FEDER: COMMUNICATION-EFFICIENT BYZANTINE-ROBUST FEDERATED LEARNING

Yukun Jiang, Sichuan University
jiangyukun@stu.scu.edu.cn

Xiaoyu Cao, Duke University
xiaoyu.cao@duke.edu

Hao Chen, University of California, Davis
chen@ucdavis.edu

Neil Gong, Duke University
neil.gong@duke.edu

ABSTRACT

In this work, we propose FedER, a federated learning method that is both *efficient* and *robust*. Our key idea is to reduce the communication cost of the state-of-the-art robust FL method via pruning the model updates. Specifically, the server collects a small clean dataset, which is split into a training set and a validation set. In each round of FL, the clients prune their model updates before sending them to the server. The server also derives a server model update based on the training set and prunes it. The server determines the pruning fraction via evaluating the model accuracy on the validation set. We further propose *mutual masking* for each client, which computes the parameters in the overlapping area of pruned client model update and server model update. The mutual mask is used to filter out the parameters of unusual dimensions in malicious updates. We also occasionally normalize the masked client model updates to limit the impact of attacks. Our extensive experiments show that FedER 1) significantly reduces the communication cost for clients in adversarial settings and 2) achieves comparable or even better robustness compared to the state-of-the-art Byzantine-robust method.

1 INTRODUCTION

First introduced by Google, federated learning (FL) has become a popular collaborative learning paradigm that enables distributed clients to train a model without sharing their data McMahan et al. (2017); Kairouz et al. (2019). Due to its distributed nature, FL is vulnerable to malicious clients, which could be fake or compromised benign clients. Considering the attack goal, some attacks aim to decrease the test accuracy of the global model that makes the model unusable (called denial-of-service attacks) Fang et al. (2020); Shejwalkar and Houmansadr (2021), while the others force the global model to output the attacker-chosen label when inputs are embedded with a predefined backdoor trigger (called backdoor attacks) Xie et al. (2019a); Bagdasaryan et al. (2020).

To defend against the aforementioned attacks, Byzantine-robust FL methods Blanchard et al. (2017); Yin et al. (2018); Xie et al. (2019b); Sattler et al. (2020); Cao et al. (2021) use Byzantine-robust aggregation rules to select or weight the client model updates before aggregation. There is an arms race between poisoning attacks and Byzantine-robust defenses, and many defense methods have been captured by advanced attacks Bhagoji et al. (2019); Fang et al. (2020) because the server has no root of trust to address malicious clients. Currently, FLTrust Cao et al. (2021) is the state-of-the-art robust method, where the server uses a small amount of local data to enable the root of trust.

However, Byzantine-robust FL could suffer from the communication overhead and the curse of dimensionality. For instance, in FLTrust Cao et al. (2021), the clients need to send the complete local updates to the server in each round, which is unaffordable for the clients when the model is large. To address this issue, we propose FedER, where the mutual masking and occasional normalization are designed to restrict the area of global update and reserve magnitudes of client model updates, respectively. We conduct extensive experiments on four real-world datasets to evaluate the performance of our FedER. We evaluate multiple existing poisoning attacks including label flipping attack, BadNet attack Gu et al. (2017), Krum attack Fang et al. (2020), and a novel backdoor attack called

AVGBN. Our results show that FedER could achieve Byzantine robustness against existing attacks as state-of-the-art method Cao et al. (2021) does, while reducing communication cost significantly.

2 The Proposed FedER

Overview of FedER: We give threat model in Appendix B. There is one server and N clients in our FedER. The server collects a small clean dataset, which we split into a training dataset (D_r^t) and a validation dataset (D_r^v). The server also maintains a server model $\mathbf{W}_{g,s}$. In each round, apart from common model training among clients, the server computes a root validation accuracy with D_r^v and a server model update $\mathbf{G}_s$ with D_r^t. The root validation accuracy is numerically equal to $p, p \in [0, \Gamma]$, where p is the pruning fraction of current round. It means that only parameters with top $1-p$ values in each model update (from the server and clients) are reserved. Besides, we set Γ as the maximum bound of p to prevent the global model from not updating. Also, we could use a fixed p in FedER, however, we found a fixed pruning fraction lacks generality on different datasets. When receiving model updates $\mathbf{G}_c^i$ from N clients, $i \in \{1, 2, \cdots, N\}$, the server calculates a ReLU-clipped cosine similarity with masked $\mathbf{G}_s$ and $\mathbf{G}_c^i$ for each client, normalizes masked $\mathbf{G}_c^i$ with occasional normalization, and computes global update according to weighted ReLU-clipped cosine similarity and normalized masked client model updates. Finally, for the server and all clients, they update their global and local models with the aforementioned global update. Next, we will elaborate on mutual masking, ReLU-clipped cosine similarity, occasional normalization, and global model updating in our FedER.

Mutual Masking: Model pruning has been used to improve communication efficiency and defend against backdoor attacks in FL by discarding neurons that are of low significance or backdoored Jiang et al. (2019); Li et al. (2020); Wu et al. (2020). Here we create a novel pruning method, called update pruning, which prunes the parameters of updates with lower values. Based on update pruning, mutual masking is designed to further restrict the area for the global update. Specifically, given a client model update $\mathbf{G}_c^i$, by selecting positions of parameters with top p values in $\mathbf{G}_c^i$, we obtain a binary client mask m_c^i. Likewise, the mask for server model update $\mathbf{G}_s$, denoted as m_s, can also be acquired. For each client i, it has a unique binary mutual mask given by

$$m_m^i = m_c^i \odot m_s, \tag{1}$$

where $\odot$ is an element-wise product. Though element-wise addition $\oplus$ (here we assume $1 \oplus 1 = 1$) can also be used to generate the additive mask m_a^i, we found our method is less robust using this additive masking measure (see Appendix G.2). As an important determinant of m_m^i, the value of p is dynamically changing that equals to the current global model's accuracy on the root validation dataset when the accuracy is not greater than Γ.

ReLU-Clipped Cosine Similarity: **Cosine similarity** is originally used to measure the angle between two vectors, and can also be used to represent whether a server model update and a client model update are of similar direction. Though FLTrust Cao et al. (2021) also employs cosine similarity to address malicious client model updates, it is based on updates with full parameters, which has a significant drawback. In particular, because there are tens of thousands of parameters in each model update, even if there are some malicious parameters, as long as the number and values of these parameters are within a reasonable range, the cosine similarity can still be greater than zero. Therefore, FLTrust is robust against attacks that produce malicious client model updates obviously deviated from the server model update (e.g., LF attack and Krum attack), while may be compromised by attacks that produce malicious client model updates slightly deviated from the server model update (e.g., BN attack and AVGBN attack). With mutual masking, we compute masked updates with partial parameters to derive a more representative cosine similarity. For masked updates $\mathbf{G}_s \odot m_m^i$ and $\mathbf{G}_c^i \odot m_m^i$, the cosine similarity between them is given by

$$\mathcal{C}_m^i = \frac{\langle \mathbf{G}_s \odot m_m^i, \mathbf{G}_c^i \odot m_m^i \rangle}{\|\mathbf{G}_s \odot m_m^i\| \cdot \|\mathbf{G}_c^i \odot m_m^i\|}, \tag{2}$$

where $\langle \cdot, \cdot \rangle$ and $\|\cdot\|$ denote the inner product of two vectors and ℓ_2 norm, respectively. However, when cosine similarity comes to be negative, no matter the client model update is benign or malicious, it leads global update to be compromised. To overcome this challenge, we sacrifice client model updates that produce negative cosine similarities with a ReLU function.

Occasional Normalization: Pruning causes the reduction of communication cost in each round, as well as the valuable information. In order to limit the impact of attacks and retain as much information of client model updates as possible, instead of normalizing them with server model update's magnitude as Cao et al. (2021), we propose an occasional normalization method, which gives them larger magnitudes (in a reasonable range) to reserve more information. As for the intuition of occasional normalization, because root training dataset only contains a few samples that are easy to be fitted by the global model, and the server model update is regarded as the standard to compute the global model. Once it is fitted, the magnitude of server model update will be much smaller than that of client model updates (see Appendix C), which greatly slows down model convergence. Since normalization lets all client model updates be of the same magnitude, the scaling attack Bagdasaryan et al. (2020) could be considered as nonexistent. Formally, for masked client model update $\mathbf{G}_c^i \odot m_m^i$, we normalize it as

$$\begin{aligned}\overline{\mathbf{G}_c^i \odot m_m^i} &= \mathbf{n} \cdot \mathbf{G}_c^i \odot m_m^i,\\ subject\ to\ \mathbf{n} &= \max\left(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3\right)\\ \mathbf{n}_1 &= \frac{\|\mathbf{G}_s \odot m_s\|}{\|\mathbf{G}_c^i \odot m_m^i\|}\\ \mathbf{n}_2 &= \frac{\sqrt{1-p} \cdot \|\mathbf{G}_s^{avg} \odot m_s\|}{\|\mathbf{G}_c^i \odot m_m^i\|}\\ \mathbf{n}_3 &= \frac{\mathbf{s}}{mean\left(abs\left(\mathbf{G}_c^i \odot m_m^i\right)\right)},\end{aligned} \tag{3}$$

where $\|\mathbf{G}_s^{avg} \odot m_s\|$ is a statistical value of averaged pruned server model updates' magnitudes for former rounds, $\mathbf{s}$ is a standard mean based on our **minor benign assumption**, $mean\left(\cdot\right)$ represents the element-wise mean, and $abs\left(\cdot\right)$ represents the element-wise absolute value. Specifically, in round $r_g, r_g \in \{1, 2, \cdots, R_g\}$, we have the following:

$$\|\mathbf{G}_s^{avg} \odot m_s\| = \sum_{r=1}^{r_g} \frac{\|\mathbf{G}_s \odot m_s\|}{\sqrt{1-p} \cdot r_g}. \tag{4}$$

It is worth noting that, though not displayed in symbols, $\|\mathbf{G}_s \cdot m_s\|$ and p are highly related to r. Due to p parameters in $\mathbf{G}_s \cdot m_s$ are set to be zero, we divide it by $\sqrt{1-p}$ to obtain an approximate "full" magnitude.

As for the minor benign assumption, we assume that in our FedER, at least ϵ (a minor factor, in our experiments, it is set to be 0.05) clients are benign. Because different client model updates have different overlapping areas with the server model update, thus here we consider the absolute element-wise mean instead of ℓ_2 norm. In particular, We define a set $\mathcal{M} = \{mean\left(abs\left(\mathbf{G}_c^1 \odot m_m^1\right)\right), mean\left(abs\left(\mathbf{G}_c^2 \odot m_m^2\right)\right), \cdots, mean\left(abs\left(\mathbf{G}_c^N \odot m_m^N\right)\right)\}$. Sorting $\mathcal{M}$ from low to high, a sorted set $\mathcal{M}^s = \{mean\left(abs\left(\mathbf{G}_c^{1,s} \odot m_m^{1,s}\right)\right), mean\left(abs\left(\mathbf{G}_c^{2,s} \odot m_m^{2,s}\right)\right), \cdots, mean\left(abs\left(\mathbf{G}_c^{N,s} \odot m_m^{N,s}\right)\right)\}$ can be acquired. Then we have

$$\mathbf{s} = \mathbf{G}_c^{\epsilon \cdot N, s}. \tag{5}$$

With the malicious fraction in range $[0, 1-\epsilon]$, it is guaranteed that $\mathbf{s}$ is not greater than the maximum absolute element-wise mean of benign client model updates, which means $\mathbf{n}_3$ is a benign factor. Meanwhile, because $\mathbf{n}_1$ and $\mathbf{n}_2$ are directly derived by server model update's magnitude, they can also be considered as safe factors. Therefore, we take $\mathbf{n} = \max\left(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3\right)$ to normalize client model updates and mitigate the impact of attacks (esp., scaling attack Bagdasaryan et al. (2020)), while accelerating model convergence.

Global Model Updating: To conduct global model updating, we first need aggregate a global update. The global update is calculated with normalized masked client model updates and weighted ReLU-clipped cosine similarities as

$$\mathbf{G}_g = \sum_{i=1}^{N} \frac{ReLU(\mathcal{C}_m^i)}{\sum_{j=1}^{N} ReLU(\mathcal{C}_m^j)} \cdot \overline{\mathbf{G}_c^i \odot m_m^i}. \tag{6}$$

Then, the server transmits $\mathbf{G}_g$ to all clients. It is intuitive that more than p parameters are set to be zero that at most $(1-p)$ parameters of global model $\mathbf{W}_g$ will be updated, which could greatly reduce communication cost for the server and each client. Formally, with global update, we update global model as $\mathbf{W}_g = \mathbf{W}_g - \alpha \cdot \mathbf{G}_g$, where α is the global learning rate.

Furthermore, we have provided description of complete workflow with pseudo code in Appendix D.

3 Evaluation

3.1 Experimental Setups

All considered FL methods are implemented using Torch[1] with float32 (32-bit) as the precision of each model parameter by default. For each dimension, we use a binary value (1-bit) to indicate whether this dimension is pruned. For other settings on datasets, evaluated attacks, evaluation metrics, and FL system settings, we mostly follow previous work Cao et al. (2021) (see Appendix E).

3.2 Experimental Results

3.2.1 Comparison with State-of-the-Art study

Table 1: Performance metrics on Fashion-MNIST-0.5

Attack	Defense	Critical Round	Critical Volume / (MB) Percent	Accuracy	ASR
No	FLTrust	1020	480.20 / 100.00%	**0.85**	
	FedER with fixed $p = 0.9$	1130	**69.82 / 14.54%**	0.80	
	FedER	**660**	126.69 / 26.38%	**0.85**	
LF	FLTrust	1150	541.40 / 100.00%	**0.85**	
	FedER with fixed $p = 0.9$	1110	**68.59 / 12.69%**	0.81	
	FedER	**700**	124.28 / 22.96%	**0.85**	
BN	FLTrust	1080	508.44 / 100.00%	**0.85**	0.09
	FedER with fixed $p = 0.9$	1180	**72.91 / 14.34%**	0.81	**0.02**
	FedER	**750**	129.87 / 25.54%	**0.85**	0.05
AVGBN	FLTrust	830	390.75 / 100.00%	**0.87**	0.97
	FedER with fixed $p = 0.9$	970	**59.94 / 15.34%**	0.83	**0.03**
	FedER	**640**	124.78 / 31.93%	0.85	0.04
Krum	FLTrust	1650	776.79 / 100.00%	0.83	
	FedER with fixed $p = 0.9$	1280	**79.09 / 10.18%**	0.82	
	FedER	**720**	126.59 / 16.30%	**0.85**	

As shown in Table 1, on Fashion-MNIST-0.5 (0.5 is the imbalance degree), compared with the state-of-the-art method FLTrust Cao et al. (2021), our FedER (with dynamic p as default) significantly reduces communication cost in FL with similar accuracy and ASR on various datasets considering existing attacks, but FedER with fixed p may result in poor accuracy. Here, we consider FLTrust as the baseline that the percent for its critical volume is always 100%. Besides, it comes to an interesting result that our method even shows higher robustness. When defending against the crafted AVGBN attack, FLTrust is compromised by AVGBN with an ASR of 0.97, but FedER receives a low ASR of 0.04, and produces around 68% less volume to reach the critical accuracy. Performance metrics on other datasets also could demonstrate comparable results (see Appendix F).

3.2.2 Ablation Study

To further evaluate the impact of maximum bound Γ for pruning fraction, masking method, occasional normalization, and malicious fraction, we conduct ablation study (see Appendix G).

4 Conclusion

In this work, we propose a federated learning method (named FedER) that enables efficient Byzantine-robustness against poisoning attacks. The server and the clients communicate pruned model updates instead of entire ones, which significantly reduces the communication cost for the server and resource-constrained clients. Extensive experiments on four datasets demonstrate that, the proposed FedER could achieve comparable or even higher Byzantine-robustness while incurring much less communication cost, compared to the state-of-the-art Byzantine-robust FL method. Our future work will focus on studying different kinds of similarities instead of simple cosine similarity.

[1]A Python deep learning library (https://pytorch.org/)

REFERENCES

Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2020. How to Backdoor Federated Learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. 2019. Analyzing federated learning through an adversarial lens. In *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, 634–643.

Battista Biggio, Blaine Nelson, and Pavel Laskov. 2012. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389* (2012).

Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. 2017. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*. 118–128.

Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. 2021. FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.

Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. 2019. Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering. In *Proceedings of the AAAI Workshop on Artificial Intelligence Safety (SafeAI)*.

Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. 2020. Local model poisoning attacks to byzantine-robust federated learning. In *Proceedings of the $USENIX$ Security Symposium ($USENIX$ Security)*. 1605–1622.

Jonathan Frankle and Michael Carbin. 2019. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733* (2017).

Song Han, Jeff Pool, John Tran, and William J Dally. 2015. Learning both Weights and Connections for Efficient Neural Network. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

Yuang Jiang, Shiqiang Wang, Victor Valls, Bong Jun Ko, Wei-Han Lee, Kin K Leung, and Leandros Tassiulas. 2019. Model pruning enables efficient federated learning on edge devices. *arXiv preprint arXiv:1909.12326* (2019).

Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2019. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977* (2019).

Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

Ang Li, Jingwei Sun, Binghui Wang, Lin Duan, Sicheng Li, Yiran Chen, and Hai Li. 2020. Lotteryfl: Personalized and communication-efficient federated learning with lottery ticket hypothesis on non-iid datasets. *arXiv preprint arXiv:2008.03371* (2020).

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning Filters for Efficient ConvNets. (2016).

Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2018a. Trojaning attack on neural networks. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018b. Rethinking the Value of Network Pruning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*. PMLR, 1273–1282.

P Molchanov, S Tyree, T Karras, T Aila, and J Kautz. 2017. Pruning convolutional neural networks for resource efficient inference. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Felix Sattler, Klaus-Robert Müller, Thomas Wiegand, and Wojciech Samek. 2020. On the byzantine robustness of clustered federated learning. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 8861–8865.

Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. 2018. Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Virat Shejwalkar and Amir Houmansadr. 2021. Manipulating the Byzantine: Optimizing Model Poisoning Attacks and Defenses for Federated Learning. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.

Octavian Suciu, Radu Marginean, Yigitcan Kaya, Hal Daume III, and Tudor Dumitras. 2018. When Does Machine Learning FAIL? Generalized Transferability for Evasion and Poisoning Attacks. In *Proceedings of the $USENIX$ Security Symposium ($USENIX$ Security)*.

Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. 2020. Data poisoning attacks against federated learning systems. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*. Springer, 480–501.

Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. 2019. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*. IEEE, 707–723.

Chen Wu, Xian Yang, Sencun Zhu, and Prasenjit Mitra. 2020. Mitigating backdoor attacks in federated learning. *arXiv preprint arXiv:2011.01767* (2020).

Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).

Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. 2019a. Dba: Distributed backdoor attacks against federated learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Cong Xie, Sanmi Koyejo, and Indranil Gupta. 2019b. Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance. In *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, 6893–6901.

Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. 2018. Byzantine-robust distributed learning: Towards optimal statistical rates. In *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, 5650–5659.

A Background and Related Work

A.1 Federated Learning

FL aims to enable collaborative model training among distributed clients with their local datasets. Assuming there are a server and N clients, where each client i possess a local dataset D_c^i. In general, with server's aggregation, our goal is to solve a distributed optimization problem

$$\min_{\mathbf{W}_g \in \mathbb{R}^d} f(\mathbf{W}_g; D_c) = \min_{\mathbf{W}_g \in \mathbb{R}^d} \sum_{i=1}^{N} f(\mathbf{W}_g; D_c^i), \tag{7}$$

where $\mathbf{W}_g$ represents the global model; $l(\cdot)$ represents the empirical loss function, e.g., cross entropy loss; and $D_c = \cup_{i=1}^{N} D_c^i$.

In non-adversarial settings, FedAVG McMahan et al. (2017) is widely recognized as a state-of-the-art method. In each round, client i receives a global model from the server, computes a client model update $\mathbf{G}_c^i$ with its local dataset D_c^i and sends $\mathbf{G}_c^i$ to the server. The server aggregates a global update following the aggregation rule below:

$$\mathbf{G}_g = \sum_{i=1}^{N} \frac{|D_c^i|}{|D_c|} \cdot \mathbf{G}_c^i, \tag{8}$$

where $|D|$ indicates the size of dataset D. Then a new global model is obtained via applying this update. Though FedAvg achieves good performance in non-adversarial settings, recent studies Bagdasaryan et al. (2020); Cao et al. (2021) show that, the global model in FedAVG is vulnerable to malicious clients, i.e., even a single malicious client can arbitrarily corrupt the global model.

To defend against adversaries, many Byzantine-robust methods are proposed. These methods essentially follow the same training process as FedAvg, while altering only the aggregation rules. Next, we will discuss some popular aggregation methods and their aggregation rules:

Krum Blanchard et al. (2017): Krum selects an update that has the minimum squared Euclidean distance to $(N - l - 2)$ other updates from all client model updates, where l is the upper limit of the number of malicious clients.

Trim-mean Yin et al. (2018): Trim-mean considers each dimension of global update individually to conduct a coordinate-wise aggregation. Specifically, for each dimension of client model updates, the server sorts parameters of this dimension from different clients, trims the parameters with larger k and smaller k values, and computes the mean of remaining $1 - 2k$ parameters as the global update parameter, where $k \in [0, 0.5)$. To achieve effective defense, k should be larger than the malicious fraction, which means Trim-mean can only be robust when more than half of clients are benign.

Median Yin et al. (2018): Median also considers each dimension of global update individually, but it does not trim any parameters. For parameters of each dimension, after sorting, Median chooses the median parameter as the global update parameter of this dimension.

FLTrust Cao et al. (2021): FLTrust is the state-of-the-art Byzantine-robust FL method. In FLTrust, the server bootstraps trust to the clients. Specifically, the server collects a small clean dataset D_c^0 and computes a server model update $\mathbf{G}^0$ based on the dataset in each round of FL. The global update is aggregated as follows:

$$\mathbf{G}_g = \sum_{i=1}^{N} \frac{ReLU(\mathcal{C}_i)}{\sum_{j=1}^{n} ReLU(\mathcal{C}_j)} \cdot \frac{\|\mathbf{G}^0\|}{\|\mathbf{G}_c^i\|} \cdot \mathbf{G}_c^i, \tag{9}$$

where $\mathcal{C}_i$ is the cosine similarity between $\mathbf{G}^0$ and $\mathbf{G}_c^i$.

A.2 Poisoning Attacks to Federated Learning

Like centralized learning, FL is vulnerable to data poisoning attacks Biggio et al. (2012); Liu et al. (2018a); Shafahi et al. (2018); Suciu et al. (2018); Tolpegin et al. (2020), where the attacker pollutes the training data such that the global model is corrupted. Moreover, due to the unique distributed training process, FL is also vulnerable to model poisoning attacks Fang et al. (2020); Bhagoji et al. (2019); Xie et al. (2019a); Bagdasaryan et al. (2020); Shejwalkar and Houmansadr (2021), in which an attacker directly manipulates the model updates sent from the malicious clients to the server. Based on the attack goal, poisoning attacks can be divided into two categories, i.e., denial-of-service (DoS) attacks Fang et al. (2020); Shejwalkar and Houmansadr (2021) and backdoor attacks Bagdasaryan et al. (2020); Xie et al. (2019a). DoS attacks aim to decrease the global model accuracy to make the model unusable, while backdoor attacks attempt to force the global model to output the attacker-chosen label for inputs with pre-defined trigger(s). Next, we will give more information about two DoS attacks, i.e., label flipping attack and Krum attack Fang et al. (2020), as well as two backdoor attacks, i.e., BadNet attack Gu et al. (2017) and averaging BadNet attack.

Label flipping (LF) attack: LF is a popular attacking method that changes data samples' labels to attacker-chosen targets. This attack is achieved by manipulating training datasets distributed among compromised clients, which is considered as a data poisoning attack. If LF attack is successful, the global model will make incorrect predictions, which leads to the low accuracy of the model.

BadNet (BN) attack Gu et al. (2017): First introduced in centralized learning, BN attack does not intend to affect global model accuracy with the clean testing dataset, but accuracy with the poisoned testing dataset. Specifically, corrupted clients train their local models with poisoned datasets, which contain some data samples with trigger(s) belonging to the attacker-chosen category. When BN attack is successful, if inputting samples with trigger(s), the global model will output the attacker-chosen prediction. While for input samples without triggers, the predictions made by the global model will not be affected. In FL, by sending malicious client model updates to the server, the global model can also be backdoored. To achieve a more effective BN attack to FL, Bagdasaryan et al. Bagdasaryan et al. (2020) proposed a scaling method, which scales malicious client model updates with a factor much greater than 1. However, this scaling method is useless when the server takes a magnitude normalization to all client model updates. Therefore, in this work, BN attack and scaling BN attack are regarded as the same.

Averaging BadNet (AVGBN) attack: With magnitude normalization and weighted aggregation in the server, the effect of BN attack is greatly mitigated. We believe that the reason for this failure of BN attack is the lack of cooperation among malicious clients. Specifically, client model updates sent by different malicious clients may be of diverse directions, and even the updates may affect each other, resulting in a very low attack effect caused by each malicious client. To make malicious client model updates orderly, empirically, we craft a cooperative BN attack by conducting a FedAVG McMahan et al. (2017) with malicious updates and sending the averaged update to the server, named AVGBN.

Krum attack Fang et al. (2020): Krum attack is a model poisoning attack to Krum Blanchard et al. (2017), but can also somehow threaten other Byzantine-robust FL Cao et al. (2021). Because Krum chooses a client model update that is closest to other updates, once the chosen update is malicious, Krum becomes vulnerable. To make the chosen update malicious, the attacker should be able to access and manipulate client local model updates. Specifically, assuming that the direction mask for the global update is m_{non} in a non-adversarial scenario, Krum attack replaces the client model updates of compromised clients with a constructed update $-\mu \cdot m_{non}$, where μ is a binary-searched value to make $-\mu \cdot m_{non}$ be chosen by Krum. This attack could deviate the global update the most towards the reverse direction of global update chosen in a non-adversarial scenario, thus making the direction between current global model $\mathbf{W}_g$ and the optimal global model $\hat{\mathbf{W}}_g$ further, finally leading to low global model accuracy.

A.3 Model Pruning & Update Pruning

To overcome over-parameterization, model pruning is widely recognized as a remedy Han et al. (2015); Molchanov et al. (2017); Li et al. (2016); Frankle and Carbin (2019). In a typical model pruning pipeline, training, pruning, and fine-tuning are regarded as the basic three stages to prune redundant parameters and keep important parameters Liu et al. (2018b). Due to FL clients' resource-constrained nature, recent years have witnessed increasing interest in employing model pruning for efficient FL Jiang et al. (2019); Li et al. (2020). Specifically, Jiang et al. Jiang et al. (2019) proposed an adaptive method that prunes model to maximize the approximate empirical risk reduction, Li et al.

Li et al. (2020) leveraged the *lottery ticket hypothesis* Frankle and Carbin (2019) that conditionally prunes model to find lottery ticket networks (LTNs). Moreover, because the backdoored neural network could produce activations of weird values with adversarial inputs Chen et al. (2019); Wang et al. (2019), interestingly, Wu et al. Wu et al. (2020) proposed a defense against backdoor attacks that prunes model to remove backdoored neurons that produce abnormal activation values.

Motivated by model pruning, we propose a new pruning paradigm for FL, which prunes model updates to reduce communication volume in each communication round, as well as the computation cost for the server to aggregate a secure global update. Specifically, we leverage unstructured pruning for model updates to get "sub-updates" that only keep parameters with higher values. Then, the server conducts Byzantine-robust aggregation based on pruned updates. Compared with the typical model pruning, though both model pruning and update pruning could achieve communication efficiency, our method achieves it by discarding redundant update parameters instead of finding a "sub-network". Compared with Wu et al. Wu et al. (2020), 1) our update pruning enables efficient FL while Wu et al. relies on entire model parameters; 2) our method is based on update-level while Wu et al. manipulates the global model directly; 3) the goal of our method is preventing the global model from being backdoored while that of Wu et al. is removing backdoors from the compromised global model.

B Threat Model

B.1 Attack Model

Following previous works Fang et al. (2020); Cao et al. (2021); Shejwalkar and Houmansadr (2021), we consider a scenario where an attacker controls some malicious clients, which can be either fake clients (because communication links are compromised) or compromised genuine clients. We assume the malicious clients can send arbitrary client model updates to the server and collude with each other. Moreover, we notice that existing attacks may have different assumptions on the attacker's knowledge Fang et al. (2020). In particular, an attacker may have full knowledge (including all clients' local training data and model updates), or partial knowledge about the FL training process. Full-knowledge attacks are shown to be stronger than partial-knowledge ones. Therefore, following FLTrust Cao et al. (2021), we consider the full-knowledge attacks to show that our FedER is robust even against the strong attacks.

B.2 Defense Model

Defender's Goal: Following the previous work Cao et al. (2021), we assume the defender's goal is to achieve robustness against poisoning attacks in adversarial settings, while remaining accurate in non-adversarial settings. Moreover, since the clients are usually resource-constrained devices, e.g., IoT devices and smartphones, we further assume the defender aims to reduce the communication cost of FL as much as possible, especially on the client side. Existing defenses like Krum Blanchard et al. (2017) or FLTrust Cao et al. (2021) do not consider the efficiency of FL.

Defender's knowledge and capability: We assume the defender is on the server side and has access to everything the server knows, including the global model and all the masks as well as masked updates, which the clients send to the server. However, the defender does not know which clients are malicious or the number of malicious clients. We further assume that the server collects a small representative dataset (i.e., the distribution of the small dataset is the same distribution as that of overall training data among clients) following Cao et al. (2021), which needs to be clean from poisoning. The cost of collecting a small size of clean samples is much lower than the cost required for an adversary to carry out an attack. This dataset can be obtained by manual labeling. We split the server's collected dataset into a training dataset and a validation dataset, which we use for calculating the server model update and choosing the pruning factor, respectively. We empirically show that the server only needs to collect a small dataset, e.g., 100 examples for training and 100 examples for validation, to achieve high accuracy and efficiency.

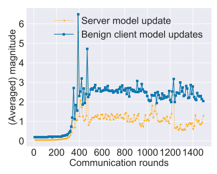
C THE ℓ_2 NORM OF THE SERVER MODEL UPDATE AND CLIENT MODEL UPDATES FOR EACH TRAINING ROUND

Figure 1 shows the ℓ_2 norm (magnitude) of the server model update and client model updates on four datasets without attacks. Here we use ℓ_2 norm to represent the magnitude of each update. For better comparison, we use the averaged magnitudes of client model updates. We observe that, for both the server model update and benign client model updates, their magnitudes increase and then decrease on all datasets except CIFAR-10. Moreover, on each dataset, the magnitude of the server model update has always been smaller than that of client model updates.

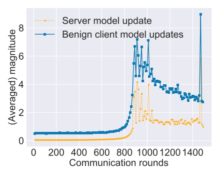
Figure 2 shows the ℓ_2 norm (magnitude) of three kinds of updates, which are respectively the server model update, benign client model updates, and malicious client model updates, where MNIST-0.5 is used and four attacks are considered. We set the malicious fraction to be 0.3 as default. For all attacks except LF attack, the magnitudes of updates increase and then decrease, while that of benign client model updates and malicious client model updates are much higher than that of server model update. For LF attack, though the server model update's magnitude receives an increase and then a reduction, the magnitudes of benign client model updates and malicious client model updates keep growing.

Overall, under all settings, the magnitudes of benign client model updates are higher than the server model update's. Hence, we design the occasional normalization to normalize the magnitudes of client model updates.

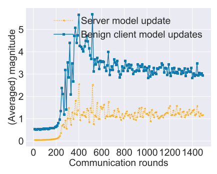
(a) MNIST-0.1



(b) MNIST-0.5



(c) Fashion-MNIST-0.5



(d) Cifar-10-0.5

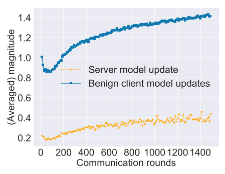
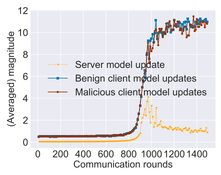
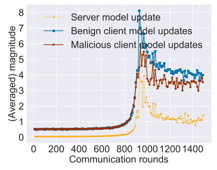


Figure 1: ℓ_2 norm of server model updates and client model updates without attack on multiple datasets.

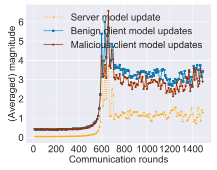
(a) LF attack



(b) BN attack



(c) AVGBN attack



(d) Krum attack

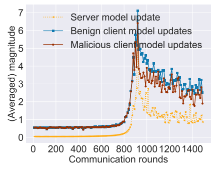


Figure 2: ℓ_2 norm of server model updates and client model updates with various attacks on MNIST-0.5.

D COMPLETE WORKFLOW OF FEDER

The complete workflow of FedER is summarized in Algorithm 2, where ModelTraining$(\cdot)$ is a local training algorithm specified in Algorithm 1, and ModelValidation$(\cdot)$ is the algorithm to evaluate the model accuracy on a given dataset. Specifically, in Algorithm 1, ModelTraining$(\cdot)$ performs stochastic gradient descent on randomly sampled batch with learning rate β for T iterations and returns a pruned model update with the corresponding pruning mask.

In Algorithm 2, the server first randomly initializes the global model. Then, the server and the clients update the global model iteratively in multiple rounds, each of which consists of three phases.

Algorithm 1 ModelTraining ($\mathbf{W}$, D, T, β, b, p)

Input: Current model $\mathbf{W}$, a training dataset D, number of iterations T, learning rate β, batch size b, and pruning fraction p.
Output: A pruned model update $\mathbf{G} \odot m$ and the corresponding pruning mask m.

```
1: $\mathbf{W}_0 = \mathbf{W}$
2: for $t = 1, 2, \cdots, T$ do
3:     Randomly sample a batch $D_b$ with $b$ samples from $D$
4:     $\mathbf{W} = \mathbf{W} - \beta \cdot \nabla f(\mathbf{W}, D_b)$        $\triangleright$ $f$ is the loss function
5: end for
6: $\mathbf{G} = \mathbf{W}_0 - \mathbf{W}$
7: Compute a pruning mask $m$ for $\mathbf{G}$ with pruning fraction $p$
8: Return $\mathbf{G} \odot m$ and $m$;
```

Particularly, in Phase I, the server first calculates a pruning fraction p for local update pruning in current round. Then, the server sends the global model to the clients in the first round, or the global update in the others. In Phase II, the server and the clients calculate their model updates using the root training dataset and the clients' local training datasets, respectively. It is worth noting that, if the clients receive a global update from the server, they need to update their client models before calculating updates. In Phase III, unique ReLU-clipped cosine similarity and a normalized client model update are calculated based on mutual masking and occasional normalization for each selected client. Finally, the server computes the global update according to the aforementioned factors and then updates the global model.

As for the security analysis, it has been well proved that, with ReLU-clipped cosine similarity and normalization, the distance between the optimal model and global model is strictly constrained Cao et al. (2021). In our FedER, we further restrict the updating area and loose the normalization magnitude, which could be considered as a trade-off between security and performance. However, the pruning process is dynamically related to the pruning fraction p and maximum bound Γ, which makes it impracticable to quantitatively analyze the distance between the global model and the optimal model. Fortunately, experimental results greatly support our idea.

E Experimental Settings

Table 2: Some default FL system parameter settings.

Symbol	Discription	Dataset			
		MNIST-0.1	MNIST-0.5	Fashion-MNIST-0.5	CIFAR-10-0.5
$\lvert D_r^t \rvert$	size of the root training dataset	500		600	500
$\lvert D_r^v \rvert$	size of the root validation dataset	100			
$\lvert D_c^i \rvert$	size of client local dataset	100			
N	# clients	100			
κ	# selected clients of each round	N			
R_g	# global training rounds	2500			1500
T_s	# server local iterations	1			
T_c	# client local iterations	1			8
b'/b	# backdoor samples / batch size	4 / 32			8 / 64
$\alpha \cdot \beta_c$	the combined learning rate	0.1		0.2	
β_s	the server local learning rate	1.0			
η	the malicious fraction	0.3			
Γ	the maximum bound of pruning fraction	0.9			

E.1 Datasets

We conduct our experiments on multiple real-world datasets, including one IID dataset and three Non-IID datasets, to evaluate the performance of our FedER. Following Cao et al. (2021), we take an imbalance degree $\lambda \in [0, 1]$ to characterize the distribution of training samples among all clients. It is worth mentioning that, in our experiments, we consider the *label distribution skew* Kairouz et al. (2019). Assuming there are ρ clusters for a dataset with φ data samples evenly from ρ categories. For cluster $\omega \in \{0, 1, \cdots, \rho{-}1\}$, $\frac{\varphi \cdot \lambda}{\rho}$ samples from category ω and $\frac{\varphi \cdot (1-\lambda)}{\rho}$ samples uniformly from

Algorithm 2 FedER

Input: A server with a root training dataset D_r^t and a root validation dataset D_r^v; N clients with N local datasets $\cup_{i=1}^N D_c^i$; number of global training rounds R_g; number of server local iterations T_s; number of client local iterations T_c; global learning rate α; server local learning rate β_s; client local learning rate β_c; number of selected clients κ in each round; batch size b; pruning fraction p and its maximum bound Γ.
Output: Global model $\mathbf{W}_g$.

```
1: Randomly initialize $\mathbf{W}_g$
2: for $r_g = 1, 2, \cdots, R_g$ do
3:     Randomly selects $\kappa$ clients $c_1, c_2, \cdots, c_\kappa$.
4:     // Phase I: Model / Update transfer
5:     $p$ = $\min$(ModelValildation($\mathbf{W}_g$, $D_r^v$), $\Gamma$)
6:     if $r_g == 1$ then
7:         Server sends $\mathbf{W}_g$ to selected clients
8:     else
9:         Server sends $\mathbf{G}_g$ to selected clients
10:    end if
11:    // Phase II: Model training
12:    // Server side training
13:    $\mathbf{W}_{g,s} = \mathbf{W}_g$
14:    $\mathbf{G}_s \odot m_s, m_s =$ ModelTraining($\mathbf{W}_{g,s}$, $D_r^t$, $T_s$, $\beta_s$, $|D_r^t|$, $p$)
15:    // Client side training
16:    for $i = c_1, c_2, \cdots, c_\kappa$ do in parallel
17:        if $r_g == 1$ then
18:            $\mathbf{W}_{g,i} = \mathbf{W}_g$
19:        else
20:            $\mathbf{W}_{g,i} = \mathbf{W}_{g,i} - \alpha \cdot \mathbf{G}_g$
21:        end if
22:        $\mathbf{G}_i \odot m_c^i, m_c^i =$ ModelTraining($\mathbf{W}_{g,i}$, $D_c^i$, $T_c$, $\beta_c$, $b$, $p$)
23:    end for
24:    // Phase III: Global update aggregation
25:    for $i = c_1, c_2, \cdots, c_\kappa$ do
26:        $m_m^i = m_c^i \odot m_s$
27:        $\mathcal{C}_m^i = \frac{\langle \mathbf{G}_s \odot m_m^i, \mathbf{G}_c^i \odot m_m^i \rangle}{\|\mathbf{G}_s \odot m_m^i\| \cdot \|\mathbf{G}_c^i \odot m_m^i\|}$
28:        Obtain $\mathbf{n}$ based on Eq. 3
29:        $\overline{\mathbf{G}_c^i \odot m_m^i} = \mathbf{n} \cdot \mathbf{G}_c^i \odot m_m^i$
30:    end for
31:    $\mathbf{G}_g = \sum_{i=1}^N \frac{ReLU(\mathcal{C}_m^i)}{\sum_{j=1}^N ReLU(\mathcal{C}_m^j)} \cdot \overline{\mathbf{G}_c^i \odot m_m^i}$
32:    $\mathbf{W}_g = \mathbf{W}_g - \alpha \cdot \mathbf{G}_g$
33: end for
34: Return $\mathbf{W}_g$
```

other categories are distributed to this cluster. Besides, the number of data samples in each cluster is the same, as well as the number of clients in each cluster. Then, data samples are distributed to clients randomly, and a data sample only belongs to one client. In other words, each cluster contains $\frac{N}{\rho}$ clients, where each client possesses $\frac{\varphi}{N}$ data samples. Apparently, a larger value of $\left|\lambda - \frac{1}{\rho}\right|$ indicates a larger degree of label distribution skew, and when λ equals to $\frac{1}{\rho}$, the label distribution is IID. Specifically, we consider datasets as follows:

MNIST-0.1: MNIST LeCun et al. (1998) contains 60,000 training and 10,000 testing 28×28 grayscale images with respect to 10-class handwritten digits, while the number of training samples for each category is not equal. Therefore, we randomly select 50,000 training samples from the original 60,000 training samples, where each category contains 5,000 samples, and distribute the selected samples to 10 clusters according to the value of λ. For MNIST-0.1, we take $\lambda = 0.1$ to simulate IID training data distribution among clients.

MNIST-0.5: For MNIST-0.5, λ is set to be 0.5, which indicates training data samples are unevenly distributed among clients.

Fashion-MNIST-0.5: Fashion-MNIST Xiao et al. (2017) comprises 60,000 training and 10,000 testing 28×28 grayscale images with respect to fashion products from 10 categories. For Fashion-MNIST-0.5, training data samples are allocated to clients with $\lambda = 0.5$, which is a kind of Non-IID setting.

CIFAR-10-0.5: CIFAR-10 Krizhevsky et al. (2009) contains 60,000 RGB images (50,000 for training, 10,000 for testing) that are classified into 10 distinct categories. For CIFAR-10-0.5, a Non-IID training data distribution among clients is considered, where we take $\lambda = 0.5$.

E.2 Evaluated Attacks

In our experiments, we consider two kinds of attacks that reduce the global model accuracy (DoS attacks) and inject backdoor triggers into the global model (backdoor attacks). For Dos attacks, we evaluate the LF attack and Krum attack. For the backdoor attacks, BN attack and our crafted AVGBN attack are considered. Specifically, we set these attacks as follows:

LF attack: The same LF attack setting in Cao et al. (2021) is used in our experiments. Given a dataset with ρ categories, for each data sample, if its original label is ω, we flip its label to $\rho - \omega - 1$.

BN attack and AVGBN attack: BN attack and AVGBN attack are both backdoor attacks to FL model, and BN attack is the same as scaling attack in Cao et al. (2021) because of the existence of normalization. Different from Cao et al. (2021) that augments client local training dataset by one, like Xie et al. (2019a), we take a small poison ratio of 0.125 (i.e., 4 for batch size 32 and 8 for batch size 64) because this setting is harder to be detected by the ReLU-based cosine similarity. Following Cao et al. (2021), for MNIST-0.1, MNIST-0.5, and Fashion-MNIST-0.5, we use a four-pixel trigger at the bottom right corner of images and set the attacker-chosen target to be 0. For CIFAR-10-0.5, we use three red, green, and blue pixels at the bottom right corner of images as the trigger, and set the target to be “bird”.

Krum attack: Krum attack replaces some client model updates with a constructed malicious update to deviate global model update. We consider the same settings in Fang et al. (2020) to conduct Krum attack in our experiments.

E.3 Evaluation Metrics

Four significant metrics are used to evaluate the performance of the global model of an FL method, i.e., 1) *Test accuracy* (or simply *accuracy*): The proportion of correct predictions made by the global model with the testing dataset. Unless otherwise mentioned, we treat test accuracy and accuracy as the same. 2) *Critical round*: The number of global training rounds when the global model reaches a certain accuracy (i.e., *critical accuracy*). For different datasets, the value of critical accuracy should not be the same. Therefore, based on some empirical estimations, the values of critical accuracy for MNIST-0.1, MNIST-0.5, Fashion-MNIST-0.5, and CIFAR-10-0.5 are set to be 0.90, 0.90, 0.80, and 0.70, respectively. 3) *Critical volume*: The volume of communication sent and received by the server in the critical round. It also shows the sum of communication volume among clients. 4) *Attack success rate (ASR)*: The proportion of predictions that are the attacker-chosen target among all predictions made by the global model with the poisoned testing dataset.

For all attacks, test accuracy, critical round, and critical volume are used to measure the basic performance of an FL method. Besides, the ASR is dedicated to further evaluating the FL method’s defense against backdoor attacks. For each FL method, the higher test accuracy and lower ASR represent the better Byzantine-robustness. Meanwhile, if they are available, the critical round and critical volume of lower value indicate better communication efficiency.

E.4 FL System Settings

For different datasets, we choose different global models to show the generality of our FedER. Specifically, we choose a CNN model with two convolutional layers and three fully connected layers for MNIST-0.1, MNIST-0.5, and Fashion-MNIST-0.5. For CIFAR-10, we choose the ResNet20 He et al. (2016) model.

Table 2 shows some FL system parameters settings that we use. In particular, we distribute 500 samples to each client for all datasets except Fashion-MNIST-0.5, where each client possess 600 samples. Following FLTrust Cao et al. (2021), by default, we assume the size of two root datasets is 100. Also, the number of clients N is set to be 100 and $\kappa = N$. For evaluated methods, they all use the parameters R_g, T_s, T_c, b, α, β_c, and β_s. We set them carefully for different datasets to achieve high accuracy and fast convergence. For instance, on Fashion-MNIST-0.5, we set $R_g = 2500$, $T_s = 1$, $T_c = 1$, $b = 32$, $\alpha = 0.2$, $\beta_c = 1.0$, and $\beta_s = 1.0$. Moreover, for FedER, we assume the maximum bound of pruning fraction is 0.9. Unless otherwise mentioned, we use the aforementioned default settings. However, the impacts of some settings will be also discussed in this section.

F Performance metrics on various datasets

Please refer to Table 3.

G Ablation Study

G.1 Impact of maximum bound for pruning fraction

Recall that in Section 2, we set a maximum bound Γ for the pruning fraction p, and p is numerically equal to the global model accuracy on root validation dataset. Due to the small size of root validation dataset, the global model accuracy on this dataset may be much higher than that on the entire testing dataset. Intuitively, a higher value of p represents a smaller global updating area. When p reaches a high value, the global model updating will be very slow or even stop (when $p = 1$). Therefore, like Li et al. (2020), we set a maximum bound Γ for p and compare the impacts of different bounds. As shown in Table 4, we vary the $\Gamma \in \{0.8, 0.9, 0.95, 0.99\}$ on MNIST-0.5 considering different attacking scenarios. We observe that, without attacks, $\Gamma = 0.8$ receives the highest accuracy and $\Gamma = 0.9$ reaches the lowest critical volume. For considered attacks, all settings remain robust against them, while $\Gamma = 0.9$ shows the best communication efficiency. Meanwhile, within a reasonable range (e.g., $(0.8, 0.90)$), different values for Γ have limited impact. Nevertheless, when Γ is set to be an extreme value (e.g., 0.99), the accuracy reduces.

G.2 Impact of Masking Method

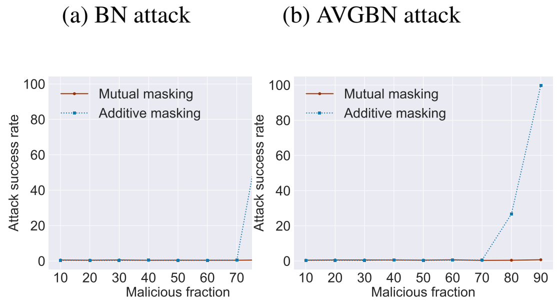


Figure 3: Impact of masking method on attack success rate on MNIST-0.5. The accuracy of all masking methods are similar under backdoor attacks, which we omit for simplicity.

Recall that in Section 2, we use the element-wise product to generate a mutual mask instead of using the element-wise addition to generate an additive mask. With mutual masking, the global

Table 3: Performance metrics on various datasets
(MNIST-0.1, MNIST-0.5, and CIFAR-10-0.5).

(a) MNIST-0.1

Attack	Defense	Critical Round	Critical Volume (MB) / Percent	Accuracy	ASR
No	FLTrust	520	244.81 / 100.00%	**0.96**	
	FedER with fixed $p = 0.9$	410	**25.33 / 10.35%**	0.95	
	FedER	**320**	81.22 / 33.18%	0.95	
LF	FLTrust	550	258.93 / 100.00%	**0.96**	
	FedER with fixed $p = 0.9$	470	**29.04 / 11.22%**	0.95	
	FedER	**430**	91.31 / 35.26%	0.95	
BN	FLTrust	550	258.93 / 100.00%	**0.96**	0.06
	FedER with fixed $p = 0.9$	420	**25.95 / 10.02%**	0.95	**0.00**
	FedER	**370**	92.75 / 35.82%	0.95	**0.00**
AVGBN	FLTrust	500	235.39 / 100.00%	**0.96**	**0.00**
	FedER with fixed $p = 0.9$	440	**27.19 / 11.55%**	0.95	**0.00**
	FedER	**370**	93.71 / 39.81%	0.95	**0.00**
Krum	FLTrust	600	282.47 / 100.00%	0.93	
	FedER with fixed $p = 0.9$	460	**28.42 / 10.06%**	**0.94**	
	FedER	**380**	87.00 / 30.80%	**0.94**	

(b) MNIST-0.5

Attack	Defense	Critical Round	Critical Volume / (MB) Percent	Accuracy	ASR
No	FLTrust	1080	580.44 / 100.00%	**0.96**	
	FedER with fixed $p = 0.9$	470	**29.04 / 5.00%**	0.95	
	FedER	**430**	94.06 / 16.20%	0.95	
LF	FLTrust	1120	527.27 / 100.00%	**0.96**	
	FedER with fixed $p = 0.9$	540	**33.37 / 6.33%**	0.95	
	FedER	**480**	100.76 / 19.11%	0.95	
BN	FLTrust	1110	522.57 / 100.00%	**0.95**	**0.00**
	FedER with fixed $p = 0.9$	540	**33.37 / 6.39%**	**0.95**	**0.00**
	FedER	**430**	102.40 / 19.60%	**0.95**	**0.00**
AVGBN	FLTrust	730	343.67 / 100.00%	**0.95**	0.03
	FedER with fixed $p = 0.9$	620	**38.31 / 11.15%**	**0.95**	**0.00**
	FedER	**470**	130.77 / 38.05%	**0.95**	**0.00**
Krum	FLTrust	1090	513.15 / 100.00%	**0.95**	
	FedER with fixed $p = 0.9$	690	**42.63 / 8.31%**	0.93	
	FedER	**400**	88.29 / 17.21%	0.94	

(c) CIFAR-10-0.5

Attack	Defense	Critical Round	Critical Volume / (MB) Percent	Accuracy	ASR
No	FLTrust	650	1359.11 / 100.00%	**0.78**	
	FedER with fixed $p = 0.9$	N/A	N/A	0.64	
	FedER	**340**	**358.15 / 26.35%**	**0.78**	
LF	FLTrust	690	1442.74 / 100.00%	**0.76**	
	FedER with fixed $p = 0.9$	N/A	N/A	0.64	
	FedER	**390**	**382.20 / 26.50%**	**0.76**	
BN	FLTrust	740	1547.29 / 100.00%	**0.78**	0.04
	FedER with fixed $p = 0.9$	N/A	N/A	0.63	0.06
	FedER	**310**	**333.33 / 21.54%**	**0.78**	**0.02**
AVGBN	FLTrust	**660**	1380.02 / 100.00%	**0.76**	**0.04**
	FedER with fixed $p = 0.9$	N/A	N/A	0.61	0.10
	FedER	750	**768.27 / 55.67%**	0.75	**0.04**
Krum	FLTrust	N/A	N/A	0.69	
	FedER with fixed $p = 0.9$	N/A	N/A	0.63	
	FedER	**830**	**734.47 / N/A**	**0.71**	

Table 4: Impact of maximum bound of pruning fraction on some performance metrics on MNIST-0.5.

Attack	Γ	Critical Round	Critical Volume (MB)	Accuracy	ASR
No	0.8	**390**	95.82	**0.96**	
	0.9	430	**94.06**	0.95	
	0.95	430	94.20	0.94	
	0.99	470	96.58	0.93	
LF	0.8	**420**	101.63	**0.95**	
	0.9	480	**100.76**	**0.95**	
	0.95	530	103.60	0.94	
	0.99	490	101.38	0.93	
BN	0.8	**400**	104.96	**0.95**	**0.00**
	0.9	430	**102.40**	**0.95**	**0.00**
	0.95	470	105.37	0.94	**0.00**
	0.99	480	105.97	0.94	0.01
AVGBN	0.8	470	137.05	**0.95**	**0.00**
	0.9	470	**130.77**	**0.95**	**0.00**
	0.95	470	132.55	0.94	0.01
	0.99	**460**	131.68	0.93	0.01
Krum	0.8	**380**	91.51	**0.95**	
	0.9	400	**88.29**	0.94	
	0.95	400	88.97	0.93	
	0.99	480	93.59	0.93	

(a) LF attack (b) BN attack (c) AVGBN attack (d) Krum attack

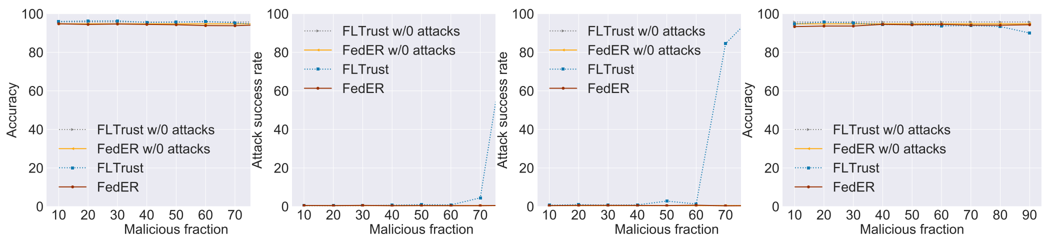


Figure 4: Impact of malicious fraction on the test accuracy and attack success rate of different FL methods on MNIST-0.5. The accuracy of all methods are similar under backdoor attacks, which we omit for simplicity.

update only keeps the dimensions that are kept in the pruned server model update. However, with additive masking, some dimensions pruned in the pruned server model update but kept in pruned client model updates are used to compute the global update, which makes some neurons that should remain low values to be updated. Because the positive or negative of the cosine similarity depends only on the mutual masked part, we call the updates outside the mutual mask part $(m_a^i - m_m^i)$ out-of-control updates. As shown in Fig. 3, these out-of-control updates may allow adversaries to successfully carry out backdoor attacks. For two considered backdoor attacks, though the additive masking method receives low ASRs when 10% to 70% of clients are malicious, it can be backdoored when major (around 80%) clients are conducting attacks. By contrast, the mutual masking method remains robust against these backdoor attacks even though with 90% malicious clients. We omit DoS attacks because two masking methods have similar performance under LF and Krum attacks. Hence, we employ the mutual masking for FedER.

G.3 Impact of Occasional Normalization

Recall that in Section 2, we take an occasional value $\mathbf{n} = \max(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3)$ to normalize the magnitudes of client model updates. In Table 5, we compare the performance of different normalization

Table 5: Impact on occasional normalization on some performance metrics on MNIST-0.5.

Attack	$\mathbf{n}$	Critical Round	Critical Volume (MB)	Accuracy	ASR
No	$\mathbf{n}_1$	1400	413.44	0.90	
	$\mathbf{n}_2$	N/A	N/A	0.86	
	$\mathbf{n}_3$	530	100.88	**0.95**	
	$\max(\mathbf{n}_1, \mathbf{n}_2)$	1430	414.91	0.92	
	$\max(\mathbf{n}_1, \mathbf{n}_3)$	**420**	**93.02**	**0.95**	
	$\max(\mathbf{n}_2, \mathbf{n}_3)$	500	98.89	**0.95**	
	$\max(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3)$	430	94.06	**0.95**	
LF	$\mathbf{n}_1$	N/A	N/A	0.90	
	$\mathbf{n}_2$	N/A	N/A	0.82	
	$\mathbf{n}_3$	**480**	101.47	**0.95**	
	$\max(\mathbf{n}_1, \mathbf{n}_2)$	1610	440.14	0.91	
	$\max(\mathbf{n}_1, \mathbf{n}_3)$	530	103.86	**0.95**	
	$\max(\mathbf{n}_2, \mathbf{n}_3)$	490	101.74	**0.95**	
	$\max(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3)$	**480**	**100.76**	**0.95**	
BN	$\mathbf{n}_1$	1690	454.89	0.90	0.01
	$\mathbf{n}_2$	N/A	N/A	0.83	0.01
	$\mathbf{n}_3$	490	107.11	**0.95**	**0.00**
	$\max(\mathbf{n}_1, \mathbf{n}_2)$	1430	436.82	0.91	0.01
	$\max(\mathbf{n}_1, \mathbf{n}_3)$	460	104.66	**0.95**	0.01
	$\max(\mathbf{n}_2, \mathbf{n}_3)$	500	108.03	**0.95**	0.01
	$\max(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3)$	**430**	**102.40**	**0.95**	**0.00**
AVGBN	$\mathbf{n}_1$	1430	325.37	0.90	0.01
	$\mathbf{n}_2$	2400	547.78	0.89	0.01
	$\mathbf{n}_3$	520	136.30	**0.95**	0.01
	$\max(\mathbf{n}_1, \mathbf{n}_2)$	1040	300.60	0.92	0.01
	$\max(\mathbf{n}_1, \mathbf{n}_3)$	490	132.61	**0.95**	**0.00**
	$\max(\mathbf{n}_2, \mathbf{n}_3)$	510	135.60	**0.95**	**0.00**
	$\max(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3)$	**470**	**130.77**	**0.95**	**0.00**
Krum	$\mathbf{n}_1$	N/A	N/A	0.90	
	$\mathbf{n}_2$	N/A	N/A	0.83	
	$\mathbf{n}_3$	2000	364.52	0.93	
	$\max(\mathbf{n}_1, \mathbf{n}_2)$	2250	454.26	0.90	
	$\max(\mathbf{n}_1, \mathbf{n}_3)$	430	90.58	**0.94**	
	$\max(\mathbf{n}_2, \mathbf{n}_3)$	480	99.33	**0.94**	
	$\max(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3)$	**400**	**88.29**	**0.94**	

methods with malicious fraction of 0.3 on MNIST-0.5. Specifically, we consider seven settings for $\mathbf{n}$, which are respectively $\mathbf{n}_1$ (the normalization in Cao et al. (2021)), $\mathbf{n}_2$, $\mathbf{n}_3$, $\max(\mathbf{n}_1, \mathbf{n}_2)$, $\max(\mathbf{n}_1, \mathbf{n}_3)$, $\max(\mathbf{n}_2, \mathbf{n}_3)$, and $\max(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3)$ (the normalization in our FedER). For the no attack setting, four settings achieve the highest accuracy while $\max(\mathbf{n}_1, \mathbf{n}_3)$ and $\max(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3)$ are more efficient than the others. For four attacks, the setting in our FedER ($\max(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3)$) produces the least amount of communication to reach the critical accuracy, meanwhile, it could achieve the highest accuracy among all potential settings. Therefore, we take $\mathbf{n} = \max(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3)$ in our FedER.

G.4 Impact of Malicious Fraction

When conducting attacks, the fraction of clients that an adversary could control matters. As shown in Fig. 4, we compare FedER with FLTrust under considered attacks. For Dos attacks (LF and Krum attacks), FLTrust and our FedER achieve the similar defense result with negligible reduction in accuracy compared with no attack scenario. However, for backdoor attacks, FedER performs better than FLTrust when malicious fraction comes to 70%. For instance, under AVGBN attack, the ASR for FLTrust is 0.84, while that for FedER is 0.00. When malicious fraction further grows to 90%, FedER still remains an ASR that is almost equal to zero, showing higher robustness than FLTrust.