What if the Adversary is Truly Byzantine? Optimal Attacks in Distributed Machine Learning

Haowen Liu*
haowen.liu@epfl.ch

Abstract

Distributed machine learning (ML) has shown impressive success on training large models and preserving data privacy and been widely accepted by academia and industry. Byzantine resilience has thus become a critical requirement to ensure that distributed optimization algorithms are protected against malicious machines, a.k.a., *Byzantine* workers. Such misbehaving workers can hinder the learning procedure by sharing erroneous information. Although many defense schemes have been proposed so far, the absence of a strong attack benchmark provides a false sense of security to those techniques. Indeed, many existing attacks, though successful in breaking some defenses, lack theoretical foundations and do not embody the worst-case behavior of a *Byzantine* worker.

In this work we provide the first principled approach to designing *truly* Byzantine attacks on a distributed learning procedure. We formulate the Byzantine attack problem as a generic optimization problem and show that it admits exact and near-optimal solutions under mild assumptions on the defense mechanism and the considered loss function. We then introduce a methodical approach to solve the Byzantine attack problem by utilizing Nonlinear Programming (NLP) to solve the optimization problem. Experiments are conducted across mean estimation examples, MNIST logistic regression tasks and normal MNIST CNNs tasks, showing that our optimal attacks outperforms all state-of-the-art attacks across different tasks and robust aggregation algorithms. The results also indicates that there is still room for refinement in the current attack strategies, while existing defense mechanisms may not be fully equipped to thwart more advanced attacks.

1 Introduction

Distributed machine learning (ML) has witnessed an increasing rise in popularity in the recent years. A widely adopted structure for executing distributed ML tasks is the parameter server model [Dean et al. (2012)]. In this setup, the dataset is distributed across multiple machines, referred to as workers, which communicate solely through a centralized parameter server that is trusted [McMahan et al. (2017)].

A natural outcome of distributing the learning process across multiple machines is the emergence of safety and reliability issues. Some machines may falter, while others could transmit incorrect data [Blanchard et al. (2017)]. In the most severe cases, an adversary could gain control over some machines and disseminate malicious information to disrupt the learning. Absent adequate precautions, this can result in seriously flawed models. In the context of Distributed Stochastic Gradient Descent (DSGD) [Boyd et al. (2011); Dean et al. (2012)], a single faulty worker, termed as Byzantine, is sufficient to deviate the gradient's estimate computed by the server. To tackle Byzantine gradients, several defense mechanisms have been proposed to substitute the averaging operation in the parameter server architecture, e.g., Krum [Blanchard et al. (2017)], Geometric Median (RFA)

*Semester project supervised by Youssef Allouah at DCL, EPFL, Switzerland. June 9th, 2023

[Chen et al. (2017)], Coordinate-wise Median (CM) Yin et al. (2018), Coordinate-wise Trimmed Mean (TM) [Yin et al. (2018)], and Centered Clipping [Karimireddy et al. (2021)].

However, robust aggregation rules are continually challenged by practical attacks. Various attacks have been designed to defeat existing aggregation rules on known classification tasks with success: label flipping (LF) [Barreno et al. (2006); Yin et al. (2018)], bit flipping or sign flipping (BF) [Yin et al. (2018)], ALIE [Baruch et al. (2019)], Inner Product Manipulation (IPM) [Xie et al. (2020)], mimic [Karimireddy et al. (2020)], MinMax and MinSum [Shejwalkar and Houmansadr (2021)]. Unfortunately, most known attacks are heuristic, rooted in existing defenses, and lack a measurable strength metric. This gives rise to a misleading sense of security when testing against these attacks, which might not realistically simulate potent real-world attacks. This work aims at (i) formalizing the adversary's goal in attacking DSGD as an optimization problem, and (ii) providing a methodology to design strong attacks. Since we are interested in the worst-case attack, i.e., a Byzantine adversary, we assume full-knowledge of the adversary. We then leverage Nonlinear Programming (NLP) [Bertsekas (1997); Fletcher and Powell (1963)] to tackle the optimization problem, seeking to identify nearly optimal attack strategies. We also conduct experiments using state-of-the-art[2] defense mechanisms to showcase the strength of the attacks designed using our methodology. As such, an important contribution of this work is to provide a solid benchmark for the stress tests of future defense mechanisms.

2 Background

2.1 Byzantine Machine Learning

Byzantine Machine Learning (ML) delineates an aspect of distributed learning where resilience to components behaving maliciously or faultily, termed as Byzantine workers, is required [Blanchard et al. (2017); Farhadkhani et al. (2022)]. Such elements can adulterate the learning procedure by disseminating erroneous or deceptive information.

The recent years have witnessed a considerable expansion in Byzantine ML, driven by an escalating intricacy of distributed ML tasks and a need for security against potential cyber threats. As distributed learning architecture becomes ubiquitous, the importance of preserving data privacy and ensuring system integrity in adversarial scenarios amplifies. Despite its nascent stage, Byzantine ML has seen significant progression, with numerous defense mechanisms and robust algorithms proposed to counteract Byzantine behaviors. Nonetheless, the ever-evolving nature of potential threats, along with the inherent complexity of distributed systems, poses Byzantine resilience as a formidable challenge.

The significance of Byzantine ML is manifold. In an epoch characterized by an increasing reliance on data and ML models to inform critical decisions, safeguarding the resilience and integrity of these models is of paramount importance. Byzantine breaches can lead to critically defective models, resulting in adverse consequences [Blanchard et al. (2017); Baruch et al. (2019); Shejwalkar and Houmansadr (2021)], ranging from distorted insights to substantial financial losses, and even endangering human safety in certain domains. Therefore, sustained advancements in Byzantine ML, particularly in the development of robust defenses and comprehensive understanding of prospective attack strategies, remain indispensable for the future of secure, distributed machine learning.

2.2 Defenses

In the realm of distributed machine learning, ensuring robustness against adversarial or faulty nodes, collectively referred to as Byzantine behaviors, is paramount. In this context, the primary defenses are broadly classified into *robust aggregation* methods and *pre-aggregation* techniques. These defensive mechanisms aim to maintain the reliability of the learning process in the face of potentially disruptive Byzantine behaviors. The robust aggregation methods primarily focus on the effective synthesis and aggregation of gradients from worker nodes, while the pre-aggregation techniques perform initial processing to ameliorate data heterogeneity issues before robust aggregation.

Robust aggregation methods form the backbone of Byzantine Machine Learning defenses. These techniques focus on aggregating gradients or model updates from multiple worker nodes in a manner that minimizes the influence of Byzantine nodes. Prominent among these are methods such as

[2]Including those using distributed momentum, which have not been broken yet.

Krum [Blanchard et al. (2017)], which selects a model with the minimum distance to $n - f$ other models, and Median-based methods [Chen et al. (2017); Yin et al. (2018)] that consider the median value of each gradients across all nodes. Another technique, Coordinate-wise Trimmed Mean [Yin et al. (2018)], operates by trimming off the extreme values on each coordinate before taking the average. These methods contribute to the robustness of the system by effectively identifying and mitigating potentially disruptive information from Byzantine nodes.

However, these aggregation defenses often rely on the assumption of data homogeneity, which may not hold true in practical scenarios. To address this, pre-aggregation techniques come into play. These methods provide a preliminary processing step to tackle data heterogeneity issues before robust aggregation. One such technique is the Nearest Neighbor Mixing (NNM) [Allouah et al. (2023)] which replaces each vector with the average of its $n - f$ nearest neighbors. Another technique, known as Bucketing [Karimireddy et al. (2020)], 'mixes' the data from all workers, reducing the chance of any subset of the data being consistently ignored. These pre-aggregation techniques thus adapt and prepare the data, making traditional Byzantine defenses more resilient in the face of data heterogeneity, and enhancing the overall robustness of distributed machine learning systems.

2.3 Attacks

The landscape of Byzantine attacks in distributed machine learning encompasses a wide spectrum of strategies, with varying degrees of attacker knowledge and adaptability. These attacks are broadly categorized into two types: *non-adaptive attacks* and *adaptive attacks* [Shejwalkar and Houmansadr (2021)]. Non-adaptive attacks are orchestrated by adversaries with limited knowledge and do not adapt their strategies according to the behaviors of other participants or the system. On the other hand, adaptive attacks are initiated by adversaries with extensive knowledge and are capable of altering their tactics based on the overall system context.

Non-adaptive attacks are characterized by adversaries whose actions are predefined and non-responsive to the system or other participants' behaviors. One example is proposed in [Baruch et al. (2019)], where attackers craft Byzantine values carefully, staying within the bounds of existing defenses to subtly induce small changes in many parameters. The attack showcases the potential of these malicious activities to significantly degrade model accuracy or even embed backdoors. Another work [Xie et al. (2020)] underscores the importance of the direction of gradient descent in addition to the robustness of estimators, challenging the Byzantine tolerance of traditional robust aggregation rules like Coordinate-wise Median and Krum. These instances illustrate the capacity of non-adaptive attacks to exploit the blind spots of existing defenses, revealing critical areas for improvements in defensive mechanisms.

In contrast, adaptive attacks are masterminded by adversaries possessing broad knowledge about the system, including the server's robust aggregation algorithm, and data distribution across all participants. They are capable of dynamically adjusting their attack strategies, making them particularly potent and challenging to defend against. The study [Shejwalkar and Houmansadr (2021)] provides a framework for model poisoning, crafting aggregation-tailored attacks for each state-of-the-art aggregation algorithm, while also proposing two aggregation-agnostic attacks MinMax and MinSum. Additionally, [Li et al. (2022)] presents an attack framework that integrates distribution learning and policy learning, where malicious workers can learn an approximation of the empirical distribution across all workers and generate attack actions based on the learned policy. These adaptive attacks underscore the need for defenses to anticipate and counter complex, evolving strategies, ensuring the robustness and reliability of distributed machine learning systems.

3 Problem Statement

We consider the parameter server architecture with n workers $w_1, \ldots, w_n$, and a trusted central server. The workers only communicate with the server and there is no inter-worker communication. We consider a classification task with input space $\mathcal{X}$ and output space $\mathcal{Y}$. We let $\mathcal{D}$ denote the data generating distribution over $\mathcal{X} \times \mathcal{Y}$. We denote the space of model parameters by Θ, and assume it to be compact. For a given model parameter $\theta \in \Theta$, a data point $z \sim \mathcal{D}$ has a real-valued loss function $q(\theta, z)$. We assume that only the workers have access to the data. The server aims to compute, by collaborating with the workers, a parameter θ^* that minimizes the expected loss function $Q(\theta)$

defined to be

$$Q(\theta) = \mathbb{E}_{z\sim\mathcal{D}}[q(\theta, z)],\ \forall\theta \in \Theta. \tag{1}$$

When all the workers are honest, i.e., follow the prescribed instructions correctly, the above optimization problem can be solved using the distributed variant of vanilla SGD presented below [Bertsekas and Tsitsiklis (2015)].

3.1 Standard Distributed SGD

This is an iterative algorithm where, at each step $t = 1,\ 2, \ldots$, the server maintains a parameter vector θ_t which is broadcast to all the workers. Each worker w_i then returns an *unbiased* stochastic estimate $g_t^{(i)}$ of the gradient $\nabla Q(\theta_t)$ by independently sampling a data point z from $\mathcal{D}$:

$$g_t^{(i)} = \nabla q(\theta,\ z). \tag{2}$$

Ultimately, the server updates θ_t by using the average of the received gradients as follows,

$$\theta_{t+1} = \theta_t - \gamma_t \frac{1}{n}\sum_{i=1}^{n} g_t^{(i)} \tag{3}$$

where $\gamma_t \geq 0$ is referred to as the *learning rate*. When all the workers are honest, the above algorithm provably converges to a critical point of function Q, under the classical assumptions on the Lipschitz smoothness of Q and bounded variance of stochastic gradients [Ghadimi and Lan (2013)].

3.2 Byzantine Robust Distributed SGD

We consider the scenario where at most f workers of unknown identity are *Byzantine*. These workers need not obey the prescribed algorithm, and may send arbitrary information to the server. Following the Byzantine threat model [Lamport et al. (1982)], we assume these workers to be *omniscient*, meaning they have full knowledge of the protocol and the workers' identities. However, Byzantine workers are not *omnipotent*, meaning that they cannot make honest workers disobey the protocol. As they can *collude*, we assume for simplicity that all Byzantine workers are controlled by a single adversary.

To solve the learning problem in the presence of Byzantine workers, a standard approach for the server is to use a Byzantine robust version of the distributed SGD algorithm (or BR-SGD) [Blanchard et al. (2017)]. Essentially, at each iteration t of BR-SGD the server uses a *robust aggregation* of the received gradients $g_t^{(1)}, \ldots, g_t^{(n)}$, denoted by $F\left(g_t^{(1)}, \ldots, g_t^{(n)}\right)$, instead of the simple averaging. Specifically, the updated rule (3) is modified as follows:

$$\theta_{t+1} = \theta_t - \gamma_t F\left(g_t^{(1)}, \ldots, g_t^{(n)}\right).$$

An effective aggregation rule F seeks to filter out the incorrect gradients sent by the Byzantine workers. Some of the most prominent such aggregation rules include *minimum-diameter averaging* (MDA) [El-Mhamdi et al. (2020)], *Median* [Yin et al. (2018)], and *Trimmed mean* [Yin et al. (2018)].

3.3 Optimal Adversary

At each step $t = 1, \ldots, T$ of BR-SGD, We can formalize the Byzantine workers by an adversary sending f malicious gradients computed using a decision function ψ_t:

$$g_t^{(n-f+1)}, \ldots, g_t^{(n)} = \psi_t(P_t),$$

where P_t is the trajectory's past; that is, all the parameters and gradients exchanged up to time t:

$$P_t \coloneqq \left((\theta_k)_{1\leq k\leq t}, (g_k^{(i)})_{\substack{i\in\mathcal{R}\\ 1\leq k\leq t}}, (g_k^{(j)})_{\substack{j\in\mathcal{B}\\ 1\leq k\leq t-1}}\right).$$

The full procedure of BR-SGD including the adversary's actions is summarized in Algorithm 1.

The goal of the adversary is to maximize the loss of the models encountered during the training procedure, by finding an appropriate strategy. This strategy corresponds to the sequence of decision

Algorithm 1: Byzantine Robust Stochastic Gradient Descent (BR-SGD)

Initialization: **Server** chooses an arbitrary initial parameter vector $\theta_1 \in \Theta$, a set of T learning rates $\{\gamma_1, \ldots, \gamma_T\}$, a robust aggregation rule $F : \mathbb{R}^{d \times n} \to \mathbb{R}^d$.

for $t = 1, \ldots, T$ **do**

Server broadcasts θ_t to all the workers;

Honest worker w_i samples and sends to the server a stochastic gradient

$$g_t^{(i)} \sim \mathcal{G}_i(\theta_t); \tag{4}$$

Adversary computes and sends to the server f Byzantine gradients

$$g_t^{(n-f+1)}, \ldots, g_t^{(n)} = \psi_t(P_t)\ ; \tag{5}$$

Server updates the parameter vector as follows

$$\theta_{t+1} = \theta_t - \gamma_t F\Big(g_t^{(1)}, \ldots, g_t^{(n)}\Big)\ ; \tag{6}$$

end

return $\hat{\theta}$ *sampled uniformly at random from* $\{\theta_1, \ldots, \theta_T\}$

functions $(\psi_1, \ldots, \psi_T)$. Given an aggregation function F, we formalize the objective of the adversary by the following optimization problem:

$$\begin{aligned}
\underset{\psi_1,\ldots,\psi_T}{\text{Maximize}} \quad & \mathbb{E}\left[\min_{t\in[T]} Q(\theta_t)\right] \\
\text{Subject to:} \quad & \theta_{t+1} = \theta_t - \gamma_t F\Big(g_t^{(1)}, \ldots, g_t^{(n)}\Big), \\
\forall t \in [T], \quad & g_t^{(n-f+1)}, \ldots, g_t^{(n)} = \psi_t(P_t), \\
& P_t = \left((\theta_k)_{1\le k\le t}, (g_k^{(i)})_{\substack{i\in\mathcal{R}\\ 1\le k\le t}}, (g_k^{(j)})_{\substack{j\in\mathcal{B}\\ 1\le k\le t-1}}\right) \in H_t, \\
& \psi_t : H_t \to \mathbb{R}^{d\times f} \text{ is Borel measurable.}
\end{aligned} \tag{7}$$

Above, we have denoted $H_t := \mathbb{R}^{d\times t} \times \mathbb{R}^{d\times(n-f)\times t} \times \mathbb{R}^{d\times f\times(t-1)}$ to be the space of the variables seen by the adversary at time t within history P_t. The decision variables of optimization Problem (7) correspond to decision rules ψ_t that take the history of the trajectory and the previous decisions as input, and output a decision consisting of f Byzantine crafted gradients. These decision rules are assumed to be Borel measurable in order for the expectation to be well-defined. Note also that the expectation in the objective is taken over the randomness of the algorithm.

By aiming to maximize the expected value of $\min_{t\in[T]} Q(\theta_t)$, the adversary guarantees maximization of the loss independently of the model output by the server at the completion of the training process.

4 Design of Attacks

In the previous chapter, we have formulated the Byzantine optimal attack strategy into a general optimization problem. In this chapter, we will design a principled attack, which aims to solve this optimization problem using nonlinear programming, thereby seeking the optimal attack strategy. The formula for the nonlinear programming problem to be solved by this attack is as follows:

$$\begin{aligned}
\underset{\mathbf{a}_1:\mathbf{a}_T}{\text{Maximize}} \quad & \sum_{t=1}^{T} \alpha_t \mathcal{L}(\theta_t, \mathbf{s}_t) \\
\text{Subject to:} \quad & \theta_{t+1} = \theta_t - \gamma_t F(\mathbf{g}_t, \mathbf{a}_t), \\
& \mathbf{g}_t = \nabla_{\theta_t} \mathcal{L}(\theta_t, \mathbf{s}_t),
\end{aligned} \tag{8}$$

where in iteration t, $\mathbf{s}_t$ is the input fixed and given, $\mathbf{g}_t$ is gradients of honest workers, $\mathbf{a}_t$ is gradients of Byzantine workers, $\mathcal{L}$ is the loss function, γ_t is the learning rate, and α_t is the objective function coefficient.

Algorithm 2: Omniscient attack algorithm

Initialization: Number of lookahead steps ℓ, decision functions set Ψ, number of chosen plannings L.

while $t \leq T$ **do**

  for $j = 1 \dots L$ **do**

    Compute $\mathrm{cost}(\mathbf{u}_t^j)$ for every possible controls sequence $\mathbf{u}_t^j$, using Equation (9) and starting from θ_t^j;

    Update control sequences with the best L controls sequences (lowest costs);

  $t \leftarrow t + \ell$;

Select

$$u_1, \dots, u_T \leftarrow \underset{1 \leq j \leq L}{\operatorname{argmax}} \min_{t \in [T]} Q(\theta_t(\mathbf{u}_t^j));$$

return *sequence of malicious gradients* $u_1, \dots, u_T$;

Unfortunately, given that the value spaces are continuous, formula 8 is intractable in practice. This is not surprising, since finding a near-optimal attack requires searching the gradients' space, a continuous high-dimensional space, while attempting to maximize loss Q, which may have a highly non-convex landscape. Nevertheless, it is crucial to notice that the key ingredient of formula 8 is the use of lookahead, since it optimizes for the expected optimal cost.

To overcome the intractability of formula 8, we devise a heuristic inspired from the field of approximate dynamic programming [Bertsekas (2012)], whose main characteristic is the use of lookahead. We place ourselves in the worst-case scenario where the adversary is fully omniscient. That is, the adversary knows in advance the realizations of the stochastic gradients sampled by honest workers. In practice, this can be realized when the adversary knows the random seeds of the honest workers. In this setting, the problem of finding an optimal attack becomes a deterministic optimal control problem, whose goal is to find a sequence of controls (malicious gradients).

Our heuristic consists in using multi-step lookahead to plan controls (malicious gradients) for ℓ training steps, iteratively until the end of the training. At planning time $t \in \{1, \ell + 1, 2\ell + 1 \cdots, T\}$, the cost of the controls sequence $\mathbf{u}_t \coloneqq (u_t, \dots, u_{t+\ell-1})$, inducing trajectory $\theta_{t+1}(\mathbf{u}_t), \dots, \theta_{t+\ell}(\mathbf{u}_t)$, is

$$\mathrm{cost}(\mathbf{u}_t^j) \coloneqq \min \left(\min_{1 \leq k \leq t} Q(\theta_k^j), \min_{1 \leq i \leq \ell} Q(\theta_{t+i}(\mathbf{u}_t^j)) \right). \tag{9}$$

Each control $u_{t+i}, 0 \leq i \leq \ell - 1$ in controls sequence $\mathbf{u}_t$ is constrained, given the corresponding seen state s_{t+i} (the tuple of parameter θ_{t+i} and honest gradients $\mathbf{g}_{t+i}^{\mathcal{R}}$) as follows:

$$u_{t+i} \in \{\psi(s_{t+i}) \mid \psi \in \Psi\},$$

where Ψ corresponds to a predefined finite set of decision functions, which can include existing attacks.

At planning time t, we keep the L control sequences with the lowest costs. Each controls sequence $\mathbf{u}_t$ induces a new starting state $\theta_{t+l}(\mathbf{u}_t)$. Starting from each of the L new starting states, we plan again for ℓ steps and keep the L best control sequences. The algorithm outputs the controls sequence with the maximum minimum loss. The algorithm is given in Algorithm 2.

5 Experiments

In the following section, we present a series of experiments aimed to validate our methods and measure their performance and robustness in various settings. The experiments are designed with a progressive roadmap, starting with simple tasks and gradually moving to more complex, real-world scenarios. The overarching goal of these experiments is to elucidate the practical applicability and efficiency of our proposed approach, while also examining its limitations and potential areas for future enhancements.

Initially, we explore a simple two-dimensional mean estimation problem. This toy example allows us to investigate the fundamental behavior of our method in a controlled and comprehensible setting.

We will implement our method and experiment with various heuristics and a non-linear programming (NLP) solver to provide insights into how the proposed solution performs in this basic scenario.

Following the initial analysis, contingent upon the results, we will expand our evaluation to more realistic and challenging tasks. Our next step will be to apply our method to the standard machine learning task of recognizing handwritten digits from the MNIST dataset, using Logistic Regression as the model of choice. This experiment will offer a deeper understanding of how our method behaves in a more practical context, as well as the impact of our approach on the model's predictive performance.

Lastly, depending on the success of the previous tasks, we will further scale up our experiments to use Convolutional Neural Networks (CNNs) on the MNIST dataset. This will provide us with a more stringent test of the proposed method, as CNNs are known for their complexity and computational intensity. And we will also conduct experiments on homogeneity setting and heterogeneity setting. The results from these more challenging tasks will provide a comprehensive assessment of the scalability and robustness of our method. Our GitHub repository is `https://github.com/MekAkUActOR/Byzantine_ML_Attacks`.

5.1 Experimental Setting

2D Mean Estimation Problem: We conduct experiments on simple two-dimensional mean estimation tasks. We are use a distributed setting with a total of 12 workers (including Byzantine workers). The number of Byzantine workers is 5. At each step, one point is sampled from the training dataset for each honest worker. We use 5 robust aggregation algorithms : Krum [Blanchard et al. (2017)], trimmed mean (TM) [Yin et al. (2018)], coordinate-wise median (CM) [Yin et al. (2018)], centered-clipping (Clipping) [Karimireddy et al. (2021)], and geometric median (RFA) [Chen et al. (2017)] which are known to be robust gradient aggregation rules. We use 6 state-of-the-art attacks – ALIE [Baruch et al. (2019)], BF [Yin et al. (2018)], IPM [Xie et al. (2020)], mimic [Karimireddy et al. (2020)], MinMax and MinSum [Shejwalkar and Houmansadr (2021)] – as reference points for evaluating the effectiveness of the optimal attack strategy we have designed. All experiments are seeded for reproducibility. All Mean Estimation experiments use a neural network containing 2 neurons, and outputs are weights of two neurons. We set the learning rate to 0.5 and clip the gradients with a norm of 2. We train for 8 steps.

MNIST Logistic Regression: We conduct experiments on classification tasks using MNIST dataset with Logistic Regression method. The batch size is 32. We are use a distributed setting with a total of 8 workers (including Byzantine workers). The number of Byzantine workers is 3. At each step, one batch is sampled from the training dataset for each honest worker. We also use the 5 robust aggregation algorithms and 6 attacks as in Mean Estimation experiments for evaluating the effectiveness of the optimal attack strategy. All Logistic Regression experiments use a fully connected neural network of 1 fully connected layer with 784 inputs and a softmax at the end. We set the learning rate to 0.01. We train for 640 steps separately on two settings: without momentum and with distributed momentum (local momentum) [Farhadkhani et al. (2022)].

MNIST CNN: We conduct experiments on classification tasks using MNIST dataset with CNNs. The batch size is 32. We are use a distributed setting with a total of 15 workers (including Byzantine workers). The number of Byzantine workers is 4. At each step, one batch is sampled from the training dataset for each honest worker. We use 2 robust aggregation algorithms : Krum and coordinate-wise median (CM). And we still use the 6 attacks as in Mean Estimation tasks, with additional LF [Yin et al. (2018)]. All CNNs experiments use a structure of Conventional(1, 32, 3, 1)-ReLU-Conventional(32, 64, 3, 1)-Maxpool(2)-Dropout(0.25)-Fullyconnected(9216, 128)-ReLU-Dropout(0.5)-Fullyconnected(128, 10)-Softmax. We set the learning rate to 0.01. We train for 320 steps separately on two settings: homogeneity without momentum, and heterogeneity with D-SHB [Allouah et al. (2023)]. And we also challenge a state-of-the-art pre-aggregation method Nearest Neighbor Mixing (NNM) [Allouah et al. (2023)] with our optimal attack strategy.

5.2 Experiments on 2D Mean Estimation Tasks

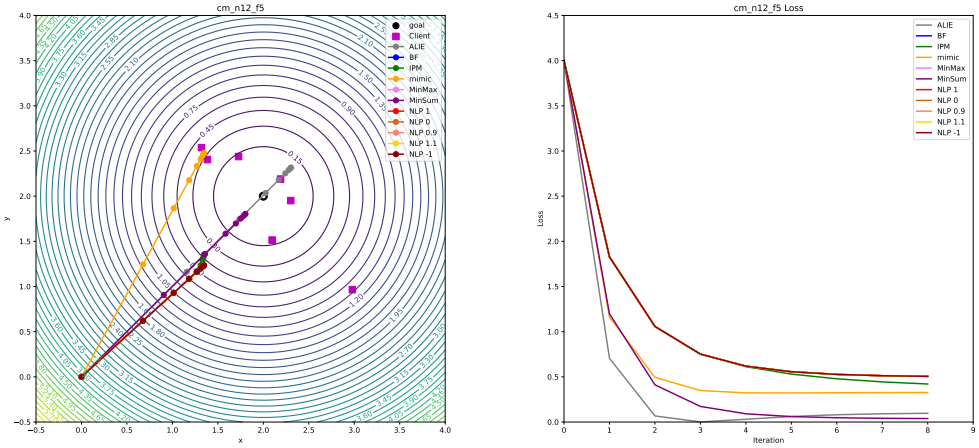
In this section, we delve into the exploratory analysis undertaken through our 2-Dimensional Mean Estimation Tasks. The experiment set-up commences at the origin, $(0, 0)$, as the starting point for the parameters (weights of two neurons in the server model). Honest workers are assigned sample

points from a normal distribution as their respective datasets, with the mean of these honest workers' samples located at $(2, 2)$. Concurrently, Byzantine workers are also randomly assigned sample points from the same normal distribution.

Moving forward, we implement a comprehensive nonlinear-programming-based attack (NLP attack). This involves the determination of malicious gradients, which are dispatched from all Byzantine workers to the parameter point across 8 iterations, so as to maximize the value of the objective function. For the purpose of this experiment, we experiment with five distinct objective functions, each represented with unique identifiers in our investigation.

"NLP 1" symbolizes the mean loss value of the parameter points across all iterations. "NLP 0" is representative of the final loss value of the parameter point. "NLP 0.9" and "NLP 1.1" respectively denote the weighted mean of loss values of parameter points across all iterations, where "0.9" and "1.1" serve as decay parameters, each iteration's weight is 0.9 or 1.1 times the weight of its preceding iteration. Lastly, "NLP -1" corresponds to the lowest loss value of the parameter point encountered throughout its trajectory.

We do experiments across a set of nonlinear programming solvers and select the one with best performance for each aggregation algorithm. And for the sake of a more manageable implementation, we operate under the assumption that all Byzantine workers send identical malicious gradients in a single iteration. This setting enables us to conduct an in-depth examination of the interaction between Byzantine workers and honest workers in such a distributed learning scenario.

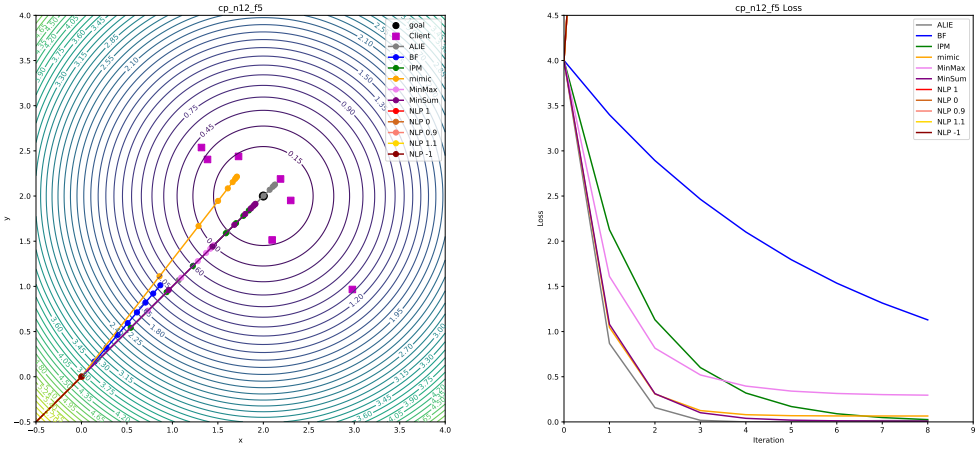


(a) Trajectory (b) Loss

Figure 1: Experiments on Mean Estimation with f = 5 Byzantine workers among n = 12. The robust aggregation algorithm is coordinate-wise median and the nonlinear programming solver is Powell [Powell (1964)]. Besides the 6 popular attacks, Byzantine workers execute 5 nonlinear-programming-based attacks. (a) Trajectories of the point under different attacks in the mean estimation process. (b) Losses (Euler distance to the mean point) of the point under different attacks.

NLP attacks are best. The NLP attacks emerge as the most effective, outperforming existing state-of-the-art attack methodologies across all robust aggregation algorithms. While the loss curves for NLP attacks do not consistently outperform other attacks on Krum throughout the entire course of training, they demonstrate superior performance in the latter half. This suggests that in Mean Estimation tasks, NLP attacks showcase the most potent attacking capability and can serve as a benchmark against a majority of robust aggregation algorithms.

Ample room for the improvement of attacks. NLP attacks show substantial superiority over other attacks on specific robust aggregation algorithms. It is observable from Figures 2, 13, and 14 that NLP attacks outshine other attacks on Centered-Clipping, Krum, and RFA. This leaves ample room for the improvement of attacks against these aggregation algorithms.



(a) Trajectory (b) Loss

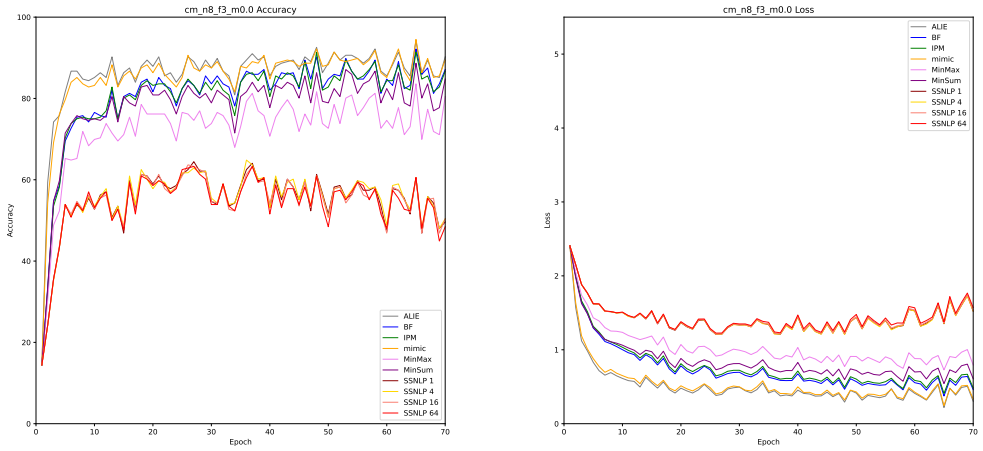
Figure 2: Experiments on Mean Estimation with f = 5 Byzantine workers among n = 12. The robust aggregation algorithm is centered-clipping and the nonlinear programming solver is SLSQP [Kraft (1988)]. Besides the 6 popular attacks, Byzantine workers execute 5 nonlinear-programming-based attacks. (a) Trajectories of the point under different attacks in the mean estimation process. (b) Losses (Euler distance to the mean point) of the point under different attacks.

Objective functions matter less. The selection of objective functions for NLP attacks does not significantly influence their attacking efficacy. Excluding Krum, the trajectories for NLP attacks with different objective functions largely coincide. In the case of Krum, NLP attacks using weighted average objective functions (including constant weights) perform optimally, while those using the minimum loss objective function lag behind. This could potentially be attributable to the performance of nonlinear programming and the particular characteristics of the dataset.

5.3 Experiments on MNIST Logistic Regression Tasks

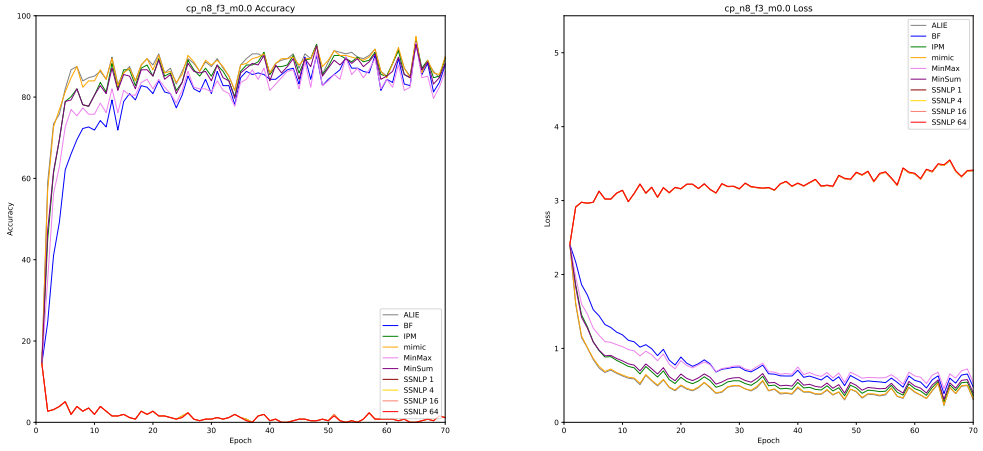
In our experiment pertaining to the MNIST Logistic Regression Tasks, due to hardware and performance constraints of the nonlinear programming solver, the full implementation of nonlinear programming was not feasible. The demand for maximization of the objective function necessitated the search for a set of gradients from all iterations transmitted by the Byzantine worker, resulting in a substantially high-dimensional search. To facilitate NLP attacks, we drew inspiration from the theoretical underpinnings of the IPM attack [Xie et al. (2020)] to significantly reduce the search dimension. At this point, the NLP solver was no longer required to seek all gradients. Instead, it sought a coefficient λ_t for each iteration t, and in this iteration, the gradients sent by the Byzantine workers were formulated as $(1-\lambda_t) * avgG_h^{(t)} + \lambda_t * (-avgG_h^{(t)})$. Here, $avgG_h^{(t)}$ stands for the average of the gradients sent by honest workers in the t-th iteration. This led to the search for a set of lambdas such that the objective function value was maximized over these T iterations.

However, searching a set of λs for the entire training process still posed a considerable challenge. Therefore, we employed the technique of multi-step lookahead in our attack design section 4. We divided the entire training process T into small segments of length l and sought a set of λs for each segment. This further decreased the dimension of the search, enabling the completion of the attack in a reasonable timeframe. We experimented with l values of $1, 4, 16, 64$, as represented by SSNLP 1, SSNLP 4, SSNLP 16, and SSNLP 64. "SSNLP" denotes "super simplified NLP attacks". We maintained the assumption that all Byzantine workers sent the same malicious gradient in the same iteration. Moreover, we utilized some of our findings from the Mean Estimation tasks, such as employing the optimal NLP solver for each robust aggregation algorithm from Mean Estimation tasks and the use of the average loss as the objective function.



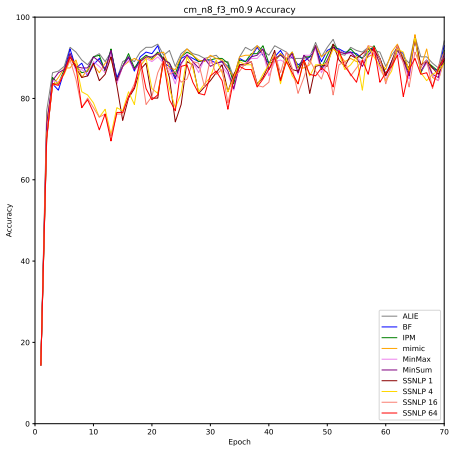
(a) Accuracy (b) Loss

Figure 3: Experiments on MNIST Logistic Regression with f = 3 Byzantine workers among n = 8, without momentum. The robust aggregation algorithm is coordinate-wise median and the nonlinear programming solver is Powell. Besides the 6 popular attacks, Byzantine workers execute 4 nonlinear-programming-based attacks.

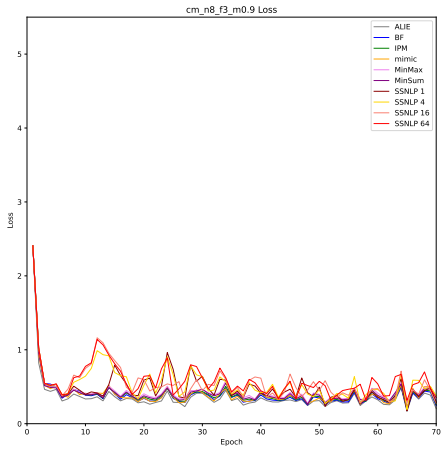


(a) Accuracy (b) Loss

Figure 4: Experiments on MNIST Logistic Regression with f = 3 Byzantine workers among n = 8, without momentum. The robust aggregation algorithm is centered-clipping and the nonlinear programming solver is SLSQP. Besides the 6 popular attacks, Byzantine workers execute 4 nonlinear-programming-based attacks.

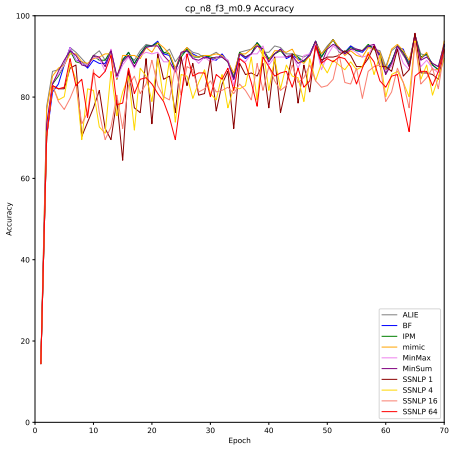


(a) Accuracy

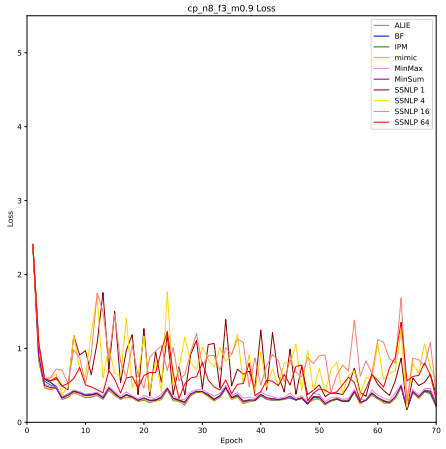


(b) Loss

Figure 5: Experiments on MNIST Logistic Regression with f = 3 Byzantine workers among n = 8, with local momentum 0.9. The robust aggregation algorithm is coordinate-wise median and the nonlinear programming solver is Powell. Besides the 6 popular attacks, Byzantine workers execute 4 nonlinear-programming-based attacks.



(a) Accuracy



(b) Loss

Figure 6: Experiments on MNIST Logistic Regression with f = 3 Byzantine workers among n = 8, with local momentum 0.9. The robust aggregation algorithm is centered-clipping and the nonlinear programming solver is SLSQP. Besides the 6 popular attacks, Byzantine workers execute 4 nonlinear-programming-based attacks.

NLP attacks perform best with no momentum in training. As evident from Figures 3 through 18, NLP attacks surpassed all existing state-of-the-art attack methods on most robust aggregation algorithms, suggesting their strongest attacking capacity in the MNIST Logistic Regression task, which can serve as a benchmark for most robust aggregation algorithms. Moreover, NLP attacks significantly outperformed other attacks on CM, Clipping, Krum, and TM. This partially overlaps with our 2-D Mean Estimation experiment results but is not completely identical. We also noticed that the loss curve of NLP attacks with a lookahead step larger than 1 on RFA was lower than other attacks, while the single-step NLP attack outperformed other attacks. This is contrary to our hypothesis and theory, as theoretically, single-step NLP attacks should not outperform multi-step NLP attacks. We suspect this is due to the insufficient performance of our customized NLP solver and the highly non-convex landscape of the search space under RFA, causing the NLP solver's solution process to easily fall into local optimal solutions.

NLP attacks still perform best with local momentum, but with a less pronounced advantage. From Figures 5 through 21, it can be seen that the loss curve of NLP attacks on robust aggregation algorithms was higher than other attacks for most of the time, but the advantage was evidently not as pronounced as when there was no momentum. This suggests that local momentum provides excellent Byzantine robustness for distributed machine learning in the context of data homomorphism, which is in line with the conclusions drawn in [Farhadkhani et al. (2022)].

With no momentum, NLP attacks improve with longer lookahead, but with minimal advantage. Except for the previously mentioned RFA, the longer the lookahead of NLP attacks, the higher the loss curve, but the advantage is not significant except on Krum. This could be due to our NLP solver's performance being unable to find a global optimal solution in a complex, highly non-convex landscape, limiting the effect of lookahead. Furthermore, the method we used to reduce dimensions may not be appropriate, further constraining the solution area and making the effect of lookahead less pronounced. This provides practical guidance for implementing NLP attacks: we can achieve similar results with shorter lookahead, which is more resource-efficient as the time complexity of NLP attacks is positively correlated with lookahead length.

The relationship between the attack effectiveness of NLP attacks and the length of lookahead is not apparent with local momentum. From the experimental results, we cannot discern a clear relationship between the attack effectiveness of NLP attacks and the length of lookahead. However, a general trend is that in the early stages of training, NLP attacks with different lookahead lengths have similar effects, but in the later stages, NLP attacks with longer lookahead have a slight advantage over those with shorter lookahead.

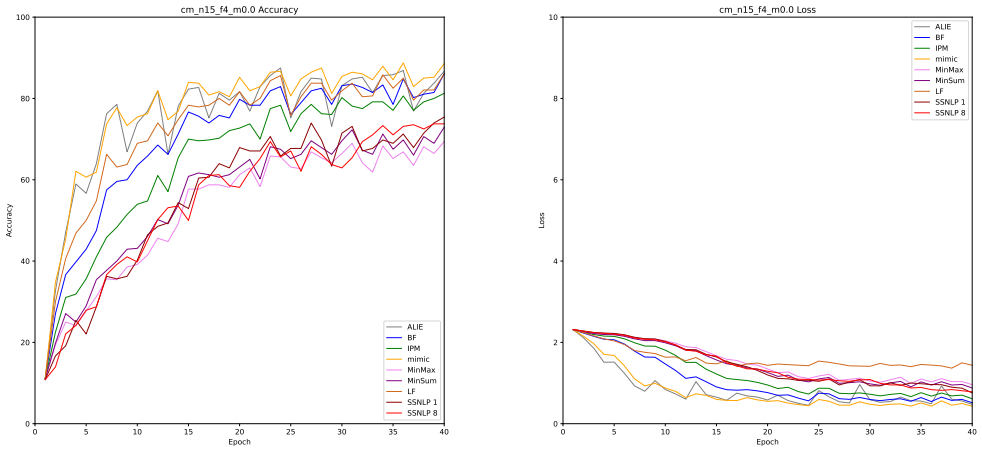
5.4 Experiments on MNIST CNNs Tasks

In this experiment, we conducted tests under two different setups: Setup 1, homogeneous datasets without applying momentum, and Setup 2, heterogeneous datasets with the application of D-SHB [Allouah et al. (2023)]. The rationale for adopting these setups are twofold: 1) From our Logistic Regression experiment, we observed strong robustness of the aggregation methods when homogeneous datasets were applied with local momentum, thus rendering further experiments under this setup less meaningful. 2) As indicated in [Allouah et al. (2023)], under heterogeneous settings without the application of D-SHB, most existing attacks already perform excellently, which also negates the need for further experiments under such circumstances.

Specifically, the data heterogeneity adopted in Setup 2 aligns with the Long-tailness non-iid as described in [Karimireddy et al. (2020)], where we utilized a long-tailness parameter of $\beta = 0.5$.

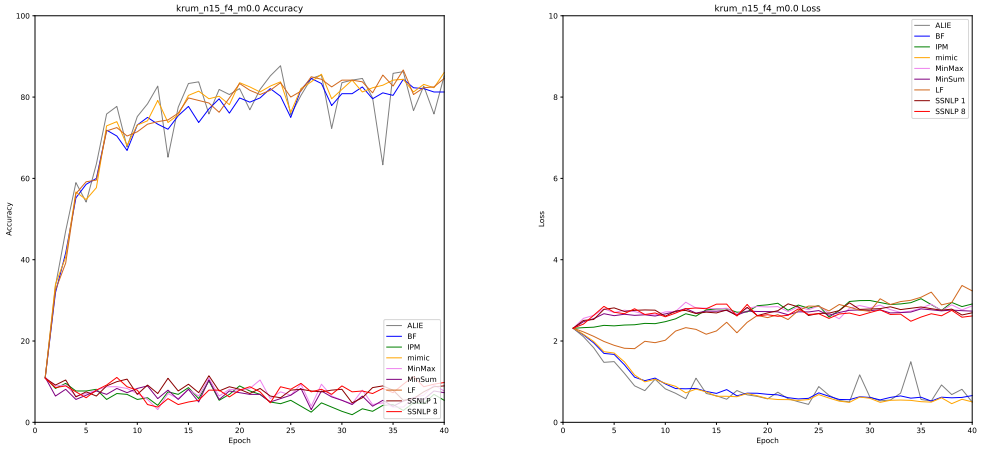
Furthermore, based on our experiences with the lookahead step length in Logistic Regression and considering the complexity and time-consuming nature of the CNNs network, we only experimented with step lengths of 1 and 8. In certain critical experiments, such as those involving heterogeneous datasets and the use of Mixing as a pre-aggregation method, we also tested a step length of 16.

Sometimes NLP attacks cause the model to output `nan`. So all `nan` losses were set to 1e+31. The training losses of LF attack in the figures below are invalid because of the code implementation of logging losses, but we can still compare LF to other attacks by the accuracy.



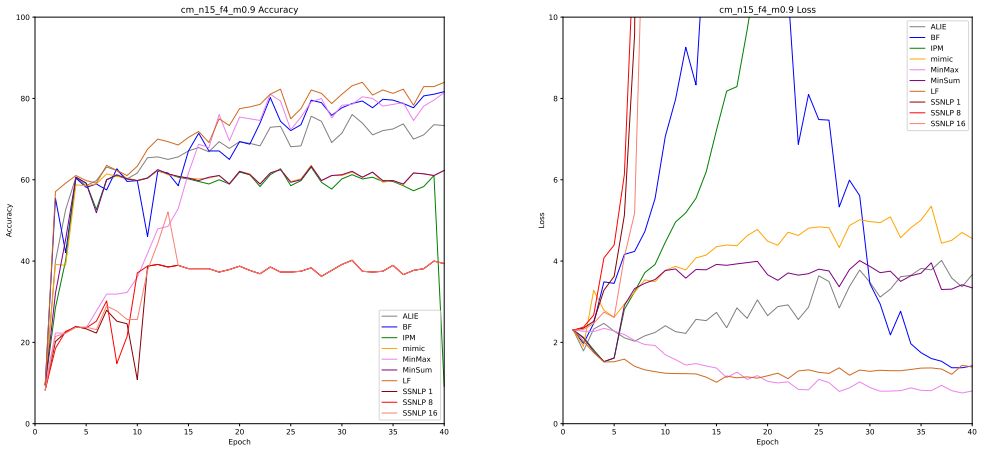
(a) Accuracy (b) Loss

Figure 7: Experiments on MNIST CNNs with f = 4 Byzantine workers among n = 15, with no momentum and in homogeneity setting. The robust aggregation algorithm is coordinate-wise median and the nonlinear programming solver is Powell. Besides the 7 popular attacks, Byzantine workers execute 2 nonlinear-programming-based attacks.



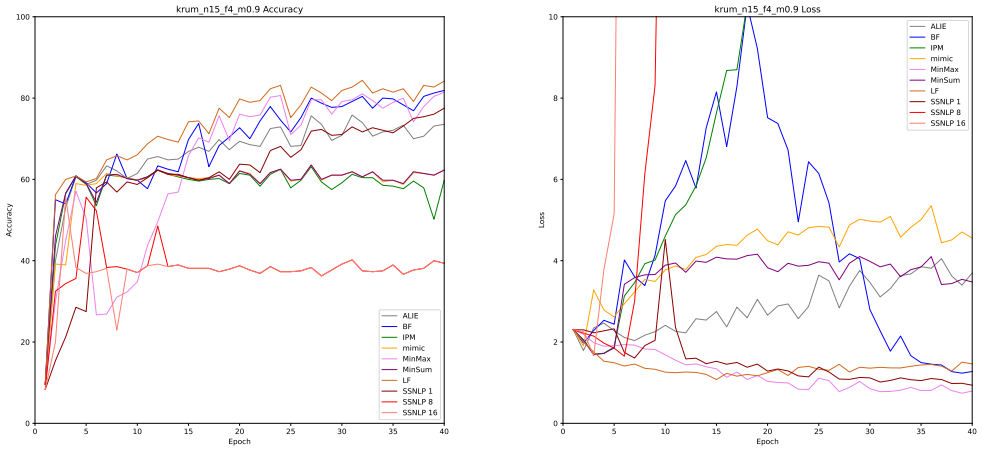
(a) Accuracy (b) Loss

Figure 8: Experiments on MNIST CNNs with f = 4 Byzantine workers among n = 15, with no momentum and in homogeneity setting. The robust aggregation algorithm is Krum and the nonlinear programming solver is Powell. Besides the 7 popular attacks, Byzantine workers execute 2 nonlinear-programming-based attacks.



(a) Accuracy (b) Loss

Figure 9: Experiments on MNIST CNNs with f = 4 Byzantine workers among n = 15, with D-SHB momentum 0.9 and in heterogeneity setting, with Mixing. The robust aggregation algorithm is coordinate-wise median and the nonlinear programming solver is Powell. Besides the 7 popular attacks, Byzantine workers execute 3 nonlinear-programming-based attacks.



(a) Accuracy (b) Loss

Figure 10: Experiments on MNIST CNNs with f = 4 Byzantine workers among n = 15, with D-SHB momentum 0.9 and in heterogeneity setting, with Mixing. The robust aggregation algorithm is Krum and the nonlinear programming solver is Powell. Besides the 7 popular attacks, Byzantine workers execute 3 nonlinear-programming-based attacks.

NLP attacks perform optimally with homogeneous data and no momentum. From Figures 7 and 8, we can see that the NLP attacks' impact on CM and Krum exceeds or matches all existing state-of-the-art attack measures on homogeneous MNIST CNNs datasets, demonstrating that NLP attacks still have the most potent attack capability as a benchmark for robust aggregation algorithms. However, we noticed that NLP attacks have lost their previous advantage over some state-of-the-art attack methods. We believe this is because the solution space of CNNs tasks with homogeneous data has become too complex for our NLP solvers, making it challenging to find the global optimum.

NLP attacks perform best with heterogeneous data and D-SHB training method. From Figures 22 and 23, it can be seen that the effect of NLP attacks on CM and Krum exceeds or equals all existing state-of-the-art attack methods, with a greater advantage than when the data is homogeneous and there is no momentum. This demonstrates that NLP attacks still maintain the strongest attack capability as a benchmark for robust aggregation algorithms on the MNIST CNNs heterogeneous data set task.

NLP attacks perform best with heterogeneous data and D-SHB and Mixing training methods, with a greater advantage. From Figures 9 and 10, it is clear that the effect of NLP attacks on CM and Krum far exceeds all existing state-of-the-art attack measures, with a much more significant advantage than the previous scenario. On the one hand, this shows that NLP attacks still maintain the strongest attack capability as a benchmark for robust aggregation algorithms on the MNIST CNNs heterogeneous data set task. On the other hand, it suggests that the Mixing method has deficiencies that current attack methods cannot exploit, whereas NLP attacks can breach Mixing. Notably, after NLP attacks have been conducted for some time, gradient explosion can occur, leading to 'nan' or 'inf' parameters in CNNs, preventing training. This suggests that the Mixing process can reduce the stability of model training. This is a concerning phenomenon, and it also implies that disrupting the stability of the model training process to interfere with the training process of distributed learning could become an attack primitive in Byzantine machine learning, warranting further research. In our Appendix A, we do some exploration on this attack primitive by designing and conducting experiments on a Stability Breaking Attack (STAB). Moreover, since Mixing is currently one of the best pre-aggregation methods, its performance under NLP attacks is not satisfactory, indicating that there is still much room for improvement in pre-aggregation methods under data heterogeneity.

6 Conclusion

Our series of experiments have demonstrated the exceptional strength of NLP attacks across different tasks and settings, surpassing or matching the best existing attack methods, but also revealed potential areas for improvement in various facets of robust aggregation methods. Despite the exceptional performance, the attack's effectiveness is still hampered by the limitations of the current NLP solvers and the way we reduce solution dimension, suggesting a future direction for enhancing the potency of such attacks, i.e., using reinforcement learning instead of nonlinear programming to solve the problem. Furthermore, our exploration has unveiled the potential of disrupting the stability of the model training process as a new attack primitive in Byzantine Machine Learning. This potential vulnerability, demonstrated by the occurrence of gradient explosion and unstable parameters in models subjected to NLP attacks, particularly under Mixing conditions, could offer a new avenue for sabotage in distributed learning systems. Ultimately, our findings underline the importance of continuous scrutiny and improvements in robust aggregation algorithms, particularly in heterogeneous data contexts, to ensure the integrity and resilience of distributed machine learning systems.

References

Allouah, Y., Farhadkhani, S., Guerraoui, R., Gupta, N., Pinot, R., and Stephan, J. (2023). Fixing by mixing: A recipe for optimal byzantine ml under heterogeneity. In *International Conference on Artificial Intelligence and Statistics*, pages 1232–1300. PMLR.

Barreno, M., Nelson, B., Sears, R., Joseph, A. D., and Tygar, J. D. (2006). Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25.

Baruch, G., Baruch, M., and Goldberg, Y. (2019). A little is enough: Circumventing defenses for distributed learning. *Advances in Neural Information Processing Systems*, 32.

Bertsekas, D. (2012). *Dynamic programming and optimal control: Volume I*, volume 1. Athena scientific.

Bertsekas, D. and Tsitsiklis, J. (2015). *Parallel and distributed computation: numerical methods*. Athena Scientific.

Bertsekas, D. P. (1997). Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334.

Blanchard, P., El Mhamdi, E. M., Guerraoui, R., and Stainer, J. (2017). Machine learning with adversaries: Byzantine tolerant gradient descent. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 119–129. Curran Associates, Inc.

Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., et al. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122.

Chen, Y., Su, L., and Xu, J. (2017). Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(2):1–25.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., et al. (2012). Large scale distributed deep networks. *Advances in neural information processing systems*, 25.

El-Mhamdi, E.-M., Guerraoui, R., Guirguis, A., Hoang, L. N., and Rouault, S. (2020). Genuinely distributed byzantine machine learning. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 355–364.

Farhadkhani, S., Guerraoui, R., Gupta, N., Pinot, R., and Stephan, J. (2022). Byzantine machine learning made easy by resilient averaging of momentums. In *International Conference on Machine Learning*, pages 6246–6283. PMLR.

Fletcher, R. and Powell, M. J. (1963). A rapidly convergent descent method for minimization. *The computer journal*, 6(2):163–168.

Ghadimi, S. and Lan, G. (2013). Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368.

Gouissem, A., Abualsaud, K., Yaacoub, E., Khattab, T., and Guizani, M. (2022). Federated learning stability under byzantine attacks. In *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 572–577. IEEE.

Guo, Q., Wu, D., Qi, Y., Qi, S., and Li, Q. (2022). Flmjr: Improving robustness of federated learning via model stability. In *Computer Security–ESORICS 2022: 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26–30, 2022, Proceedings, Part III*, pages 405–424. Springer.

Karimireddy, S. P., He, L., and Jaggi, M. (2020). Byzantine-robust learning on heterogeneous datasets via bucketing. In *International Conference on Learning Representations*.

Karimireddy, S. P., He, L., and Jaggi, M. (2021). Learning from history for byzantine robust optimization. In *International Conference on Machine Learning*, pages 5311–5319. PMLR.

Kraft, D. (1988). A software package for sequential quadratic programming. *Forschungsbericht-Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*.

Lamport, L., Shostak, R., and Pease, M. (1982). The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401.

Li, H., Sun, X., and Zheng, Z. (2022). Learning to attack federated learning: A model-based reinforcement learning attack framework. In *Advances in Neural Information Processing Systems*.

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

Powell, M. J. (1964). An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The computer journal*, 7(2):155–162.

Shejwalkar, V. and Houmansadr, A. (2021). Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning. In *NDSS*.

Xie, C., Koyejo, O., and Gupta, I. (2020). Fall of empires: Breaking byzantine-tolerant sgd by inner product manipulation. In *Uncertainty in Artificial Intelligence*, pages 261–270. PMLR.

Yin, D., Chen, Y., Kannan, R., and Bartlett, P. (2018). Byzantine-robust distributed learning: Towards optimal statistical rates. In *International Conference on Machine Learning*, pages 5650–5659. PMLR.

Zhu, C., Byrd, R. H., Lu, P., and Nocedal, J. (1997). Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on mathematical software (TOMS)*, 23(4):550–560.

Part I
Appendix

A Proof-of-Concept of Stability Breaking Attacks

Currently, there exist some explorations related to the stability of Byzantine Machine Learning. However, these studies either do not primarily focus on the stability during the training process [Guo et al. (2022)], or they fail to provide an in-depth analysis of the impact of Byzantine nodes on the robustness of the training process [Gouissem et al. (2022)].

In this appendix, we delve into a proof-of-concept illustration of Stability Breaking Attacks (STAB). Our objective is to demonstrate the practical feasibility and potential impact of such attacks on federated learning models, shedding light on a hitherto underexplored yet critical aspect of the model's vulnerability. By illuminating this novel threat vector, we aim to stimulate further exploration and encourage the development of robust countermeasures. The following discourse will outline our methodology, experimental setup, and findings.

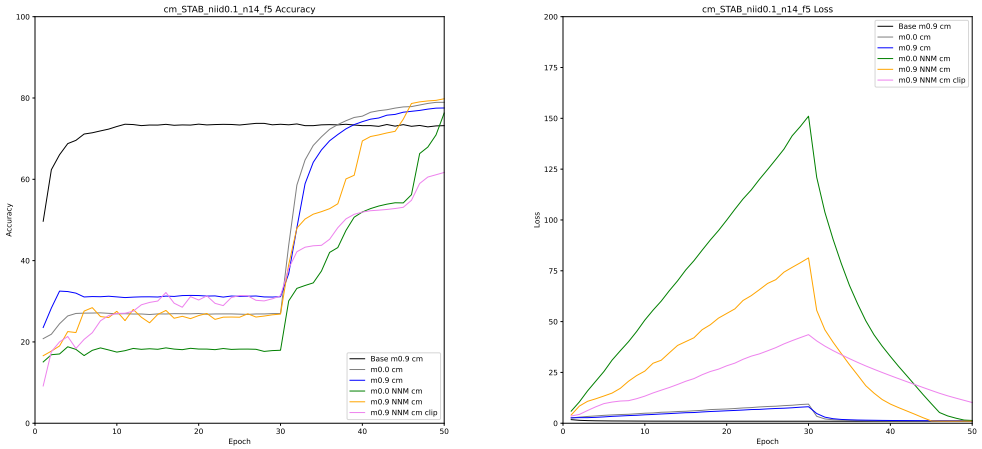
The Stability Breaking Attack (STAB) presents a variant of the NLP attack when the lookahead step length is set to 1. It is essentially founded on NLP solvers, yet with a redefined objective function aiming to maximize the gap between the largest and smallest values of the network output logits. In pursuit of concise yet generalized results, we conducted our experiments on the MNIST Logistic Regression task. Considering the MNIST Logistic Regression model solely consists of a fully-connected layer, it is relatively simpler than more complex networks such as Convolutional Neural Networks (CNNs). As a result, its inherent stability during the training phase is generally superior, rendering any stability-based attack results more persuasive.

We executed our experiments under extreme data heterogeneity conditions, where each honest node samples the MNIST dataset according to labels via Dirichlet sampling, with the alpha parameter set at 0.1. Contrary to the experimental strategy in the main text, we applied the STAB attack to various Byzantine defense combinations, investigating the efficacy of STAB under different defense configurations. Our experiments were carried out separately on defense combinations based on the CM and Krum methods.

We first proceed with an attack phase for 960 iterations, during which we implement the STAB Attacks. Following this attack phase, we then suspend the attack, allowing the model to train correctly for the next 640 iterations. This experimental procedure provides us with insights into the aftermath of STAB and the model's ability to recover and regain its trainability post-attack. The results are shown in Figure 11 and 12.

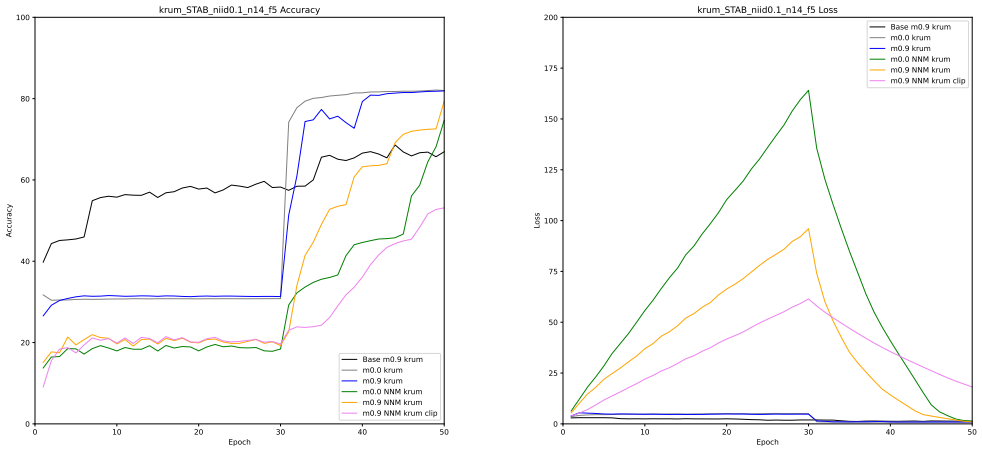
Our experiments revealed that STAB is capable of countering all defensive measures in our experimental setup. A closer examination of the details shows that the impact of STAB is more pronounced when the training method does not employ local momentum, and when Non-Negative Mixing (NNM) is utilized as opposed to when it's not. The latter finding is counterintuitive, as according to [Allouah et al. (2023)], NNM as a pre-aggregation method should ideally enhance the robustness of the training process. Yet, it performs the worst under stability-based attacks, indicating a significant weakening effect on the stability of the distributed training process under data heterogeneity. Additionally, we discovered that employing simple model stabilization methods such as gradient clipping can provide some defensive effect against the attack. However, the effect is marginal and might even affect the speed of model recovery once the attack is halted.

However, this is a very preliminary experiment, and the objective function set for the STAB attack in this paper is based on the intuition of generating large gradients, which lacks a robust theoretical foundation. Further research is required to delve deeper into this attack primitive, to uncover its essence and potential.



(a) Accuracy (b) Loss

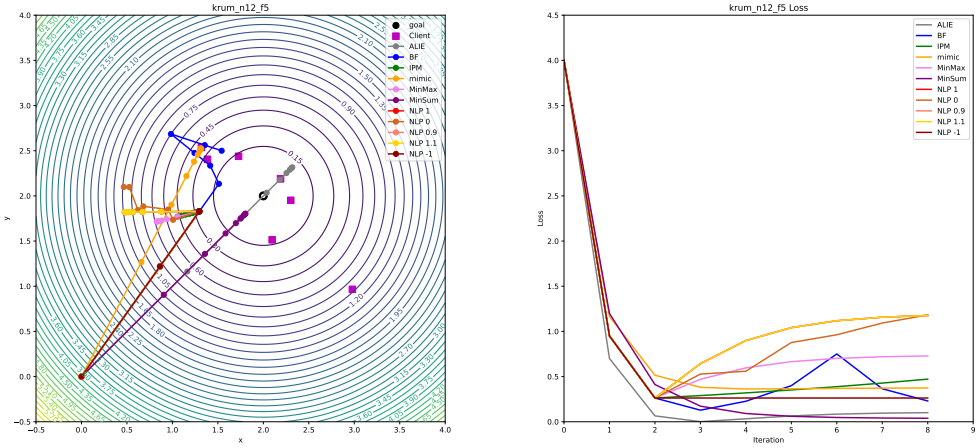
Figure 11: Experiments on MNIST Logistic Regression with f = 5 Byzantine workers among n = 14. The robust aggregation algorithm is CM and the nonlinear programming solver is Powell. The heterogeneity of datasets is $\alpha = 0.1$.



(a) Accuracy (b) Loss

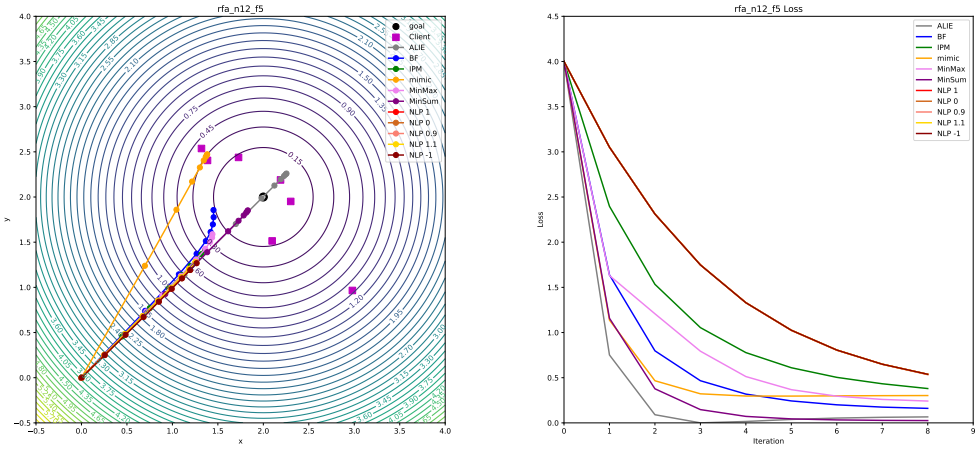
Figure 12: Experiments on MNIST Logistic Regression with f = 5 Byzantine workers among n = 14. The robust aggregation algorithm is Krum and the nonlinear programming solver is Powell. The heterogeneity of datasets is $\alpha = 0.1$.

B Figures



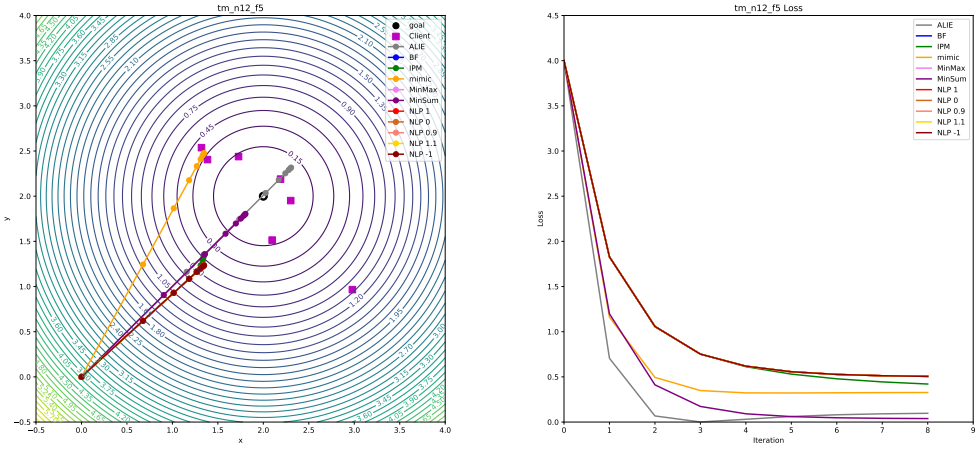
(a) Trajectory (b) Loss

Figure 13: Experiments on Mean Estimation with f = 5 Byzantine workers among n = 12. The robust aggregation algorithm is Krum and the nonlinear programming solver is Powell. Besides the 6 popular attacks, Byzantine workers execute 5 nonlinear-programming-based attacks. (a) Trajectories of the point under different attacks in the mean estimation process. (b) Losses (Euler distance to the mean point) of the point under different attacks.



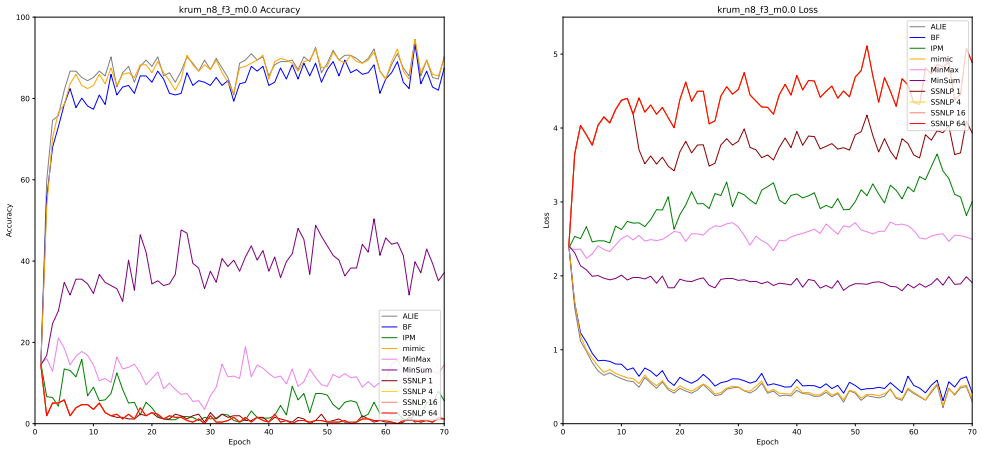
(a) Trajectory (b) Loss

Figure 14: Experiments on Mean Estimation with f = 5 Byzantine workers among n = 12. The robust aggregation algorithm is geometric median (RFA) and the nonlinear programming solver is L-BFGS-B [Zhu et al. (1997)]. Besides the 6 popular attacks, Byzantine workers execute 5 nonlinear-programming-based attacks. (a) Trajectories of the point under different attacks in the mean estimation process. (b) Losses (Euler distance to the mean point) of the point under different attacks.



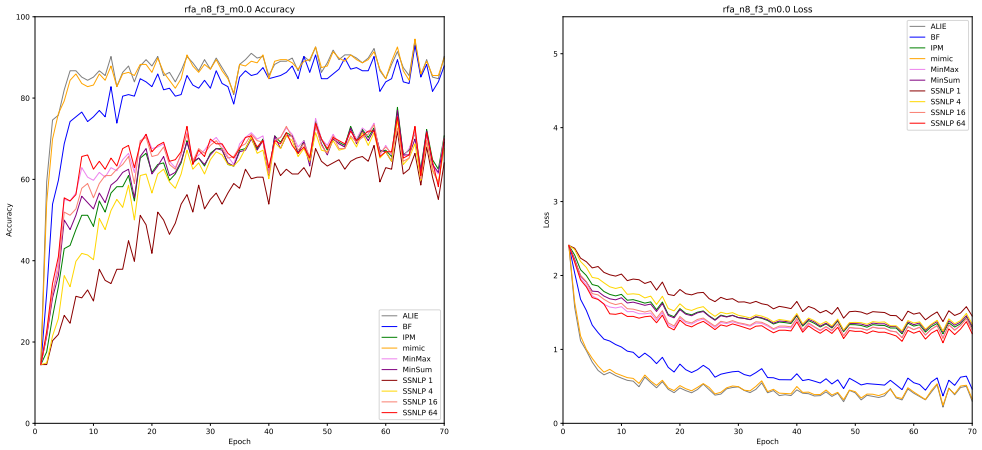
(a) Trajectory (b) Loss

Figure 15: Experiments on Mean Estimation with f = 5 Byzantine workers among n = 12. The robust aggregation algorithm is trimmed mean and the nonlinear programming solver is Powell. Besides the 6 popular attacks, Byzantine workers execute 5 nonlinear-programming-based attacks. (a) Trajectories of the point under different attacks in the mean estimation process. (b) Losses (Euler distance to the mean point) of the point under different attacks.



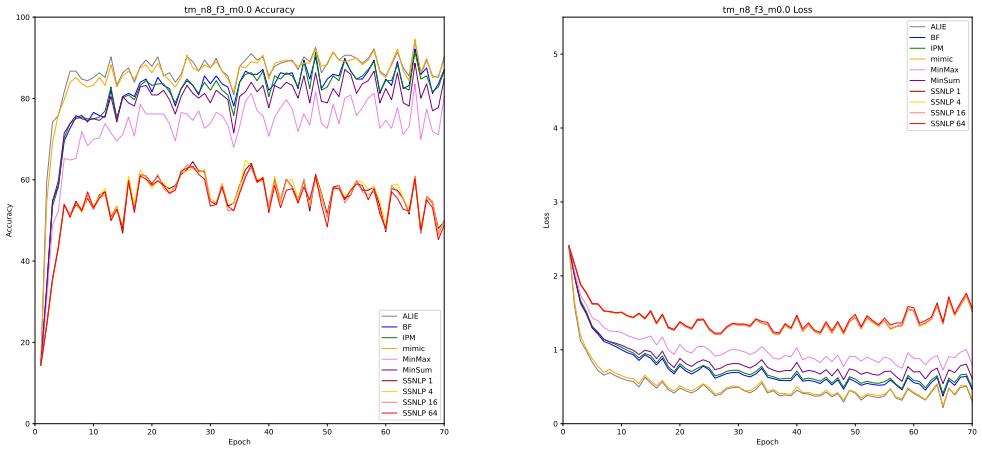
(a) Accuracy (b) Loss

Figure 16: Experiments on MNIST Logistic Regression with f = 3 Byzantine workers among n = 8, without momentum. The robust aggregation algorithm is Krum and the nonlinear programming solver is Powell. Besides the 6 popular attacks, Byzantine workers execute 4 nonlinear-programming-based attacks.



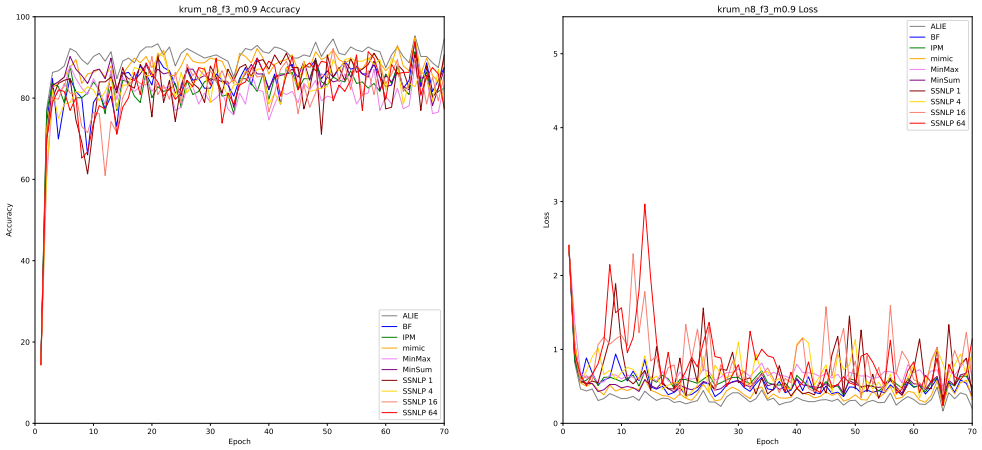
(a) Accuracy (b) Loss

Figure 17: Experiments on MNIST Logistic Regression with f = 3 Byzantine workers among n = 8, without momentum. The robust aggregation algorithm is geometric median (RFA) and the nonlinear programming solver is L-BFGS-B. Besides the 6 popular attacks, Byzantine workers execute 4 nonlinear-programming-based attacks.



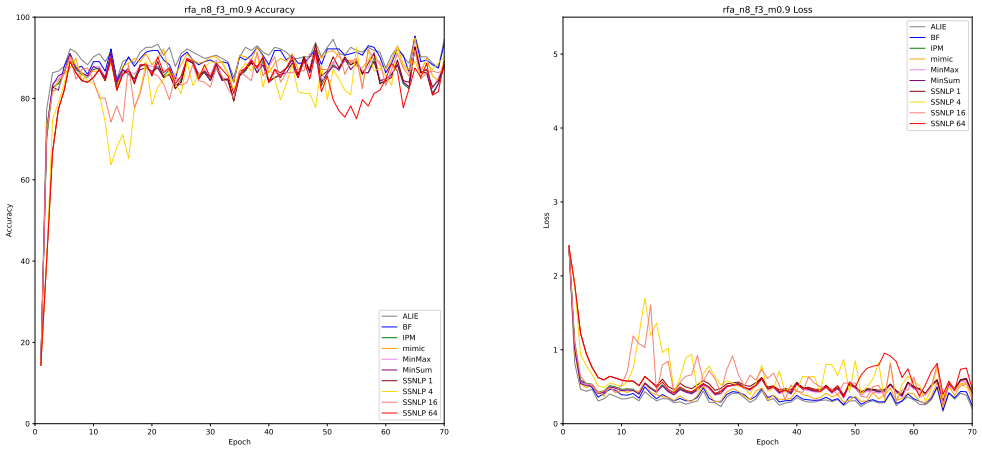
(a) Accuracy (b) Loss

Figure 18: Experiments on MNIST Logistic Regression with f = 3 Byzantine workers among n = 8, without momentum. The robust aggregation algorithm is trimmed mean and the nonlinear programming solver is Powell. Besides the 6 popular attacks, Byzantine workers execute 4 nonlinear-programming-based attacks.



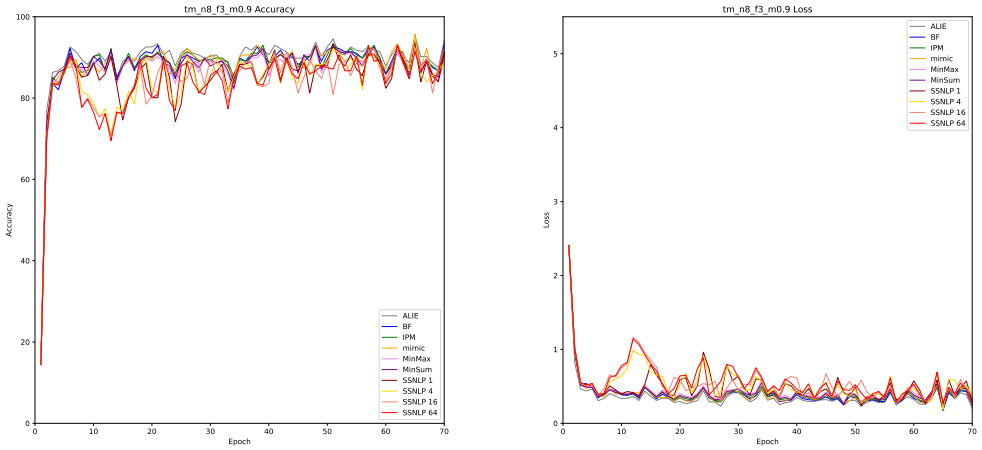
(a) Accuracy (b) Loss

Figure 19: Experiments on MNIST Logistic Regression with f = 3 Byzantine workers among n = 8, with local momentum 0.9. The robust aggregation algorithm is Krum and the nonlinear programming solver is Powell. Besides the 6 popular attacks, Byzantine workers execute 4 nonlinear-programming-based attacks.



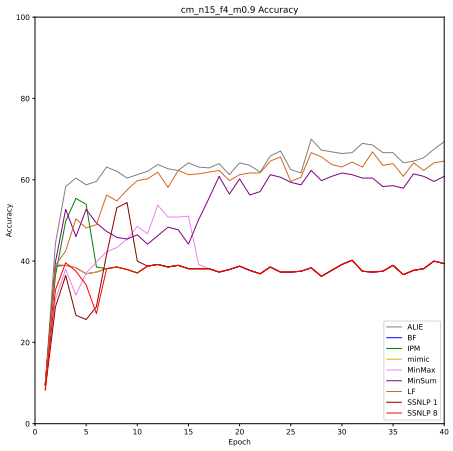
(a) Accuracy (b) Loss

Figure 20: Experiments on MNIST Logistic Regression with f = 3 Byzantine workers among n = 8, with local momentum 0.9. The robust aggregation algorithm is geometric median (RFA) and the nonlinear programming solver is L-BFGS-B. Besides the 6 popular attacks, Byzantine workers execute 4 nonlinear-programming-based attacks.

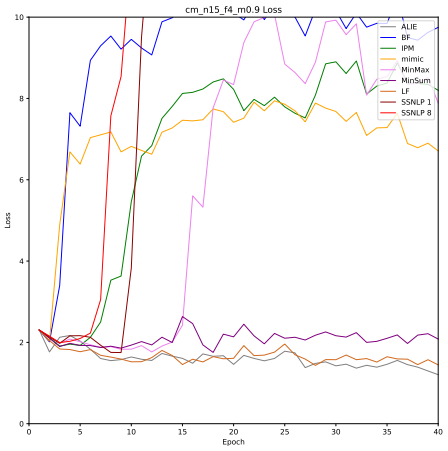


(a) Accuracy (b) Loss

Figure 21: Experiments on MNIST Logistic Regression with f = 3 Byzantine workers among n = 8, with local momentum 0.9. The robust aggregation algorithm is trimmed mean and the nonlinear programming solver is Powell. Besides the 6 popular attacks, Byzantine workers execute 4 nonlinear-programming-based attacks.

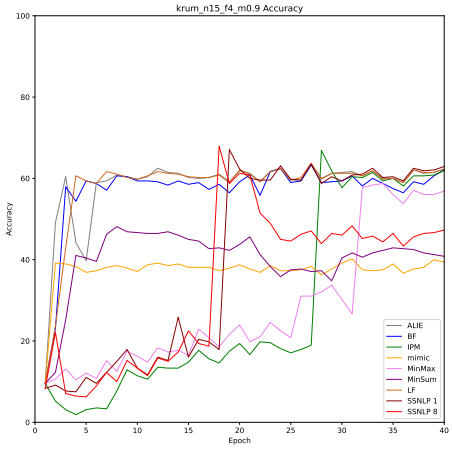


(a) Accuracy

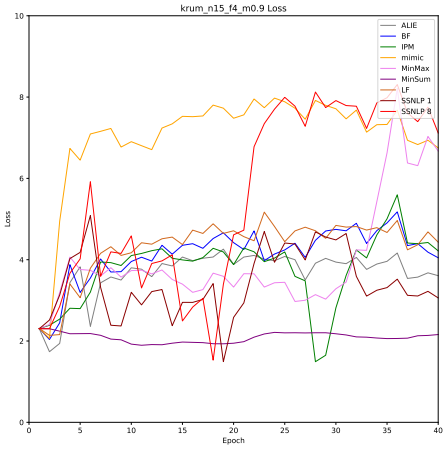


(b) Loss

Figure 22: Experiments on MNIST CNNs with f = 4 Byzantine workers among n = 15, with D-SHB momentum 0.9 in heterogeneity setting. The robust aggregation algorithm is coordinate-wise median and the nonlinear programming solver is Powell. Besides the 7 popular attacks, Byzantine workers execute 2 nonlinear-programming-based attacks.



(a) Accuracy



(b) Loss

Figure 23: Experiments on MNIST CNNs with f = 4 Byzantine workers among n = 15, with D-SHB momentum 0.9 in heterogeneity setting. The robust aggregation algorithm is Krum and the nonlinear programming solver is Powell. Besides the 7 popular attacks, Byzantine workers execute 2 nonlinear-programming-based attacks.