text{ArgSort}(S) \triangleright \text{Ranking of the client} \\
14: & \text{Server: Majority Vote} \\
15: & \text{set } \text{vote threshold } \theta^{s}(R_{u}^{E}) \triangleright \text{vote on the scores based on the global ranking} \\
16: & \text{end for} \\
17: & \text{function } \text{vote}(R_{u}^{E}) \triangleright \text{Majority vote aggregation} \\
18: & V \leftarrow \text{SUM}(\text{ArgSort}(R_{u}^{E})) \triangleright \text{Vote} \\
19: & \text{return } \text{ArgSort}(A) \\
\end{align*}
\]

In this section, we provide the design of our federated...
rank learning (FRL) algorithm (Algorithm 2). FRL clients collaborate (without sharing their data) to find a subnetwork within a randomly initialized, untrained supernetwork, with scores $\theta'$ and weights $\theta^w$. In each round, FRL first finds a unanimous (global) ranking of the supernetwork edges and then uses the subnetwork of the top ranked edges as the global model.

The objective of FRL is to find a global ranking $R_g$ and convert it to a global binary mask, $m$, such that resulting subnetwork, $\theta^w \odot m$, minimizes the average loss of all clients. FRL optimization can be formalized as follows:

$$\min F(\theta^w, R_g) = \min_{R_g} \sum_{i=1}^{N} \lambda_i L_i(\theta^w \odot m)$$

subject to $m[R_g < k] = 0$ and $m[R_g \geq k] = 1$ (1)

where $N$ is the total number of FRL clients, $L_i$ is the loss function for the $i^{th}$ client, $\lambda_i$ is the importance, e.g., weight, of the $i^{th}$ client; we use $\lambda_i = \frac{1}{N}$, i.e., all clients have the same weight. $m$ is the final binary mask, where edges with top $k$ ranks (layer-wise) get ‘1’ while others get ‘0’. We use $m$ to compute final global model by superimposing $m$ on $\theta$, i.e., we use the subnetwork $\theta \odot m$ as the final global model.

In Figure 2, we demonstrate a single FRL round using a supernetwork with six edges $e_i \in [0, 5]$ and three clients $C_i \in [1, 3]$ who aim to find a subnetwork of size $k=50\%$ of the original supernetwork.

4.1 Server: Initialization (only for round $t = 1$)

In the first round, the FRL server chooses a random seed $\text{SEED}$ to generate initial random weights $θ^w$ and scores $θ'$ for the global supernetwork $θ$; note that, $θ^w$, $θ'$, and $\text{SEED}$ remain constant during the entire FRL training. Next, the FRL server shares $\text{SEED}$ with FRL clients, who can then locally reconstruct the initial weights $θ^w$ and scores $θ'$ using $\text{SEED}$. Figure 2-(1) depicts this step.

Recall that, the goal of FRL training is to find the most important edges in $θ'$ without changing the weights. Unless specified otherwise, both server and clients use the Signed Kaiming Constant algorithm [35] to generate random weights and the Kaiming Uniform algorithm [21] to generate random scores. However, in Section 7.5.2, we also explore the impacts of different weight initialization algorithms on the performance of FRL. We use the same seed to initialize weights and scores.

At the beginning, the FRL server finds the global rankings of the initial random scores (Algorithm 2 line 4), i.e., $R^i_g = \text{ARGSORT}(θ')$. We define rankings of a vector as the indices of elements of vector when the vector is sorted from low to high, which is computed using $\text{ARGSORT}$ function.

4.2 Clients: Calculating the ranks (for each round $t$)

In the $t^{th}$ round, FRL server randomly selects $n$ clients among total $N$ clients, and shares the global rankings $R^i_g$ with them. Each of the selected $n$ clients locally reconstructs the weights $θ^w$’s and scores $θ'$’s using $\text{SEED}$ (Algorithm 2 line 9). Then, each FRL client reorders the random scores based on the global rankings, $R^i_g$ (Algorithm 2 line 10); we depict this in Figure 2-(2a).

Next, each of the $n$ clients uses reordered $θ'$ and finds a subnetwork within $θ'$ using Algorithm 1; to find a subnetwork, they use their local data and $E$ local epochs (Algorithm 2 line 11). Note that, each iteration of Algorithm 1 updates the scores $S$ starting from $θ'$. Then client $u$ computes their local rankings $R^u_g$ using the final updated scores ($S$) and $\text{ARGSORT}()$, and sends $R^u_g$ to the server. Figure 2-(2a) shows how each of the selected $n$ clients reorders the random scores using global rankings. For instance, the initial global rankings for this round are $R^i_g = [2, 3, 0, 5, 1, 4]$, meaning that edge $e_1$ should get the highest score ($s_1 = 1.2$), and edge $e_2$ should get the lowest score ($s_2 = 0.2$).

Figure 2-(2b) shows, for each client, the scores and rankings they obtained after finding their local subnetwork. For example, rankings of client $C_1$ are $R^1_g = [4, 0, 2, 3, 5, 1]$, i.e., $e_4$ is the least important and $e_1$ is the most important edge for $C_1$. Considering desired subnetwork size to be $50\%$, $C_1$ uses
Malicious Client Executes: reputations of each layer. Hence, FRL significantly reduces the space of the most important edge. Consequently, \( V(\cdot) \) function assigns reputation=0 to edge \( e_4 \), reputation=1 to \( e_0 \), reputation=2 to \( e_2 \), and so on. In other words, for rankings \( R_u^t \) of \( C_u \) and edge \( e_t \), \( V(\cdot) \) computes the reputation of \( e_t \) as its index in \( R_u^t \). Finally, \( V(\cdot) \) computes the total reputation of \( e_t \) as the sum of reputations from each of the local rankings. In Figure 2-\( 2b \), reputations of \( e_0 \) are 1 in \( R_1^t \), 1 in \( R_2^t \), and 0 in \( R_3^t \), hence, the total reputation of \( e_0 \) is 2. We depict this in Figure 2-\( 2g \); here, the final total reputations for edges \( e_t \) of \( u \in \{0,5\} \) are \( A = [2, 12, 3, 11, 8, 9] \). Finally, the server computes global rankings \( R_t^{k+1} \) to use for round \( t + 1 \) by sorting the final total reputations of all edges, i.e., \( R_t^{k+1} = \text{ARGSORT}(A) \).

Note that, all FRL operations that involve sorting, reordering, and voting are performed in a layer-wise manner. For instance, the server computes global rankings \( R_u^t \) in round \( t \) for each layer separately, and consequently, the clients selected in round \( t \) reorder their local randomly generated scores \( \theta^u \) for each layer separately.

5 Robustness of FRL to poisoning

FRL and FL are distributed learning algorithms with mutually untrusting clients. Hence, a poisoning adversary may own or compromise some of FRL (FL) clients, called malicious clients, and mount a targeted or untargeted poisoning attack. As discussed in Section 1, we mainly focus on the more severe untargeted attacks and show that FRL is significantly more robust by design to such poisoning attacks. However, for completeness we also evaluate robustness of FRL against targeted attacks in Appendix C.

Intuition behind robustness of FRL: Existing FL algorithms, including robust algorithms, are shown to be vulnerable to various poisoning attacks [39]. One of the key reasons behind the susceptibility of existing algorithms is that their model updates can have a large continuous space of values. For instance, to manipulate vanilla FedAvg, malicious clients send very large updates [9], and to manipulate Multi-krum and Trimmed-mean, [18, 38] propose to perturb a benign update in a specific malicious direction. On the other hand, in FRL, clients must send a permutation of indices \( \in [1, n_t] \) for each layer. Hence, FRL significantly reduces the space of the possible malicious updates that an adversary can craft. Majority voting in FRL further reduces the chances of successful attack. Intuitively, this makes FRL design robust to poisoning attacks. Below, we make this intuition more concrete.

The worst-case untargeted poisoning attack on FRL: Here, the poisoning adversary aims to reduce the accuracy of the final global FRL subnetwork on most test inputs. To achieve this, the adversary should replace the high ranked edges with low ranked edges in the final subnetwork. For the worst-case analysis of FRL, we assume a very strong adversary (i.e., threat model): 1) each of the malicious clients has some data from benign distribution; 2) malicious clients know the entire FRL algorithm and its parameters; 3) malicious clients can collude. Under this threat model we design a worst case attack on FRL (Algorithm 3), which executes as follows: First, malicious clients compute rankings on their benign data and use \( V(\cdot) \) algorithm to compute an aggregate rankings. Finally, each of the malicious clients uses the reverse of the aggregate rankings to share with the FRL server in given round. The adversary should invert the rankings layer-wise as the FRL server will aggregate the local rankings per layer too, and it is not possible to mount a model-wise attack.

Algorithm 3 FRL Poisoning

```
1: Input: number of malicious clients \( M \), number of malicious local epochs \( E^i \), seed \( \text{SEED} \), global ranking \( R_\theta \), learning rate \( \eta \), subnetwork size \( k\% \)
2: function MALICIOUSUPDATE(M, \text{SEED}, R_\theta, E^i, \eta, k%):  
3: for \( m \in [M] \) do  
4: Malicious Client Executes:  
5: \( \theta^m, \theta^u \leftarrow \text{Initialize random scores and weights using SEED} \)
6: \( \theta^u[R_\theta] \leftarrow \text{SORT}(\theta^u) \)
7: \( S \leftarrow \text{Edge-PopUp}(E^i, \partial \theta^u, \theta^u, \eta, k\%) \)
8: \( R_{\text{mu}} \leftarrow \text{ARGSORT}(S) \) \( \triangleright \) Ranking of the malicious client  
9: end for  
10: Aggregation:  
11: \( R_u \leftarrow V(\theta^u[R_{\text{mu}}]) \) \( \triangleright \) Majority vote aggregation  
12: return \( \text{REVERSE}(R_u) \) \( \triangleright \) reverse the ranking  
13: end function
```
the clients have $e_i$ (a good edge) in their local subnetworks. In other words, the rankings that each client sends to the server is just a bit mask showing that each edge should or should not be in the final subnetwork. The server makes a majority vote on the bit masks, and if an edge has more than half votes, it will be in the global subnetwork. Our VOTE(.) mechanism has more strict robustness criterion, as it uses more nuanced reputations of edges instead of bit masks. Hence, the upper bound on failure probability in this section also applies to the FRL VOTE(.) function.

The probability that our voting system fails is the probability that more than half of the votes do not include $e_i$ in their subnetworks. The upper bound on the probability of failure would be $1/2 \sqrt{\frac{np(1-p)}{(n-p+\alpha(1-2p)-1/2)^2}}$, where $n$ is the number of clients being processed, $p$ is the probability that a benign client puts $e_i$ in their top $k$ ranks, and $\alpha$ is the fraction of malicious clients. We defer the detailed proof to Appendix A. Figure 3 shows the upper bound on the failure probability of VOTE(.) for different values of $\alpha$ and $p$. As can be seen, the higher the probability $p$, the higher the robustness of FRL.

6 Communication efficiency of FRL

In FL, and especially in the cross-device setting, clients have limited communication bandwidth. Hence, FL algorithms must be communication efficient. We discuss here the communication cost of FRL algorithm. In the first round, the FRL server only sends one seed of 32 bits to all the FRL clients, so they can construct the random weights ($\theta^0$) and scores ($\theta^t$). In a round $t$, each of selected FRL clients receives the global rankings $R^t_i$ and sends back their local rankings $R^t_u$. The rankings are a permutation of the indices of the edges in each layer, i.e., of $[0, n_t - 1]$ if $\ell \in [L]$ where $L$ is the number of layers and $n_t$ is the number of parameters in $\ell$th layer.

![Figure 3: Upper bound on the failure probability of VOTE(.) function in FRL. $\alpha$ is the percentages of malicious clients and $p$ is the probability that a benign client puts a good edge in its top $k$ ranks.](image)

![Figure 4: Upload (U) and download (D) Communication cost analysis. The download cost (D) of all SFRLs would be the same as FRL. Download communication cost of SignSGD would be the same as FedAvg too.](image)

We use the naive approach to communicate layer-wise rankings, where each FRL client exchanges a total of $\sum_{\ell \in [L]} n_t \times \log(n_t)$ bits. Because, for the layer $\ell$, the client receives and sends $n_t$ ranks where each one is encoded with $\log(n_t)$ bits. On the other hand, a client exchanges $\sum_{\ell \in [L]} n_t \times 32$ bits in FedAvg, when 32 bits are used to represent each of $n_t$ weights in layer $\ell$. In Section 7.3, we measure the performance and communication cost of FRL with other existing FL compressors SignSGD [7] and TopK [2,3].

Sparse-FRL: Here, we propose Sparse-FRL, a simple extension of FRL to further reduce the communication cost. In Sparse-FRL, a client sends only the most important ranks of their local rankings to the server for aggregation. For instance, in Figure 2, client $C_1$ sends $R^t_i = [4, 0, 2, 3, 5, 1]$ in case of FRL. But in sparse-FRL, with sparsity set to 50%, client $C_1$ sends just the top 3 rankings, i.e., $R^t_i = [3, 5, 1]$. For each client, the sparse-FRL server assumes 0 reputation for all of the edges not included in the client’s rankings, i.e., in Figure 2, sparse-FRL server will assign reputation=0 for edges $e_4$, $e_0$, and $e_2$. Then the server uses VOTE(.) to compute total reputations of all edges and then sort them to obtain the final aggregate global rankings, i.e., $R^t_s = [3, 5, 1, 0, 0, 0]$, to use for subsequent rounds. We observe in our experiments, that sparse-FRL performs very close to FRL, even with sparsity as low as 10%, while also significantly reducing the communication cost.

Lower-bound of communication cost of FRL: Since the FRL clients send and receive layer-wise rankings of indices, i.e., integers $i \in [0, n_t - 1]$, for layer $\ell$, there are $n_t!$ possible permutations for layer $\ell \in [L]$. If we use the best possible compression method in FRL, an FRL client needs to send and receive $\sum_{\ell \in [L]} \log(n_t!)$ bits. Therefore, the download and upload bandwidth for each FRL client would be $\sum_{\ell \in [L]} \log(n_t! + n_t \cdot (n_t - 1) \cdot ... \cdot 2 \cdot 1) = \sum_{\ell \in [L]} \sum_{i=1}^{n_t} \log(i)$ bits. Please note that in our experiment, FRL clients send and receive the rankings without any further compression, and
We use MNIST, CIFAR10, and FEMNIST datasets. Most we compare FRL utility and robustness to an extension of hyperparameters for FRL and other baselines (Section 7.1.2). We independently tune the parameters settings to Appendix B.1. We can improve communication cost of FRL to its lower-bound (FRL-LB in Figure 4). In this Figure, we can see that SFRL can provide competitive upload communication cost and lower download communication cost compared to SignSGD when the clients are sending only 10% of top local rankings (SFRL 10%).

7 Empirical evaluation

In this section, we extensively evaluate the utility, robustness, and communication cost of our FRL algorithm. This section is organized as follows: in Section 7.1 we discuss the experimental setup. In Section 7.2 we investigate the robustness and utility of FRL, followed by Section 7.3 in which we discuss the communication cost of FRL. Next, in Section 7.4, we compare FRL utility and robustness to an extension of edge-popup to FL. We conclude with Section 7.5, where we provide an ablation study on FRL. We have released the code for reproducibility. 2

7.1 Experimental setup

7.1.1 Datasets and their distribution

We use MNIST, CIFAR10, and FEMNIST datasets. Most real-world FL settings have heterogeneous client data, hence following previous works [22, 37], we distribute MNIST and CIFAR10 datasets among 1,000 clients in non-iid fashion using Dirichlet distribution with parameter \( \beta = 1 \). Note that, increasing \( \beta \) results in more iid datasets. FEMNIST is naturally distributed non-iid among 3,400 clients. We further split the datasets of each client into training (80%) and test (20%).

We run all the experiments for 2000 global rounds of FRL and FL, while selecting 25 clients in each round. At the end of the training, we calculate the test accuracy of all the clients on the final global model, and we report the mean and standard deviation of all clients’ test accuracies in our experiments. Due to space restrictions, we defer further details of datasets, model architectures, and corresponding hyperparameters settings to Appendix B.1. We independently tune the hyperparameters for FRL and other baselines (Section 7.1.2).

We also evaluate the utility of FRL on a significantly more complicated task, Tiny-ImageNet [1] in Section 7.5.4.

7.1.2 Baseline FL algorithms

We compare the FRL with following FL baselines: Federated averaging: In non-adversarial FL settings, i.e., without any malicious clients, the dimension-wise Average (FedAvg) [25, 31] is an effective AGR. In fact, due to its efficiency, Average is the only AGR implemented by FL applications in practice [30, 34].

SignSGD: is a quantization method used in distributed learning to compress each dimension of gradient updates into 1 bit instead of 32 or 64 bits. To achieve this, in SignSGD [7] the clients only send the sign of their gradient updates to the server, and the server runs a majority vote on them. SignSGD is designed for distributed learning where all the clients participate in each round, so all the clients are aware of the most updated weight parameters of the global model. However, SignSGD only reduces upload communication (clients → server). But, does not reduce download communication (server → clients), i.e., to achieve good performance of the global model, the server sends all the weight parameters (each of 32 bits) to the newly selected clients in each round. Hence, SignSGD is as inefficient as FedAvg in download communication.

TopK: is a sparsification method used in distributed learning that transmits only a few dimensions of each model update to the server. In TopK [2, 3], the clients first sort the absolute values of their local model updates, and send the Top K% largest model update dimensions to the server for aggregation. TopK suffers from the same problem as SignSGD: for performance reasons, the server should send the entire updated model weights to the new selected clients.

Multi-krum: [9] proposed Multi-krum AGR as a modification to their own Krum AGR. Multi-krum selects an update using Krum and adds it to a selection set, \( S \). Multi-krum repeats this for the remaining updates (which remain after removing the update that Krum selects) until \( S \) has \( c \) updates such that \( n - c > 2m + 2 \), where \( n \) is the number of selected clients and \( m \) is the number of compromised clients in a given round. Finally, Multi-krum averages the updates in \( S \).

Trimmed-mean: Yin et al. [45] proposed Trimmed-mean that aggregates each dimension of input updates separately. It sorts the values of the \( j^{th} \)-dimension of all updates. Then it removes \( m \) (i.e., the number of compromised clients) of the largest and smallest values of that dimension, and computes the average of the rest of the values as its aggregate for the dimension \( j \).

7.2 Analyses of robustness to poisoning

We compare FRL with state-of-the-art robust aggregation rules (AGRs): Mkrum [9], and Trimmed-mean [45]. Table 1
Table 1: Comparing the robustness of various FL algorithms: FRL and Sparse-FRL (SFRL) (in **bold**) outperform the state-of-the-art robust AGRs and SignSGD against the strongest of untargeted poisoning attacks.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>AGR</th>
<th>0% malicious</th>
<th>10% malicious</th>
<th>20% malicious</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MNIST + LeNet</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000 clients</td>
<td>FedAvg</td>
<td>98.8 (3.2)</td>
<td>10.0 (10.0)</td>
<td>10.0 (10.0)</td>
</tr>
<tr>
<td></td>
<td>Trimmed-mean</td>
<td>98.8 (3.2)</td>
<td>95.1 (7.7)</td>
<td>87.6 (9.5)</td>
</tr>
<tr>
<td></td>
<td>Multi-krum</td>
<td>98.8 (3.2)</td>
<td>98.6 (3.3)</td>
<td>97.9 (4.1)</td>
</tr>
<tr>
<td></td>
<td>SignSGD</td>
<td>97.2 (4.6)</td>
<td>96.6 (5.0)</td>
<td>96.2 (5.6)</td>
</tr>
<tr>
<td></td>
<td>FRL</td>
<td><strong>98.8</strong> (3.1)</td>
<td><strong>98.8</strong> (3.1)</td>
<td><strong>98.7</strong> (3.3)</td>
</tr>
<tr>
<td></td>
<td>SFRL Top 50%</td>
<td>98.2 (3.8)</td>
<td>97.04 (4.4)</td>
<td>95.1 (7.8)</td>
</tr>
<tr>
<td><strong>CIFAR10 + Conv8</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000 clients</td>
<td>FedAvg</td>
<td>85.4 (11.2)</td>
<td>10.0 (10.1)</td>
<td>10.0 (10.1)</td>
</tr>
<tr>
<td></td>
<td>Trimmed-mean</td>
<td>84.9 (11.0)</td>
<td>56.3 (16.0)</td>
<td>20.5 (13.2)</td>
</tr>
<tr>
<td></td>
<td>Multi-krum</td>
<td>84.7 (11.3)</td>
<td>58.8 (15.8)</td>
<td>25.6 (14.4)</td>
</tr>
<tr>
<td></td>
<td>SignSGD</td>
<td>79.1 (12.8)</td>
<td>39.7 (15.9)</td>
<td>10.0 (10.1)</td>
</tr>
<tr>
<td></td>
<td>FRL</td>
<td>85.3 (11.3)</td>
<td>79.0 (12.4)</td>
<td>69.5 (14.8)</td>
</tr>
<tr>
<td></td>
<td>SFRL Top 50%</td>
<td>77.6 (13.0)</td>
<td>41.7 (15.4)</td>
<td>39.7 (15.2)</td>
</tr>
<tr>
<td><strong>FEMNIST + LeNet</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3400 clients</td>
<td>FedAvg</td>
<td>85.8 (10.2)</td>
<td>6.3 (5.8)</td>
<td>6.3 (5.8)</td>
</tr>
<tr>
<td></td>
<td>Trimmed-mean</td>
<td>85.2 (11.0)</td>
<td>72.7 (15.7)</td>
<td>56.2 (20.3)</td>
</tr>
<tr>
<td></td>
<td>Multi-krum</td>
<td>85.2 (10.9)</td>
<td>80.9 (12.2)</td>
<td>23.7 (12.8)</td>
</tr>
<tr>
<td></td>
<td>SignSGD</td>
<td>79.3 (12.4)</td>
<td>76.7 (13.2)</td>
<td>55.1 (14.9)</td>
</tr>
<tr>
<td></td>
<td>FRL</td>
<td>84.2 (10.7)</td>
<td>83.0 (10.9)</td>
<td>65.8 (17.8)</td>
</tr>
<tr>
<td></td>
<td>SFRL Top 50%</td>
<td>75.2 (12.7)</td>
<td>70.5 (14.4)</td>
<td>60.39 (14.8)</td>
</tr>
</tbody>
</table>

gives the performances of robust AGRs, SignSGD, and FRL with different percentages of malicious clients using attacks proposed by Shejwalkar et al. [38], Bernstein et al. [7], and Algorithm 3 respectively. To attack FRL, we choose 25 random malicious clients in each FRL round to generate the malicious update. Here, we make a rather impractical assumption in favor of the previous robust AGRs: we assume that the server knows the exact % of malicious clients in each FL round. Note that, FRL does not require this knowledge.

**FRL achieves higher robustness than state-of-the-art robust AGRs:** We note from Table 1 that, FRL is more robust to the presence of malicious clients who mount untargeted poisoning attacks, compared to Multi-Krum and Trimmed-mean, when percentages of malicious clients are 10% and 20%. For instance, on CIFAR10, 10% malicious clients can decrease the accuracy of FL models to 56.3% and 58.8% for Trimmed-mean and Multi-Krum respectively; 20% malicious clients can decrease the accuracy of the FL models to 25.6% for Trimmed-mean and Multi-Krum respectively. On the other hand, FRL performance decreases to 79.0% and 69.5% for 10% and 20% attacking ratio, respectively. We make similar observations for MNIST and FEMNIST datasets: for FEMNIST, 10% (20%) malicious clients reduce accuracy of the global model from 85.8% to 72.7% (56.2%) for Trimmed-mean and Multi-Krum respectively; 20% malicious clients can decrease the accuracy of the FL models to 25.6% for Trimmed-mean and Multi-Krum respectively. On the other hand, FRL performance decreases to 79.0% and 69.5% for 10% and 20% attacking ratio, respectively.

**FRL is more robust than SignSGD:** Next, we note from Table 1 that, FRL is more robust against untargeted poisoning attacks compared to SignSGD when percentages of malicious clients are 10% and 20%. For instance, on CIFAR10, 10% (20%) malicious clients can decrease the accuracy of SignSGD model to 39.8% (10.0%). On the other hand, FRL performance decreases to 79.0% and 69.5% for 10% and 20% attacking ratio respectively. We make similar observations for MNIST and FEMNIST datasets: for FEMNIST, 10% (20%) malicious clients reduce accuracy of the global model from 85.8% to 76.7% (55.1%) for SignSGD, while FRL accuracy decreases to 83.0% (65.8%).

**Sparse-FRL robustness:** We evaluate robustness of SFRL Top 50% against 10% and 20% malicious clients. As we can see from Table 1, by sending only top half of the local rankings, the accuracy goes from 85.3% (FRL) to 77.6% (SFRL). SFRL also can provide robustness to some extent, but adversary has more influence on the global ranking since half of the rankings are missing. For instance, on CIFAR10, 10% (20%) malicious clients can decrease the accuracy of global ranking to 41.7% (39.7%) from 77.6%. Also for FEMNIST, 10% (20%) malicious clients can decrease the accuracy of global ranking to 70.5% (60.39%) from 75.2%. We can see when malicious clients’ percentages are higher, SFRL can perform better compared to existing robust AGR.
Table 2: Comparing the accuracy and communication cost of FedAvg, SignSGD, TopK, FRL, and Sparse-FRL (SFRL) with different percentages of sparsity (in bold). Parentheses in the accuracy column show standard deviation of the accuracy.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Accuracy</th>
<th>Upload/Download (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MNIST + LeNet + 1000 clients</td>
<td></td>
</tr>
<tr>
<td>FedAvg</td>
<td>98.8 (3.1)</td>
<td>6.20/ 6.20</td>
</tr>
<tr>
<td>FRL</td>
<td>98.8 (3.2)</td>
<td>4.05/ 4.05</td>
</tr>
<tr>
<td>SFRL Top 50%</td>
<td>98.2 (3.8)</td>
<td>2.03/ 4.05</td>
</tr>
<tr>
<td>SFRL Top 10%</td>
<td>89.5 (9.2)</td>
<td>0.40/ 4.05</td>
</tr>
<tr>
<td>SignSGD</td>
<td>97.2 (4.6)</td>
<td>0.19/ 6.20</td>
</tr>
<tr>
<td>TopK 50%</td>
<td>98.8 (3.2)</td>
<td>3.29/ 6.20</td>
</tr>
<tr>
<td>TopK 10%</td>
<td>98.7 (3.2)</td>
<td>0.81/ 6.20</td>
</tr>
<tr>
<td>CIFAR10 + Conv8 + 1000 clients</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FedAvg</td>
<td>85.4 (11.2)</td>
<td>20.1/ 20.1</td>
</tr>
<tr>
<td>FRL</td>
<td>85.3 (11.3)</td>
<td>13.1/ 13.1</td>
</tr>
<tr>
<td>SFRL Top 50%</td>
<td>77.6 (13.0)</td>
<td>6.5/ 13.1</td>
</tr>
<tr>
<td>SFRL Top 10%</td>
<td>75.7 (14.4)</td>
<td>1.3/ 13.1</td>
</tr>
<tr>
<td>SignSGD</td>
<td>79.1 (13.6)</td>
<td>0.63/ 20.1</td>
</tr>
<tr>
<td>TopK 50%</td>
<td>82.1 (11.8)</td>
<td>10.09/ 20.1</td>
</tr>
<tr>
<td>TopK 10%</td>
<td>77.8 (13.0)</td>
<td>2.64/ 20.1</td>
</tr>
<tr>
<td>MNIST + LeNet + 3400 clients</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FedAvg</td>
<td>85.8 (10.2)</td>
<td>6.23/ 6.23</td>
</tr>
<tr>
<td>FRL</td>
<td>84.2 (10.7)</td>
<td>4.06/ 4.06</td>
</tr>
<tr>
<td>SFRL Top 50%</td>
<td>75.2 (12.7)</td>
<td>2.03/ 4.06</td>
</tr>
<tr>
<td>SFRL Top 10%</td>
<td>59.2 (15.0)</td>
<td>0.40/ 4.06</td>
</tr>
<tr>
<td>SignSGD</td>
<td>79.3 (12.9)</td>
<td>0.19/ 6.23</td>
</tr>
<tr>
<td>TopK 50%</td>
<td>85.7 (9.9)</td>
<td>3.37/ 6.23</td>
</tr>
<tr>
<td>TopK 10%</td>
<td>85.5 (10.0)</td>
<td>0.81/ 6.23</td>
</tr>
</tbody>
</table>

**FRL versus FedAvg and TopK:** We omit the results of non-robust aggregations, FedAvg and TopK, because even a single malicious client [9] can jeopardize their performances.

**Robustness of FRL against targeted poisoning:** Although our primary focus is on defending against untargeted attacks, for completeness, we also evaluate the FRL’s robustness against targeted poisoning, and especially against state-of-the-art backdoor attacks [4, 42, 44] in Appendix C. Backdoor attacks are successful in traditional continuous updates based FL (e.g., FedAvg), because in a given round, the adversary can scale its malicious update and cancel the effect of benign clients’ updates, thereby forcing the global model to be their malicious model [4]. However, by design, FRL is robust to such scaling as it uses rankings based updates where scaling is impossible.

However, we empirically show that FRL is more robust than tradition FL algorithms, even against the backdoor attacks that do not use scaling [4, 42]. For example, with 2% of malicious clients and non-iid distributed CIFAR10 data, semantic backdoor attacks [4] achieve 82.7% (84.4%) average backdoor (clean) task accuracy against FedAvg, while they achieve 49.2% (84.1%) backdoor (clean) task accuracy against FRL.

**7.3 Communication cost analysis**

In FRL, both clients and server communicate just the edge ranks instead of weight parameters. Thus, FRL reduces both upload and download communication cost. Table 2 illustrates the utility, i.e., the mean and standard deviation of all clients’ test accuracies and, communication cost of FRL and state-of-the-art quantization (i.e., SignSGD [7]) and sparsification (i.e., TopK [2, 3]) communication-reduction methods.

**FRL versus SignSGD:** SignSGD in FL reduces only the upload communication, but for efficiency reasons, the server sends all of the weight parameters (each of 32 bits) to the newly selected clients. Hence, SignSGD has very efficient upload communication, but very inefficient download communication. For instance, on CIFAR10, for both upload and download, FRL achieves 13.1MB each while SignSGD achieves 0.63MB and 20.1MB, respectively.

**FRL versus TopK:** We compare FRL and TopK where \( K \in \{10, 50\}\% \). FRL is more accurate than Topk for MNIST and CIFAR10: on CIFAR10, FRL accuracy is 85.3%, while TopK accuracies are 82.1% and 77.8% with \( K=50\% \) and \( K=10\% \), respectively. Similar to SignSGD, Topk more efficiently reduces upload communication, but has very high download communication. Therefore, the combined upload and download communication cost per client per round is 26.2MB for FRL and 30.79MB for TopK with \( K=50\% \); note that, even then TopK performs worse than FRL.

**Communication cost reduction due to Sparse-FRL (SFRL):** We now evaluate SFRL explained in Section 6. In SFRL with top 50% ranks, clients send the top 50% of their ranks to the server, which reduces the upload bandwidth consumption by half. Please note that the download cost of SFRL is the same as FRL since the FRL server should send all the global rankings to the selected clients in each round. We note from Table 2 that, by sending fewer ranks, SFRL reduces upload communication at a small cost of performance. For instance, on CIFAR10, SFRL with top 50% reduces the upload communication by 50% at the cost reducing accuracy from 85.4% to 77.6%.

**7.4 Comparison with naïve extension of edge-popup algorithm to FL**

As discussed in Section 4.2, prior works [36] on superrmask training and its following works [14, 43] do not consider rankings, and instead find subnetworks in randomly initialized networks by just training the scores, in a centralized training setting. Algorithm 4 shows the naïve extension of [36] to FL, where the clients train and exchange scores (32bits floats). Like FedAvg, these scores are from a continuous space of float numbers (as opposed to parameter ranks in FRL). At the end of each FL round, the server averages the local score updates to aggregate them and produces global scores (different from our majority vote aggregation). It is important to note that our majority vote aggregation only works on a set of ranking inputs.

Exchanging such scores is as vulnerable to poisoning as regular FL because they are not scale-free and discrete similar
Algorithm 4: Edge-popup based FL (EFL)

1: Input: number of rounds $T$, number of local epochs $E$, number of users $n$, seed $\text{SEED}$, learning rate $\eta$, subnetwork size $k$
2: Server: Initialization
3: $\theta^n_0$, $\theta^n_r$ ← Initialize random scores and weights using $\text{SEED}$
4: for $t \in [1, T]$ do
5: $U \leftarrow$ set of $n$ randomly selected clients out of $N$ total clients
6: for $u$ in $U$ do
7: Clients: Calculating the scores using EP
8: $\theta^u_0$ ← Initialize weights using $\text{SEED}$
9: $S_{\theta^u_0}^u$ ← Edge-PopUp($E$, $D^u_{\theta^u_0}$, $\theta^n_0$, $\theta^n_{t-1}$, $k$, $\eta$) $\triangleright$ Client $u$ uses Algorithm 1 to train a supermask starting from global scores $\theta^n_{t-1}$
10: end for
11: Server: Averaging the scores
12: $\theta^n_{t+1}$ ← AVG($S_{\theta^u_0}^u$)
13: end for

Table 3: Comparing the robustness of EFL, a naïve edge-popup based FL (Algorithm 4), with robustness of FRL.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Type of Non-IID</th>
<th>Metric</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>Dirichlet</td>
<td>Mean</td>
<td>FRL</td>
</tr>
<tr>
<td>LeNet</td>
<td>Distribution $\beta = 1$</td>
<td>98.8</td>
<td>98.8</td>
</tr>
<tr>
<td>MNIST</td>
<td>Randomly 2 classes assigned to each client</td>
<td>84.3</td>
<td>98.8</td>
</tr>
<tr>
<td>CIFAR10</td>
<td>Dirichlet</td>
<td>Mean</td>
<td>FRL</td>
</tr>
<tr>
<td>Conv8</td>
<td>Distribution $\beta = 1$</td>
<td>85.4</td>
<td>85.3</td>
</tr>
<tr>
<td>N=1000</td>
<td>Randomly 2 classes (assigned to each client)</td>
<td>33.3</td>
<td>85.3</td>
</tr>
</tbody>
</table>

Table 4: Comparing the performance of FRL and FedAvg in cross-device FL setting using two non-iid data distribution methods. We distribute data among 1000 clients with two methods described briefly below: please check Section 7.5.1 for more details.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Type of Non-IID</th>
<th>Metric</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>Dirichlet</td>
<td>Mean</td>
<td>FRL</td>
</tr>
<tr>
<td>LeNet</td>
<td>Distribution $\beta = 1$</td>
<td>98.8</td>
<td>98.8</td>
</tr>
<tr>
<td>MNIST</td>
<td>Randomly 2 classes assigned to each client</td>
<td>84.3</td>
<td>98.8</td>
</tr>
<tr>
<td>CIFAR10</td>
<td>Dirichlet</td>
<td>Mean</td>
<td>FRL</td>
</tr>
<tr>
<td>Conv8</td>
<td>Distribution $\beta = 1$</td>
<td>85.4</td>
<td>85.3</td>
</tr>
<tr>
<td>N=1000</td>
<td>Randomly 2 classes (assigned to each client)</td>
<td>33.3</td>
<td>85.3</td>
</tr>
</tbody>
</table>

7.5 Ablation study

In this section, we perform an extensive ablation study to understand performances of FRL under various settings. Specifically, we evaluate the performances of FRL while varying non-iid data distributions methods (Section 7.5.1), weight initialization algorithms (Section 7.5.2), sparsity (Section 7.5.3), and size of supernetwork (Section 7.5.4).

7.5.1 FRL under different heterogeneous data distribution methods

So far, we evaluated all of our experiments when the data is distributed non-iid using Dirichlet distribution with parameter $\beta = 1$. In this method of non-iid data distribution, all clients will get at least a few samples from each data class with non-zero probabilities that Dirichlet distribution generates. However, this non-iid data distribution need not represent all the practical FL settings. In fact, there may exist non-iid distributions that make training FL models more difficult. Therefore in this section, we consider a more difficult setting where the data distribution is more non-iid.

Assigning only two classes to each client: We experiment with the more extreme non-iid distribution considered by McMahan et al. [31]. More specifically, to distribute of MNIST and CIFAR10 data among 1000 clients, we sort all the (i.e., combined train and test) MNIST and CIFAR10 data according to their classes and then we partition them into 2000 shards. Hence, each shards of training MNIST has 30 images and each CIFAR10 shard has 25 images of a single class. Then we assign two random shards to each client resulting in each client having data from at most two classes. Therefore, in CIFAR10 experiments, each client has 50 training images, and 10 test images, and in MNIST experiments, each client has 60 training images and 10 test images. We only use this assignment in Section 7.5.1.

Table 4 shows the performances of FRL and FedAvg using...
different methods of non-iid assignment. We distribute the data between 1000 clients using: (I) Dirichlet distribution with $\beta = 1$ similar to [22, 37] and (II) the method described above from [31]. In Table 4, we note that FRL achieves similar performances as FedAvg for different heterogeneous data distribution methods. For instance, on CIFAR10, FedAvg and FRL achieve similar performances of 85.4% and 85.3% respectively when data is distributed according to (I). Similarly, when data is distributed according to (II), FedAvg and FRL achieve similar performances of 70.6% and 70.9%, respectively.

We make similar observations for MNIST as well: FedAvg achieves 98.8% and 98.4% for the two methods of data distribution respectively, while FRL achieves 98.8% and 98.3% accuracy.

### 7.5.2 FRL under different weight initializations

Table 5: Comparing the performance of FRL with different random weight initialization algorithms with the performance of vanilla FedAvg for cross-device setting. Using Signed Kaiming Constant ($U_K$) as weight initialization gives the best performance for all the datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Metric</th>
<th>Algorithm</th>
<th>FedAvg W_{init} ~</th>
<th>FRL $X_N$</th>
<th>$N_K$</th>
<th>$U_K$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST LeNet</td>
<td>Mean</td>
<td>-</td>
<td>98.8</td>
<td>96.6</td>
<td>98.7</td>
<td>98.8</td>
</tr>
<tr>
<td>N=1000</td>
<td>STD</td>
<td>3.1</td>
<td>5.2</td>
<td>3.2</td>
<td>3.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Min</td>
<td>75.0</td>
<td>57.1</td>
<td>75.0</td>
<td>75.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>CIFAR10 Conv8</td>
<td>Mean</td>
<td>-</td>
<td>85.4</td>
<td>63.6</td>
<td>82.0</td>
<td>85.3</td>
</tr>
<tr>
<td>N=1000</td>
<td>STD</td>
<td>11.2</td>
<td>15.6</td>
<td>11.9</td>
<td>11.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Min</td>
<td>33.3</td>
<td>0</td>
<td>0</td>
<td>33.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>FEMNIST LeNet</td>
<td>Mean</td>
<td>-</td>
<td>85.8</td>
<td>69.2</td>
<td>82.9</td>
<td>84.2</td>
</tr>
<tr>
<td>N=3400</td>
<td>STD</td>
<td>10.2</td>
<td>14.2</td>
<td>11.1</td>
<td>10.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Min</td>
<td>10.0</td>
<td>0</td>
<td>14.3</td>
<td>7.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

In FRL, the weight parameters are randomly initialized at the start and remain fixed throughout the training. An appropriate initialization is instrumental to the success of FRL, since the FRL clients are sending the local rankings of these edges; more important edges get higher ranks. They generate these rankings by feeding the subnetwork of top rank edges and calculating the gradient of the loss with respect to the scores, so distribution of these random weights has a high impact on the calculated loss. We use three different distribution for initializing the weight parameters as follows:

**Glorot Normal** [20] where we denote by $X_N$. Previous work [46] used this initialization to demonstrate that sub-networks of randomly weighted neural networks can achieve impressive performance.

**Kaiming Normal** [21] where we denote by $N_K$ defined as $N_K = N(0, \sqrt{2/n_{i-1}})$ where $N$ shows normal distribution. $n_i$ shows the number of parameters in the $i$th layer.

**Singed Kaiming Constant** [35] where all the weights are a constant $\sigma$ but they are assigned $\{-\sigma, +\sigma\}$ randomly. This constant, $\sigma$, is the standard deviation of Kaiming Constant. We show this initialization with $U_K$ as we are sampling from $\{-\sigma, +\sigma\}$ where $\sigma = \left(\sqrt{2/n_{i-1}}\right)$.

Table 5 shows the results of running FRL for three datasets under the three aforementioned initialization algorithms. We compare FRL with FedAvg and report the mean, standard deviation, minimum, and maximum of the accuracies for the clients’ accuracies in FRL and FedAvg at the end of training. As we can see under three different random initialization, using Signed Kaiming Constant ($U_K$) results in better performance. We note from Table 5 that FRL with Signed Kaiming Constant ($U_K$) initialization achieves performance very close to the performance of FedAvg.

Note that, since the FRL clients update scores in each round, unlike initialization of weights, initialization of scores does not have significant impact on the final global subnetwork search. Therefore, we do not explore different randomized initialization algorithms for scores and simply use Kaiming Uniform initialization for scores.

Ramanujan et al. [35] also considered these three initialization to find the best subnetwork in centralized machine learning setting. They also showed that using Signed Kaiming Constant gives the best supermasks. Our results align with their conclusions, hence we use Signed Kaiming Constant to initialize the weights and Kaiming Uniform to initialize the scores of the global supernetwork.

### 7.5.3 FRL with varying sizes of subnetworks

In FRL, each client uses Edge-popup (Algorithm 1) and their local data to find a local ranking by finding a subnetwork within a randomly initialized global network, which we call supernetwork. Edge-popup algorithm uses parameter $k$ which represents the % of all the edges in a supernetwork which will remain in the final subnetwork. For instance, $k = 50\%$ denotes that each client finds a subnetwork within a supernetwork that has half the number of edges as in the supernetwork.

Figure 5 illustrates how the performance of FRL varies with the sizes of local subnetworks that the clients share with the server. In other words, when we vary the sparsity $k$% of edge popup algorithm during local subnetwork search $k \in [10, 20, 30, 40, 50, 60, 70, 80, 90]$$\%$. Interestingly we note that, FRL performs the worst when clients use all ($k=100\%$) or none ($k=0\%$) of the edges. This is because, it is difficult to find a subnetwork with small number of edges. While using all of the edge essentially results in using a random neural
network. As we can see FRL with $k \in [40, 70] \%$, gives the best performances for all the three datasets. Hence, we set $k=50\%$ by default in our experiments.

7.5.4 FRL with larger networks

We already discussed the performance of the FRL for MNIST, CIFAR10, and FEMNIST datasets using moderately large neural networks with number of parameters ranging from 1.62M to 5.27M. In this section, we conduct experiments to demonstrate the effectiveness of FRL for very large networks: ResNet18 and ResNet34, with number of parameters 11.11M and 21.26M, respectively. Along with CIFAR10, we conduct experiments using a larger and significantly more challenging Tiny-ImageNet dataset. Table 7 shows the network architectures we use for these experiments. We have a batch normalization layer in these networks after each convolution layer. Following the setting of [14, 35, 43], we set the batch normalization to the non-affine mode, i.e., the scale and bias terms are set to $\gamma = 1$ and $\beta = 0$. We use the hyperparameters described in Section B.1 for all the models.

Tiny-ImageNet Table 6 shows the performance of FRL and FedAvg on Tiny-ImageNet for different network sizes, where ResNet18 and ResNet34 have 11.26 and 21.36 millions parameters respectively. From this table, we can see that FRL can achieve similar test accuracy as FedAvg: for example, by using ResNet18, FRL achieves 31.0% test accuracy while FedAvg achieves 30.8% test accuracy.

Table 6: FRL with larger networks for CIFAR10 and Tiny-ImageNet distributed over 1000 FL clients.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Network</th>
<th># of Params</th>
<th>FRL Acc (%)</th>
<th>FedAvg Acc (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR10 $N = 1000$</td>
<td>Conv8</td>
<td>5.27M</td>
<td>85.3</td>
<td>85.4</td>
</tr>
<tr>
<td></td>
<td>ResNet18</td>
<td>11.11M</td>
<td>89.3</td>
<td>89.9</td>
</tr>
<tr>
<td></td>
<td>ResNet34</td>
<td>21.26M</td>
<td>90.2</td>
<td>91.4</td>
</tr>
<tr>
<td>Tiny-ImageNet $N = 1000$</td>
<td>ResNet18</td>
<td>11.26M</td>
<td>31.0</td>
<td>30.8</td>
</tr>
<tr>
<td></td>
<td>ResNet34</td>
<td>21.36M</td>
<td>30.9</td>
<td>30.8</td>
</tr>
</tbody>
</table>


CIFAR10. In Table 6, we show the final test accuracy of different models, Conv8 with 5.27M parameters, ResNet18 with 11.11M parameters, and ResNet34 with 21.26M parameters. We can see that FRL can achieve similar test accuracies as FedAvg for different model sizes. For example, For ResNet18, FRL achieves 89.3% test accuracy while FedAvg achieves 89.9% test accuracy.

8 Conclusions

We designed a novel collaborative learning algorithm, called Federated Rank Learning (FRL), to address the issues of robustness to poisoning and communication efficiency in existing FL algorithms. We argue that a core reason for the susceptibility of existing FL algorithms to poisoning is the large continuous space of values in their model updates. Hence, in FRL, we use ranks of edges of a randomly initialized neural network contributed by collaborating clients to find a global ranking and then use a subnetwork based only on the top edges. Use of rankings in a fixed range restricts the space available to poisoning adversaries to craft malicious updates, and also allows FRL to reduce the communication cost. We show, both theoretically and empirically, that ranking based collaborative learning can effectively mitigate the robustness issue as well as reduce the communication costs involved.

Acknowledgements

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Agreement No. HR00112190125. Approved for public release; distribution is unlimited. This work was also supported by the NSF grant 2131910.
References


A Theoretical analysis of FRL’s robustness

In this section, we detail the proof of robustness of FRL. In other words, we prove an upper bound on the failure probability of robustness of FRL, i.e., the probability that a good edge will be removed from the final subnetwork when malicious clients mount the worst case attack. Following SignSGD [7], for this proof, we assume a simpler VOTE(.) function where
We also assume that the malicious clients do not collude.

Assume that edge $e_i$ is a good edge, i.e., having $e_i$ in the final subnetwork improves the performance of the final subnetwork. Let $Z$ be the random variable that represents the number of clients who vote for the edge $e_i$ to be in the final subnetwork, i.e., the number of clients whose local subnetwork of size $k\%$ of the entire supernetwork (Algorithm 2 line 11) contains $e_i$. Therefore, $Z \in [0, n]$ where $n$ is the number of clients being processed in a given FRL round.

Let $G$ and $B$ be the random variable that represent the number of benign and malicious clients that vote for edge $e_i$, respectively; the malicious clients inadvertently exclude the good edge $e_i$ in their local subnetwork based on their benign training data.

There are total of $\alpha n$ malicious clients, where $\alpha$ is the fraction of malicious clients that $B$ of them decides that $e_i$ is a bad edge and should not be removed. Each of the malicious clients computes the subnetwork on its own benign training data, so $B$ of them do not conclude that $e_i$ is a good edge. Hence, $Z = G + B$. We can say that $G$ and $B$ have binomial distribution, i.e., $G \sim \text{binomial}([1 - \alpha]n, p)$ and $B \sim \text{binomial}([\alpha n, 1 - p])$ where $p$ is the probability that a benign client has this edge in their local subnetwork and $\alpha$ is the fraction of malicious clients. Note that the probability that our voting in simplified FRL fails is $P[\text{failure}] = P[Z \leq \frac{n}{2}]$, i.e., when more than half of the clients vote against $e_i$, i.e., they do not include $e_i$ in their local subnetworks. We can find the mean and variance of $Z$ as follows:

$$E[Z] = (1 - \alpha)np + \alpha n(1 - p)$$  \hspace{1cm} (2)

$$\text{Var}[Z] = (1 - \alpha)np(1 - p) + \alpha np(1 - p) = np(1 - p)$$  \hspace{1cm} (3)

[12] provides an inequality where for a random variable $X$ with mean $\mu$ and variance $\sigma^2$ we have $P[|X - \mu| > \lambda] \leq \frac{1}{1 + \frac{\lambda^2}{\sigma^2}}$. Using this inequality, we can write:

$$P[Z \leq \frac{n}{2}] = P[E[Z] - Z \geq E[Z] - n/2] \leq \frac{1}{1 + \frac{(E[Z] - n/2)^2}{\text{Var}[Z]}}$$  \hspace{1cm} (4)

because $1 + x^2 \geq 2x$, we have:

$$P[Z \leq \frac{n}{2}] \leq \frac{\sqrt{\text{Var}[Z]}}{(E[Z] - n/2)^2}$$  \hspace{1cm} (5)

$$= \frac{1}{2\sqrt{np(1 - p)}} \left( \frac{np(1 - p)}{(n\alpha + \alpha(1 - 2p) - n/2)^2} \right)$$

$$= \frac{1}{2\sqrt{(n\alpha + \alpha(1 - 2p) - n/2)^2}}$$

What this means is that the probability that the simplified $\text{OTE}(.)$ fails is upper bounded as in (5). We show the effect of the different values of $\alpha$ and $p$ in Figure 3. We can see from Figure 3, if the benign clients can train better supermasks (better chance that a good edge ended in their subnetwork), the probability that the attackers succeed is lower (more robustness). $\text{OTE}(.)$ in FRL (Section 4.3) is more sophisticated and puts more constraints on the malicious clients, hence the about upper bound also applies to FRL.

## B Additional details on experimental setup

In this section, we provide the details of experimental setup that are missing from Section 7.

### B.1 Datasets and model architectures

We use four benchmark datasets widely used in prior works on federated learning robustness:

- **MNIST** is a 10-class class-balanced classification task with 70,000 grayscale images, each of size $28 \times 28$. We experiment with LeNet architecture given in Table 7. For local training in each FRL/FL round, each client uses 2 epochs. For training weights (experiments with FedAvg, SignSGD, TopK), we use SGD optimizer with learning rate of 0.01, momentum of 0.9, weight decay of 1e-4, and batch size 8. For training ranks (experiments with FRL), we use SGD with learning rate of 0.4, momentum 0.9, weight decay 1e-4, and batch size 8.

- **CIFAR10** [26] is a 10-class classification task with 60,000 RGB images (50,000 for training and 10,000 for testing), each of size $32 \times 32$. We experiment with a VGG-like architecture given in Table 7, which is identical to what [35] used. For local training in each FRL/FL round, each client uses 5 epochs. For training weights (experiments with FedAvg, SignSGD, TopK), we use SGD with learning rate of 0.1, momentum of 0.9, weight decay of 1e-4, and batch size of 8. For training ranks (experiments with FRL), we optimize SGD with learning rate of 0.4, momentum of 0.9, weight decay 1e-4, and batch size 8.

- **FEMNIST** [11, 15] is a character recognition classification task with 3,400 clients, 62 classes (52 for upper and lower case letters and 10 for digits), and 671,585 grayscale images. Each client has data of their own handwritten digits or letters. We use LeNet architecture given in Table 7. For local training in each FRL/FL round, each client uses 2 epochs. For training weights (experiments with FedAvg, SignSGD, TopK), we use SGD with learning rate of 0.15, momentum of 0.9, weight decay of 1e-4, and batch size of 10. For training ranks (experiments with FRL), we optimize SGD with learning rate of 0.2, momentum of 0.9, weight decay of 1e-4, and batch size of 10.

- **Tiny-ImageNet** [1] contains 100K, 10k, 10k training, validation and test images respectively for 200 classes downsized to 64 x 64 colored images, so each class has 500 training, and
Table 7: In our experiments, we use the following, state-of-the-art model architectures from [14, 35, 43].

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Layer Name</th>
<th>Number of parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeNet [43]</td>
<td>Conv(32)</td>
<td>294</td>
</tr>
<tr>
<td></td>
<td>Conv(64)</td>
<td>18432</td>
</tr>
<tr>
<td></td>
<td>FC(128)</td>
<td>1605632</td>
</tr>
<tr>
<td></td>
<td>FC(10) or FC(50)</td>
<td>1280 or 7936</td>
</tr>
<tr>
<td>Conv8 [35]</td>
<td>Conv(64)</td>
<td>38592</td>
</tr>
<tr>
<td>(CIFAR10)</td>
<td>Conv(128)</td>
<td>222912</td>
</tr>
<tr>
<td></td>
<td>Conv(256)</td>
<td>884736</td>
</tr>
<tr>
<td></td>
<td>Conv(512) or Conv(512)</td>
<td>3588944</td>
</tr>
<tr>
<td></td>
<td>FC(1024) or FC(512)</td>
<td>592384</td>
</tr>
<tr>
<td>ResNet18 [43]</td>
<td>FC(10) or FL(200)</td>
<td>5120 or 102400</td>
</tr>
<tr>
<td>(CIFAR10 and</td>
<td>[Conv(64), Conv(64)]</td>
<td>149184</td>
</tr>
<tr>
<td>Tint-ImageNet)</td>
<td>\times 2</td>
<td></td>
</tr>
<tr>
<td>ResNet34 [43]</td>
<td>[Conv(64), Conv(64)]</td>
<td>149184</td>
</tr>
<tr>
<td>(CIFAR10 and</td>
<td>\times 2</td>
<td></td>
</tr>
<tr>
<td>Tint-ImageNet)</td>
<td>[Conv(128), Conv(128)] \times z</td>
<td>524288</td>
</tr>
<tr>
<td></td>
<td>[Conv(256), Conv(256)] \times z</td>
<td>2097152</td>
</tr>
<tr>
<td></td>
<td>[Conv(512), Conv(512)] \times z</td>
<td>8388608</td>
</tr>
<tr>
<td></td>
<td>Avg-Pool</td>
<td></td>
</tr>
</tbody>
</table>

50 validation and 50 test images. We distribute the training data over 1000 clients in a non-iid fashion using Dirichlet distribution with parameter $\beta = 1$. We run FedAvg and FRL for 4000 global rounds, and in each round, the FL server selects 25 clients randomly. For local training weights (experiments with FedAvg), we use SGD optimizer with a learning rate of 0.01, momentum 0.9, weight decays of 1e-5, batch size 8, and local epochs 2. For training ranks (experiment with FRL), we use SGD with a learning rate of 0.8, momentum of 0.9, weight decay of 1e-5, batch size 8, and local epochs 1. We optimize the hyperparameters based on the validation data and report the final accuracy on the test data.

**B.2 Model architectures**

Table 7 shows the state-of-the-art model architectures and corresponding datasets that we use in our experiments. We also show the number of parameters in each of their layers.

**C FRL against targeted poisoning**

So far, we evaluated the robustness of FRL against untargeted attacks. In this section, we evaluate the robustness of FRL against targeted poisoning, and specifically against backdoor poisoning attacks. We consider state-of-the-art backdoor attacks of three types: **semantic** [4], **artificial** [44], and **edge-case** [42] backdoor attacks. In this section, we first briefly discuss the three backdoor attack types; next, we discuss our evaluation setup, and finally, we present the experiment results at the end.

**C.1 Existing FL backdoor attacks**

Backdoor attacks [39] aim to misclassify any input that contains a specific signal called backdoor trigger, while correctly classifying inputs not containing the trigger. Based on the type of triggers, there are three kinds of backdoor attacks: **Semantic backdoor attacks** [4] aim to misclassify a specific set of inputs that naturally contain a backdoor trigger. Figure 6(a) shows inputs from CIFAR10 data with the background of black stripes on a yellow wall as the naturally present, semantic trigger. **Artificial backdoor attacks** [4, 44] aim to misclassify any input containing a manually added trigger. Figure 6(b) shows inputs with F shaped white pixels as the artificial trigger. **Edge-Case backdoor attacks** [42] aim to misclassify inputs that are from the tail of the training data distribution. Hence, adversary can compute their model updates using mislabeled edge-case data and manipulate the global model to learn the incorrect correlation between data and labels. Figure 6(c) shows edge-case inputs for CIFAR10 distribution which are Southwest airplane images from Internet.

**C.2 Evaluation setup**

**C.2.1 Training hyperparameters**

We compare the performances of the FRL and FedAvg against the above backdoor attacks. We evaluate the accuracy of the (poisoned) global model on the main (inputs without trigger) and backdoor (inputs with trigger) tasks. Following [4, 42], for all the experiments, we start from a pre-trained model with 80.0% test accuracy and train it for 1,000 more FL rounds when malicious clients are present. We use $N = 1000$ clients, distribute CIFAR10 as in Section 7, and randomly select $n = 25$ clients in each round. We report the final test accuracy and the average of backdoor test accuracies over the 1000 FL rounds when percentages of malicious clients is in {1, 2, 5, 10}. We assume that each malicious client has some benign data (per the distribution scheme from Section 7) and all the backdoored data. We use the same hyperparameters discussed in Section 7 for training CIFAR10 on Conv8 model.

**C.2.2 Backdoor attack hyperparameters**

Model Replacement in FL backdoor Attacks: FL backdoor poisoning attacks use a strategy called model replacement where the malicious client first finds a malicious update that contains the backdoor, then it scales the model parameters to cancel the contributions from the other honest clients. For
example, if there are $m$ malicious clients selected in FL round $t$, each malicious client $u$ calculates its backdoored update $\theta_u^t$, and re-scaled it to $\lambda \theta_u^t$ where $\lambda = \frac{m}{n}$ where $n$ is the number of selected clients in each FL round. Model replacement strategy requires that the global model is close to convergence, so the malicious clients can replace the global model with their backdoored model, which performs well on the main task. Following this strategy, the backdoor accuracy would be very high in FedAvg, even if one of the malicious clients is chosen in one round. However, in FRL, re-scaling is not possible as the clients are sending their local rankings where each ranking are a permutation of the indices of the edges in each layer, i.e., of $[0, n_t - 1]\forall \ell \in [L]$ where $L$ is the number of layers and $n_t$ is the number of parameters in $\ell$th layer. To have a fair comparison with FedAvg, we did not use a model replacement strategy for FedAvg too.

**Semantic backdoors:** We choose images with vertically striped walls in the background (Figure 7 (a)) as the backdoors. Of these 12 images with this trigger in the CIFAR10 dataset, we use 9 of them for training the backdoors while keeping the other three for testing the backdoor accuracy. The malicious clients want the global model to predict these images as the bird (class label=2). We measure the backdoor accuracy on 1000 randomly rotated and cropped versions of the three backdoor images held out of the adversary's training.

**Artificial backdoors:** We add a particular pixel pattern to the top left corner of the first nine (not bird) images of the CIFAR10 dataset (Figure 7 (b)) and change their labels to bird (label=2). To evaluate these attacks, we pick 256 random (not bird) images from CIFAR10 and add this pattern to them with the label of class bird.

**Edge-Case backdoors:** We collect 980 images from public web by searching for Southwest airplanes (similar to what [42] did) and resize the images to $32 \times 32$ (Figure 7 (c)). We set their target labels as truck (class label=9). We use 784 of these images for training and keep 196 of them for the evaluation of the backdoor.

### C.3 Evaluation results

**Semantic backdoor attacks:** Figure 7 (a) shows the performance of FedAvg and FRL on the main task and the backdoor task when different percentages of malicious clients want to put a semantic backdoor in the global model. This figure shows that FRL is more robust against semantic backdoor attacks for different percentages of malicious clients. For example, with 2% of malicious clients, training FedAvg results in 84.4% final test accuracy with 82.7% average backdoor accuracy, while training FRL results in 84.1% and 49.2% accuracy on the main task and backdoor task, respectively. The existence of a more significant number of malicious clients (e.g., 10%) results in higher backdoor accuracy for both FedAvg and FRL as the malicious clients have more influence on the global model to introduce their backdoor; with existence of 10% of malicious clients, training FedAvg and FRL achieves 95.7% and 91.2% average backdoor accuracy respectively.

**Artificial backdoor attacks:** These attacks are ineffective when the adversary cannot use model replacement strategy (i.e., cannot re-scale their parameters). In FRL, malicious clients cannot scale their updates, as they submit a local ranking (from a discrete space of updates). To be fair, we also did not use re-scaling in our experiments for FedAvg. We did not report the results for this attack, as the backdoor accuracy would be 0% for both FRL and FedAvg with no parameter re-scaling. It means that the global model always predict the right label (not the adversary target label "bird") for the test backdoor images.

**Edge-Case backdoor attacks:** Figure 7 (b) shows that FRL is more robust against Edge-case backdoor attacks for different percentages of malicious clients. For example, with 2% of malicious clients, training FedAvg results in 83.7% final test accuracy with 77.3% average backdoor accuracy, while training FRL results in 84.0% and 64.6% accuracy on the main task and backdoor task, respectively. Similar to semantic backdoors, a larger number of malicious clients (e.g., 10%) results in higher backdoor accuracy for both FedAvg and FRL; with 10% of malicious clients, training FedAvg and FRL achieves 94.0% and 90.3% average backdoor accuracy respectively.