The Limitations of Federated Learning in Sybil Settings

Clement Fung
Carnegie Mellon University
clementf@andrew.cmu.edu

Chris J.M. Yoon
University of British Columbia
yoon@alumni.ubc.ca

Ivan Beschastnikh
University of British Columbia
bestchai@cs.ubc.ca

Abstract

Federated learning over distributed multi-party data is an emerging paradigm that iteratively aggregates updates from a group of devices to train a globally shared model. Relying on a set of devices, however, opens up the door for sybil attacks: malicious devices may be controlled by a single adversary who directs these devices to attack the system.

We consider the susceptibility of federated learning to sybil attacks and propose a taxonomy of sybil objectives and strategies in this setting. We describe a new DoS attack that we term *training inflation* and present several ways to carry out this attack. We then evaluate recent distributed ML fault tolerance proposals and show that these are insufficient to mitigate several sybil-based attacks. Finally, we introduce a defense against targeted sybil-based poisoning called *FoolsGold*, which identifies sybils based on the diversity of client updates. We show that FoolsGold exceeds state of the art approaches when countering several types of poisoning attacks. Our work is open source and is available online: https://github.com/DistributedML/FoolsGold

1 Introduction

To train multi-party machine learning (ML) models from user-generated data, clients share their training data with services, which can be computationally expensive and privacy-violating. Federated learning (FL) [10, 39, 40] is a recent solution to both problems: data is kept on the client device and only model parameters are transferred to a central aggregator while training. Clients maintain a basic level of privacy by computing their model updates locally and independently, enabling collaborative ML in resource-constrained settings such as over a mobile network or in IoT(Internet of Things) deployments [24, 29, 49, 57].

However, FL widens the attack surface of the machine learning process: clients, who previously acted only as passive data providers, can now observe intermediate model states and adaptively contribute arbitrary information as part of decentralized training. For example, adversaries posing as honest clients can send erroneous updates to *poison* the trained model [3, 8, 15, 19, 23, 28, 36], *invert* or reconstruct the data of honest clients [26, 41, 42, 48, 54, 56], or *gain access to models* without usefully contributing [34].

Previous work has shown that adversaries who control more clients in a federated learning deployment can carry out poisoning attacks with more damage [52], and they can mount more elaborate and elusive attacks with coordinated malicious clients [58]. Such *sybil-based* attacks [18], in which an adversary controls multiple malicious clients, have been mentioned only in passing in existing work on FL [6, 30, 33, 35]. In this paper we focus on sybil attacks and contribute a taxonomy of sybil strategies that can be used to exacerbate known attacks against FL. As part of this process we define a new category of sybil-specific denial of service (DoS) attack that we term *training inflation*.

Having defined a space of sybil strategies and corresponding malicious objectives (e.g., model poisoning) in the context of FL, we consider the defenses. Existing work has explored several ways to defend the FL training process [9, 50, 60, 61]. We evaluate these defenses in the context of sybil-based attacks. Our results demonstrate that none of these defenses are effective, particularly against a large number of sybils.

Consequently, we propose a defense called **FoolsGold** to counter one class of sybil-based attacks on FL: *targeted* poisoning by sybil clones. In a targeted poisoning attack, these clones contribute updates towards a specific poisoning objective. In expectation over the training process, this targeting reveals sybils through behavior that is more similar to each other than the similarity observed amongst the honest clients. FoolsGold's insight is to use this characteristic behavior to detect and reject poisoned contributions by adapting client learning rates based on *client contribution similarity*.

Our evaluation shows that FoolsGold defends FL from targeted poisoning by a large number of sybils, with only minimal change to the server-side algorithm and no change to the client-side algorithms. We evaluate FoolsGold on 4 diverse data sets (MNIST [31], VGGFace2 [14], KDDCup99 [17], Amazon Reviews [17]) and 3 model types (1-layer Softmax, Squeezenet, VGGNet) and find that our approach mitigates poisoning attacks under a variety of conditions, including different distributions of client data, varying poisoning targets, and various sybil strategies.

In summary, we make the following contributions:

- ⋆ We provide a taxonomy of sybil-based attacks on federated learning, which includes sybil strategies paired with an objective based on existing work on attacks against FL. We use this taxonomy to motivate open research problems in this space, and contribute a new type of DoS attack that we term *training inflation*.
- ⋆ We evaluate existing defenses against malicious sybil-based attacks on ML (Multi-Krum [9], median [62], trimmed mean [62]).
- ⋆ We design, implement, and evaluate a novel defense against sybil-based targeted poisoning attacks for federated learning that uses an adaptive learning rate per client based on inter-client contribution similarity.
- ⋆ In the context of colluding sybils, we design and evaluate intelligent poisoning attacks performed across sybils, and show that FoolsGold can defend against them, while suggesting strategies for further mitigation.

2 Background

Machine Learning (ML). Many ML problems are the minimization of a loss function in a large Euclidean space. For an ML classification task that predicts a discrete class; prediction errors result in a higher loss. Given a set of training data and a proposed model, ML algorithms *train*, or find an optimal set of parameters, for the given training set.

Stochastic gradient descent (SGD). One approach in ML is to use stochastic gradient descent (SGD) [12], an iterative algorithm that selects a batch of training examples, uses them to compute gradients on the parameters of the current model, and takes gradient steps in the direction that minimizes the loss function. The algorithm then updates the model parameters and another iteration is performed. SGD is a general learning algorithm that can be used to train a wide variety of models, including deep neural networks [12]. We assume SGD as the optimization algorithm in this paper. In SGD, the model parameters w are updated at each iteration t as follows:

$$w_{t+1} = w_t - \eta_t \left(\frac{1}{b} \sum_{(x_i,y_i)\in B_t} \nabla l(w_t, x_i, y_i) + \lambda ||w_t||_p\right) \quad (1)$$

where η_t represents a local learning rate, λ is a regularization parameter on an L_p norm of the parameters that prevents over-fitting, B_t represents a gradient batch of training data examples (x_i, y_i) of size b and ∇l represents the gradient of the loss function.

Federated learning [39]. We assume a standard federated learning setting, in which training data is distributed across multiple clients and the aggregator does not see any training data.

The distributed learning process is performed by a set of clients over *synchronous update rounds*, in which an aggregation of the k client updates, weighted by their proportional dataset size $\frac{n_k}{n}$, is applied to the model atomically.

$$w_{g,t+1} = w_{g,t} + \sum_k \frac{n_k}{n} \Delta w_{k,t}$$

Even if the training data is distributed such that it is not independent and identically distributed (non-IID), federated learning can still attain convergence. For example, federated learning can train an MNIST [31] digit recognition classifier in a setting where each client only holds data of 1 of the digits (0-9).

Federated learning comes in two forms: FEDSGD, in which each client sends every SGD update to the server, and FEDAVG, in which clients locally aggregate multiple SGD iterations before sending updates to the server, which is more communication efficient [39].

3 Threat model for sybil-based attacks on FL

Setting assumptions. We are focused on FL and therefore assume that data is distributed across clients and hidden, such as in an IoT deployment with multiple devices distributed in people's homes. The adversary can only access and influence the model state through the FL API. They cannot observe the training data of other honest clients. The adversary can observe the global change in model state to learn the total averaged update across all clients, but they cannot view individual honest client updates.

We assume that the server is uncompromised and is not malicious. In general, sybils can also be prevented non-algorithmically through techniques such as CAPTCHAs or device-specific asymmetric keys. We assume that such account or device verification services in the FL system do not exist, or that the adversary has the means to bypass these solutions.

Attacker capability. A system like FL, that allows clients to join and leave, is susceptible to sybil attacks [18] in which an adversary gains influence by joining a system using multiple colluding aliases. Sybil attacks are especially prevalent in IoT sensor networks [44], an emerging use case for FL [13, 33]. In this work, we assume that an adversary leverage sybils to mount more powerful attacks on FL.

While an attacker's influence on the system may increase with more sybils, the information the sybils learn from the system remains constant: a snapshot of the global model. The attack strategies are therefore more limited. The same holds for defenses: since no auxiliary information is assumed, sybil

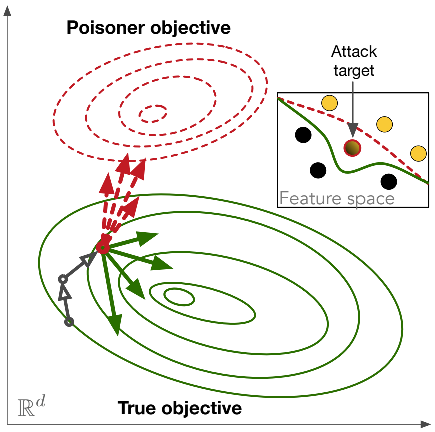

Figure 1: Targeted poisoning attack in SGD. The dotted red vectors are sybil contributions that drive the model towards the poisoner's objective. The solid green vectors are contributed by honest clients that drive towards the true objective.

Target	Attack type	Attack objective
Model quality	Untargeted poisoning **[19,59]**	Decrease model accuracy
	Targeted poisoning **[3,6,36,58,§7.3,§7.4]**	Decrease model accuracy on target class only
Privacy	Data inversion **[26,48,56]**	Learn data from clients
	Membership inference **[41,42,54]**	Determine if a client has certain data
Utility	Model free riding **[34]**	Access model without usefully contributing
Resource	Training time inflation **§5.2**	Increase training time
	Bandwidth inflation	Increase bandwidth used by server/clients
	CPU inflation	Increase CPU usage at server/clients

Table 1: Types of attacks against Federated Learning.

defenses based on e.g., information from social networks or peer-to-peer systems are not applicable [22, 53, 55, 63, 64].

Next, we review different adversarial objectives and examples of attack variants that achieve these objectives in the context of FL.

4 Attack objectives and strategies in FL

FL is susceptible to a variety of attack objectives and strategies. We now review the attack objectives in Table 1. In Section 4.2 we will review the strategies in Table 2 and discuss their intersection in Table 3.

Dimension	Sybil strategy	Description
Data	**Clones**	Sybils perform optimization steps on *identical local dataset*
	Act-alikes	Sybils perform optimization steps on *different local dataset*
	Clowns	Sybils use *synthetic gradients* not based on dataset
Coordination	**Uncoordinated**	Sybils act *without coordination*
	Swarm	Sybils coordinate and *weakly synchronize* states
	Puppets	Sybils coordinate and *synchronize on each round*
Churn	**Remainers**	Sybils join and stay
	Churners	Sybils join and leave

Table 2: High-level sybil strategies in Federated Learning.

4.1 Attack objectives

Attacking trained model quality. Adversaries may attack the quality of the model by supplying strategic inputs during training. In *untargeted poisoning*, the adversary aims to produce a model with low test accuracy that performs poorly across all classes [19, 59].

In *targeted poisoning*, the adversary directs the shared model towards a more specific objective. The adversary's goal is to produce a model where a subset of inputs are classified as a different, incorrect class. For example, this subset could be a source class (label-flipping attack [5]) or a set of images with a curated pattern (backdoor attack [3, 23]). To avoid detection of such poisoning attacks, the prediction accuracy of classes unrelated to the attack should not change. In FL, each client has an equal share of the aggregated gradient and attackers can attack any class with enough influence by generating additional sybils as shown in Figure 1.

Later in the paper we will demonstrate the vulnerability of FL to targeted poisoning by sybils (Section 5.3) and design and evaluate a new defense, *FoolsGold*, to defend against these types of attacks (Sections 6 and 7).

Attacking privacy of honest clients. Clients in FL possess a subset of data that is not explicitly shared with other clients. This enables learning over private datasets [39]. Although adversaries cannot access the training data at honest clients, they may attack the model and infer sensitive client information from the changes in the model state. An adversary may even influence the shared model to have it leak more private information in future iterations.

In a *data inversion* attack, the adversary reconstructs the training data of a targeted honest client [27] by generating a sample that closely represents the training data of a specific client. Alternatively, in a *membership inference* attack, the adversary determines whether or not a client has a specific datapoint in their dataset [41].

Table 2 *Sybil strategies*			Table 1 *Attack types*					
Churn	**Data**	**Coordination**	**U.Poison**	**T.Poison**	**D.Inversion**	**M.Infer**	**M.Free**	**T.Inflate**
Remainers	Clones	Uncoordinated		FoolsGold **§6**		[41]		
		Swarm						
		Puppets						
	Act-alikes	Uncoordinated		[58]				
		Swarm						
		Puppets						
	Clowns	Uncoordinated		[3,6,36]	[26,48,56]	[41,42,54]	[34]	**§5.2**
		Swarm						
		Puppets	[19,59]	**§7.3, §7.4**				**§5.2**
Churners	*All*	*All*	*Unexplored*					

Table 3: Known combinations of sybil strategies (Table 2) and attacks (Table 1). FoolsGold (Section 6) is highlighted as a contribution in this paper that defends against remainer clones whose goal is targeteted poisoning. The other highlighted+bolded sections in this paper are new attack variants we contribute in this paper.

Attacking system utility. FL assumes that the utility of model contributions is equally distributed amongst participating clients: all clients contribute data of relative equal value to the shared model and receive a similar increase in utility when gaining access to the shared model. In a *model free-riding* attack, the adversary participates in the FL process while providing contributions of negligible value [34]. When the training process completes, the adversary gains access to the shared model, despite providing no value to the system.

Attacking resources in the system. Alleviating resource usage in FL is an active area of research. Recent work has introduced model compression techniques and dropout schemes to extend FL to resource constrained settings [13].

With this in mind, we propose a class of attacks against the FL process itself. The aim of these attacks is to inflate the *training time*, *used bandwidth*, or *used compute cycles*. These attack variants can be seen as a form of denial of service, since they waste resources intended for FL at the server and honest clients. While the utility and efficacy of these attacks depend on the configuration of the FL system [13], we show in Section 5.2 that for some standard heuristic of early stopping criteria, sybils can inflate the training time arbitrarily by influencing these heuristics.

4.2 Sybil attack strategies

There are many ways in which sybils may be used to implement the attacks from the previous section. In this section we provide a preliminary taxonomy of *how* an adversary may mount a sybil-based attack (Table 2). We also provide an intersection of these strategies and their known attack variants (from Table 1) in Table 3. We consider three dimensions that we believe are key to understanding sybil attacks (and defending against them): data distribution among sybils, level of sybil coordination, and level of sybil churn.

Data distribution among sybils. In FL, clients, whether honest or malicious, provide *updates* to the shared model. These updates are assumed to be the result of an optimization algorithm based on client training data. They may be based on identical datasets at sybils (*clones*), different datasets (*act-alikes*), or may even be generated synthetically by sybils through an algorithm (*clowns*). Most known attacks on FL use synthesized gradients to achieve their attack objective [3, 19, 59].

Level of sybil coordination. An adversary that generates an increasing number of sybils, increases their influence on the system. This influence may be a type of a brute force strategy (*uncoordinated*), in which the honest clients are simply overpowered by the sybils on their desired objective. Alternatively, sybils may communicate amongst themselves before generating the next series of model updates. The state shared between sybils may be weakly consistent (*swarm*), through infrequent synchronization, or strongly consistent (*puppets*) through frequent synchronization, enabling more advanced attacks that require complex computation and communication.

The majority of attacks on FL use uncoordinated sybils. While some attacks use synchronized strategies [19, 59], these are limited to untargeted poisoning. In Sections 7.3 and 7.4, we demonstrate the power of these strategies by designing novel synchronized attacks against our FoolsGold defense.

Churn level of sybils. Although sybil churn is presently not used by defenses or attacks on FL, the churn level may impact defenses that rely on stateful client tracking during the learning process. Sybils that remain in the system (*remainers*) for an extended period of time may use their stability to build up their reputation with the server. Sybils that join, leave, and re-join the system (*churners*) may use churn to prevent the server from linking their activity across sessions.

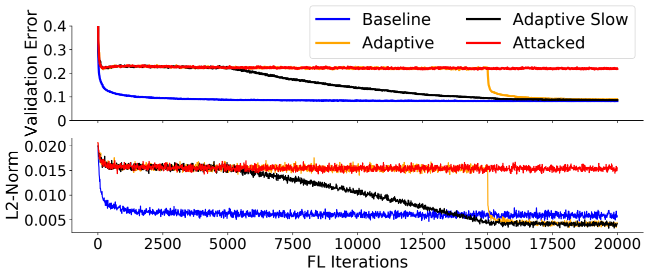

Figure 2: Training inflation attack on FL with 5 sybils.

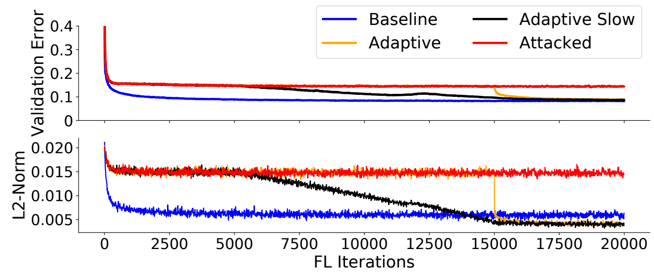

Figure 3: Training inflation attack with 5 sybils on FL with Multi-Krum when $f = 2$.

5 Can current defenses handle sybils?

We show next that, even when only assuming attacks from uncoordinated clones, current defenses are inadequate in defending FL from sybils.

5.1 Existing defenses for FL

In the general ML setting, defenses rely on access to the training data [4] or access to the training process itself [7, 25, 45] to defend against adversaries. Since FL does not have access to these elements, FL defenses are limited to those that use *robust aggregation schemes*, as these operate on the server only. Several such aggregation techniques have been proposed to defend against Byzantine adversaries in FL, including Multi-Krum [9], median [62], and trimmed mean [62].

At each iteration of **Multi-Krum aggregation**, the total Euclidean distance from the $n-f-2$ nearest neighbors is calculated for each client contribution. The f updates with the highest distances are removed and the average of the remaining updates is calculated.

When using the **median aggregation** technique, the element-wise median (the global update value for a parameter ΔW_j is the median of Δw_j) across all clients is used as the global update. Similarly, when using the **trimmed mean aggregation** technique, the highest and lowest β values for each feature are removed prior to computing the aggregated mean.

For all three techniques above, a successful defense requires an explicit bound on the maximum number of Byzantine clients. We show that if an adversary can spawn an arbitrarily large number of sybils, these techniques fail to work.

5.2 Training inflation attacks by sybil clones

Sybils can attack FL training by performing a *training inflation attack*. In this section, we show that when typical convergence heuristics are used, sybil clones can inflate the training time for as long as they wish, consuming shared resources on both the server and clients.

An ML training process needs a stopping condition, such as a fixed-length heuristic like the number of iterations or the number of training epochs. For better training efficiency, the process may also use a dynamic early stopping heuristic, typically based on the norm of the gradients [37], or the validation error [47].

We show both the average L2 norm of model updates and the validation error across iterations on both a baseline FL system (Figure 2) and an FL system with Multi-Krum (Figure 3). In evaluating Multi-Krum as a defense, when the parameter f is greater than the number of sybils, the attack is prevented. Figure 3 shows that Multi-Krum is ineffective for the case when an adversary can command more than f sybils.

In this experiment, 10 honest clients with a uniform sample of the MNIST dataset train a 1-layer softmax classifier; 5 sybils join the system and perform untargeted poisoning that causes the training to continue indefinitely ("Attacked"). Since this strategy is likely to be noticed by the server, we also show that sybils may choose to stop poisoning the model, either by sending negligibly small gradients ("Adaptive") or by slowly reducing their own learning rate ("Adaptive Slow"). In all attack variants, the model either does not converge, or only converges when the adversary allows it to.

5.3 Poisoning attacks by sybil clones

We demonstrate the ineffectiveness of prior defenses by performing a targeted poisoning attack with sybil clones in a non-IID FL setting, where 10 clients train an MNIST 1-layer softmax classifier; each client holds a distinct digit from the original training dataset.

We perform the attack against a variety of aggregation techniques: Multi-Krum (using $f = sybils$, the best case scenario for the defense), median, trimmed-mean (using $\beta = [10\%, 20\%]$), a baseline evaluation (using the mean across clients) and FoolsGold, our proposed solution that relies on client similarity, described in detail in Section 6 (using FEDSGD and FEDAVG)[1].

[1] In prior work [20], we also performed a targeted sybil-based poisoning attack against RONI [5], a defense that relies on a validation dataset, but

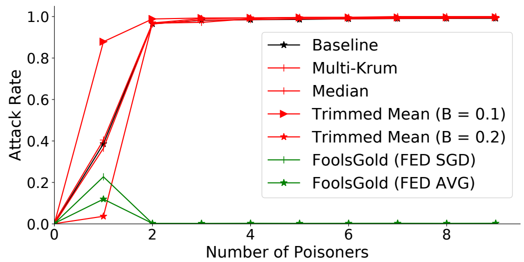

Figure 4: **Label-flipping** attack rate for varying number of sybils, for FL (Baseline), Multi-Krum, Median, Trimmed Mean and FoolsGold.

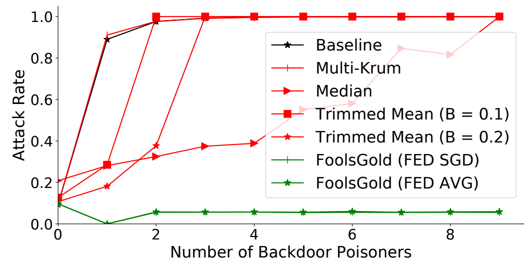

Figure 5: **Backdoor** attack rate for varying number of sybils, for FL (Baseline), Multi-Krum, Median, Trimmed Mean and FoolsGold.

To perform the *label-flipped attack*, each sybil client holds the same dataset of 1s, all labeled as 7s. A successful label-flipping attack would produce a model that incorrectly classifies all 1s as 7s. To perform the *backdoor attack*, we use a single white pixel in the bottom-right corner as the backdoor pattern [23]. Each sybil client holds a random uniform subset of the MNIST data, where each image is marked with a white pixel in the bottom-right corner of the image, and labeled as a 7. A successful backdoor attack results in a model where all images with the backdoor inserted (white bottom-right pixel) would be predicted as a 7, regardless of the other information in the image.

For both attacks, the number of sybils executing the attack increases from 0 to 9. Figures 4 and 5 show the performance of the approaches against the label-flipping and backdoor attacks respectively, where the attack rate is defined as the proportion of targeted examples (originally labeled 1s or images with the backdoor pixel) in the test set that are misclassified as 7s (the poisoning objective).

As soon as the proportion of sybil-based poisoners for a single class increases beyond the corresponding number of honest clients that hold that class (which is 1 in this case), the attack rate increases significantly for naive averaging (labeled as “Baseline”).

The performance of Multi-Krum is especially poor in the non-IID setting. When the variance of updates among honest clients is high, and the variance of updates among sybils is lower, Multi-Krum removes honest clients from the system. Multi-Krum is unsuitable for defending FL against sybils in its intended non-IID setting. Similarly, both the median and the trimmed mean are inadequate once the number of sybils increases. In all cases, a large number of sybils will skew this summary statistic, causing FL to fail.

Summary statistics are not a viable solution to sybils. Clearly, when sybils are present in a FL system, relying on a summary statistic (such as the mean, median, mode or any distribution-based summarizing technique) is an inadequate defense. When the number of sybils is high enough, these statistics are manipulated by the adversary and in fact will cause honest contributions to be labeled as anomalous and removed.

FoolsGold is a defense that does not require explicit parameterization of the number of attackers. The key assumption in FoolsGold is that: when performing targeted poisoning, sybils contribute updates that appear more similar than the expected similarity found between honest clients. When an abnormally high similarity is observed, those client contributions are penalized with a lower learning rate. We further discuss the design and motivation of FoolsGold in Section 6.

In contrast to the summary statistics described above, FoolsGold penalizes attackers further as the proportion of similar updates increases, and in Figures 4 and 5 FoolsGold remains robust even with 9 sybils. Since this attack uses client-contribution similarity, FoolsGold performs the worst when defending against one poisoner. We mitigate this weakness in Section 7.7.

In the rest of the paper we focus on targeted poisoning attacks and defenses in the context of clone-based sybils.

6 FoolsGold: countering targeted sybil poisoning attacks

We now describe FoolsGold, a defense that uses client similarity to prevent sybil-based targeted poisoning attacks in FL[2]. Unlike other defenses, FoolsGold does not require knowledge of the number of sybils, does not require modifications to the client-side protocol, and only uses state from the learning process itself. The FoolsGold algorithm and motivation are also described in our prior work [20].

6.1 FoolsGold threat model

In our setting, sybils observe global model state and send any arbitrary gradient contribution to the aggregator at any iteration. We assume that some honest clients have unpoisoned training data, requiring that every class in the model

found that RONI was trivially beaten by the sybil-based poisoning attack and we therefore omit these results for space.

[2] System implementation and experiments are available at https://github.com/DistributedML/FoolsGold

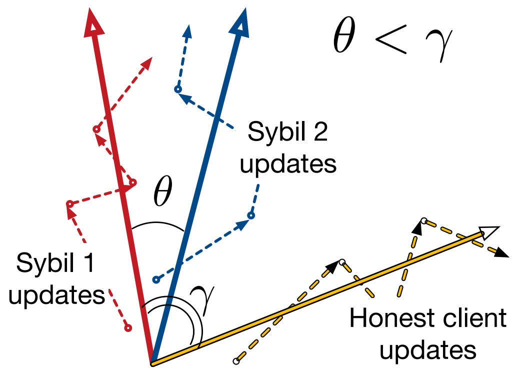

Figure 6: Dashed lines are gradient updates from three clients (2 sybils, 1 honest). Solid lines are aggregated update vectors. The angle between the aggregated update vectors of sybil clients (θ) is smaller than between those of the honest client and a sybil (γ). Cosine similarity would reflect this similarity.

is represented by at least one honest client's dataset. Without these honest clients, no contribution-based defense is possible since the model would be unable to learn anything about these classes in the first place.

One possible attack strategy involves scaling malicious updates to overpower honest clients [3, 61]. However, since state of the art magnitude-based detection methods succeed in preventing these attacks [9, 61], we do not consider this strategy in our work.

Secure-aggregation for FL provides privacy by obfuscating client updates [11]. For any server-side defense to operate through observation of malicious updates, we must assume that these types of obfuscations are not used. We also require that FL is performed synchronously, as is assumed by most other attacks and defenses in FL [9, 19, 59, 61, 62].

Client-side differential privacy has also been proposed in FL [21]: from the server's perspective, the aggregation rules are performed in the same way. Furthermore, since the algorithm is performed on the client device, we assume that sybils are not required to respect this protocol when performing attacks.

6.2 FoolsGold design

In the FL protocol (Algorithm 1), gradient updates are collected and aggregated in synchronous update rounds[3].

When each client's training data is non-IID and has a unique distribution, we assume that *honest clients can be distinguished from act-alike sybils by the diversity of their gradient updates*: sybils will contribute updates that appear more similar to each other than those among honest clients. Although this assumption is most clear in the federated non-IID setting, FoolsGold does not rely on the data being non-IID; we also explore FoolsGold's performance in varying IID settings in Section 7.2.

Data: Initial Model w_0 and SGD updates $\Delta_{i,t}$ from each client i at iteration t. Confidence parameter κ

for *Iteration T* **do**
 // Per client learning rate for iteration T
 Initialize α
 for *All clients i* **do**
  // Updates history
  Let H_i be the aggregate historical vector $\sum_{t=1}^{T} \Delta_{i,t}$
  // Feature importance
  Let S_T be the weight of important features at iteration T
  for *All other clients j* **do**
   Let cs_{ij} be the S_T-weighted cosine similarity between H_i and H_j
  end
  Let $v_i = \max_j(cs_i)$
 end
 for *All clients i* **do**
  for *All clients j* **do**
   // Pardoning
   if $v_j > v_i$ **then**
    $cs_{ij} *= v_i/v_j$
   end
  end
  // Per-row maximums
  $\alpha_i = 1 - \max_j(cs_i)$
 end
 // Normalize learning rates to 0-1 range
 $\alpha = \alpha / \max_i(\alpha)$
 // Element-wise logit function
 $\alpha = \kappa(\ln[(\alpha)/(1-\alpha)] + 0.5)$
 // Federated SGD iteration
 $w_T = w_{T-1} + \sum_i \alpha_i, \Delta_i$
end

Algorithm 1: FoolsGold learning algorithm.

FoolsGold adapts the learning rate α_i per client[4] based on (1) the update similarity among indicative features in any given iteration, and (2) historical information from past iterations.

Cosine similarity. We use cosine similarity to measure the angular distance between updates. This is preferred to Euclidean distance since sybils can manipulate the magnitude of a gradient to achieve dissimilarity, but the direction of a gradient is indicative of the update's objective.

Feature importance. From the perspective of a poisoning attack, there are three types of features in the model: (1) features that are relevant to the correctness of the model, but must be modified for a successful attack, (2) features that are relevant to the correctness of the model, but irrelevant for the attack, and (3) features that are irrelevant to both the attack

[3]From the aggregator perspective, this is true regardless of FEDAVG or FEDSGD. We show that FoolsGold can be applied in both settings.

[4]Note that we use α for the FoolsGold assigned learning rate, and η for the traditional, local learning rate. These are independent of each other.

and the model effectiveness.

Similar to other decentralized poisoning defenses [50], we look for similarity only in the indicative features (type 1 and 2) in the model. This prevents adversaries from manipulating irrelevant features while performing an attack, which is evaluated in Section 7.3.

The indicative features are found by measuring the magnitude of model parameters in the output layer of the global model, since this maps directly to its influence on the prediction probability [51]. The updates on these features can be removed based on a threshold (hard) or re-weighed based on their influence on the model (soft), and are normalized across all classes.

For deep neural networks, we do not consider the magnitude of values in the non-output layers of the model, which do not map directly to output probabilities and are more difficult to reason about. Recent work on feature influence in deep neural networks [1, 16, 32] may better capture the intent of sybil-based poisoning attacks in deep learning and we leave this analysis as future work.

Update history. FoolsGold maintains a history of updates from each client by aggregating the updates over multiple iterations (line 4). To better estimate client similarity, FoolsGold computes the pairwise similarity between aggregated historical updates instead of the updates from just the current iteration.

Figure 6 shows that even for two sybils with a common target objective, updates at a given iteration may diverge due to the variance of SGD. However, the cosine similarity between the sybils' aggregated historical updates tends to converge towards the malicious objective, providing a more accurate estimate of the client intent.

We interpret the cosine similarity on the indicative features, a value between -1 and 1, as a representation of how strongly two clients are acting as sybils. We define v_i as the maximum pairwise similarity for a client i, ensuring that as long as one such interaction exists, we can devalue the contribution while staying robust to an increasing number of sybils.

Pardoning. Since we have weak guarantees on the cosine similarities between an honest client and sybils, honest clients may be incorrectly penalized under this scheme. We introduce a pardoning mechanism that avoids penalizing such honest clients by re-weighing the cosine similarity by the ratio of v_i and v_j (line 14), reducing false positives. The new client learning rate α_i is then found by inverting the maximum similarity scores along the 0-1 domain. Since we assume at least one client in the system is honest, we rescale the vector such that the maximum adaption of the learning rate is 1 (line 19). This ensures that at least one client will have an unmodified update and encourages that a system with only honest nodes will not penalize their contributions.

Logit. However, even for very similar updates, the cosine similarity may be less than one. An attacker may exploit this by increasing the number of sybils to remain influential. We therefore want to encourage a higher divergence for values that are near the tails of this function, and avoid penalizing honest clients with a low, non-zero similarity value. Thus, we use the logit function (the inverse sigmoid function) centered at 0.5 (line 20), to encourage these properties. We also expose a confidence parameter κ that scales the logit function and show in Appendix A that κ and can be set as a function of the expected data distribution among clients.

When taking the result of the logit function, any value exceeding the 0-1 range is clipped and rounded to its respective boundary value. Finally, the gradient update is calculated by applying the final re-scaled learning rate to the global model.

Dataset	Examples	Classes	Features	Model
MNIST	60,000	10	784	1-layer
VGGFace2	7,380	10	150,528	ImageNet
KDDCup	494,020	23	41	1-layer
Amazon	1,500	50	10,000	1-layer

Table 4: Datasets used in this evaluation.

7 FoolsGold evaluation

We evaluate FoolsGold on a federated learning prototype implemented in 600 lines of Python. The prototype includes 150 lines for FoolsGold, implementing Algorithm 1. We use scikit-learn [46] to compute cosine similarity of vectors. For each experiment below, we partition the original training data into disjoint training sets, locally compute SGD updates on each dataset, and aggregate the updates using the described FoolsGold method to train a globally shared classifier.

We evaluate our prototype on four classification datasets (described in Table 4): MNIST [31], a digit classification problem, VGGFace2 [14], a facial recognition problem, KDDCup [17], which contains classified network intrusion patterns, and Amazon [17], which contains text from product reviews.

Each dataset was selected for one of its particularities. MNIST was chosen as the baseline dataset for evaluation since it was used extensively in the original federated learning evaluation [39]. The VGGFace2 dataset was chosen as a more complex learning task that requires deep neural networks to solve. For simplicity in evaluating poisoning attacks, we limit this dataset to the top 10 most frequent classes only. The KDDCup dataset has a relatively low number of features, and contains a massive class imbalance: some classes have as few as 5 examples, while some have over 280,000. Lastly, the Amazon dataset is unique in that it has few examples and contains text data: each review is one-hot-encoded, resulting in a large feature vector of size 10,000.

For all the experiments in this section, we perform targeted poisoning attacks that attempt to encourage a *source label/pattern* to be classified as a *target label* while training through federated learning[5]. In our baseline experiments, each class is solely owned by a single client, which is consistent with the federated learning baseline. In all experiments the number of honest clients matches the number of classes used in the dataset: 10 for MNIST and VGGFace2, 23 for KDDCup, and 50 for Amazon. For more-IID settings, we modify the distribution of client data and consider settings where classes overlap between clients in Section 7.2.

For MNIST, KDDCup, and Amazon, we train a 1-layer softmax classifier. For VGGFace2, we use two popular pre-trained Imagenet architectures from the torchvision package [38]: SqueezeNet1.1, a compressed model of 727,000 parameters designed for edge devices; and VGGNet11, a larger model of 128,000,000 parameters. When comparing client similarity for FoolsGold, we only use the features in the final output layer's gradients (fully connected layer in VGGNet and 10 1x1 convolutional kernels in SqueezeNet).

In MNIST, the data is already divided into 60,000 training examples and 10,000 test examples [31]. For VGGFace2, KDDCup and Amazon, we randomly partition 70% of the total data as training data and 30% as test data. The test data is used to evaluate two metrics that represent the performance of our algorithm: the *attack rate*, which is the proportion of attack targets (source labels/patterns) that are incorrectly classified as the target label, and the *test accuracy*, which is the proportion of examples in the test set that are correctly classified.

The MNIST and KDDCup datasets were executed with 3,000 iterations and a batch size of 50 unless otherwise stated. For Amazon, due to the high number of features and low number of samples per class, we train for 100 iterations and a batch size of 10. For VGGFace2, we train for 500 iterations with batch size of 8, momentum 0.9, weight decay 0.0001, and learning rate 0.001. These values were found using cross validation in the training set. During training, images were resized to 256x256 and randomly cropped and flipped to 224x224. During testing, images were resized to 256x256 and a 224x224 center was cropped.

We showed FoolsGold's effectiveness both when FEDSGD and FEDAVG are used by the clients. Since the difference between FEDSGD and FEDAVG was negligible in Figures 4 and 5, we continued to use FEDSGD for all future experiments.

We report the mean across 5 experiments in all cases. For each experiment, FoolsGold is parameterized with a confidence parameter $\kappa = 1$, and does not use the historical gradient or the significant features filter (we evaluate these design elements independently in Section 7.3 and 7.5, respectively).

[5] For the MNIST, VGGFace2, and Amazon datasets, we evaluated all source/target class pairs and found that the performance difference between these attacks was marginal.

Attack	Description	Dataset
A-1	1 sybil attacks.	All
A-5	5 sybils attack.	All
A-5x5	5 sets of 5 sybils, concurrent attacks.	MNIST, Amazon, VGGFaces2
A-OnOne	5 sybils executing 5 attacks on the same target class.	KDDCup99
A-99	99% sybils, same attack.	MNIST

Table 5: Canonical attacks used in our evaluation.

7.1 Canonical attack scenarios

Our evaluation uses a set of 5 canonical attack scenarios across the four datasets (Table 5). Attack **A-1** is a traditional poisoning attack: a single client joins the federated learning system with poisoned data. Attack **A-5** is the same attack performed with 5 sybil clients in the system. Each client sends updates for a subset of its data through SGD, meaning that their updates are not identical. Attack **A-5x5** evaluates FoolsGold's ability to thwart multiple attacks at once: 5 sets of client sybils attack the system concurrently, and we assume that the classes in these attacks do not overlap[6].

Since KDDCup99 is a unique dataset with severe class imbalance, instead of using an A-5x5 attack we choose to perform a different attack, **A-OnOne**. In KDDCup99, data from various network traffic patterns are provided. Class 'Normal' identifies patterns without any network attack, and is proportionally large ($\sim$ 20% of the data). When attacking KDDCup99, we assume that adversaries mislabel malicious attack patterns, which are proportionally small, (on average $\sim$ 2% of the data) and poison the malicious class towards the 'Normal' class. A-OnOne is a unique attack for KDDCup in which 5 different malicious patterns are each labeled as 'Normal', and each attack is performed concurrently.

Finally, we use **A-99** to illustrate the robustness of FoolsGold to a massively powerful adversary that generates 990 sybils to overpower a network of 10 honest clients, all of them performing the same attack against MNIST.

Since we use these canonical attacks throughout this work, we first evaluate FoolsGold against each attack on their respective datasets and models. Figure 7 plots the attack rate and test accuracy for each attack in Table 5. We also show results for the system without attacks: the original federated learning algorithm (FL-NA) and the system with the FoolsGold algorithm (FG-NA).

Figure 7 shows that for most attacks, FoolsGold effectively prevents the attack while maintaining high test accuracy. As FoolsGold faces larger groups of sybils, it has more information to more reliably detect similarity between sybils. FoolsGold performs worst on the A-1 attacks in which only one malicious client attacks the system. This is expected; without

[6] We do not perform a 1-2 attack in parallel with a 2-3 attack, since evaluating the 1-2 attack would be biased by the performance of the 2-3 attack.

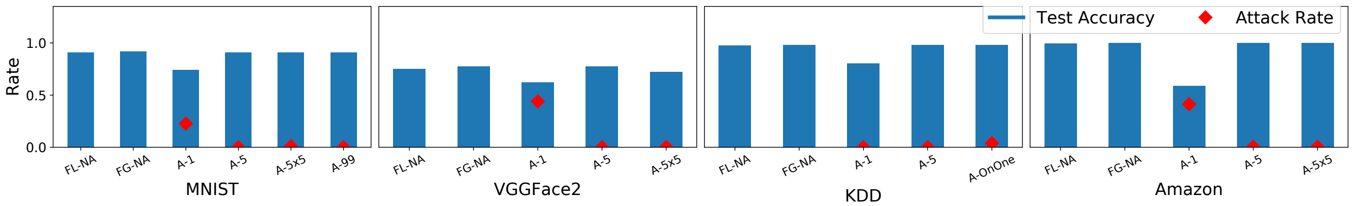

Figure 7: Test accuracy (blue bars) and attack rate (red ticks) for canonical attacks against the relevant canonical datasets.

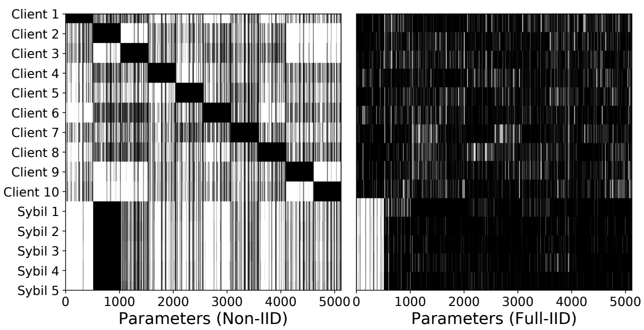

Figure 8: A visualization of the historical gradient of each clients on the Squeezenet model, in the non-IID (left) and IID (right) case. The top 10 rows show the gradient vector of honest clients; the bottom 5 rows for sybils performing a targeted 0-1 attack.

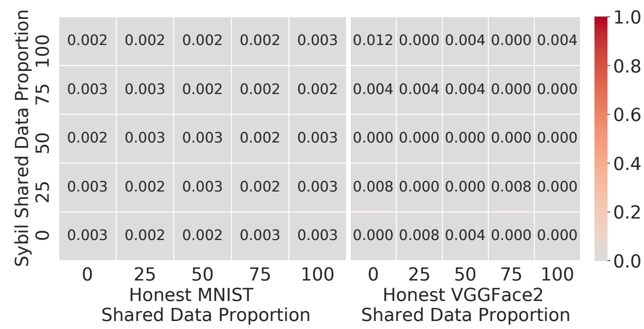

Figure 9: The attack rates on the MNIST (with softmax model) and VGGFace2 (with SqueezeNet model) datasets for varying sybil and honest client data distributions. 0 means that the data is completely disjoint by class; 100 means that the data is uniformly distributed.

multiple colluding sybils, malicious and honest clients are indistinguishable to FoolsGold.

Another point of interest is the prevalence of false positives. In A-1 KDDCup, our system incorrectly penalized an honest client for colluding with the attacker, lowering the prediction rate of the honest client as the defense was applied. We observe that the two primary reasons for low test accuracy are either a high attack rate (false negatives) or a high target class error rate (false positives). We also discuss false positives from data similarity in Section 7.2.

7.2 FoolsGold on varying IID settings

By design, FoolsGold relies on the assumption that training data is sufficiently dissimilar between clients. However, a more realistic scenario may involve settings where local client data distributions overlap more significantly.

To understand how FoolsGold handles these situations, we execute a VGGFace2 A-5 experiment under varying client distributions. Figure 8 shows each client's FoolsGold vector after 3000 training iterations, where each row shows the flattened historical gradient of a client on their final softmax layer. The top 10 rows correspond to honest clients and the bottom 5 rows correspond to sybils. The left half of Figure 8 shows the historical vector $\sum_{t=0}^{T} \Delta_{i,T}$ in a non-IID setting. Since each honest client has data corresponding to a unique class, their gradients are highly dissimilar, and since the sybils have the same poisoning dataset for a 0-1 attack, their gradients are highly similar. Therefore, FoolsGold can easily detect the sybil gradients.

The right half of Figure 8 shows the historical vector $\sum_{t=0}^{T} \Delta_{i,T}$ in a full-IID setting. Despite the honest clients holding uniform samples of the training data, the stochasticity of SGD introduces variance between them. The sybil gradients are still uniquely distinct as a result of their poisoned data, as seen in the bottom left corners of Figure 8. Even with only 10% poisoned data, executing a targeted poisoning attack produces fundamentally similar gradient updates in both full-IID and non-IID settings, enabling FoolsGold to succeed.

To test FoolsGold under diverse data distribution assumptions, we conduct an A-5 0-1 attack (with 5 sybils) on MNIST and VGGFace2 with 10 honest clients while varying the proportion of shared labels between both the sybils and honest clients. We varied these proportions over a grid ($D_{sybil}, D_{honest} \in \{0, 0.25, 0.5, 0.75, 1\}$), where D refers to the ratio of disjoint data to shared data. $D = 0$ refers to a non-IID setting where each client's local dataset is composed of a single class, and $x = 1$ refers to a setting where each client's local dataset is uniformly sampled from all classes. For all other cases, clients hold a proportion of D_{honest} uniform data and $(1-D_{honest})$ non-IID data from a single class. For sybils, we first create an honest client using the above mechanism, and flip all 0 labels to a 1 to perform a targeted attack. Therefore, when $D_{sybil}=0$, sybils will hold a full dataset of 0-1 poisoned data, and when $D_{sybil}=1$, sybils will hold a dataset of 10% poisoned 0-1 data, and 90% uniform data from classes 1-9.

FoolsGold defends against poisoning attacks for all (D_{sybil}, D_{honest}) combinations: the maximum attack rate was less than 1% for both the MNIST and VGGFace2 datasets, using both SqueezeNet and VGGNet. We show these results in Figure 9. In summary, an attacker cannot subvert FoolsGold by manipulating their malicious data distribution[7]. Instead, they must directly manipulate their gradient outputs, which we explore next.

If an attacker is aware of the FoolsGold algorithm, they may attempt to send updates in ways that encourage additional dissimilarity. This is an active trade-off: as attacker updates become less similar to each other (lower chance of detection), they become less focused towards the poisoning objective (lower attack utility).

Next, we consider and evaluate two synchronized sybil strategies in which attackers may subvert FoolsGold: (1) perturbing contributed updates to maximize dissimilarity, and (2) infrequently and adaptively sending poisoned updates.

7.3 Attacks using intelligent perturbations

A set of intelligent sybils could synchronize and send pairs of updates with careful perturbations that are designed to sum to zero. For example, if an attacker draws a perturbation vector ζ, two malicious updates a_1 and a_2 could be contributed as v_1 and v_2, such that $v_1 = a_1 + \zeta$ and $v_2 = a_2 - \zeta$.

Since the perturbation vector ζ has nothing to do with the poisoning objective, its inclusion will add dissimilarity to the malicious updates and decrease FoolsGold's effectiveness in detecting them. Also note that the sum of these two updates is still the same: $v_1 + v_2 = a_1 + a_2$. This strategy can also be scaled beyond 2 sybils by taking orthogonal perturbation vectors and their negation: for any subset of these vectors, the cosine similarity is 0 or -1, while the sum remains 0.

As explained in Section 6, this attack is most effective if ζ is only applied to features of type (3): those which are not important for the model or the attack. The attack is mitigated by filtering for indicative features in the model. Instead of looking at the cosine similarity between updates across all features in the model, we look at a weighted cosine similarity based on feature importance.

To evaluate the importance of this mechanism to the poisoning attack, we execute the intelligent perturbation attack

[7]In prior work [20], we also evaluated FoolsGold against other method for increased SGD diversity: mixing honest data with malicious data and using settings with a decreased SGD batch size.

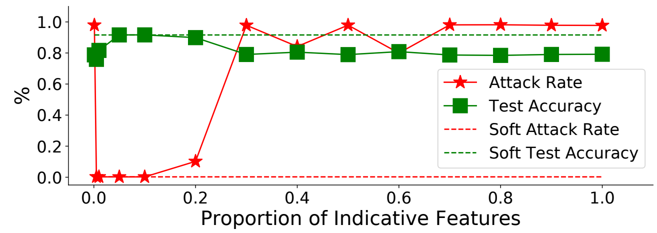

Figure 10: The performance of the optimal perturbation attack on MNIST with varying indicative feature ratio.

described above on MNIST, which contains several irrelevant features (the black background) in each example: a pair of sybils send v_1 and v_2 with intelligent perturbation ζ. We then vary the proportion of model parameters that are defined as indicative from 0.001 (only top 8 features on MNIST) to 1 (all features).

Figure 10 shows the attack rate and the test accuracy for varying proportions of indicative features against an A-5 attack. We first observe that when using all of the features for similarity (far right), the poisoning attack is successful.

Once the proportion of indicative features decreases below 0.1 (10%), the dissimilarity caused by the intelligent perturbation is removed from the cosine similarity and the poisoning vector dominates the similarity, causing the intelligent perturbations strategy to fail with an attack rate of near 0. We also observe that if the proportion of indicative features is too low (0.01), the test accuracy also begins to suffer. When considering such a low number of features, honest clients appear to collude as well, causing false positives.

We also evaluated the soft feature weighing mechanism, which weighs each contribution proportionally based on the model parameter itself. The results of the soft weighting method on the same intelligent MNIST poisoning attack are also shown in Figure 10. For both the attack rate and test accuracy, the soft filtering mechanism is comparable to the optimal performance of the hard filtering mechanism.

7.4 Attacks with adaptive updates

We devised another synchronized attack against FoolsGold that manipulates its memory component. If an adversary knows that FoolsGold uses similarity on the update history, and is able to locally compute its own pairwise cosine similarity among sybils, they can collude and compute this information themselves, deciding only to send poisoned updates when their historical similarity is low. We define a parameter M for the attack strategy that represents the threshold on inter-sybil similarity for sybils to send a poisoned update. When M is lower, sybils are less likely to be detected by FoolsGold and will send their updates less often; however, this will also lower the influence the sybils have on the global model.

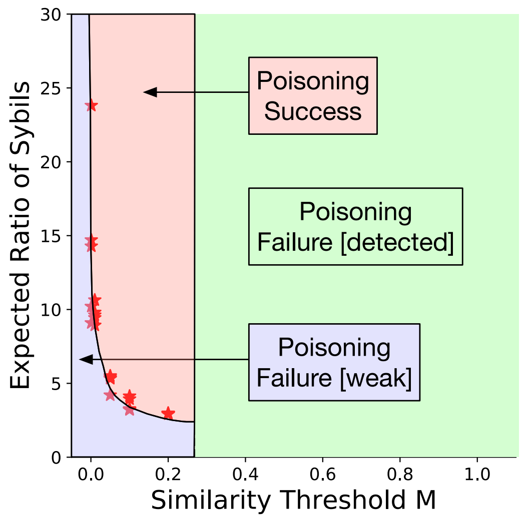

Figure 11: Relationship between similarity threshold and expected ratio of sybils per honest opposing client for the synchronized adaptive attack stategy against FoolsGold with 2 sybils on MNIST.

An adversary could generate an excess number of sybils for a successful attack, but given that the adversary is uncertain about the influence needed to overpower the honest clients, this is a difficult trade-off to predict for an optimal attack.

To demonstrate this, the intelligent perturbation attack above is executed by 2 sybils on MNIST, with FoolsGold using the soft weighing of features in its cosine similarity (the optimal defense for MNIST against the intelligent perturbation attack). Figure 11 shows the relationship between M and the resulting expected ratio of sybils needed to match the influence for each honest opposing client.

For instance, if we observed that the sybils only sent poisoning gradients 25% of the time, they would need 4 sybils to induce a comparable influence on the model. Given a prescribed similarity threshold M, the values shown are the expected number of sybils required for the optimal attack. The attack is optimal because using less sybils does not provide enough influence to poison the model, while using more sybils is inefficient.

This is shown in Figure 11 with three shaded regions: in the green region to the right ($M > 0.27$), the threshold is too high and any poisoning attack is detected and removed. In the blue region on the bottom left, the attack is not detected, but there is an insufficient number of sybils to overpower the honest opposing clients. Lastly, in the top left red region, the attack succeeds, potentially with more sybils than required.

With a sufficiently large number of sybils and appropriately low threshold, attackers can subvert our current defense for our observed datasets. Although this strategy can break FoolsGold, finding the appropriate threshold is challenging as it is dependent on many other factors: the number of honest clients in the system, the proportion of indicative features considered by FoolsGold, and the distribution of data. The exact number of sybils required to successfully poison the model is unknown to attackers without knowledge of the number of honest clients and their honest training data.

7.5 Effects of design elements

Each of the three main design elements (history, pardoning and logit) described in Section 6 addresses specific challenges. In the following experiments we disabled one of the three components and recorded the test error, attack rate, and target class test error of the resulting model.

History. Attacks with intelligent perturbations and adaptive updates increase the variance of updates in each iteration. The increased variance in the updates sent by sybils cause the cosine similarities at each iteration to be an inaccurate approximation of a client's sybil likelihood. Our design uses history to address this issue, and we evaluate it by comparing the performance of FoolsGold with and without history using an A-5 MNIST attack with 80% poisoned data and batch size of 1 (factors which will induce a high variance).

Pardoning. We claim that honest client updates may be similar to the updates of sybils, especially if the honest client owns the data for the targeted class. To evaluate the necessity and efficacy of our pardoning system, we compare the performance of FoolsGold on KDDCup with the A-AllOnOne attack with and without pardoning.

Logit. An important motivation for using the logit function is that adversaries could otherwise arbitrarily increase the number of sybils to mitigate any non-zero weighting of their updates. We evaluate the performance of FoolsGold with and without the logit function for the A-99 MNIST attack.

Figure 12 shows the overall test error, sybil attack rate, and target class test error for the six different evaluations. The attack rate for the A-AllOnOne KDDCup attack is the average attack rate for the 5 sets of sybils.

Overall, the results align with our claims. For the A-5 MNIST case, we find that history successfully mitigates attacks that otherwise would pass through in the no-history system. Comparing the results of the A-AllOnOne KDDCup attack, we find that, without pardoning, the test error of both the target class and the overall test error increase while the attack rate was negligible for both cases, indicating a high rate of false positives for the target class. Finally, for the A-99 MNIST attack, without the logit function, the adversary was able to mount a successful attack by overwhelming FoolsGold with sybils, showing that the logit function is necessary to prevent this attack.

7.6 FoolsGold performance overhead

We evaluate the runtime overhead incurred by augmenting a federated learning system with FoolsGold. We run the system

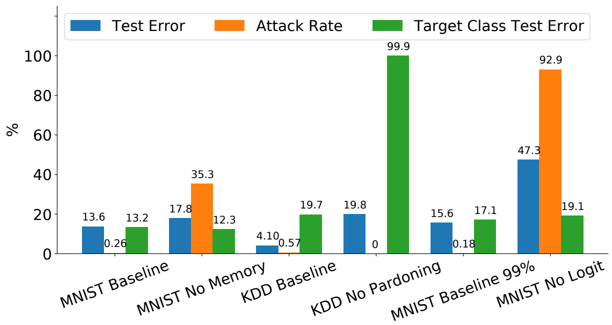

Figure 12: Metrics for FoolsGold with various components independently removed.

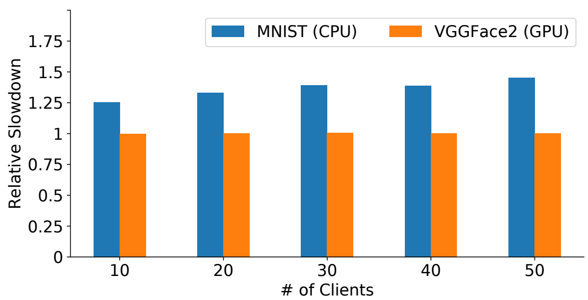

Figure 13: Running time overhead of FoolsGold as compared to Federated Learning (baseline of 1), for MNIST (on a CPU) and VGGFace2 (on a GPU).

with and without FoolsGold with 10 – 50 clients by training an MNIST classifier on a commodity CPU and a VGGFace2 deep learning model on a Titan V CPU.

Figure 13 plots the relative slowdown added by FoolsGold for CPU and GPU based workloads. On a CPU, the most expensive part of the FoolsGold algorithm is computing the pairwise cosine similarity. Our Python prototype is not optimized and there are known optimizations to improve the speed of computing angular distance at scale [2]. When training a deep learning model on a GPU, the cost of training is high enough that the relative slowdown from FoolsGold is negligible. We profiled micro-benchmarks for the total time taken to execute the FoolsGold algorithm and found that it took less than 1.5 seconds, even with 50 clients.

7.7 Combating a single client adversary

We consider the single-shot model replacement attack [3] in which a single adversarial client sends the direct vector to the poisoning objective. This attack does not require sybils and can therefore bypass FoolsGold.

We performed an experiment that augmented FoolsGold with a properly parameterized Multi-Krum solution, with $f = 1$. Figure 14 shows the training accuracy and the attack rate

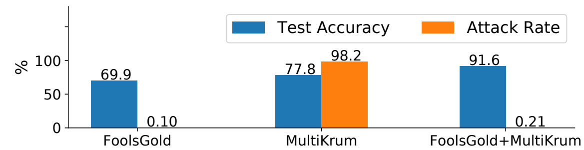

Figure 14: The performance of Multi-Krum with FoolsGold when combating a combination of an A-5 attack and the model replacement attack [3] on MNIST.

for FoolsGold, Multi-Krum, and the two systems combined when facing concurrent A-5 and model replacement attacks.

We see that Multi-Krum and FoolsGold do not interfere with each other. The Multi-Krum algorithm prevents the model replacement attack, and FoolsGold prevents the sybil attack. Independently, these two systems fail to defend both attacks concurrently, either by failing to detect the model replacement attack (for FoolsGold) or by allowing the sybils to overpower the system (for Multi-Krum).

FoolsGold is specifically designed for handling targeted poisoning attacks from a group of clone-based sybils: we believe the current state of the art is better suited to mitigate attacks from single actors.

8 Conclusion

The decentralization of ML is driven by growing privacy and scalability challenges. Federated learning is a state of the art proposal adopted in production [40], and is increasingly being used in mobile and edge networks [33]. However, using federated learning in such settings has opened the door for adversaries to attack the system with sybils [44]. We showed that federated learning is vulnerable to sybil-based attacks and that existing defenses are ineffective. To defend against one of these strategies (sybil-based clones that remain in the system), we proposed *FoolsGold*, a defense that uses client *contribution similarity*. Our results indicate that FoolsGold mitigates targeted poisoning attacks and is effective even when sybils overwhelm the honest clients.

Despite FoolsGold's performance against targeted poisoning attacks, a number of sybil-based strategies are not well defended by FoolsGold, such as coordinated attacks, some of which we showed (adaptive attacks, intelligent perturbations) and some of which have been proposed in recent work (distributed backdoors [58]).

In addition, we suggest that there is a much higher potential for sybils to be used to execute attacks on distributed multi-party ML systems such as federated learning. We hope that our work inspires further research on the implications of sybils on these systems and leads to other defenses that are robust to sybils.

Acknowledgments

This work was supported by the Huawei Innovation Research Program (HIRP), Project No: HO2018085305. We also acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), 2014-04870. Chris contributed to the project while being sponsored by the NSERC USRA program.

References

[1] M. Ancona, E. Ceolini, A. C. Oztireli, and M. Gross. A Unified View of Gradient-based Attribution Methods for Deep Neural Networks. In *Workshop on Interpreting, Explaining and Visualizing Deep Learning*, NIPS, 2017.

[2] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and Optimal LSH for Angular Distance. In *Advances in Neural Information Processing Systems 28*, NIPS. 2015.

[3] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How To Backdoor Federated Learning. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, AISTATS, 2020.

[4] Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, and Jaehoon Amir Safavi. Mitigating Poisoning Attacks on Machine Learning Models: A Data Provenance Based Approach. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, AISec, 2017.

[5] Marco Barreno, Blaine Nelson, Anthony D. Joseph, and J. D. Tygar. The Security of Machine Learning. *Machine Learning*, 81(2), 2010.

[6] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing Federated Learning through an Adversarial Lens. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 634–643. PMLR, 2019.

[7] Battista Biggio, Igino Corona, Giorgio Fumera, Giorgio Giacinto, and Fabio Roli. Bagging Classifiers for Fighting Poisoning Attacks in Adversarial Classification Tasks. In *Multiple Classifier Systems*, 2011.

[8] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning Attacks Against Support Vector Machines. In *Proceedings of the 29th International Conference on International Conference on Machine Learning*, ICML, 2012.

[9] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent. In *Advances in Neural Information Processing Systems 30*, NIPS, 2017.

[10] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé M Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards Federated Learning at Scale: System Design. In *Conference on Systems and Machine Learning*, SysML, 2019.

[11] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS, 2017.

[12] Léon Bottou. Large-Scale Machine Learning with Stochastic Gradient Descent. In *19th International Conference on Computational Statistics*, COMPSTAT, 2010.

[13] Sebastian Caldas, Jakub Konečny, H Brendan McMahan, and Ameet Talwalkar. Expanding the Reach of Federated Learning by Reducing Client Resource Requirements. *arXiv preprint arXiv:1812.07210*, 2019.

[14] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman. VGGFace2: A Dataset for Recognising Faces across Pose and Age. In *International Conference on Automatic Face and Gesture Recognition*, FG, 2018.

[15] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[16] A. Datta, S. Sen, and Y. Zick. Algorithmic Transparency via Quantitative Input Influence: Theory and Experiments with Learning Systems. In *Proceedings of the 2016 IEEE Symposium on Security and Privacy*, S&P, 2016.

[17] Dua Dheeru and Efi Karra Taniskidou. UCI Machine Learning Repository, 2017.

[18] John (JD) Douceur. The Sybil Attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, IPTPS, 2002.

[19] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Local Model Poisoning Attacks to Byzantine-Robust Federated Learning. In *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

[20] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. Mitigating Sybils in Federated Learning Poisoning. *arXiv preprint arXiv:1808.04866*, 2018.

[21] Robin C. Geyer, Tassilo Klein, and Moin Nabi. Differentially Private Federated Learning: A Client Level Perspective. *NIPS Workshop: Machine Learning on the Phone and other Consumer Devices*, 2017.

[22] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP, 2017.

[23] T. Gu, B. Dolan-Gavitt, and S. Garg. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. *arXiv preprint arXiv:1708.06733*, 2017.

[24] J. Hamm, A. C. Champion, G. Chen, M. Belkin, and D. Xuan. Crowd-ML: A Privacy-Preserving Learning Framework for a Crowd of Smart Devices. In *Proceedings of the IEEE 35th International Conference on Distributed Computing Systems*, ICDCS, 2015.

[25] Bo Han, Ivor W. Tsang, and Ling Chen. On the Convergence of a Family of Robust Losses for Stochastic Gradient Descent. In *European Conference on Machine Learning and Knowledge Discovery in Databases - Volume 9851*, ECML PKDD, 2016.

[26] Briland Hitaj, Giuseppe Ateniese, and Fernando Pérez-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*, CCS, 2017.

[27] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R. Ganger, Phillip B. Gibbons, and Onur Mutlu. Gaia: Geo-distributed Machine Learning Approaching LAN Speeds. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, 2017.

[28] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. D. Tygar. Adversarial Machine Learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, AISec, 2011.

[29] Linshan Jiang, Rui Tan, Xin Lou, and Guosheng Lin. On Lightweight Privacy-preserving Collaborative Learning for Internet-of-things Objects. In *Proceedings of the International Conference on Internet of Things Design and Implementation*, IoTDI, 2019.

[30] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and Open Problems in Federated Learning. *arXiv preprint arXiv:1912.04977*, 2019.

[31] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11), 1998.

[32] K. Leino, S. Sen, A. Datta, M. Fredrikson, and L. Li. Influence-Directed Explanations for Deep Convolutional Networks. In *2018 IEEE International Test Conference*, ITC, 2018.

[33] Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Niyato, and Chunyan Miao. Federated Learning in Mobile Edge Networks: A Comprehensive Survey. *arXiv preprint arXiv:1909.11875*, 2019.

[34] Jierui Lin, Min Du, and Jian Liu. Free-riders in Federated Learning: Attacks and Defenses. *arXiv preprint arXiv:1911.12560*, 2019.

[35] Lingjuan Lyu, Han Yu, and Qiang Yang. Threats to Federated Learning: A Survey. *arXiv preprint arXiv:2003.02133*, 2020.

[36] Saeed Mahloujifar, Mohammad Mahmoody, and Ameer Mohammed. Multi-party Poisoning through Generalized p-Tampering. *arXiv preprint arXiv:1809.03474*, 2018.

[37] Maren Mahsereci, Lukas Balles, Christoph Lassner, and Philipp Hennig. Early Stopping without a Validation Set. *arXiv preprint arXiv:1703.09580*, 2017.

[38] Sébastien Marcel and Yann Rodriguez. Torchvision the Machine-vision Package of Torch. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM, 2010.

[39] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017.

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

[41] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov. Exploiting Unintended Feature Leakage in Collaborative Learning. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy*, S&P, 2019.

[42] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive Privacy Analysis of Deep Learning: Stand-alone and Federated Learning under Passive and Active White-box Inference Attacks. *arXiv preprint arXiv:1812.00910*, 2018.

[43] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust Stochastic Approximation Approach to Stochastic Programming. *SIAM J. on Optimization*, 19:1574–1609, 2009.

[44] J. Newsome, E. Shi, D. Song, and A. Perrig. The Sybil attack in Sensor Networks: Analysis & Defenses. In *Third International Symposium on Information Processing in Sensor Networks*, IPSN, 2004.

[45] Olga Ohrimenko, Felix Schuster, Cedric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious Multi-Party Machine Learning on Trusted Processors. In *25th USENIX Security Symposium (USENIX Security 16)*, 2016.

[46] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[47] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

[48] Ahmed Salem, Apratim Bhattacharya, Michael Backes, Mario Fritz, and Yang Zhang. Updates-leak: Data Set Inference and Reconstruction Attacks in Online Learning. *arXiv preprint arXiv:1904.01067*, 2019.

[49] Sumudu Samarakoon, Mehdi Bennis, Walid Saad, and Merouane Debbah. Distributed Federated Learning for Ultra-Reliable Low-Latency Vehicular Communications. *arXiv preprint arXiv:1807.08127*, 2019.

[50] Shiqi Shen, Shruti Tople, and Prateek Saxena. Auror: Defending Against Poisoning Attacks in Collaborative Deep Learning Systems. In *Proceedings of the 32Nd Annual Conference on Computer Security Applications*, ACSAC, 2016.

[51] Reza Shokri and Vitaly Shmatikov. Privacy-Preserving Deep Learning. In *Proceedings of the 2015 ACM SIGSAC Conference on Computer and Communications Security*, CCS, 2015.

[52] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H. Brendan McMahan. Can You Really Backdoor Federated Learning? In *2nd International Workshop on Federated Learning for Data Privacy and Confidentiality*, FL - NeurIPS, 2019.

[53] Nguyen Tran, Bonan Min, Jinyang Li, and Lakshminarayanan Subramanian. Sybil-resilient Online Content Voting. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, NSDI, 2009.

[54] Stacey Truex, Ling Liu, Mehmet Emre Gursoy, Lei Yu, and Wenqi Wei. Towards Demystifying Membership Inference Attacks. *arXiv preprint arXiv:1807.09173*, 2018.

[55] Bimal Viswanath, Muhammad Ahmad Bashir, Muhammad Bilal Zafar, Simon Bouget, Saikat Guha, Krishna P. Gummadi, Aniket Kate, and Alan Mislove. Strength in Numbers: Robust Tamper Detection in Crowd Computations. In *Proceedings of the 3rd ACM Conference on Online Social Networks*, COSN, 2015.

[56] Lixu Wang, Shichao Xu, Xiao Wang, and Qi Zhu. Eavesdrop the Composition Proportion of Training Labels in Federated Learning. *arXiv preprint arXiv:1910.06044*, 2019.

[57] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan. Adaptive Federated Learning in Resource Constrained Edge Computing Systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1205–1221, 2019.

[58] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. DBA: Distributed Backdoor Attacks against Federated Learning. In *International Conference on Learning Representations*, ICLR, 2020.

[59] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Fall of Empires: Breaking Byzantine-tolerant SGD by Inner Product Manipulation. In *Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence*, UAI, 2019.

[60] Cong Xie, Sanmi Koyejo, and Indranil Gupta. SLSGD: Secure and Efficient Distributed On-device Machine Learning. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, ECMLPKDD, 2019.

[61] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Zeno: Distributed Stochastic Gradient Descent with Suspicion-based Fault-tolerance. In *Proceedings of the 36th International Conference on Machine Learning*, ICML, 2019.

[62] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates. In *Proceedings of the 35th International Conference on Machine Learning*, ICML, 2018.

[63] Haifeng Yu, Phillip B. Gibbons, Michael Kaminsky, and Feng Xiao. SybilLimit: A Near-optimal Social Network Defense Against Sybil Attacks. TON, 2010.

[64] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, and Abraham Flaxman. SybilGuard: Defending Against Sybil Attacks via Social Networks. In *SIGCOMM*, 2006.

A Convergence analysis

Theorem: Given the process in Algorithm 1, the convergence rate of the participants (malicious and honest) is $O(\frac{1}{T^2})$ over T iterations.

Proof of theorem: We know from the convergence analysis of SGD [43] that for a constant learning rate, we achieve a $O(\frac{1}{T^2})$ convergence rate.

Let M be the set of malicious clients in the system and G be the set of honest clients in the system. Assume that the adapted learning rates provided at each iteration α_i are provided by a function $h(i,t)$, where i is the client index and t is the current training iteration. As long as $h(i,t)$ does not modify the local learning rate of the honest clients and removes the contributions of sybils, the convergence analysis of SGD applies as if the training was performed with the honest clients' data.

$$\forall i \in M, h(i,t) \rightarrow 0 \qquad \text{(cond1)}$$
$$\forall i \in G, h(i,t) \rightarrow 1 \qquad \text{(cond2)}$$

We will show that, under certain assumptions, FoolsGold satisfies both conditions of $h(i,t)$. We prove each condition separately.

Condition 1: Let v_i be the ideal gradient for any given client i from the initial shared global model w_0, that is: $w_0 + v_i = w_i^*$ where w_i^* is the optimal model relative to any client i's local training data. Since we have defined all sybils to have the same poisoning goal, all sybils will have the same ideal gradient, which we define to be v_m.

As the number of iterations in FoolsGold increases, the historical gradient $H_{i,t}$ for each sybil approaches v_m, with error from the honest client contributions ε:

$$\forall i \in M : \lim_{t\rightarrow\infty} H_{i,t} = v_m + \varepsilon$$

Since the historical update tends to the same vector for all sybils, the expected pairwise similarity of these updates will increase as the learning process continues. As long as the resulting similarity, including the effect of pardoning between sybils, is below β_m, FoolsGold will adapt the learning rate to 0, satisfying (cond1).

β_m is the point at which the logit function is below 0 and is a function of the confidence parameter κ:

$$\beta_m \geq 1 - \frac{e^{-0.5\kappa}}{1+e^{-0.5\kappa}}$$

Condition 2: Regarding the ideal gradients of honest clients, we assume that the maximum pairwise cosine similarity between the ideal gradient of honest clients is β_g. As long as β_g is sufficiently low such that FoolsGold assigns a learning rate adaptation of 1 for all honest clients, the second condition of $h(i,t)$ is met. β_g is the point at which the logit function is greater than 1 and is also a function of the confidence parameter κ:

$$\beta_g \leq 1 - \frac{e^{0.5\kappa}}{1+e^{0.5\kappa}}$$

If the above condition holds, FoolsGold will classify these clients to be honest and will not modify their learning rates. This maintains the constant learning rate and satisfies (cond2).