Exploring the Trade-off between Domain Generalization and Byzantine Robustness in Federated Learning

Haozhe Si
haozhes3@illinois.edu

1 Introduction

Federated learning is a decentralized and distributed machine learning framework that protects the data privacy of users [7, 16, 10]. Briefly speaking, there is one *server* and multiple *clients* in this framework, and each client keeps its own data. The server wants to train a model over clients' data in a way that does not directly access these data for privacy protection. The algorithms for federated learning share a similar scheme: (i) the server sends model parameters θ to clients (ii) each client updates its own copy of the model on its data for k steps and then sends the updated model parameters $\{\theta_i^k\}_{i=1}^N$ back to the server (iii) the server updates the model by aggregating copies of parameters it receives. Notice in these steps, there is no communication of data between clients and the server (or among the clients), which admits protection of end users' data privacy.

In general, the datasets collected and kept by different clients are non-IID with some shared structure (e.g., mobile phone users in the same university/city may collect similar images, but the image contents will be different). In addition, there are always new clients added to the network of clients that a server communicates with, and it is desired that the model learned by the server can generalize well to new users. This is the goal of *federated domain generalization* [3, 8], which aims to distributedly learn a model over data from multiple clients that can generalize well to other unseen clients, given that there exists some shared structure among clients.

On the other hand, the distributed learning methodology brings vulnerability to the federated learning framework. To be specific, an adversarial client can inject malicious data into the aggregated model by training the local model using poisoned data [17, 18]. The most common data poisoning methods include (i) backdoor attacks [9], which force the model to make adversary-specified predictions when an embedded trigger is observed, and (ii) label flipping attacks [1], which corrupt the decision boundaries of the models. In the federated learning setting, different types of attacks may occur in different clients with varying ratios. State-of-the-art Byzantine-robust algorithms [2, 4] find robust aggregation methods that filter out malicious updates. Intuitively, non-IID clients tend to be down-scaled in aggregation and under-represented. To design a trustworthy federated learning framework against data poisoning while preserving the ability of generalization, we need to distinguish the feature distribution shift caused by the poisoned samples from the natural distribution shift. Therefore, one natural question is,

> *Can Byzantine-robust federated learning algorithm achieve domain generalization on unseen domains? If not so, how can we design an algorithm that makes the trade-off between Byzantine robustness and domain generalization?*

To answer this question, we first evaluate the performance of the state-of-the-art Byzantine-robust federated learning algorithm, FLTrust [2], in a federated domain generalization scenario. By showing that the robust aggregation algorithm is conservative, we proposed a novel algorithm that applies a generalization regularizer to the optimization goal and a robustness-generalization trade-off parameter that enforce the aggregator to pay more attention to the OOD clients.

Project Report for CS 562, Fall 2022.

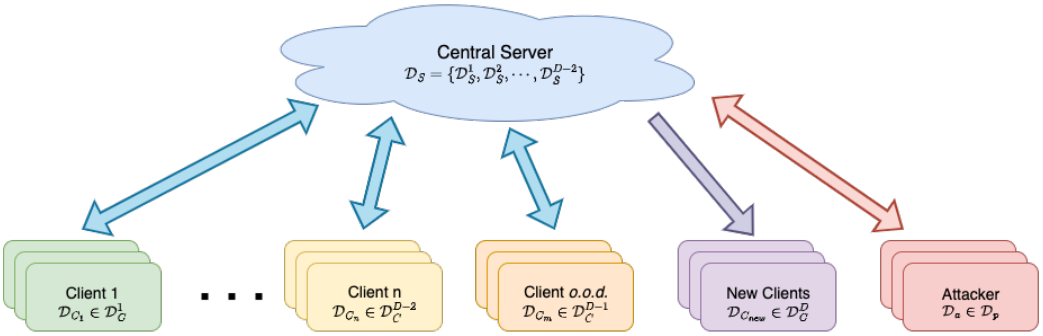

Figure 1: The setup of our proposed scenario for federated learning with o.o.d. clients and malicious clients.

Experiments are conducted on DomainNet [14]. Results show that our proposed method improves the worst group accuracy in a binary classification task across 6 domains, while the attack success rate of *Distributed Backdoor Attack* [18] inevitably increased due to the trade-off. We also perform ablation studies on different attacking rates in each malicious client and different trade-off parameters.

The contribution of our project is three-fold:

- We build a challenging scenario for Byzantine-robust federated learning by taking unseen domains into consideration. In this setting, the Byzantine-robust federated learning algorithm needs to deal with the o.o.d. clients and the malicious clients simultaneously.
- We show that the state-of-the-art FLTrust algorithm fails in domain generalization tasks by being too conservative in aggregation.
- We propose a novel method for training domain-generalizable models in a federated setting and define a trade-off parameter that controls the robustness and the generalization of the model.

2 Problem Setup

In our proposed scenario we proposed the challenging scenario where a federated learning model is trained with the existence of o.o.d. clients and malicious clients. The overall setup is shown in Figure 1. To be specific, in our framework, we have the server $\mathcal{S}$ and N clients $\mathcal{C}_i$ for $i \in N$. Following the training pipeline in [11], (i) the server distributes the global models to the clients; (ii) the clients train local models using private data and update to the server; and (iii) the server aggregate the local models. To make the algorithm Byzantine-robust, we adopt the FLTrust [2] algorithm, which requires the server to maintain a clean dataset $\mathcal{D}_{\mathcal{S}}$. In aggregation, the public dataset serves as the root of trust. Meanwhile, to evaluate the robustness of the algorithm, we introduce the attacker clients $\mathcal{C}_{\mathcal{A}}$. To take domain generalization into consideration, our dataset will be from $k+1$ domains, $i.e.$, $\mathcal{D} = \{\mathcal{D}^1, \cdots, \mathcal{D}^k\}$. Specifically, we leave one domain $\mathcal{D}^{+1}$ out as the unseen domain to perform the evaluation for domain generalization. In addition, the server dataset only contains data from $k-1$ domains, $i.e.$, $\mathcal{D}_{\mathcal{S}} = \{\mathcal{D}^i, \cdots, \mathcal{D}^{k-1}\}$, while the client contains data from k domains. While every client only contains data from one domain, we can separate our benign clients into IID clients $\mathcal{C}_I$ whose data is $\mathcal{D}_I^j$, $j \in [1, k-1]$, and o.o.d. clients $\mathcal{C}_O$ whose data is $\mathcal{D}_O^k$. Therefore, in the local update, data from all k domains are involved in training, however, in the global aggregation, the root of trust only contains data from $k-1$ domains. This setting will make the server treat the clients from the o.o.d. domain as attackers, and consequently defect the domain generalization.

3 Observation on FLTrust

To justify our setup, we perform a simple visualization using the FLTrust algorithm on the DomainNet dataset. The aggregation rule of FLTrust algorithm is shown in Figure 2. Given the server training on the public data and the local update on the private data, the FLTrust considers both the server

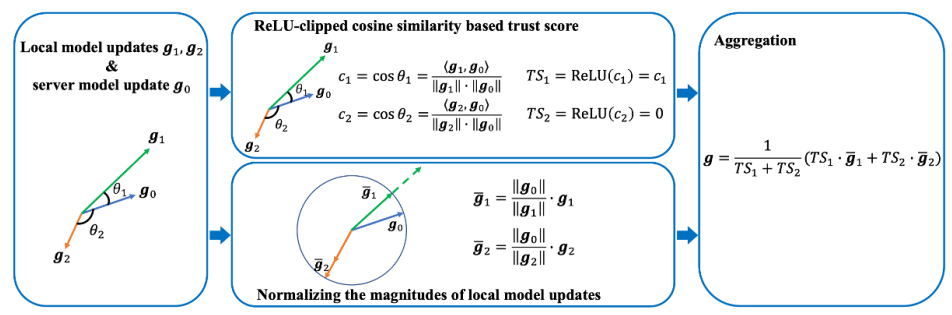

Figure 2: Illustration of FLTrust aggregation rule.

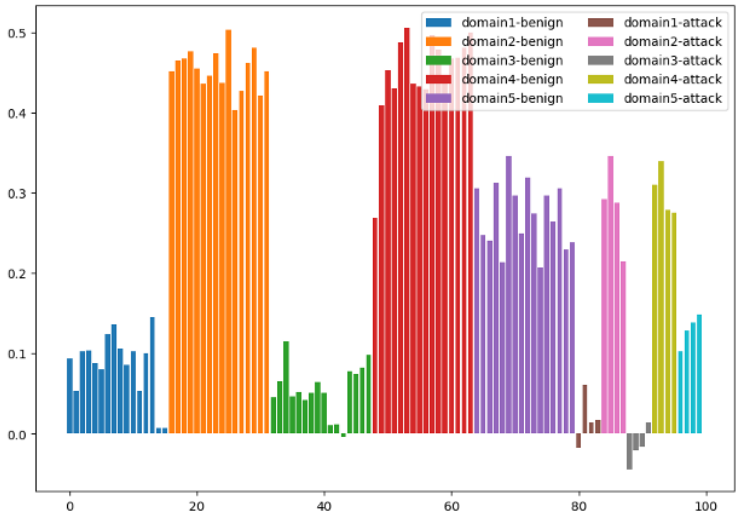

Figure 3: The cosine similarity between the local updates and global updates using FLTrust.

model update and the clients' local model updates and filters out the local update that deviates from the update direction of the server update. To measure how the local update deviates from the server update, FLTrust calculates the cosine similarity between the updates. Therefore, in this experiment, we locally train models on all the clients in our domain generalization setting for one epoch and measure their updates. We also perform one round of the server update and calculate their cosine similarity. The resulting cosine similarity is as shown in Figure 3. While Domain-1 and Domain-3 are the two most challenging domains in DomainNet, $i.e.$, their feature distribution is very different from the rest domains, we can see their cosine similarly with the global update is also very small. This is because the FLTrust algorithm cannot distinguish the update direction misalignment caused by o.o.d. data and poisoned data. FLTrust is too conservative by filtering out all the misaligned updates, which is not an issue if the domain data share similar distribution with the server data. However, if we simply perform FLTrust on the domain generalization setting, the weight for the o.o.d. domains will be downscaled, which leads to a failure on domain generalization tasks. We thus propose an algorithm that can encourage domain generalization and trade off the robustness with generalization.

4 Methods

In this section, we elaborate on our proposed methods. First, we introduce our improvements on the model structure and optimization goal. Then, we explain how we trade-off the robustness and generalization. The detailed algorithm is shown in Algorithm 1.

Algorithm 1 `FLTrust-Anchor`

Require: The server dataset $\mathcal{D}_{\mathcal{S}}$. The k client datasets $\mathcal{D}_{\mathcal{C}} = \{\mathcal{D}_{\mathcal{C}}^{i} | i = 1, \cdots, k\}$.

Server executes:
  for global epoch $t = 1, 2, \cdots, T$ **do**
    for each client $i = 1, 2, \cdots, k$ **in parallel do**
      $\mathbf{g}_i^{t+1}, \text{MMD}_k^{t+1} \leftarrow \text{ClientUpdate}(i, \theta^t)$
    $\mathbf{g}_s^{t+1} \leftarrow \text{ServerUpdate}(\mathbf{w}^t)$
    $\mathbf{w}_{br-dg} \leftarrow \text{ComputeAggregationWeight}(\mathbf{g}_c^{t+1}, \mathbf{g}_s^{t+1}, \text{MMD}^{t+1}, \gamma)$
    $\theta^{t+1} \leftarrow \theta^t + \sum_{i=1}^{k} \mathbf{w}_{br-dg}^i \cdot \mathbf{g}_i^{t+1}$
End

ClientUpdate(i, θ)**:** ▷ *Execute on client k*
  Receive $\theta = \{w_f, w_c\}$ from server
  $w_0 \leftarrow w_f$
  for local epoch $e = 1, 2, \cdots, E$ **do**
    Update w_f on $\mathcal{D}_{\mathcal{C}}^{i}$ and the fixed anchor **a** to minimize $\mathcal{L} = CE + \mathcal{L}_{\text{MMD}}$
  $\mathbf{g} \leftarrow w_f - w_0$
  Upload the trained **g** the final L_{MMD} to server.
End

ComputeAggregationWeight($\mathbf{g}_c$, $\mathbf{g}_s$, MMD, γ)**:**
  for each client $i = 1, 2, \cdots, k$ **do**
    $\mathbf{w}_{br}^i \leftarrow \frac{\text{ReLU}(\cos(\mathbf{g}_s, \mathbf{g}_{c_i}))}{\sum_{i=1}^{n} \text{ReLU}(\cos(\mathbf{g}_s, \mathbf{g}_{c_i}))}$
    $\mathbf{w}_{dg}^i \leftarrow \frac{\exp(\text{MMD}_i)}{\sum_{j=1}^{k} \exp(\text{MMD}_j)}$
    $\mathbf{w}_{br-dg}^i = (1 - \gamma) \cdot \mathbf{w}_{br}^i + \gamma \cdot \mathbf{w}_{dg}^i$
  Return $\mathbf{w}_{br-dg}$.
End

4.1 Anchor-Base Simplex-ETF Classifier

Intuitively, domain generalization can be achieved if the feature extractor learns a domain-invariant feature. In another word, we want the learned features share a similar distribution across all the domains. Therefore, we propose that, by adding a regularization term, we can align the features to the same distribution. We call the target distribution an *Anchor*. An illustration is shown in Figure 4. Given the anchor, we calculate the maximum mean discrepancy loss between the learned domain feature and the anchor distribution. In optimization, we minimize the MMD loss to regularize the features extracted by $F(\cdot)$, so that the distribution of the extracted features can be shifted towards the anchor $\mathbf{a} = \mathcal{N}(\mu, \Sigma)$.

To choose an appropriate anchor, we take advantage of the fact that neural features tends to collapse to a single point with infinite long time of training [12]. The feature from different classes will also form a simplex-ETF. In addition, the speed of collapse can be accelerated if we directly fix the classifier into a simplex-ETF [5]. Given all features in training across all domains can form the same simplex-ETF, we take the vertices if the simplex-ETF as the mean of the class-conditional anchors, and fix the model classifier into the simplex-ETF classifier. Intuitively, all the feature across the domains will align with the vertices of the simplex-ETF, and domain generalization can be consequently achieved.

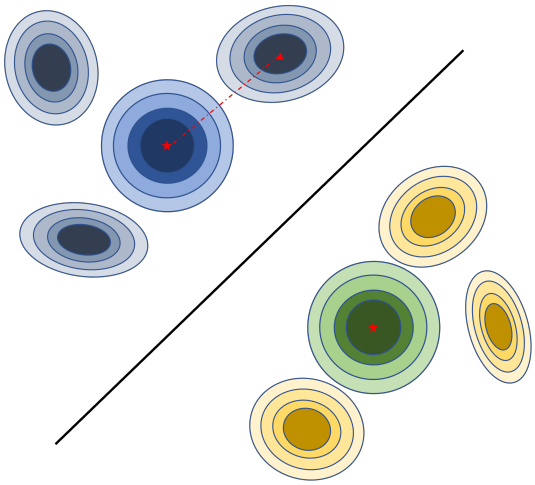

Figure 4: Illustration of the simplex-ETF anchors. The star indicates the mean of the anchor distribution. The triangle indicates the mean of the domain feature distribution. The red dash line is their maximum mean discrepancy. The black line is the fixed classifier.

4.2 Trade-off between Robustness and Generalization

The aggregation rule in FLTrust is being too conservative by scaling down or filtering our all the local updates that do not have the same direction as the server update. To be specific,

$$w_{br}^i = \frac{\text{ReLU}(\cos(g_s, g_{c_i}))}{\sum_{i=1}^n \text{ReLU}(\cos(g_s, g_{c_i}))} \tag{1}$$

where w_{br}^i is the Byzantine robustness weight for the i^{th} client. g_s is the gradient vector for the server update and g_{c_i} is the gradient vector for local update of the i^{th} client. FLTrust calculates the cosine similarity of the clients' updates and the server update. It then filter out all the updates that has negative cosine similarity by taking the ReLU operation. Then, all the weights are normalized to sum to 1. However, as we shown in Section 3, such filter according to cosine similarity cannot distinguish the clients from the poorly generalized domains and the poisoned domains. Therefore, we want to slightly relax such criterion, by scaling up the local updates that do not generalize well, $i.e.$, the MMD loss is large. Specifically,

$$w_{dg}^i = \frac{\exp(\text{MMD}(z^i, \mathcal{N}(\mu_a, \sigma_a)))}{\sum_{i=1}^n \exp(\text{MMD}(z^i, \mathcal{N}(\mu_a, \sigma_a)))}, \tag{2}$$

where w_{dg}^i is the domain generalization weight for the i^{th} client. We calculate the MMD of the clients' features z^i and the fixed anchors $\mathcal{N}(\mu_a, \sigma_a)$. We then calculate the softmax of the MMDs to build the domain generalization weight. To control the trade-off between the Byzantine-robustness weight and the domain generalization weight, we adopt a trade-off parameter γ, and the resulting robustness-generalization weight becomes

$$w_{br-dg} = (1-\gamma) \cdot w_{br} + \gamma \cdot w_{dg}. \tag{3}$$

By controlling γ, we can decide whether be more conservative against poisoned data or be more generous for domain generalization. In FLtrust, the local gradients are also normalized according to the magnitude of the server gradients. Thus, the weight for aggregation is

$$w^i = w_{br-dg}^i \cdot \frac{\|g_{c_i}\|}{\|g_s\|} \tag{4}$$

where w^i is the aggregation weigh for the i^{th} client's update.

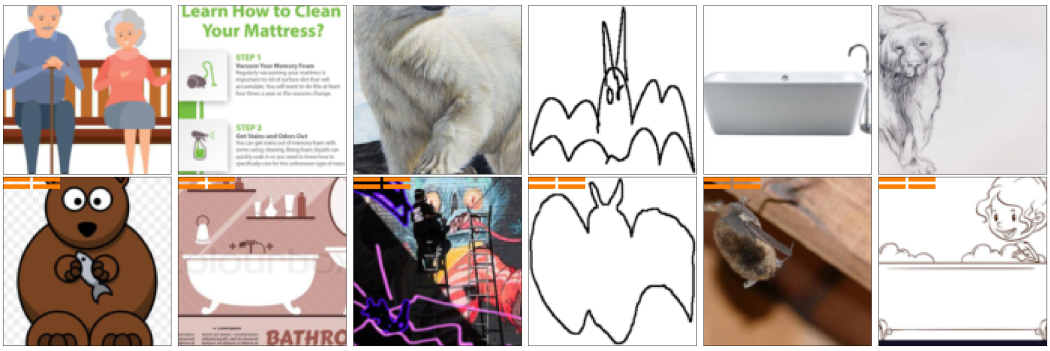

Figure 5: Some sample data in our problem setting. Each column represents data from one domain. The first row is the clean data, the second row is the poisoned data with the global trigger. In the training phase, only one of the four triggers will be injected into the target class.

5 Empirical Study

5.1 Experiments

We perform empirical validation of the proposed algorithm on a multi-domain dataset, DomainNet [14], which contains images of objects from six different domains: *cliparts, inforgraph, painting, quickdraw, real, sketch*. Among these, *inforgraph* and *quickdraw* are the two most challenging domains.

Task In order to study domain generalization in the federated learning setting with data poisoning, we perform a custom binary classification task on DomainNet:

- **Target**: Maximizing the Worst Group Accuracy (WGA) of a Furniture vs. Mammal task on the six domains, while minimizing the Attack Success Rate (ASR).
- **Poisoning Attack**: Distributed Backdoor attack [18], which injects local triggers into different clients during training, and evaluates the attack success rate using the global triggers. We will attack the Furniture class by injecting triggers to the image in the class.
- **Dataset Splitting**:
  - ☐ Server: 1% of the training data from domain *painting, quickdraw, real, sketch*. Data from domain *cliparts, inforgraph* are invisible to the server.
  - ☐ Clients: The rest data from domain *inforgraph, painting, quickdraw, real, sketch*. Each domain has 20 clients. Among the 20 clients, 4 clients are attackers and 16 are benign clients. The clients containing data from domain *inforgraph* is the o.o.d. domain. To sum up, there are 100 clients, 80 of them are benign clients and 20 are attackers.
  - ☐ Evaluation: We use the testing split of the dataset. Notice that in the evaluation, we first test on the original testing set to evaluate domain generalization. Then, we inject global triggers into the Mammal class and evaluate the attack success rate on the poisoned data. Some samples of the data is shown in Figure 5.

Empirical Implementation We implement our code in PyTorch [13] on a single NVIDIA A6000 GPU. Specifically, we adopt the following techniques and hyper-parameters

- Network structure: ResNet-50 [6] with fixed simplex-ETF classifier [5].
- Optimizer: SGD [15] with learning rate $1e-3$ and weight decay $5e-4$.
- Training Paradigm: We train our model 100 global epochs. In each round, 10 out of the 100 clients are sampled, and each client has a batch of 64. If the client is a benign client, it locally trains for 10 iterations. If the client is an attacker, it locally trains for 20 iterations and scales up the local gradient 50 times. In attacker, 20 out of the 64 images in each batch are poisoned images.
- The trade-off parameter: $\gamma = 0.4$

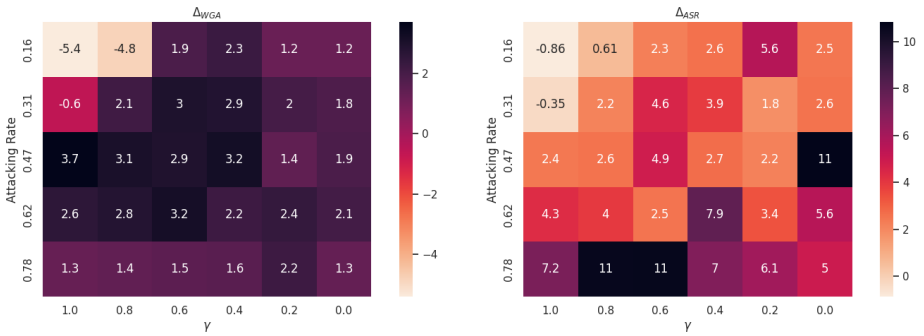

Figure 6: The change of WGA and average ASR after applying our proposed algorithm with varying attacking rate and trade-off parameter γ.

Empirical Results We present our empirical results in Table 1 and Table 2. As we can see, our proposed algorithm indeed improves the WGA over the baseline (FLTrust) in our domain generalization task on DomainNet data. Meanwhile, as expected, the attack success rate of our method is increased. These results clearly show that our proposed algorithm makes the trade-off between domain generalization and Byzantine robustness.

Method	clipart	infograph	paint	quickdraw	real	sketch	avg. acc.	worst acc.
FLTrust	82.88	71.46	85.39	72.36	95.63	82.23	83.00	71.46
Ours	82.84	74.32	85.49	74.54	94.32	82.31	**83.49**	**74.32**

Table 1: Accuracy of our proposed task on DomainNet.

Method	clipart	infograph	paint	quickdraw	real	sketch	avg. ASR	best ASR
FLTrust	14.61	11.47	19.86	5.49	3.77	15.15	**9.00**	**19.86**
Ours	21.02	12.41	29.15	5.77	7.83	23.84	12.91	23.84

Table 2: Attack success rate of our proposed task on DomainNet.

5.2 Ablation Study

To explore how the attacking rate and trade-off parameter will affect domain generalization and Byzantine robustness, we performed an ablation study by varying the attacking rate and the trade-off parameter. The results are shown in Figure 6. The left figure shows the WGA improvement (can be negative) of our proposed method over the original FLTrust with different attacking rates and trade-off parameters γ. The right figure shows the average ASR will raise as the WGA improves. We can tell from the figure generally we want a larger γ when the attacking rate is high. To be specific, we notice that when the attacking rate is 0.47, the improvement is in WGA it the higher if complete aggregate the local models according to the MMD loss. Intuitively, this is because DBA succeeded in attacking federated models by having the framework to learn the spurious feature (trigger). By requiring all the domains to learn the same distribution, we can also train a robust model that only depends on causal features but not the triggers. However, this hypothesis needs more experiments and theories to prove.

6 Summary

In this project, we proposed a novel and challenging scenario: when o.o.d. clients and malicious clients coexist in federated learning, how can we achieve domain generalization while being Byzantine-robust against the poisoned data? To demonstrate that this task is non-trivial, we first show that the state-of-the-art FLTrust algorithm fails in aggregating clients from o.o.d. domains. We then proposed our method to trade off Byzantine robustness and domain generalization. We

show the effectiveness of our method on a binary classification task. Due to the limitation of time and computation resources, we only perform experiments on one dataset using one robust federated aggregation algorithm and one federated data poisoning method. In the future, more experiments can be conducted to explore how our proposed method can be applied to other attacking and defending scenarios. We believe balancing domain generalization and Byzantine robustness is an important topic to be worked on before federated learning can be applied in the industry, and we hope our project can inspire future works to further explore this field.

References

[1] B. Biggio, B. Nelson, and P. Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.

[2] X. Cao, M. Fang, J. Liu, and N. Z. Gong. Fltrust: Byzantine-robust federated learning via trust bootstrapping. *arXiv preprint arXiv:2012.13995*, 2020.

[3] Y. Cao and Q. Gu. Generalization bounds of stochastic gradient descent for wide and deep neural networks. In *Advances in Neural Information Processing Systems*, pages 10835–10845, 2019.

[4] C. Fung, C. J. Yoon, and I. Beschastnikh. Mitigating sybils in federated learning poisoning. *arXiv preprint arXiv:1808.04866*, 2018.

[5] X. Han, V. Papyan, and D. L. Donoho. Neural collapse under mse loss: Proximity to and dynamics on the central path. *arXiv preprint arXiv:2106.02073*, 2021.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] J. Konečnỳ, B. McMahan, and D. Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

[8] Q. Liu, C. Chen, J. Qin, Q. Dou, and P.-A. Heng. Feddg: Federated domain generalization on medical image segmentation via episodic learning in continuous frequency space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1013–1023, 2021.

[9] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang. Trojaning attack on neural networks. 2017.

[10] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[11] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[12] V. Papyan, X. Han, and D. L. Donoho. Prevalence of neural collapse during the terminal phase of deep learning training. *Proceedings of the National Academy of Sciences*, 117(40):24652–24663, 2020.

[13] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32:8026–8037, 2019.

[14] X. Peng, Q. Bai, X. Xia, Z. Huang, K. Saenko, and B. Wang. Moment matching for multi-source domain adaptation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1406–1415, 2019.

[15] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[16] R. Shokri and V. Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321, 2015.

[17] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu. Data poisoning attacks against federated learning systems. In *European Symposium on Research in Computer Security*, pages 480–501. Springer, 2020.

[18] C. Xie, K. Huang, P.-Y. Chen, and B. Li. Dba: Distributed backdoor attacks against federated learning. In *International Conference on Learning Representations*, 2019.