Learning to Attack Federated Learning: A Model-based Reinforcement Learning Attack Framework

Anonymous Author(s)
Affiliation
Address
email

Abstract

We propose a model-based reinforcement learning framework to derive untargeted poisoning attacks against federated learning (FL) systems. Our framework first approximates the distribution of the clients' aggregated data using model updates from the server. The learned distribution is then used to build a simulator of the FL environment, which is utilized to learn an adaptive attack policy through reinforcement learning. Our framework is capable of learning strong attacks automatically even when the server adopts a robust aggregation rule. We further derive an upper bound on the attacker's performance loss due to inaccurate distribution estimation. Experimental results on real-world datasets demonstrate that the proposed attack framework significantly outperforms state-of-the-art poisoning attacks. This indicates the importance of developing adaptive defenses for FL systems.

1 Introduction

Federated learning (FL) is a promising machine learning framework that allows multiple devices with private data to jointly train a learning model (coordinated by a server) without sharing their local data. It has recently been applied to consumer digital products [27], credit risk prediction [1], drug discovery [28], and digital health [33]. However, federated learning systems are vulnerable to adversarial attacks [25] such as model poisoning attacks [4, 42, 11, 2], data poisoning attacks [3, 14, 16], and inference attacks [29, 18, 49]. To this end, various robust aggregation rules such as coordinate-wise median [44], trimmed mean [44], Krum [5], Bulyan [30], and FLTrust [7] have been proposed. However, these defenses are mainly evaluated against manually crafted myopic attack policies [36]. Their robustness in the face of advanced attacks remains unknown.

Due to the distributed nature of FL systems, a malicious device typically has limited knowledge about benign devices and system dynamics. To fully reveal the vulnerabilities of FL systems, it is therefore crucial to develop strong attacks that can best utilize the limited global knowledge. In this work, we take a first step in this direction by considering the white-box attack setting where the attacker has some global knowledge about the FL system and the server's algorithm, but has no access to the private data of benign devices, a reasonable assumption for real-world FL systems. To derive strong adaptive attacks, we propose to leverage the power of model-based reinforcement learning (RL) by integrating *distribution learning* and *policy learning*. A key observation of our approach is that although accurate information about individual devices can be hard to obtain in FL, it is often possible to infer their aggregated data distribution from publicly available model updates, which is sufficient to derive strong attacks. In particular, the set of malicious devices first cooperatively estimate the aggregated data distribution through gradient matching [18, 15]. The learned distribution is then used to build a simulator of the FL environment, which is utilized to derive an adaptive attack policy through reinforcement learning. We focus on untargeted model poisoning in this work, where

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

the malicious devices aim to reduce the accuracy of the global model as much as possible by sending crafted gradient information to the server. However, our proposed framework can potentially be applied to other types of attacks in both the white-box and the more challenging black-box settings.

Our model-based approach distinguishes from existing work on reinforcement learning based adversarial attacks against machine learning systems [38, 48]. In particular, we consider a more realistic threat model where the attacker might not always be selected due to subsampling nor does it have prior information about the distribution of the aggregated data. The attackers need to efficiently learn the distribution along the federated learning process in real time. In contrast, previous works typically assume more powerful attackers that can attack at any time and have full knowledge about the environment. Thus, they typically adopt a purely model-free approach, which is infeasible in attacking FL systems due to the large number of samples needed to be effective.

Our contributions. We advance the state-of-the-art in the following aspects. First, we propose a novel reinforcement learning attack framework against federated learning systems by integrating distribution learning and policy learning. Second, we theoretically quantify the effect of inaccurate distribution learning and heterogeneous local data distributions on the optimality of policy learning. Third, our experiments on real-world datasets demonstrate that our RL-based attack method consistently outperforms existing model poisoning attacks [4, 11, 42] and significantly reduces the global model accuracy even when robust aggregation rules are applied. These findings indicate the importance of developing adaptive defenses for FL systems.

2 Approach Overview

In this section, we describe the federated learning setting used in our work, the threat model, and the proposed RL-based attack framework.

Federated learning. We consider an FL setting that is similar to *federated averaging (FedAvg)* [26]. The FL system consists of a server and K workers (also known as devices or clients) in which each worker has some private data. Coordinated by the server, the set of workers cooperate to train a machine learning model within $\mathcal{T}$ epochs by solving the following problem: $\min_\theta f(\theta)$ where $f(\theta) := \sum_{k=1}^{K} p_k F_k(\theta)$. Here $F_k(\cdot)$ is the local objective of worker k and p_k is the weight assigned to worker k and satisfies $p_k \geqslant 0$ and $\sum_k p_k = 1$. The local objective $F_k(\theta)$ is usually defined as the empirical risk over worker k's local data with model parameter $\theta \in \Theta$. That is, $F_k(\theta) = \frac{1}{N_k} \sum_{j=1}^{N_k} \ell(\theta; z_{jk})$, where N_k is the number of data samples available locally on worker k, $\ell(\cdot;\cdot)$ is the loss function, and $z_{jk} := (x_{jk}, y_{jk})$ is the jth data sample that is drawn *i.i.d.* from some distribution P_k. It is typical to set $p_k = \frac{N_k}{N}$, where $N = \sum_k N_k$ is the total number of data samples across workers.

If all the local workers' data distributions are the same (i.e., $P_k = P_{k'}$ for all $k, k' \in [K]$), we call the workers' data are *i.i.d.*; otherwise, the data are *non-i.i.d.*. We write $\widehat{P}_k$ as the empirical distribution of the N_k data samples drawn from P_k, and let $\widehat{P} := \sum_{k=1}^{K} \frac{N_k}{N} \widehat{P}_k$ denote the mixture empirical distribution across workers.

The FL algorithm (see Algorithm 1 in Appendix B.2) works as follows: at each time step t, a random subset $\mathcal{S}^t$ of size w is uniformly sampled without replacement from the workers set $[K]$ by the server for synchronous aggregation [23]. The process of selecting workers for aggregation is called *subsampling*. Let $\kappa = w/N$ denote the *subsampling rate*, i.e., the ratio of the selected workers number w to the total number of workers K. Each selected worker $k \in [w]$ then samples a minibatch b_k of size B from its local data distribution $\widehat{P}_k$. The worker then calculates the average local gradient $g_k^{t+1} \leftarrow \frac{1}{B} \sum_{z \in b_k} \nabla_\theta \ell(\theta^t; z)$ and sends the gradient to the server. The server then uses an aggregation rule to compute the aggregated gradient $g^{t+1} \leftarrow Aggr(g_{k_1}^{t+1}, ..., g_{k_w}^{t+1})$ where $k_i \in \mathcal{S}^t$, and then updates the global model parameters $\theta^{t+1} \leftarrow \theta^t - \eta g^{t+1}$ where η is the learning rate. The newly updated model parameters θ^{t+1} is then sent to the selected workers to perform another FL iteration.

Threat Model. We assume that among the K workers, $M (1 \leqslant M < K)$ of them are malicious. Let $\mathcal{A}$ denote the set of malicious workers. These attackers are assumed to be *fully cooperative* and share the same goal of compromising the FL system. We consider untargeted

model poisoning attacks in this work where the M cooperative attackers send crafted local updates $\{\widetilde{g}_k^t\}_{k\in\mathcal{A}}$ to the server in order to maximize the empirical loss $f(\theta)$. They are coordinated either by one leading attacker or an external agent. We refer such agent as a *leader agent*.

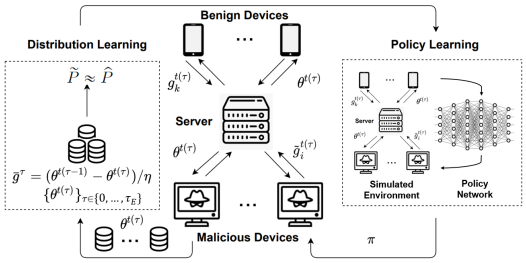


Figure 1: An overview of the RL-based attack framework against federated learning.

We assume that the attackers 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 $\{\widehat{P}_k\}_{k\in\mathcal{A}}$ (but not the benign workers' local data distributions). We further assume that the attackers obtain information about the server's training algorithm (i.e., the white-box attack setting). This information includes the server's learning rate η, the subsampling rate κ, the total number of workers K, the aggregation rule $Aggr$, and the total number of training epochs $\mathcal{T}$.

RL-based online attack framework. Our attack framework consists of the following three phases.

- **Distribution learning**: The malicious workers first jointly learn an approximation of $\widehat{P}$ from the model updates $\{\theta^t\}$ received from the server, using a gradient leakage based inference attack [15, 49].
- **Policy learning**: The leader agent then builds a simulator of the FL environment using the attackers' local data and the learned distribution. An optimal attack policy is then derived through reinforcement learning using data sampled from the simulator. Note that policy learning can start together with distribution learning and continue after a reasonable distribution is learned.
- **Attack execution**: The learned policy is distributed to all the malicious workers to generate attack actions. Note that attack execution can start once an initial policy is learned, which can be updated during attack execution.

We note that all the three phases happen while the federated learning process is ongoing, thus the lengths of these phases are important hyperparameters to be determined. For example, with more observations, an attacker can learn a more accurate distribution, which will help obtain a better attack policy. However, when the total time window available to attacks is limited, a longer distribution learning phase reduces the attack opportunities in Phase 3. Compared with a purely model-free approach, our model-based approach is more sample efficient, which is especially important for federated learning as a malicious worker can only attack when it is sampled by the server.

3 Model-based reinforcement learning attack framework

In this section, we first formulate the model poisoning attack problem as a Markov decision process (MDP). We then discuss our model-based reinforcement learning attack framework in more details.

3.1 Attackers' problem as a Markov decision process

The attackers' problem is formulated as an undiscounted MDP, denoted by $\mathcal{M} = (S, \boldsymbol{A}, T, r, H)$, 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 $t(\tau)$, which is shared between all malicious workers.
- $\boldsymbol{A} = A^M$ is the space of the attackers' joint actions where each attacker share the same action space A. If attacker i is selected at $t(\tau)$, 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.
- $T : S \times \boldsymbol{A} \to \mathcal{P}(S)$ is the state transition function that represents the probability of reaching a state $s' \in S$ from the current state $s \in S$ when attackers choose actions $a_1^\tau, ..., a_M^\tau$, respectively.

- $r : S \times \boldsymbol{A} \times S \to \mathbb{R}_{\geqslant 0}$ 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 determined by the joint empirical distribution across workers $\{\widehat{P}_k\}_{k\in[K]}$ (fixed but unknown to the attackers) and other system parameters.
- 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 will be able to execute attacks.

The attackers' goal is to find a joint 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 $\pi_i : S \to \mathcal{P}(A)$ denotes a stationary policy of attacker i that maps the state to a probability measure over A. Using the definition of r^τ, this objective is equivalent to finding a joint 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 depend on the joint data distribution $\{\widehat{P}_k\}_{k\in[K]}$, which is unknown to the attackers. Although model-free reinforcement learning can bypass this difficulty, it requires a large amount of samples to be effective, which is infeasible in the online attacking scenario we consider. We therefore adopt model-based reinforcement learning as a principled approach for designing adaptive attacks in the online setting.

An important observation of our model-based approach is that although the joint empirical distribution $\{\widehat{P}_k\}_{k\in[K]}$ is unknown, the attackers can learn an approximation of the mixture distribution $\widehat{P} = \sum_{k=1}^{K} \frac{N_k}{N} \widehat{P}_k$, denoted by $\widetilde{P}$, from model updates shared by the server, which is often sufficient to 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 $\widetilde{\mathcal{M}} = (S, \boldsymbol{A}, T', r', H)$ where T' and r' are derived from $\widetilde{P}$. Thus, our proposed model-based reinforcement learning attack framework naturally consists of the distribution learning, policy learning, and attack execution phases.

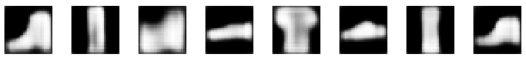
3.2 Distribution learning

Initially, the attackers do not perform model-poisoning attacks. Instead, they jointly learn a mixture distribution $\widetilde{P}$ from the model updates $\{\theta^t\}$ using a gradient leakage based inference attack [15, 49]. Various gradient leakage attacks have been proposed in the literature. In this work, we adapt the *inverting gradients* (IG) method [15] to distribution learning. The IG method reconstructs data samples by optimizing a loss function based on the angle (i.e., cosine similarity) between the gradient generated from true data and that from the reconstructed data. The primary goal of IG is to reconstruct the original data samples, which is more ambitious than what the attackers need in our setting. On the other hand, recent works on gradient leakage including IG have focused on the server side, where the true gradients of each individual worker can be easily obtained from model updates. In contrast, the attackers only obtain approximated and aggregated gradient information from consecutive model updates received from the server, due to model aggregation and subsampling. Despite these differences, our experiment results show that the $\widetilde{P}$ learned using IG can help derive an effective attacker policy (see Figure 4(a)).

(a) Reconstructed MNIST noisy images

(b) Reconstructed MNIST denoised images

(c) Reconstructed Fashion-MNIST noisy images



(d) Reconstructed Fashion-MNIST denoised images

Figure 2: Examples of reconstructed images (before and after denoising)

As shown in Algorithm 2 in Appendix B.2, for each epoch $t(\tau)$ that at least one attacker is selected, the leader agent obtains the model update from one of the attackers and calculates the batch-level gradient as $\bar{g}^\tau := (\theta^{t(\tau-1)} - \theta^{t(\tau)})/(\eta(t(\tau) - t(\tau-1)))$. The leader agent then starts with (randomly generated) dummy data and dummy labels, which are updated iteratively by solving the following optimization problem: $\arg\min_{x\in\mathbb{R}^n, y\in\mathbb{R}} 1 - \frac{\langle \nabla_\theta \ell(\theta;(x,y)), \bar{g}^\tau \rangle}{\|\nabla_\theta \ell(\theta;(x,y))\| \cdot \|\bar{g}^\tau\|} + \beta \mathrm{TV}(x)$, where n is the dimension of data sample x, $\mathrm{TV}(x)$ is the total variation [34] of x, and β is a fixed parameter. The process

terminates after max_iter iterations, then outputs the updated data as the reconstructed data samples. We observed that although the data samples generated by IG resemble true samples, they contain a small amount of noise as shown in Figures 2(a) and 2(c), making the learned distribution less representative of the true distribution. To reduce noise, we adapt the method of denoising auto-encoder [40]. We utilize the clean data owned by attackers and add Gaussian noise to them to simulate the noise in the reconstructed data samples. The clean data and the synthetic noisy data are then paired to train an auto-encoder for denoising, which is then used to remove the noise in reconstructed data samples (see Figures 2(b) and 2(d)). The approximated mixture distribution $\widetilde{P}(\tau)$ consists of the reconstructed data up to $t(\tau)$ and the M attackers' local data. The distribution learning phase starts at the first FL epoch and continues for τ_E steps. The learned distribution is shared with all the attackers.

In addition to the IG method, several other existing methods can also be utilized. For instance, the *deep leakage from gradients* method (DLG) [49], the latent projection method [20], the DeepInversion method [46] and the *GradInversion* method [45]. It is worth noting that the GradInversion method is capable of reconstructing individual images with high fidelity from averaging gradients even for complex datasets like ImageNet [10], deep networks, and large batch sizes. These approaches can potentially be adopted to learn $\widetilde{P}$ in more challenging settings.

3.3 Policy learning

Once the leader agent obtains the approximated distribution $\widetilde{P}$, it can simulate the behavior of the server and that of normal workers. In particular, to simulate the behavior of a normal worker in each FL training step, a minibatch of size B is $i.i.d.$ sampled from $\widetilde{P}$. With experiences sampled from the simulated environment, the leader agent can learn a joint attack policy that maximizes the empirical loss using a state-of-the-art (deep) reinforcement learning algorithm, e.g., TD3 [13] or PPO [35].

Note that the leader agent does not need to wait until a good distribution has been learned to start training the policy. Instead, policy learning can start together with distribution learning. Initially, the leader agent uses the attackers' local data to train the policy, which will be continuously updated while more data samples are being generated. To reduce the training overhead, we assume that all the malicious workers share the same attack policy (i.e., $\pi_1 = \pi_2 = \cdots = \pi_M$) in this work, which achieves good attack performance in our experiments. Extension to general joint policies will be considered in our future work.

When we train a small neural network with federated learning, it is natural to use $(\theta^{t(\tau)}, \mathcal{A}^{t(\tau)})$ as the state, and the gradient $\widetilde{g}^{t(\tau)}$ 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 an 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 compress the state and action spaces for high-dimensional models as follows.

To compress the state space, we first observe that as the set of attackers are fully cooperative and share the same policy, there is no need to distinguish them in the state when the server does not track the behavior of individual workers. Thus, we replace $\mathcal{A}^{t(\tau)}$ by the number of attackers sampled in $t(\tau)$, denoted by $m^{t(\tau)}$. Further, instead of using the entire set of model parameters $\theta^{t(\tau)}$ in the state. the parameters of the last hidden layer of the current neural network model is used instead. This is because the last hidden layer passes on values to the output layer and typically carries information about important features of the model [37]. Note that the approximated state is used to define the policy only. The true state is still the full FL model that determines transition probabilities and rewards.

It is more challenging to compress the action space. We observe that a policy that manipulates the model parameters of the last hidden layer only works well for certain aggregation rules such as Krum and coordinate-wise median. However, it becomes less effective for stronger defenses such as coordinate-wise median with clipping. Given that it is challenging to identify a small subset of model parameters that can lead to most damage to model accuracy when manipulated, we adopt a different approach in this work.

The main idea is to search for a model update direction that can lead to a large empirical loss using gradient ascent, with its parameters identified by reinforcement learning. To this end, we define the

local search objective as $L(\theta) := F(\theta) + \lambda \cos(\theta - \theta^{t(\tau)}, g(\theta^{t(\tau)}))$ where $F(\theta) = \mathbb{E}_{z\sim\widetilde{P}}[\ell(\theta; z)]$ models the empirical loss and $g(\theta^{t(\tau)}) = \mathbb{E}_{z\sim\widetilde{P}}[\nabla_\theta \ell(\theta^{t(\tau)}; z)]$ captures the average update direction from normal devices, both are estimated from $\widetilde{P}$. The parameter λ is used to balance the loss and the deviation from the normal direction. To solve the problem, we start with $\theta = \theta^{t(\tau)}$ and generate G trajectories, where each trajectory involves E model updates. Each model update involves a single gradient ascent step using a minibatch sampled from $\widetilde{P}$. Let θ_k denote the new model parameters found by the k-th trajectory after E updates. The attacker i's action is then computed as $\widetilde{g}_i^{t(\tau)} = -\frac{\gamma}{G}\sum_{k=1}^{G}(\theta_k - \theta^{t(\tau)})$, which is identical to all the attackers. The parameter γ is used to control the magnitude of the crafted gradient, which is needed as most robust aggregation rules apply a certain type of filtering rule to mitigate the effect of malicious attacks. We assume that G is fixed while γ, E, λ are parameters to be learned by reinforcement learning. Thus, the action space for the attackers' MDP becomes a 3-dimensional real space.

3.4 Attack execution

Both distribution learning and policy learning can start from the first epoch of federating learning and continue while federating learning is ongoing. The simulated environment is updated when a new estimated distribution $\widetilde{P}$ is learned. Although the attackers may choose to start attacking during distribution learning, we observe that this can blur the gradient information and make distribution learning less accurate. Thus, we assume that each attacker starts attacking once the distribution learning phase is finished, and applies the latest learned attack policy during the remaining epochs of the federated learning process. During attack execution, each selected attacker first notifies other attackers so that every one knows the number of attackers that are sampled in that epoch. Each selected attacker then generates a crafted gradient according to the process described above with the parameters obtained from the latest learned policy.

The attackers' total training time (including distribution learning and policy learning) should be significantly less than the total FL training time so that the attackers have time to execute the attacks. In real-world FL training, the server usually must wait for some time (typically ranging from 1 minute to 10 minutes) before it receives responses from the clients [43, 6, 19]. In contrast, the leader agent does not incur such time cost in training attackers' policies using a simulated FL environment. Therefore, an epoch in policy learning is typically much shorter than an FL epoch, making it possible to train the attack policy with a large number of episodes. In addition, the leader agent is usually equipped with GPUs, or other parallel computing facilities and can run multiple training episodes in parallel [8].

4 Impact of Inaccurate Distribution Learning and Data Heterogeneity

Our model-based RL attack employs the estimated data distribution $\widetilde{P}$ to simulate the behavior of benign workers, which can suffer from two types of errors. First, $\widetilde{P}$ can be far away from the true mixture distribution $\widehat{P}$ due to inaccurate distribution learning. Second, benign workers may vary in their local data distributions $\widehat{P}_k$, which cannot be fully captured by a single mixture distribution. In this section, we study how the attack performance is affected by these two factors, which provides insights on properly distributing resources between the three phases of our attack method.

Our analysis is adapted from recent works that study the impact of model inaccuracy on the performance of model-based reinforcement learning [47, 24, 48] by addressing two key challenges. First, we need to establish the connection between the inaccuracy in data distribution $\widetilde{P}$ and the inaccuracy in the corresponding MDP as both the reward function and the transition dynamics depend on $\widetilde{P}$. Second, although there are different ways to measure the distance between two models [47], it makes more sense to use the Wasserstein distance to measure the distance between two data distributions. This, however, requires bounding the Lipschitz constant of the optimal value function [47]. Although this is a challenging task for general RL tasks, we are able to show that this is indeed the case in our setting under the following assumptions. The first assumption models the inaccuracy of distribution learning as well as the heterogeneity of benign worker's local data.

Assumption 1. *$W_1(\widetilde{P}, \widehat{P}_k) \leqslant \delta$ for any benign worker k.*

We further need the following standard assumptions on the loss function.

Assumption 2. *Let Z denote the domain of data samples across all the workers. For any $s_1, s_2 \in S$ and $z_1, z_2 \in Z$, the loss function $\ell : S \times Z \to \mathbb{R}$ satisfies:*

1. $|\ell(s_1, z_1) - \ell(s_2, z_2)| \leqslant L\|(s_1, z_1) - (s_2, z_2)\|_2$ *(Lipschitz continuity w.r.t. s and z);*
2. $\|\nabla_s \ell(s_1, z_1) - \nabla_s \ell(s_1, z_2)\|_2 \leqslant L_z\|z_1 - z_2\|_2$ *(Lipschitz smoothness w.r.t. z);*
3. $\ell(s_2, z_1) \geqslant \ell(s_1, z_1) + \langle \nabla_s \ell(s_1, z_1), s_2 - s_1\rangle + \frac{\alpha}{2}\|s_2 - s_1\|_2^2$ *(strongly convex w.r.t. s);*
4. $\ell(s_2, z_1) \leqslant \ell(s_1, z_1) + \langle \nabla_s \ell(s_1, z_1), s_2 - s_1\rangle + \frac{\beta}{2}\|s_2 - s_1\|_2^2$ *(strongly smooth w.r.t. s);*
5. *$\ell(\cdot, \cdot)$ is twice continuously differentiable with respect to s.*

where $\|(s_1, z_1) - (s_2, z_2)\|_2^2 = \|s_1 - s_2\|_2^2 + \|z_1 - z_2\|_2^2$. For simplicity, we further make the following assumption on the FL environment, although our analysis can be readily applied to more general settings.

Assumption 3. *The server adopts FedAvg without subsampling ($w = K$). All workers have same amount of data ($p_k = \frac{1}{K}$) and the local minibatch size $B = 1$. In each epoch of federated learning, each normal worker's local minibatch is sampled independently from the local empirical data distribution $\widehat{P}_k$.*

Let $\mathcal{M} = (S, \boldsymbol{A}, T, r, H)$ denote the true MDP for the attackers, and $\widetilde{\mathcal{M}} = (S, \boldsymbol{A}, T', r', H)$ the simulated MDP when the local distribution of any benign worker is estimated as $\widetilde{P}$. Let $T(s'|s, a)$ and $r(s, a, s')$ denote the transition dynamics and reward function for $\mathcal{M}$, respectively. The following theorem captures the attack performance loss due to inaccurate distribution learning (see Appendix C for the proof).

Theorem 1. *Let $\mathcal{J}_{\mathcal{M}}(\pi) := \mathbb{E}_{\pi,T,\mu_0}[\sum_{t=0}^{H-1} r(s^t, a^t, s^{t+1})]$ denote the expected return over H attack steps under $\mathcal{M}$, policy π and initial state distribution μ_0. Let π^* and $\widetilde{\pi}^*$ be the optimal policies for $\mathcal{M}$ and $\widetilde{\mathcal{M}}$ respectively, with the same initial state distribution μ_0. Then,*

$$|\mathcal{J}_{\mathcal{M}}(\pi^*) - \mathcal{J}_{\mathcal{M}}(\widetilde{\pi}^*)| \leqslant 2H\epsilon\delta[(L + L_v)L_z\eta + 2L],$$

where $\epsilon = \frac{K-M}{K}$ is the fraction of benign nodes, $L_v \leqslant \sum_{t=0}^{H-1}(K_F)^t(L + LK_F)$ and $K_F \leqslant \epsilon \max\{|1 - \eta\alpha|, |1 - \eta\beta|\}$.

In practice, the learning rate η is typically small enough so that $\max\{|1 - \eta\alpha|, |1 - \eta\beta|\} \leqslant 1$. In this case, L_v is bounded by $\frac{L(1+K_F)}{1-K_F} \leqslant \frac{L(1+\epsilon)}{1-\epsilon}$. Therefore, we have $|\mathcal{J}_{\mathcal{M}}(\pi^*) - \mathcal{J}_{\mathcal{M}}(\widetilde{\pi}^*)| = O(H\frac{\epsilon}{1-\epsilon}\eta\delta)$. To ensure convergence, we typically have $\eta = O(\frac{1}{\sqrt{H}})$ [32], thus $|\mathcal{J}_{\mathcal{M}}(\pi^*) - \mathcal{J}_{\mathcal{M}}(\widetilde{\pi}^*)| = O(\frac{\epsilon}{1-\epsilon}\delta\sqrt{H})$. This result clearly shows how the loss of attack performance depends on the fraction of benign nodes, the inaccuracy of distribution learning, and the time horizon.

5 Experiments

In this section, we compare our RL-based attack with state-of-the-art model poisoning attacks on real-world datasets.

5.1 Experiment setup

Datasets. We conduct extensive experiments on three real-world datasets: MNIST [21], Fashion-MNIST [41], and EMNIST [9]. Due to space limitation, experiment results for Fashion-MNIST and EMNIST are provided in the supplementary material (see Appendix D.2). For the *i.i.d.* setting, we randomly split the dataset into K groups, each of which consists of the same number of training samples. For the *non-i.i.d.* setting, we follow the method of [11] to quantify the heterogeneity of local data distribution across clients. Suppose there are C classes in the dataset, e.g., $C = 10$ for the MNIST and Fashion-MNIST datasets. We evenly split the worker devices into C groups (with the M attackers evenly distributed across the C groups), where each group is assigned $1/C$ of training samples as follows. A training instance with label c is assigned to the c-th group with probability $q \geqslant 1/C$ and to every other group with probability $(1 - q)/(C - 1)$. Within each group, instances are evenly distributed. A higher q indicates a higher degree of *non-i.i.d.*. We set $q = 0.5$ as the default *non-i.i.d.* degree. To demonstrate the power of distribution learning, we assume that the set

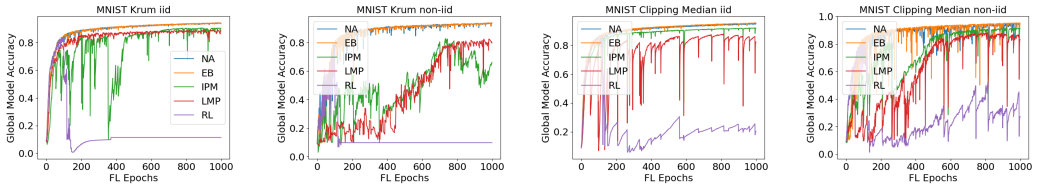


Figure 3: A comparison of global model accuracy under Krum and Clipping Median for both *i.i.d.* data and *non-i.i.d.* data. All parameters are set as default.

of attackers share m true data points sampled from the training instances assigned to them. We set $m = 200$ as the default value.

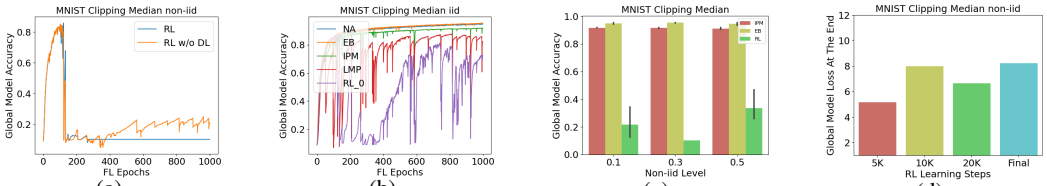
Baselines. We compare our RL-based attack (RL) with no attack (NA), and the state-of-the-art model poisoning FL attack methods: explicit boosting (EB) [4], inner product manipulation (IPM) [42], and local model poisoning attack (LMP) [11]. IPM manipulates the attackers' gradients so that the inner product between the aggregation result and the true gradient is negative. This requires access to the average of normal workers' gradients in each FL epoch, which is usually unavailable in practice. LMP generates myopic attacks by solving an optimization problem in each FL epoch. In addition to the server's aggregation rule, it also requires access to normal workers' local models. Although LMP with partial knowledge is also presented in [11], it performs substantially worse than the full knowledge case when the server uses the coordinate-wise median defense. We compare the RL-based attack with the more powerful full knowledge LMP below.

We consider three representative robust aggregation rules of different types [36]: Krum [5], which applies a vector-wise filtering to model updates, coordinate-wise median [44], which adopts a dimension-wise filtering, and FLTrust [7], which requires the server to have access to a small amount of root data. In the experiments, we actually consider an extension of the vanilla coordinate-wise median where a norm-bound clipping [39] is first applied before aggregation. This gives a more powerful defense as we observed in experiments. We set the default norm threshold to 2.

Default FL and RL settings. We adopt the following default parameters for the FL models: number of total workers $= 100$, number of attackers $= 20$, learning rate $\eta = 0.01$, subsampling rate $= 10\%$, the number of total FL epochs $= 1,000$. For our RL-based attack, both the distribution learning and policy learning phase start at the first FL epoch. The former ends at the 100th FL epoch when RL-based attack starts (all other attacks start at epoch 0). The policy learning phase ends at the 400th epoch. Since both the action space and state space are continuous in our setting, we choose the state-of-the-art Twin Delayed DDPG (TD3) [13] and Proximal Policy Optimization (PPO) [35] algorithms for training the attack policy in our experiments and find that TD3 gives better results in most cases. Below we report the results for TD3. We fix the initial model and the random seeds for subsampling and local data sampling for fair comparisons. See Appendix D for details of the datasets, experimental setups, and additional results.

5.2 Attack performance

Figure 3 shows how the test accuracy of the global model varies over FL epochs under different attacks when the server uses Krum and Clipping Median as the aggregation rule, respectively. Results for FLTrust is provided in the supplementary material (Appendix D.2). We observe that our RL-based attack performs significantly better in all the settings, despite the fact that IPM and LMP use the model updates of normal clients while RL does not. Note that for Krum, RL-based attack quickly drives the global model to a poor state ($\sim 10\%$ accuracy) once the attack starts at epoch 100 under both i.i.d. and non-i.i.d. local data distributions. Attacks become harder under Clipping Median due to the norm-bound clipping but our RL-based attack still reduces global model accuracy to around 50% on average. This is mainly because it targets long-term return while all other baselines are myopic. For example, in Figure 3(c), the global model accuracy drops significantly under all the attacks when five malicious devices are sampled around epoch 200. After that, the RL method keeps the accuracy at a low level, while other baselines' accuracy rebounds rapidly.



(a) (b) (c) (d)

Figure 4: Attack performance on MNIST with Clipping Median under (a) RL with and without distribution learning; (b) RL_0 (zero initial data for policy learning) and other baselines; (c) different non-iid degrees; and (d) different policy learning lengths. Non-iid degree $q = 0.3$ in (a) and (d). Other parameters are set as default.

5.3 Ablation studies

Importance of distribution learning. Figure 4(a) compares the global model accuracy of RL-based attack with distribution learning (RL) and that without distribution learning (RL w/o DL). We observe in both cases, the model accuracy decreases dramatically after attacks start at FL epoch 100. Further, the accuracy of RL w/o DL slightly increases up to 20%, while the accuracy of RL stays below 10%, which confirms our expectation that distribution learning allows the attackers to learn better attack policies. Figure 4(b) shows the attack performance of various baselines and RL_0, a variant of our RL-based attack where the attackers only have 200 *unlabeled* true images (used to train the denoising autoencoder only), thus completely relying on distribution learning to generate labeled samples needed for policy learning. We observe that the RL approach still outperforms other baseline methods, further indicating the power of distribution learning.

Impact of non-i.i.d. degree. Figure 4(c) shows the impact of data heterogeneity on attack performace. We use 5 different random seeds for all attacks and show the error bars. We observe that all the baselines obtain similar performance under different non-i.i.d. degrees and the impact of randomness in the testing environment on their performance is limited. On the other hand, we observe that the RL policies for $q = 0.1$ and $q = 0.5$ exhibit a large variance. But even the worst-case performance of our attack outperforms the best cases of all the baselines. Further, the RL policy for $q = 0.3$ is extremely good and exhibits very low variance. We expect that the variation across different RL policies is in part because the attackers simply use the last trained policy at the end of policy learning for attack execution, which does not necessarily give the best performance among all the intermediate policies trained (see also Figure 4(d)).

Impact of training length on policy learning. Figure 4(d) shows how the global model loss at the end of an FL training episode (in the simulated environment) varies over the RL policy training steps. We observe that longer training usually provides a better attack policy, although the training process is not stable. To fix this, one approach is to set up a separate testing environment to identify best trained policies. As mentioned above, our RL-based attack achieves promising performance even when the attackers always use the last policy obtained at the end of policy learning.

6 Conclusion

We propose a new approach for developing non-myopic attacks that can effectively compromise FL systems even with advanced defense mechanisms applied, by utilizing model-based reinforcement learning as a principled approach. While we focus on untargeted model poisoning against FL systems in this paper, our attack framework can be extended to targeted attacks (e.g., backdoor attacks) and to objectives beyond global model accuracy (e.g., fairness across clients [31, 22]). Further, our attack framework can be integrated with meta-learning [12, 17] to generalize the learned policy to different training tasks. For instance, the leader attacker can first learn a distribution over different tasks and train a model-agnostic policy with the learned distribution by utilizing model-agnostic meta-learning (MAML) [12]. Given a new task, the leader attacker can then train an attack policy fast by adapting from the model-agnostic policy with few shots. We envision that this approach can be especially useful for developing black-box attacks and for attacking multi-task FL. Another direction is to investigate novel methods to defend our adaptive attack methods. One possible solution would be to dynamically adjust FL parameters such as the subsampling rate or the aggregation rule.

References

[1] Utilization of FATE in Risk Management of Credit in Small and Micro Enterprises. https://www.fedai.org/cases/utilization-of-fate-in-risk-management-of-credit-in-small-and-micro-enterprises/.

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

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

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

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

[6] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

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

[8] Alfredo V Clemente, Humberto N Castejón, and Arjun Chandra. Efficient parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1705.04862*, 2017.

[9] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pages 2921–2926. IEEE, 2017.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

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

[12] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[13] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

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

[15] 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.

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

[17] Abhishek Gupta, Benjamin Eysenbach, Chelsea Finn, and Sergey Levine. Unsupervised meta-learning for reinforcement learning. *arXiv preprint arXiv:1806.04640*, 2018.

[18] 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*, pages 603–618, 2017.

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

[20] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

[21] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[22] Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization. In *International Conference on Machine Learning (ICML)*, 2021.

[23] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189*, 2019.

[24] Yuping Luo, Huazhe Xu, Yuanzhi Li, Yuandong Tian, Trevor Darrell, and Tengyu Ma. Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. *arXiv preprint arXiv:1807.03858*, 2018.

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

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

[27] Brendan McMahan and Daniel Ramage. Federated Learning: Collaborative Machine Learning without Centralized Training Data. https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.

[28] Brendan McMahan and Daniel Ramage. Machine Learning Ledger Orchestration For Drug Discovery (MELLODDY). https://www.melloddy.eu/.

[29] 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*, pages 691–706, 2019.

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

[31] Mehryar Mohri, Gary Sivek, and Ananda Theertha Suresh. Agnostic federated learning. In *International Conference on Machine Learning*, pages 4615–4625. PMLR, 2019.

[32] Boris T. Polyak. *Introduction to optimization*. Optimization Software, 1987.

[33] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletari, et al. The future of digital health with federated learning. *NPJ digital medicine*, 3(1):1–7, 2020.

[34] 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.

[35] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[36] Virat Shejwalkar, Amir Houmansadr, Peter Kairouz, and Daniel Ramage. Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning. In *IEEE Symposium on Security and Privacy*, 2022.

[37] 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*, pages 1891–1898, 2014.

[38] Yiwei Sun, Suhang Wang, Xianfeng Tang, Tsung-Yu Hsieh, and Vasant Honavar. Node injection attacks on graphs via reinforcement learning. *arXiv preprint arXiv:1909.06543*, 2019.

[39] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H Brendan McMahan. Can you really backdoor federated learning? *arXiv preprint arXiv:1911.07963*, 2019.

[40] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 1096–1103, New York, NY, USA, 2008. Association for Computing Machinery.

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

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

[43] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018.

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

[45] Hongxu Yin, Arun Mallya, Arash Vahdat, Jose M Alvarez, Jan Kautz, and Pavlo Molchanov. See through gradients: Image batch recovery via gradinversion. *arXiv preprint arXiv:2104.07586*, 2021.

[46] Hongxu Yin, Pavlo Molchanov, Jose M Alvarez, Zhizhong Li, Arun Mallya, Derek Hoiem, Niraj K Jha, and Jan Kautz. Dreaming to distill: Data-free knowledge transfer via deepinversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8715–8724, 2020.

[47] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.

[48] Xuezhou Zhang, Yuzhe Ma, Adish Singla, and Xiaojin Zhu. Adaptive reward-poisoning attacks against reinforcement learning. In *International Conference on Machine Learning*, pages 11225–11234. PMLR, 2020.

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

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Section 4.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix C.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The code and the data are proprietary.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix D.1.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Figure 4(c).
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix D.1.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Section 5.1.
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]