COORDINATED ATTACKS AGAINST FEDERATED LEARNING: A MULTI-AGENT REINFORCEMENT LEARNING APPROACH

Wen Shen, Henger Li & Zizhan Zheng
Department of Computer Science, Tulane University
{wshen9,hli30,zzheng3}@tulane.edu

ABSTRACT

We propose a model-based reinforcement learning framework against federated learning systems. Our method first approximates the distribution of the aggregated data through cooperative multi-agent coordination. It then learns an attack policy through multi-agent reinforcement learning. Experimental results demonstrate that the proposed attack framework achieves strong performance even if the server deploys advanced defense mechanisms. Our work sheds light on how to attack federated learning systems through multi-agent coordination.

1 INTRODUCTION

Federated learning (FL) is a powerful machine learning framework that allows a server to train machine learning models across multiple devices that hold local data samples, without exchanging them. Unfortunately, federated learning systems are vulnerable to threats (Lyu et al., 2020) such as model poisoning attacks (Fang et al., 2020; Bagdasaryan et al., 2020; Bhagoji et al., 2019), data poisoning attacks (Baruch et al., 2019; Fung et al., 2018; Gu et al., 2017), and inference attacks (Melis et al., 2019; Hitaj et al., 2017; Zhu et al., 2019). These attacks are effective even when the server applies robust aggregation rules such as coordinate-wise median (Yin et al., 2018), trimmed mean (Yin et al., 2018), Krum (Blanchard et al., 2017), or Bulyan (Mhamdi et al., 2018). Nevertheless, a recent study (Cao et al., 2020) shows that the server can collect a small clean training dataset to bootstrap trust to defend a variety of attacks (Fang et al., 2020; Bagdasaryan et al., 2020). Their results show that the FLTrust (Cao et al., 2020) defense is capable of achieving a high level of robustness against a large fraction of adversarial attackers. In this work, we propose a novel *multi-agent reinforcement learning* (MARL) attack framework that is effective even if the server deploys the state-of-the-art defense mechanism such as FLTrust.

Our MARL attack framework consists *distribution learning* and *policy learning* (See Figure 1). The attackers first learn a model of the aggregated data distribution, and then simulate the behavior of the server and the benign workers using the learned distribution. In doing so, the attackers jointly learn an attack policy through multi-agent reinforcement learning. Experiments on a real-world dataset demonstrate that our proposed MARL method consistently outperforms existing model poisoning attacks (Bhagoji et al., 2019; Fang et al., 2020; Xie et al., 2020b). It achieves strong attack performance even if the level of subsampling is as low as 5% or the number of attackers is as small as 5.

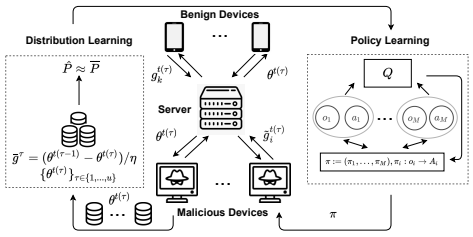

Figure 1: An overview of the MARL attack framework for federated learning.

Our method distinguishes itself in that it learns a non-myopic policy through fully cooperative multi-agent reinforcement learning. In stark contrast, existing attack methods (e.g., (Bhagoji et al., 2019; Fang et al., 2020; Xie et al., 2020b)) typically craft myopic attack strategies based on heuristics. Further, most of them consider independent attackers without coordination. Recently, a distributed backdoor attack (DBA) method is proposed in Xie et al. (2020a) where a global trigger pattern is manually decomposed into local patterns that are embedded to different attackers. Compared with DBA, our MARL method enables the attackers to jointly learn an attack policy through coordinating

the behavior of attackers in distribution learning, policy learning and attack execution, while their distributed backdoor attacks are coordinated in attack execution (i.e., trigger injection) only.

2 MARL ATTACK FRAMEWORK AGAINST FEDERATED LEARNING

Federated learning. We consider an FL setting that is similar to *federated averaging (FedAvg)* (McMahan et al., 2017). The FL system consists of a server and K devices (also known as workers) in which each device has some private data. Coordinated by the server, the set of devices cooperate to train a machine learning model within $\mathcal{T}$ epochs by solving the following problem: $\min_\theta f(\theta)$ where $f(\theta) := \sum_{k=1}^{K} p_k F_k(\theta)$, $p_k \geqslant 0$ and $\sum_k p_k = 1$. The local objective $F_k(\theta)$ is usually defined as the empirical risk over device k's local data with model parameter $\theta \in \Theta$. That is, $F_k(\theta) = \frac{1}{N_k} \sum_{j_k=1}^{N_k} \ell(\theta; (x_{j_k}, y_{j_k}))$, where N_k is the number of data samples available locally on device k, $\ell(\cdot, \cdot)$ is the loss function, and $(x_{j_k}, y_{j_k}) := z_j(k)$ is the j_kth data sample that is drawn i.i.d. from some distribution P_k. We write $\widehat{P}_k$ as the empirical distribution of the N_k data samples drawn from P_k. Let $\widehat{P} := \{\widehat{P}_k\}_{k\in[K]}$ denote the joint empirical data distribution across workers, and $N = \sum_k N_k$ the total number of data samples across workers. It is typical to set $p_k = \frac{N_k}{N}$. The empirical distribution of aggregated data of all devices is defined as $\bar{P} = \sum_{k=1}^{K} \frac{N_k}{N} \widehat{P}_k$.

The FL algorithm (see Algorithm 1 in the Appendix) works as follows: at each time step t, a random subset of w devices is selected by the server for synchronous aggregation. Each selected device $j \in [w]$ then samples a minibatch b of size B from its local data distribution $\widehat{P}_j$. It then calculates the average local gradient $g_j^{t+1} \leftarrow \frac{1}{B} \sum_{z\in b} \nabla_\theta \ell(\theta^t; z)$ and sends the gradient to the server. The server then uses an aggregation rule to compute the aggregated gradient $g^{t+1} \leftarrow Aggr(g_1^{t+1}, ..., g_K^{t+1})$ and then updates the global model parameters $\theta^{t+1} \leftarrow \theta^t - \eta g^{t+1}$ where η is the learning rate.

Threat Model. We assume that among the K workers, $M (1 \leqslant M < K)$ of them are malicious. Further, these attacker are fully cooperative and share the same goal of compromising the FL system. They are coordinated either by one leading attacker or an external agent. We refer such agent as a *leader agent*. We assume the attackers only know the global model parameters received from the server, $\{\theta^t\}$, the local training algorithm (including the batch size B) and their local data distributions $\{\widehat{P}_i\}_{i\in[M]}$. In practice, the attackers may communicate with each other to share their local information such as local data distributions, the status of whether being selected by the server or not, their unique identifiers, and each attacker's action. Such internal communication allows each selected attacker i to obtain the following information at each time step t: the number of attackers being selected by the server $m^t (0 \leqslant m^t \leqslant M)$, and its relative rank $\sigma_i^t \in \{0, ..., m^t - 1\}$ in the set of the selected attackers $[m^t]$. The relative rank is obtained by sorting the selected attackers according to their identifiers in ascending order. In addition, we assume the attackers obtain information about the total number of devices K, the number of selected workers w, and a lower bound of the total number of training epochs $\mathcal{T}$.

We consider untargeted model poisoning attacks where M cooperative attackers send crafted local updates $\{\widetilde{g}_i^t\}_{i\in[M]}$ to the server that aims to maximize the empirical loss, i.e., $\max_\theta f(\theta)$.

Markov decision process. We formulate the attackers' optimization problem as an episodic Markov decision process (MDP). We represent it as a tuple (S, A, T, r, H), where

- S is the state space. Let $\tau \in \{0, 1, ...\}$ denote the index of the attack step and $t(\tau) \in [\mathcal{T}]$ the corresponding FL epoch when at least one attacker is selected by the server. The state at step τ is defined as $s^\tau := (\theta^{t(\tau)}, \mathcal{A}^{t(\tau)})$ where $\mathcal{A}^{t(\tau)}$ is the set of attackers selected at time $t(\tau)$. We further define the *observation* of attacker i at τ as $o_i^\tau := (\theta^{t(\tau)}, m^{t(\tau)}, \sigma_i^{t(\tau)})$, which can be derived from s^τ. Let O_i denote the space of o_i^τ.
- A is the space of the attackers' joint actions. If attacker i is selected at τ, its action $a_i^\tau := \widetilde{g}_i^{t(\tau)+1} \in \mathbb{R}^d$ is the local update that attacker i sends to the server at time step $t(\tau)$, where d is the dimension of the model parameters. The only action available to an attacker not selected at $t(\tau)$ is $\perp$, indicating that the attacker does not send any information in that step. Let A_i denote the domain of attack i's actions, we have $A := A_1 \times \cdots \times A_M$.
- $T : S \times A \to \mathcal{P}(S)$ is the state transition function that represents the probability of reaching a state $s' \in S$ from the state $s \in S$ when attackers choose actions $a_1^\tau, ..., a_M^\tau$, respectively.

- $r : S \times A \times S \to \mathbb{R}_{\geqslant 0}$ is the reward function. We define the reward at step τ as $r^\tau := f(\theta^{t(\tau+1)}) - f(\theta^{t(\tau)})$, which is determined by the joint actions of attackers and shared by all the attackers. Both the transition probability T and the reward function r are jointly determined by the joint empirical distribution across workers $\widehat{P}$ (fixed but unknown to the attackers), the number of workers K, the number of workers w selected for each time step, the size of local minibatch B, the algorithm used by each worker, the aggregation rule used by the server, and the attackers' actions $a_1^\tau, ..., a_M^\tau$.
- H is the number of attack steps in each episode. H is a hyperparameter that can be adjusted, but we require $t(H) < \mathcal{T}$ so that the attackers have time to execute attacks.

The attackers' objective is to jointly find an attack policy $\pi = (\pi_1, ..., \pi_M)$ that maximizes the expected total rewards over H attack steps, i.e., $\mathbb{E}[\sum_{\tau=0}^{H-1} r^\tau]$, where $\pi_i : O_i \to \mathcal{P}(A_i)$ denotes a stationary policy of attacker i that maps its observation o_i to a probability measure over A_i. Using the definition of r^τ, this objective is equivalent to finding a policy π that maximizes $\mathbb{E}_{\theta^{t(H)}}[f(\theta^{t(H)})]$. A key obstacle to solving the MDP is that both the probability transition function T and the reward function r in the MDP depend on the joint data distribution $\widehat{P}$, which is unknown to the attackers. An important observation is that the attackers can instead learn an approximation $\widetilde{P}$ of the mixture distribution $\bar{P}$ from model updates, which is often sufficient to derive effective model poisoning attacks. The idea is that the attacker can simulate the behavior of benign agents and the server by assuming that each benign agent samples data from $\widetilde{P}$. This gives rise to a new MDP (S, A, T', r', H) where T' and r' are derived from $\widetilde{P}$. We propose to apply multi-agent reinforcement learning to solve the MDP instead of dynamic programming due to efficiency concerns.

MARL attack framework. Our proposed model-based reinforcement learning attack framework naturally consists of *distribution learning* and *policy learning*, both of which happen while federating learning is ongoing. The attackers execute the learned attack policy once it has been trained.

Distribution learning. Initially, the attackers do not perform attacks. Instead, they jointly learn the aggregated data distribution $\widetilde{P}$ using the *inverting gradients* (IG) method (Geiping et al., 2020). The IG method reconstructs images by optimizing a loss function based on the angles (+++cosine similarity) of gradients to find images that lead to a similar change in model prediction as the ground truth.

When a set of attackers $\mathcal{A}^{t(\tau)}$ are selected at $t(\tau)$, they share the model updates $\theta^{t(\tau)}$ with the leader agent. The leader agent utilizes the pooled model updates $\{\theta^{t(\tau')}\}_{\tau' \leqslant \tau}$, their gradients, the dummy data and labels (we use the attackers' local data and labels as dummy data), and the loss function ℓ as the input to compute an approximation $\widetilde{P}^\tau$ of the mixture distribution $\bar{P}$ with the IG method. We measure the Wasserstein distance of the approximated distributions between two consecutive attack steps, $\widetilde{P}^{\tau-1}$ and $\widetilde{P}^\tau$. The algorithm terminates once the Wasserstein distance is below a threshold ν. After distribution learning, the leader agent shares the learned distribution $\widetilde{P}$ to all the attackers. See Algorithm 2 in the Appendix for more details of distribution learning

Policy learning. As the attackers are fully cooperative and their actions are continuous, we adopt the framework of centralized training with decentralized execution and use the *multi-agent deep deterministic policy gradient* (MADDPG) method (Lowe et al., 2017) to train attack policies. Since the attackers share the same reward function and differs in the observations only in our setting, it suffices to train a single centralized action-value function $Q^\mu(s, a_1, ..., a_M)$ for all the attackers and a shared deterministic policy $\mu : \tilde{O} \to \tilde{A}$ where $\tilde{O} = \bigcup_i O_i$ and $\tilde{A} = \bigcup_i A_i$. As the leader agent has all the information needed to simulate system dynamics, the centralized training can be implemented by the leader agent and no communication with other attackers is needed during policy training. The learned policy is shared with all the attackers at the end of policy training and executed in a decentralized way during attacks. See Algorithm 3 in the Appendix for details of the MADDPG.

At each step τ, the leader agent generates the next state $s^{\tau+1}$ and the reward r^τ given the current state s^τ and the joint attack actions (shared by attackers), based on the learned distribution $\widetilde{P}$. The experience will then be used to update the centralized action-value function, which will in turn be used to update the joint attack policy. In doing so, the leader agent simulates the behavior of the benign workers and the server. To simulate the benign workers, the leader agent samples a minibatch that is i.i.d. drawn from the same learned distribution $\widetilde{P}$ for each benign worker and computes the respective gradients given the current model update $\theta^{t(\tau)}$. Since the attackers have no information

about N_k, the leader agent assumes that each benign worker has the same amount of data (i.e., $p_k = \frac{1}{K}$). To simulate the server's behavior, the leader agent applies the same aggregation rule as the server to compute the next model update. In practice, the leader agent is usually equipped with GPUs or other parallel computing facilities and can run multiple training episodes in parallel. Note that the total training epochs (including distribution learning and policy learning) should be less than the number of FL training epochs $\mathcal{T}$ (or a lower bound of $\mathcal{T}$) so that the attackers have time to execute the attacks. We further propose a method to reduce the state and action space as well as a gradient rescaling technique to counter the effect of robust aggregation rules (See the Appendix).

Attack execution. After policy learning, each selected attacker i sends the crafted gradients according to the learned policy μ and its local observation o_i in the remaining FL epochs. Since attacker i's local observation depends on its rank information, the set of attackers need to communicate with each other in each epoch to share the number of selected attackers as well as their unique identifiers.

3 Experiments

We conduct experiments on a real-world dataset: Fashion-MNIST (Xiao et al., 2017). We compare the performance of different attack methods: no attack (NA), random attack (RN), inner product manipulation (IPM) (Xie et al., 2020b), local model poisoning attack (LMP) (Fang et al., 2020), explicit boosting (EB) (Bhagoji et al., 2019), and three variants of our reinforcement learning attack menthod, namely, RL with a single attacker (RL), RL with multiple independent attackers (IRL), and the proposed MARL attack. See the Appendix for details of the experimental settings.

Results show that our model-based reinforcement learning methods (RL, IRL and MARL) consistently outperform existing methods such as LMP, EB and IPM (See Figure 2). The primary reason is that our methods learn a non-myopic attack policy using the learned distributions instead of crafting model updates myopically with heuristics. Among the three RL methods, the proposed MARL method performs the best due to better coordination among the attackers. Its advantage expands as the server employs more effective defense mechanisms.

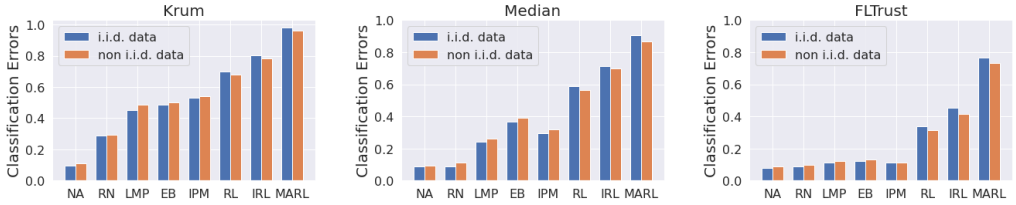

Figure 2: A comparison of classification error rates for FL with three different aggregation rules.

When the server uses advanced defense mechanism such as FLTrust, existing attack methods often perform poorly. Such situation still holds even for a relatively large number of attackers or a relatively high subsampling level (See Figure 3). In sharp contrast, our proposed MARL method achieves a classification error rate of 0.657 even when the total number of attackers is limited to 5. In comparison, the IRL method obtains 0.292 while all other methods get less than 0.15. Even if the subsampling is as low as 5%, MARL obtains a classification error rate of 0.562 while all other methods obtain an error rate below 0.2. A main factor contributing to MARL's superior performance is that it requires significantly less time for distribution learning without sacrificing the accuracy of the learned distribution due to multi-agent coordination (See Table 1 in the Appendix for a comparison).

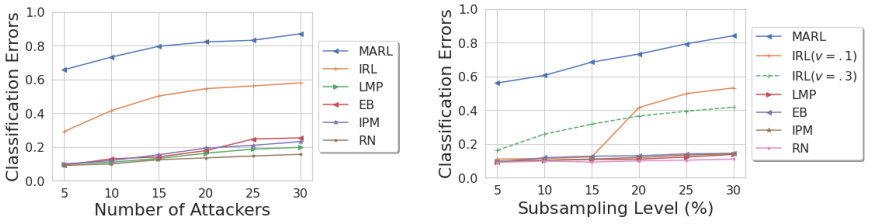

Figure 3: A comparison of classification error rates for FL with FLTrust on non-i.i.d. data.

4 Conclusion

We propose a multi-agent reinforcement learning attack framework to learn a non-myopic attack policy that can effectively compromise FL systems even with advanced defense mechanisms applied.

Our experiments show that the proposed MARL attack framework achieves a superior performance over existing attack methods due to non-myopic policy learning and better coordination among the attackers. While our focus has been on untargeted attacks against FL systems, our attack framework can be extended to targeted attacks or backdoor attacks. Another direction is to investigate novel methods to automatically learn optimal parameters such as the threshold of distribution learning and the number of policy learning epochs. Future work is needed to identify how best to do so.

ACKNOWLEDGMENTS

This work was supported in part by NSF grant CNS-1816495. We would like to thank the anonymous reviewers for their constructive feedback.

REFERENCES

Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 2938–2948. PMLR, 2020.

Moran Baruch, Gilad Baruch, and Yoav Goldberg. A little is enough: Circumventing defenses for distributed learning. *arXiv preprint arXiv:1902.06156*, 2019.

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

Peva Blanchard, Rachid Guerraoui, Julien Stainer, et al. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Advances in Neural Information Processing Systems*, pp. 119–129, 2017.

Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. FLTrust: Byzantine-robust federated learning via trust bootstrapping. *arXiv preprint arXiv:2012.13995*, 2020.

Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning attacks to byzantine-robust federated learning. In *29th USENIX Security Symposium*, pp. 1605–1622, 2020.

Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. Mitigating sybils in federated learning poisoning. *arXiv preprint arXiv:1808.04866*, 2018.

Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients–how easy is it to break privacy in federated learning? *arXiv preprint arXiv:2003.14053*, 2020.

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

Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep models under the gan: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 603–618, 2017.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*, 2017.

Lingjuan Lyu, Han Yu, and Qiang Yang. Threats to federated learning: A survey. *arXiv preprint arXiv:2003.02133*, 2020.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pp. 1273–1282. PMLR, 2017.

Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy*, pp. 691–706, 2019.

El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. The hidden vulnerability of distributed learning in byzantium. *arXiv preprint arXiv:1802.07927*, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268, 1992.

Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation from predicting 10,000 classes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1891–1898, 2014.

Han Xiao, Kashif Rasul, and Roland Vollgraf. 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. DBA: Distributed backdoor attacks against federated learning. In *International Conference on Learning Representations*, 2020a.

Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Fall of empires: Breaking byzantine-tolerant sgd by inner product manipulation. In *Uncertainty in Artificial Intelligence*, pp. 261–270. PMLR, 2020b.

Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. *arXiv preprint arXiv:1803.01498*, 2018.

Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *Advances in Neural Information Processing Systems*, pp. 14774–14784, 2019.

APPENDIX

Algorithm 1 Federated Learning

Input: Initial weight θ^0, K workers indexed by k, size of subsampling m, local minibatch size B, step size η, number of global training steps $\mathcal{T}$
Output: $\theta^{\mathcal{T}}$
Server executes:
  for $t = 0$ to $\mathcal{T} - 1$ **do**
    $[m] \leftarrow$ randomly select m workers from K workers
    for each worker $j \in [m]$ **in parallel do**
      $g_j^{t+1} \leftarrow$ **WorkerUpdate**(j, θ^t)
    end for
    $g^{t+1} \leftarrow Aggr(g_1^{t+1}, ..., g_K^{t+1})$
    $\theta^{t+1} \leftarrow \theta^t - \eta g^{t+1}$
  end for
WorkerUpdate(j, θ):
  Sample a minibatch b of size B
  $g \leftarrow \frac{1}{B} \sum_{z \in b} \nabla_\theta \ell(\theta, z)$
  return g to server

More details about distribution learning. At the beginning of distribution learning, the leader agent uses all the M attackers' local data to serve as the dummy data D_{dummy}. The attackers only need access to the gradients of the aggregated data, not each individual device's gradients. The former can be derived using common information by $\bar{g}^\tau = (\theta^{t(\tau-1)} - \theta^{t(\tau)})/\eta$, where $\theta^{t(\tau-1)}$ and $\theta^{t(\tau)}$ are previous and current FL model parameters received from the server, and η is the server's learning rate. For each epoch that at least one attacker is selected, the leader agent then uses the inverting gradients algorithm (Geiping et al., 2020) to reconstruct the normal workers' data based

on the dummy data and the derived gradient $\bar{g}$ by minimizing a loss function that is adapted from the inverting gradients algorithm (Geiping et al., 2020). The optimization objective aims to reconstruct data points by iteratively solving:

$$\arg \min_{x \in \mathbb{R}^n, y \in \mathbb{R}} 1 - \frac{\langle \nabla_\theta \ell(\theta; (x, y)), \bar{g} \rangle}{||\nabla_\theta \ell(\theta; (x, y))|| \cdot ||\bar{g}||} + \beta TV(x), \tag{1}$$

where n is the dimension of x, $TV(x)$ is the total variation (Rudin et al., 1992) of x and β is total variation parameter. For each data point, the inverting gradients algorithm terminates after max_iter iterations. After data reconstruction in each epoch τ, all the reconstructed data will be added to the set of dummy data. The approximated mixture distribution $\widetilde{P}$ consists of the reconstructed data and the M attackers' local data. To determine whether an approximated distribution is sufficiently accurate, we measure the Wasserstein distance $W(\widetilde{P}(\tau-1), \widetilde{P}(\tau))$ between the approximated distributions of the two consecutive attack steps. We use the data points across all the devices in the previous step and the current step to approximate the previous and current mixture distributions. The distribution learning algorithm terminates when the Wasserstein distance is below a predefined threshold ν.

Algorithm 2 Distribution Learning

Input: Wasserstein distance threshold for termination ν, number of iterations for inverting gradients max_iter, step size for FL η and step size for inverting gradients η', model parameters $\{\theta^{t(\tau)}\}$

Output: $\widetilde{P}$

Wasserstein distance $W(\widetilde{P}(-1), \widetilde{P}(0)) \leftarrow \infty$, $\tau \leftarrow 0$

$D_{dummy} \leftarrow M$ attackers' local data

while $W(\widetilde{P}(\tau-1), \widetilde{P}(\tau)) > \nu$ **do**

  $\tau \rightarrow \tau + 1$

  Compute the aggregated gradients using $\bar{g}^\tau \leftarrow (\theta^{t(\tau-1)} - \theta^{t(\tau)})/\eta$

  for $(x, y) \in D_{dummy}$ **do**

    $(x_0, y_0) \leftarrow (x, y)$

    for $i = 0$ to $max\_iter - 1$ **do**

      $\nabla_\theta \ell(\theta^{t(\tau)}; (x_i, y_i)) \leftarrow \partial \ell(\theta^{t(\tau)}; (x_i, y_i))/\partial \theta$

      $\mathcal{L}_i \leftarrow 1 - \frac{\langle \nabla_\theta \ell(\theta^{t(\tau)}; (x_i, y_i)), \bar{g}^\tau \rangle}{||\nabla_\theta \ell(\theta^{t(\tau)}; (x_i, y_i))|| \cdot ||\bar{g}^\tau||} + \beta TV(x_i)$

      $x_{i+1} \leftarrow x_i - \eta' \nabla_{x_i} \mathcal{L}_i, y_{i+1} \leftarrow y_i - \eta' \nabla_{y_i} \mathcal{L}_i$

    end for

  end for

  Add newly all reconstructed data points (x, y) to D_{dummy}

  Compute the current approximated mixture distribution $\widetilde{P}(\tau)$ with all the reconstructed data points and the M attackers' local data

  Compute the Wasserstein distance $W(\widetilde{P}(\tau-1), \widetilde{P}(\tau))$

end while

Cooperative multi-agent deep deterministic policy gradient. We adapt the *Multi-Agent Deep Deterministic Policy Gradient* (MADDPG) (Lowe et al., 2017) algorithm to our fully cooperative setting with subsampling as follows. We let all the agents (attackers) share the same action-value function $Q^\mu(s, a_1, ..., a_M)$ and the same policy $\mu_\phi(\cdot|o_i)$ with parameters ϕ. Note that although the attackers share the same policy, their actions in each step vary due to the unique observations they receive. Using the chain rule, we can derive the gradient of the expected return $J(\phi) = \mathbb{E}[\sum_{\tau=0}^{H-1} r^\tau]$ as follows:

$$\nabla_\phi J(\phi) = \mathbb{E}_{s,a \sim \mathcal{D}}[\sum_{i=1}^{M} \nabla_\phi \mu_\phi(o_i^j) \nabla_{a_i} Q^\mu(s^j, a_1^j, \ldots, a_i, \ldots, a_M^j)|_{a_i = \mu(o_i^j)}],$$

where the experience reply buffer $\mathcal{D}$ contains tuples $(s, r, s', a_1, \ldots, a_M)$. Note that for attacker i not selected by the server in a certain step, its action does not affect the Q^μ value, which implies that $\nabla_{a_i} Q^\mu = 0$ for any a_i when the state indicates that attacker i is not sampled. Hence, the policy

gradient formula makes sense even when subsampling is applied. Similar to Lowe et al. (2017), the shared action-value function Q^{μ} is updated by minimizing the loss:

$$\mathcal{L}(\phi) = \mathbb{E}_{s,a,r,s'}\{[y - Q^{\mu}(s, a_1, \ldots, a_M)]^2\}, \quad y = r + Q^{\mu'}(s', a'_1, \ldots, a'_M)|_{a'_k=\mu'(o_k)}.$$

where μ' is the target policy with delayed parameters ϕ'.

Algorithm 3 Cooperative Multi-Agent Deep Deterministic Policy Gradient

for episode $= 1$ to $max\_episode$ **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state s
  for $\tau = 1$ to H **do**
    for each attacker i, select action $a_i = \mu_\phi(o_i) + \mathcal{N}_\tau$ w.r.t the current policy and exploration
    Execute actions $a = (a_1, \ldots, a_M)$ and observe reward r and new state s'
    Store (s, a, r, s') in replay buffer $\mathcal{D}$
    $s \leftarrow s'$
    Sample a random minibatch of C samples (s^j, a^j, r^j, s'^j) from $\mathcal{D}$
    Set $y^j = r^j + Q^{\mu'}(s'^j, a'_1, \ldots, a'_M)|_{a'_k=\mu'(o_k^j)}$
    Update critic by minimizing the loss:
      $\mathcal{L}(\phi) = \frac{1}{C}\sum_j [y^j - Q^{\mu}(s^j, a_1^j, \ldots, a_M^j)]^2$
    Update actor using the sampled policy gradient:
      $\nabla_\phi J \approx \frac{1}{C}\sum_j \sum_i \nabla_\phi \mu_\phi(o_i^j) \nabla_{a_i} Q^{\mu}(s^j, a_1^j, \ldots, a_i, \ldots, a_M^j)|_{a_i=\mu(o_i^j)}$
    Update target network parameters $\phi' \leftarrow \alpha\phi + (1-\alpha)\phi'$
  end for
end for

State and action space reduction. When we train a small neural network with the federated learning system, it is natural to use $(\theta^{t(\tau)}, \mathcal{A}^{t(\tau)})$ as the state, and the gradient $\widetilde{g}_i^{t(\tau)+1}$ as the action. When we use the federated learning system to train a large neural network, however, this approach does not scale as it results in extremely large search space that requires both large runtime memory and long training time, which is usually prohibitive. To solve this problem, we propose to approximate the state $(\theta^{t(\tau)}, \mathcal{A}^{t(\tau)})$ and the action $g_i^{t(\tau)+1}$ for high-dimensional data. To approximate the state, we use parameters of the last hidden layer of the current neural network model to replace $\theta^{t(\tau)}$ in state $(\theta^{t(\tau)}, \mathcal{A}^{t(\tau)})$. This is because because the last hidden layer passes on values to the output layer and typically carry information about important features of the model (Sun et al., 2014). Note that the true state is still the full FL model that determines transition probabilities and rewards. We further define the action a_i^τ as a one-dimension scaling factor $a_i^\tau \in [-1, 1]$, and set $\widetilde{g}_i^{t(\tau)+1} = a_i^\tau \times g_i^{t(\tau)+1}$, which is used to compute the next state and reward.

Gradient rescaling. To reduce the chance that the gradients from the attackers are filtered out by the defense mechanisms (e.g., coordinate-wise median, Krum, or FLTrust), we further rescale the gradients. When computing the gradients to be sent to the server, the attack method first calculates the maximum value $\zeta_{\max}$ and the minimum value $\zeta_{\min}$ that occur in any dimension of $\nabla_\theta l(\theta^{t(\tau)}; z)$ for any z in the data samples generated in the distribution learning stage. The original scaling factor a^τ is in $[-1, 1]$. The rescaled scaling factor $\tilde{a}^\tau = a^\tau \times \frac{\zeta_{\max}-\zeta_{\min}}{|\zeta_{\max}|} \times 0.5 + \frac{\zeta_{\max}+\zeta_{\min}}{|\zeta_{\max}|} \times 0.5$.

Training time. The training time mainly comes from learning an accurate estimation of the mixture data distribution $\bar{P}$, and learning an attack policy μ. For a typical subsampling rate (e.g., 20%), the distribution learning stage usually takes only a few federated learning epochs of time before convergence. The distribution learning time also depends on the degree of data heterogeneity. (We follow the definition of the degree of non-i.i.d. in (Fang et al., 2020)). A higher degree of heterogeneity requires a longer time for distribution learning. The time for policy learning in stage two largely depends on the number of training epochs used to simulate the FL dynamics in each training episode, and the number of training episodes used by the attackers. The larger of the two factors, the more training time is required. To this end, we assume that the leader agent has access to GPUs or other parallel computing facilities so that it can run multiple training episodes in parallel. We further

observe that with the large number of episodes trained in parallel, there is no need to simulate the complete FL process for $\mathcal{T}$ epochs. It suffices to consider much shorter epochs in practice. Another important observation is that the length of a simulating epoch is typically shorter than that of a true epoch in FL since the simulated process involves no communications with the devices.

Dataset. Fashion-MNIST includes 60,000 training examples and 10,000 testing examples, where each example is a 28×28 grayscale image, associated with a label from 10 classes. We consider a federated learning system that consists of 100 workers. We assume that there are 10 attackers unless stated otherwise. For the *i.i.d.* setting, we randomly split the dataset into 100 groups, each of which consists of the same number of training samples and the same number of testing samples. For the *non-i.i.d.* setting, we follow the method of Fang et al. (2020) to quantify the heterogeneity of the data. We split the workers into 10 groups and model the non-i.i.d. federated learning by assigning a training instance with label c to the c-th group with probability pr and to all the groups with probability $1-pr$. A higher pr indicates a higher level of heterogeneity. For for non-i.i.d. data, we set the degree of non-i.i.d. $pr = 0.5$ as the default setup.

Federated learning settings. We adopt the following default parameters for the federate learning models: learning rate $\eta = 0.01$, number of total devices = 100, number of attackers = 10 (0 for NA and 1 for RL), subsampling level = 20%, and number of total epochs = 100. We train a neural network classifiers consisting of 8×8, 6×6, and 5×5 convolutional filter layers with ELU activations followed by a fully connected layer and softmax output. We set the local batch size $B = 128$. We implement the FL model with PyTorch (Paszke et al., 2019) and run all the experiments on the same 2.30GHz Linux machine with NVIDIA Tesla P100 GPU.

Distribution learning settings. In our experiments, we set the step size for inverting gradients $\eta' = 0.001$ the total variation parameter $\beta = 0.01$, the threshold for distribution learning $\nu = 0.1$ and the number of iterations for inverting gradients $max\_iter = 4,800$ (same as (Geiping et al., 2020)), and learn the data distribution from scratch.

Policy learning settings. We adopt a PyTorch implementation of the MADDPG and use the default setting for the hyper-parameters as that in Lowe et al. (2017) in our experiments unless stated otherwise. The default parameters are described as the following : number of policy training epochs = 30, number of policy training episodes $max\_episode$ = $6,000$. We train the $6,000$ episodes in parallel. Note that instead of fixing H, which corresponds to the number of attack steps in policy learning when at least one attacker is selected so that the leader agent can observe the new state and reward, we fix the number of simulating epochs in each episode in policy learning, including those epochs when no attacker is selected. Thus, H varies across episodes in experiments.

Table 1: A comparison of training time (in number of epochs) in each stage for RL-based attacks (number of attackers = 10, subsampling level = 20%, all other parameters are set as the default settings).

Method	Distribution learning	Policy learning	Execution	Classification error
MARL	6	30	64	0.735
IRL($\nu = 0.1$)	54	30	16	0.416
IRL($\nu = 0.3$)	28	30	42	0.367