Trust-Based Aggregation for Federated Machine Learning (G15)

Lilesh Kurella
University of Illinois at Urbana-Champaign
lileshk2@illinois.edu

Shruthik Musukula
University of Illinois at Urbana-Champaign
srm14@illinois.edu

Vineet Chinthakindi
University of Illinois at Urbana-Champaign
vineetc2@illinois.edu

Abstract

Federated Learning (FL) has emerged as a pivotal paradigm for distributed processing and model training. This paper addresses the limitations of FL systems and Krum's algorithm, which lacks a mechanism for evaluating the trustworthiness of individual nodes. We introduce Trust-Krum, a novel trust metric that enriches Krum's scoring by incorporating a historical rolling average of node contributions and assessing the quality of these contributions. Trust-Krum significantly enhances model robustness, reducing malicious update selection by up to 38.56% and improving convergence rates by 2-3% compared to traditional methods. This approach effectively mitigates adversarial impacts, boosting distributed ML performance. The experiment codebase can be found here.

1 Introduction

Machine Learning (ML) has become the cornerstone of technological advancement. The growth in ML use cases has led to an influx in volume, velocity, and variety of data that traditional data processing software cannot handle[13]. Federated Learning (FL), is a strategy where a distributed network of nodes each process their own subset of data and then combine their gradient updates into an aggregation method for Stochastic Gradient Descent (SGD) of the global server model[9]. Distributed ML and FL introduce new challenges, particularly in terms of system security and integrity [12].

Krum's algorithm [1], is a solution that scores gradients based on their distance from the median gradient, effectively identifying and isolating malicious updates in distributed or decentralized machine learning environments. This strategy helps impose robustness and tolerate adversarial attacks from client nodes against the global model, characterizing it as a byzantine-tolerant system. While this method bolsters system resilience, it does not actively deter nodes from repetitively compromising the global model. Krum's algorithm does not incorporate a mechanism for evaluating the trustworthiness of individual nodes.

1.1 Proposed Solution: Trust-Krum

We introduce a scalable trust metric designed to enrich Krum's algorithm by integrating a trust-based scoring mechanism for client gradients. The trust metric draws upon the quality of historical contributions from each client node, by employing a rolling average calculated over the last hundred iterations for each client node. This average encapsulates the deviation between a node's Krum score and that of the node whose gradient was ultimately chosen for the model update.

Trust validates the reliability of nodes. We identified the rolling average as an effective metric for determining trust, due to its capacity to reference past SGD aggregations and to smooth out randomness, in scenarios lacking a consistent pattern of malicious attacks from specific nodes. Through this approach, we aim to provide a more nuanced assessment of each node's reliability and contribution quality, ensuring a more robust and trustworthy FL environment.

$$S_{\text{Trust}}(t) = \text{Avg}\left[S_{\text{Krum}}(t-1), \ldots, S_{\text{Krum}}(t-100)\right] \tag{1}$$

$$S_{\text{final}}(t) = S_{\text{Krum}}(t) + S_{\text{Trust}}(t) \tag{2}$$

Equations 1 & 2 presents a new scoring metric. The final score, S_{final}, is derived by merging the Krum score, S_{Krum}, which identifies and mitigates outlier contributions, with the Trust score, S_{Trust}, which assesses a client's historical reliability.

1.2 Research Questions

Our research questions are as follows:

1. How do model accuracies change based on SGD aggregation methodologies?
2. How much more often are bad gradients selected between Krum and Trust Krum?
3. What happens to the model when the percentage of malicious attacks varies per aggregation round?
4. What happens when Krum and Trust Krum both expect zero malicious nodes?
5. What are the overhead costs of Trust Krum (time)?
6. Does clustering help specify and segregate non-adversarial nodes?

2 Related Work

One of the earliest papers that looked into creating a reputation management stream for peer-to-peer networks introduced the *EigenTrust* algorithm [8]. The algorithm treats trust within a network as a quantifiable measure assigning each node in a network a trust value based on behavior and feedback from other nodes. This concept can be borrowed for use in FL since information must be shared securely and accurately while malicious nodes are present in the system. Trust can be used used a metric to help maintain a robust global model that can be distributed to the clients in the network.

2.1 Distributed SGD Network Topologies

When dealing with distributed SGD architectures, systems commonly use synchronous approaches. [3]. In this work, we utilize a synchronous server-client model to maintain and evaluate models and the FL environment.

2.2 Reputation as an Incentive Mechanism

Current approaches to FL lack a reliable trustworthy incentive mechanism [14]. A proposed secure and trustworthy blockchain framework (*SRB-FL*) is designed to address this [11]. *SRB-FL* focuses on improving the reliability of data provided by users in a system designed for FL workloads through an incentive mechanism known as subjective multi-weight logic to bolster reliability.

2.3 Byzantine-Tolerant FL

In Google's FL system implementation, *FedAvg* [10], a server-side aggregation algorithm was implemented to determine global model updates from client-calculated gradients by averaging them and applying those averaged gradients to the global model.

To ensure Byzantine tolerance a novel aggregation algorithm called *Krum* was created to better handle arbitrary byzantine failures in a system [1]. Given a set of vectors, *Krum* (Equation 3) computes pairwise distances between all vectors, and aggregates the sum of the distances to create a score. The lowest score is chosen to represent the best gradient. Table 1 compares major characteristics of *FedAvg*, *Krum*, and our proposed strategy *Trust-Krum*.

$$\left.\begin{aligned} \text{Krum}(\{\widetilde{v}_i : i \in [n]\}) = \widetilde{v}_k, \\ k = \arg\min_{i \in [n]} \sum_{i \to j} \|\widetilde{v}_i - \widetilde{v}_j\|^2 \end{aligned}\right\} \tag{3}$$

An existing approach to combat malicious behavior by incorporating the idea of bootstrapping trust [5], *FLTrust* runs the global server model on a small portion of the dataset to get initial metrics. These early results are then used to compare against client updates to evaluate the effectiveness of the data. This is an encouraging approach but does not have an individual impact on each stochastic gradient step.

The primary objective of FL approaches is to train a global model that protects client data privacy. *FedBayes* addresses the vulnerability of FL approaches to attacks by evaluating client-provided weights against the global model's expected distribution, using mean and standard deviation. Weights with lower probabilities, determined through a cumulative distribution function, are less trusted for

Strategy	*FedAvg*	*Zeno*	*Krum*	*Trust-Krum*
Goal	Average aggregation	Byzantine fault tolerance with strong convergence properties	Ensuring model reliability	Byzantine fault tolerance
Key Feature	Simplicity and efficiency	Reliability scoring system to evaluate client updates	Robustness to malicious updates	Identifying trusted client updates
Aggregation Method	Weighted average based on local dataset sizes	Reliability scores determine update selection, prone to error in dynamic environments	Selects single update or subset closest to others	Selects closest update from a trusted client
Fault Tolerance	Low	High	High	High
Data Heterogeneity Handling	Direct averaging can be suboptimal for non-IID data	Attempts to address non-IID data, but not effective	Does not directly address data heterogeneity	Historical averaging directly handles data heterogeneity
Scalability	High, due to simplicity	Moderate, scalability impacted by the complexity of dynamic scoring	Moderate, computational overhead for distance calculations	Moderate, computational overhead from historical averaging

Table 1. Comparison of Federated Learning Approaches

updates. This approach is enhanced by considering the historical contributions of clients to assess trust [15].

A novel aggregation rule named *Zeno* is introduced for FL [4], which handles Byzantine clients while ensuring strong convergence properties. Zeno operates by assigning a reliability score to each participating node. At the central server, the average gradient is calculated from the updates of nodes with the highest scores. The server then compares this average gradient with the actual gradient values to determine the likelihood of an update being malicious. The continued idea of keeping a scoring metric in *Trust-Krum* is influenced by implementations like these.

2.4 Identifying Malicious Clients

PeerReview [7] effectively detects and isolates malicious nodes in various networked systems by maintaining secure logs, ensuring the protection of benign nodes from false accusations of malice. FoolsGold [6] offers a complementary approach to security in Federated Learning (FL) environments by mitigating Sybil attacks, where a single adversary masquerades as many entities. It operates by analyzing the similarity of client updates, down-weighting similar contributions to safeguard the learning process without a central trust authority, assuming genuine participants provide sufficiently diverse updates.

3 Experimental Setup

In this section, we describe the environment configured to evaluate the performance of the proposed algorithm. The approach used in our baseline experimentation involves the *PyTorch* framework to test the effectiveness of baseline strategies against our own. This approach focuses on collecting historical averages of a recent subset of model scores.

We anticipate that the addition of historical averages of model gradients to the Krum algorithm will enhance its effectiveness in mitigating the impact of Byzantine faults in federated learning environments. The original Krum algorithm selects, among the submitted local updates, the one that

is the closest to the average of all updates, effectively ignoring those that deviate from the norm, which could be indicative of Byzantine behavior. However, this approach does not directly consider the temporal consistency or evolution of updates over time.

Integrating historical averages of model gradients can provide several benefits, supported by theoretical insights and empirical evidence from related work in optimization and distributed learning: (1) Smoothing out noise in gradient updates, (2) Stabilize learning process leading to early convergence

3.1 Datasets and Implementation

We benchmark FL SGD with the MNIST dataset [2]. MNIST is an extensively used benchmark in machine learning research that holds images of numbers to be classified between 0-9. The simplicity of this dataset is helpful when trying to see the impact of controlled manipulations during experimentation compared to the baseline.

Model Architecture: We employ a simplified neural network architecture designed to effectively interpret images from the MNIST dataset. This approach minimizes the complexity associated with additional dynamic layers found in stochastic gradient descent enhancements. As shown in the Neural Net Layers Table 2, the input layer formats 28-pixel by 28-pixel images into a 784-dimensional vector. The first fully connected layer converts the image tensor into 128 dimensions, the second to 64, and the third into a 10-dimensional vector to represent scores for the 10 classes in the classification dataset. Then the softmax function converts those scores into class probabilities.

Layer (Type)	Input Size	Output Size
Input Layer	28x28 (784)	-
fc1	784	128
fc2	128	64
fc3	64	10

Table 2. Neural Net Layers

Modeled Adversarial Attacks: To model attacks and replicate entropy throughout our system, we defined and tried three types of adversarial attacks. We experiment with these attacks leveraging both deterministic conditions and randomness. However, after further experimentation, it was concluded that only the Gradient-Flip attack would be utilized as it induced the most model corruption.

1. *Gradient-Flip Attack:* This attack involves flipping the sign in front of the gradient tensors, aiming to degrade the performance of the model directly by reversing the direction of a client model update.
2. *Gaussian Noise Attack:* In this attack, Gaussian noise is added to the gradients, intending to confuse the model by distorting the original signal without significantly altering its appearance.
3. *Combined Gaussian-Flip Attack:* This approach merges the techniques of adding Gaussian noise and Sign-Flipping per gradient update, leveraging the combined effects to potentially bypass defenses that might be robust against either attack in isolation.

Federated Learning Architecture: In our federated learning architecture, a central server node (VM) coordinates with a network of client nodes, each running on separate VMs. Each client node performs local computations, updating its model with gradients from batch iterations. These updated models are then sent to the server node, which aggregates the updates. Using the Krum algorithm, the server computes Krum scores for each model based on the gradients, contributing to a history-based metric. These scores are normalized with the bounds: [0,1], relative to one another, with the history of normalized scores tracked as a rolling average. The server selects the model with the optimal gradient according to this new trust score and disseminates it back to all client nodes. Consequently, each client node adopts this selected gradient, synchronizing their local models with the globally chosen update.

As detailed in Algorithm 1, for each batch processed by the client node, a backward pass is executed, generating gradients that represent changes in the weights. These gradients are then submitted to the server which will utilize these gradients to

Algorithm 1 Client Training Procedure with FL per Epoch

```
for (data, target) in ClientDataLoader do
    output = worker_model(data)
    loss = criterion(output, target)
    loss.backward()
    Send Client Model Parameters to Server
    Sync Client Model with Global Parameters
end for
```

create a new set of model weights which are then sent back to the client to overwrite their current model with.

Algorithm 2 outlines the decision-making process executed on the server side to distribute agreed-upon model parameters to the clients. In our approach, we first extract the gradients from each client and then subject them to specific adversarial attacks as part of our experiment. Subsequently, we compute the Krum scores for the client models. These scores are pivotal in identifying the gradient vector that aligns most closely with the majority, thus ensuring the integrity of the model updates. To facilitate comparison across different magnitudes—which is crucial as gradients tend to diminish over successive rounds—we normalize the Krum scores to a range between 0 and 1. This normalization is achieved by assessing the scores against the minimum and maximum values within the current round. After sufficient rounds have elapsed, we employ a historical analysis to compute the rolling average of the scores from the most recent roll_len rounds. This average is then adjusted by a predefined trust index, a hyperparameter to scale the value of trust. By multiplying the rolling average by the trust index and adding this product to the current Krum score, we derive each client's trust score. This trust score is computed individually for each client. Accordingly, the server updates its global model using the gradients from the client with the lowest trust score, thus favoring the most reliable updates. Finally, after updating the global model, the new weights are disseminated back to each client node. This process replaces their existing local weights with the updated global weights, thereby synchronizing the model across all clients and advancing the collective learning process.

Algorithm 2 Server Side Procedure with FL

```
hist_dict ← {cid : [] for cid in CLIENTS}
Input: global_model, client_models
Output: New Global Model Parameters
function TrustSGD:
    Extract gradients from each worker model
    APPLY Adversarial Attack
    krum_scores = Krum(Client_Grads)
    trust_scores = []

    min_val = min(krum_scores)
    max_val = max(krum_scores)

    for (score, cid) in krum_scores do
        if max_val = min_val then
            norm_score = 0
        else
            norm_score = (score−min_val) / (max_val−min_val)
        end if

        t_score = norm_score
        h_list = hist_dict[cid]

        if len(h_dict) ≥ roll_len then
            start = len(h_list)-roll_len
            r_avg = hist_list[start:]
            t_score += (t_index * r_avg)
        end if

        trust_scores.append(t_score)
    end for
    Select gradient with minimum trust score
    APPEND gradients to global model
    return global_model.parameters()
```

4 Evaluation

To evaluate our trust algorithm's efficiency for SGD, we compared it against FedAvg, Zeno, and Krum in an FL setting under Gradient-Flip Adversarial Attacks. Section 3.1 discusses three key adversarial attacks, but our focus is on Gradient-Flip

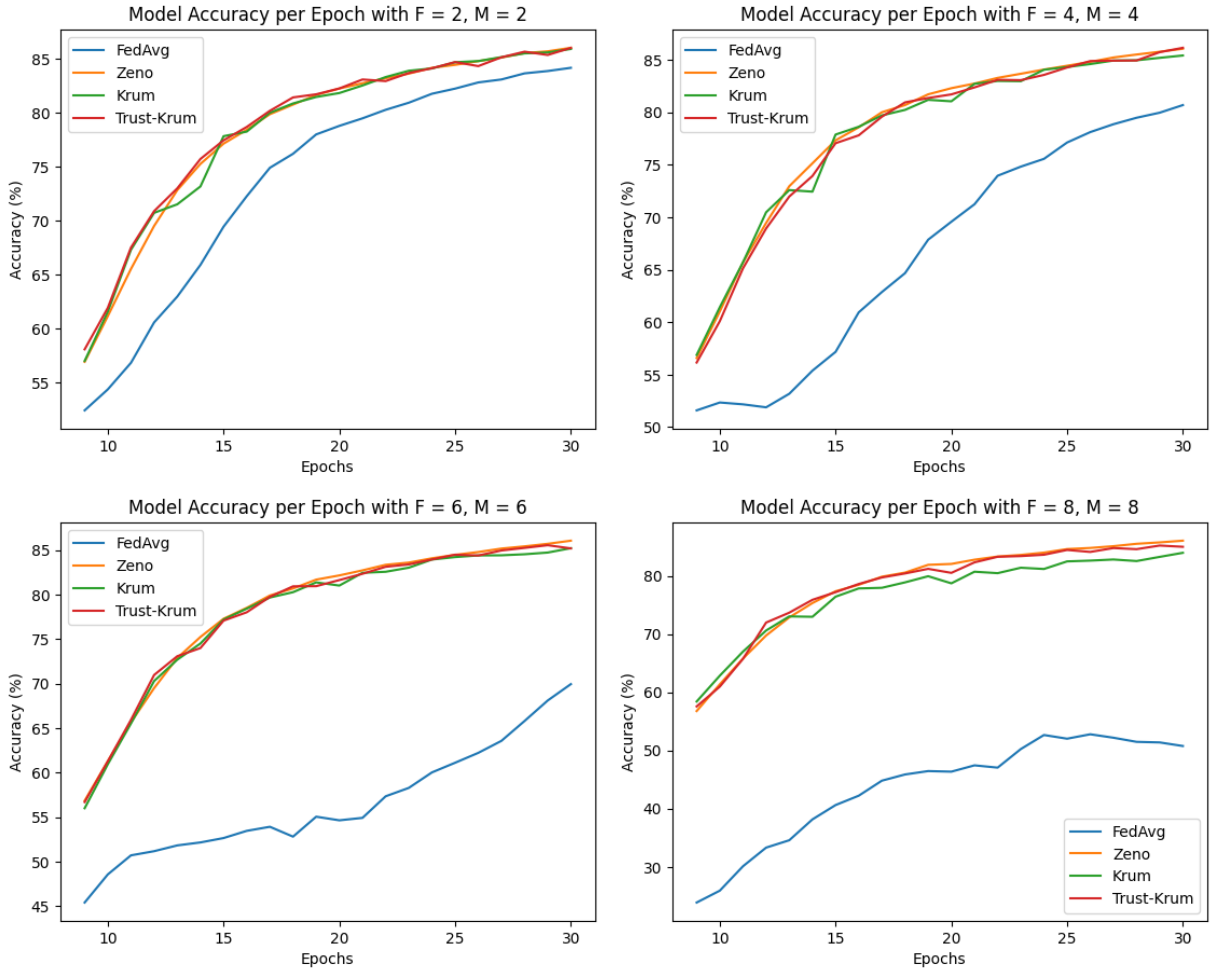

Figure 1. Convergence on i.i.d. training data, without failures. $F = M = \{2, 4, 6, 8\}$.

Attacks due to their impact on global model performance. We consistently managed hyperparameters to facilitate accurate comparisons and resilience assessment against these attacks in a controlled environment. Our experiments aim to highlight our approach's strengths and weaknesses relative to established frameworks, offering insights into its performance and potential areas for improvement.

4.1 System Overview

In our experimental setup, we configured the system with a batch size of 100, a learning rate, $\alpha = 0.01$, and ran it for 30 epochs. We included 20 clients in total, with a subset being malicious, varying in number from $\{2, 4, 6, 8\}$, across a historical window of 100 iterations, referred to as Roll-Len. Experiments were run on Linux machines with 2 CPU cores and 4 GB RAM each.

# of F, M	Trust-Krum	Krum
2	0	0
4	0	3
6	0	4
8	1	9

Table 3. Number of Byzantine Nodes Selected by Trust-Krum and Krum with consistently malicious nodes out of 20 nodes over 820 Rounds. F represents number of expected malicious nodes, while M is the true number of malicious nodes.

4.2 Baseline Results

The benefit of these gradient aggregation algorithms is not only their judgment but how quickly

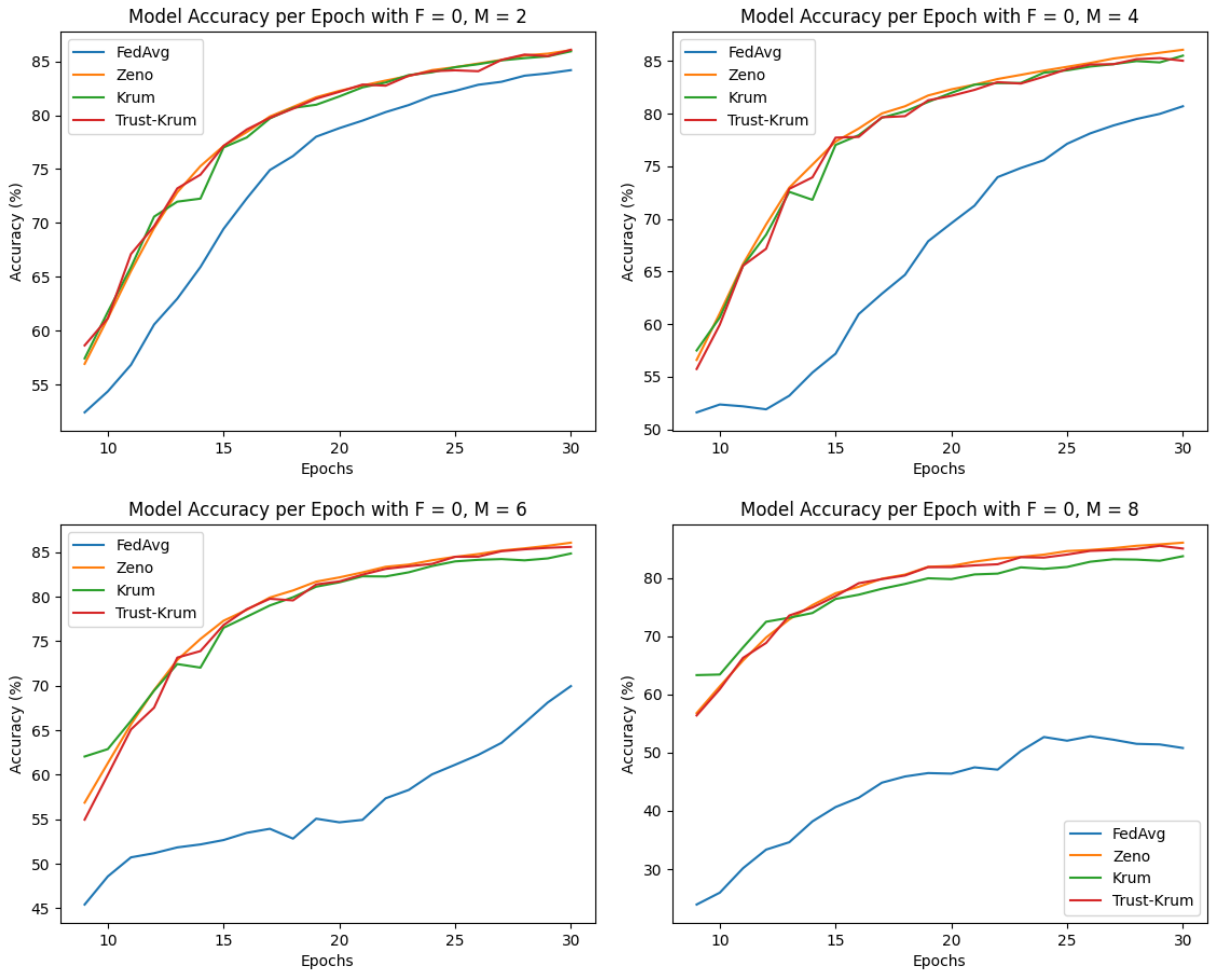

Figure 2. Convergence on i.i.d. training data, without failures. $F = 0, M = \{2, 4, 6, 8\}$.

they can help the model converge to a certain accuracy level. It can be observed in Figure 1 that all models converge to the same accuracy in a similar manner, with Zeno, Krum, and Trust-Krum exhibiting similar behaviors. In the initial experimentation above, we cover the results of Sign-Flip Attacks on a cluster containing 20 worker nodes.

As expected, both Zeno, Krum, and Trust-Krum all significantly outperform FedAvg when both model accuracy and recorded loss are used as benchmarks. By factoring in the trustworthiness of node contributions based on past performance, Trust-Krum contributes to global model stability and facilitates faster convergence. However, convergence and accuracy rates seem to be quite consistent and similar between Zeno and Trust-Krum. Observing the rates of byzantine gradient selection from Table 3, we see a significant difference with Trust-Krum and Krum. Trust-Krum's integration of historical rolling averages allows for a nuanced assessment of node reliability over time, significantly enhancing its robustness. This mechanism effectively reduces the likelihood of accepting malicious gradients, particularly noticeable as the F/M ratio increases. Trust-Krum consistently identified fewer Byzantine nodes compared to Krum, especially at higher ratios, demonstrating its ability to effectively differentiate between genuine and malicious updates.

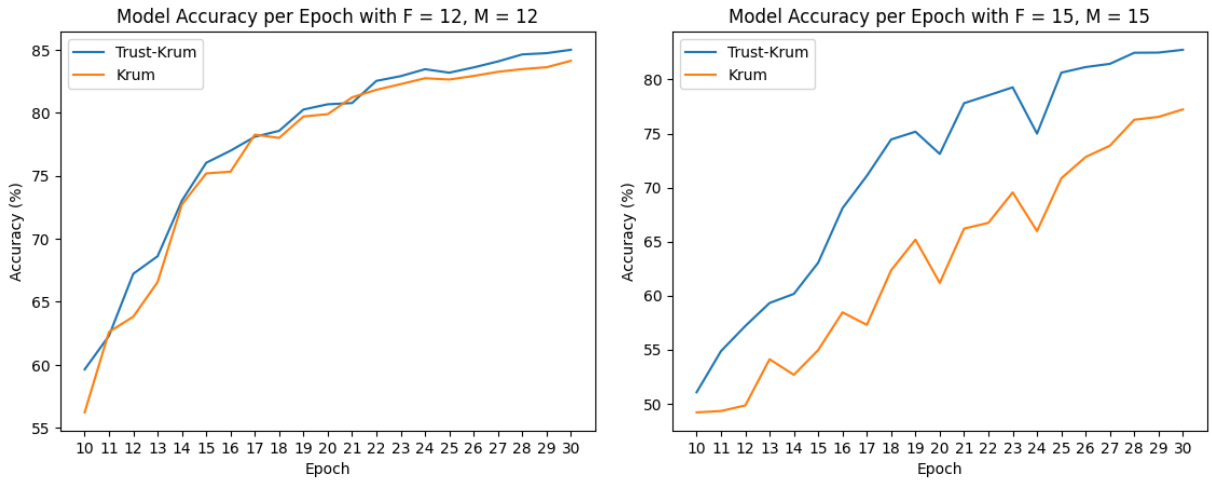

Figure 3. Model Performance with Selected Nodes having 50% chance of behaving malicious. M = {12,15}.

4.3 Underestimated Byzantine Nodes

Figure 2 illustrates the impact of Trust-Krum and other algorithms under conditions where the number of Byzantine nodes was underestimated, providing critical insights into how each algorithm performs when the system's assumptions about adversarial presence are incorrect.

In scenarios where the number of malicious nodes exceeds expectations, Trust-Krum demonstrates notable resilience. The plot in Figure 2 shows that while all models experienced a decrease in performance due to an increased number of adversarial actions, Trust-Krum maintained higher accuracy levels compared to Krum and FedAvg. This resilience is likely due to Trust-Krum's ability to leverage historical data, allowing it to better identify and mitigate the impact of nodes that consistently exhibit malicious behavior, even when their number is underrepresented in the system's initial assumptions. The data also shows that Zeno and Trust-Krum exhibit similar robustness under underestimated Byzantine conditions, with both algorithms performing better than FedAvg. However, Trust-Krum has an approximately 1-2% performance improvement over Krum due to its specific focus on historical performance and trust scoring, which seems to offer an additional layer of protection against fluctuations in node behavior that are not immediately apparent to other algorithms.

4.4 Varying Byzantine Nodes per Aggregation

Another experiment involved devising a situation where the expected number of malicious/Byzantine nodes stay the same in the system but the overall nodes that may act byzantine in a specific aggregation step vary.

In this scenario, the number of malicious nodes and expected number of malicious nodes were both set at [12,15,18] respectively where in each iteration, each of those number of nodes had a 50% chance of supplying malicious flipped gradient. Figure 3 demonstrated the graphical results of the scenario with number of potential malicious nodes being 12 and 15. In the F and M = 12 case there is a moderate benefit of Trust-Krum over regular Krum by around 2%. However, the F and M = 15 case shows a big improvement on the order of 5-7 % improvement in model accuracy as well as faster convergence. This shows that incorporating trust allows for the aggregation algorithm to also weight model contributions in previous rounds. Even though a node marked as potentially malicious may not be invoked in a specific round, aggregating previous scores from previous rounds will allow the algorithm to differentiate better between those that have had low Trust-Krum scores in the past compared to Krum which computes an individual score at each aggregation step. This can be further referenced in Figure 4 where the

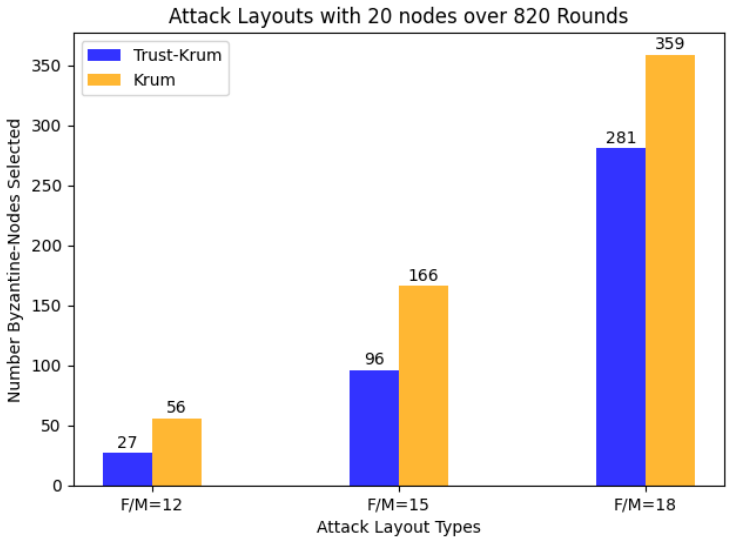

Figure 4. # of Byzantine Nodes Selected with varying # of Malicious Nodes. M = {12, 15, 18}.

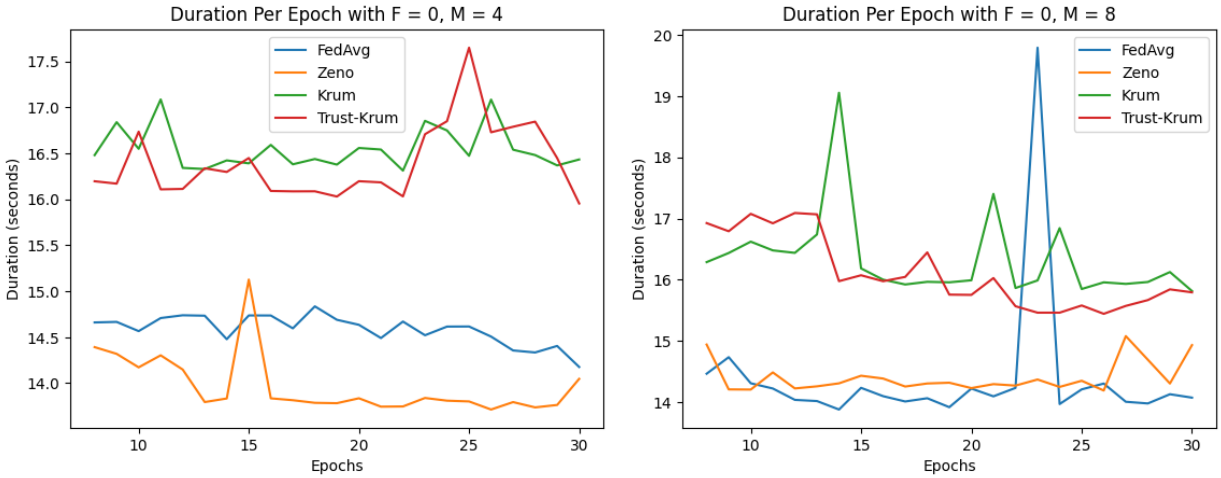

Figure 5. Duration per Epoch across all global models. M = {4, 8}.

number of actual byzantine nodes selected during all rounds were. As shown, in the case where F and M = [12,15,18], Trust-Krum reduces choosing the number of byzantine-gradients by around 50% in the F and M = 12/15 case and a by a factor of around 25% in the F and M = 18 case.

4.5 Model Overhead Analysis

Due to the additional overhead in maintaining and calculating historical data for Trust-Krum, we find it important to benchmark the time overhead against the original Krum strategy. In our experiment, we consider the scenarios where f is set to 0 and the number of malicious workers in our system of 20 nodes is equal to 4 and 8. Figure 5 in the report illustrates the duration per epoch across different global models. The results indicate that Trust-Krum exhibits a slightly higher time overhead per epoch compared to FedAvg, Zeno, and Krum. However, the increase is generally marginal, suggesting that the additional computational load does not drastically affect the overall efficiency of the system. The observed spikes in duration are attributed to variability in machine performance and

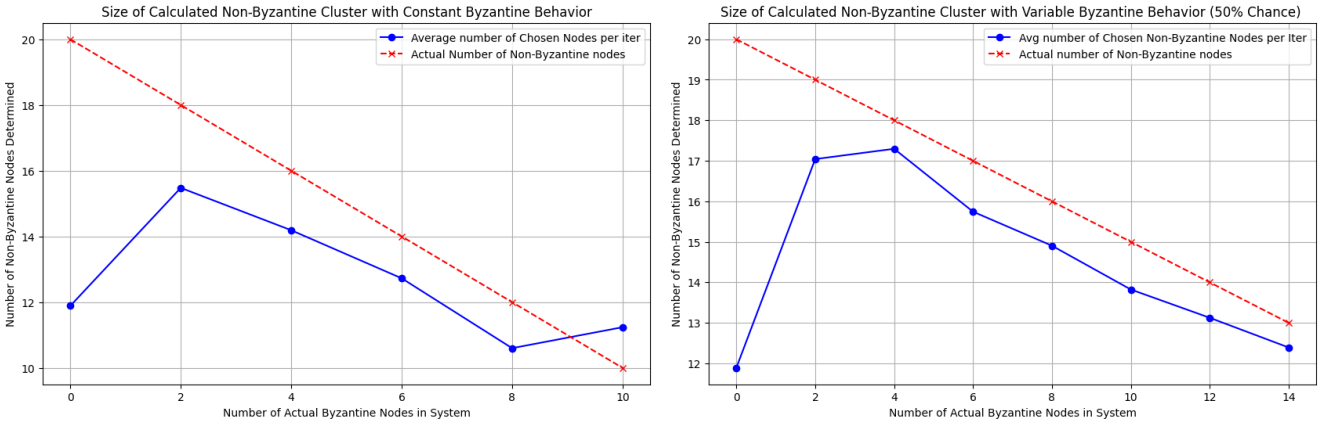

Figure 6. # of Selected Model Gradients against the # of real non-byzantine nodes in the system out of 20

network conditions rather than inherent inefficiencies in the Trust-Krum algorithm. Despite the minimal increase in computational overhead, Trust-Krum's ability to maintain higher accuracy and robustness in adversarial environments presents a favorable trade-off. The slight increase in computation time is offset by the significant gains in model security and integrity, ensuring that the federated learning model remains effective under adversarial attacks.

4.6 Unsupervised Clustering for Malicious Node Detection

After viewing the results of the benefit of utilizing Trust-Krum in varying systems, it was motivating to see if un-supervised clustering, specifically k-means clustering could be used to actually identify "different" or byzantine/malicious nodes in a system. The Krum and by extension, Trust-Krum algorithm contains a parameter f that can be passed in as an estimate of the number of malicious nodes in the system. While our results in Figure 1 and 2 demonstrate that correctly estimating the number of byzantine clients does provide a more robust model, the process of actually determining f is still noted as a hyper-parameter. The main question investigated is if k-means clustering can be utilized in order to determine this f parameter to an acceptable degree. In order to do this, at each aggregation step, k-means clustering was utilized to find the two different cluster groupings where the cluster with the lower average Trust-Krum scores was deemed as the cluster holding non-byzantine gradients. The experimentation to answer this question consisted of two different types of systems:

In the first system, there were 20 total nodes where 0,2,4,6,8,10 nodes were marked as byzantine. In this scenario, every aggregation step, the nodes that were marked as malicious actually sent malicious gradients through a gradient-flip attacks. As seen in the left graph of Figure 6, the average number of selected non-byzantine nodes was calculated and compared the the red dashed line which indicated the actual amount of non-byzantine nodes in the system. According to the graph, when the true number of malicious nodes in the system was set to 2,4,6,8 nodes the number of selected non-byzantine nodes seemed to match the slope of the red dashed line. However, overall the k-means clustering seemed to be slightly underestimating number of true malicious clients. However it was interesting to note that in the situation where there were 0 malicious clients, the k-means still ended up splitting the nodes into about two equal groupings. This could be a function of the unsupervised method as it is still looking to split nodes based on their Trust-Krum score even if the differences observed are minimal. However, the problem seems to lie in situations where number of malicious nodes approaches 50% or more. In this case it seems that according to the Krum algorithm when there are more malicious nodes compare to non-malicious, clustering should actually center around group with highest Krum-score because

now non-byzantine nodes are in the minority. To incorporate clustering to be beneficial in these scenarios this has to be taken into account as well.

In the second system, the nodes selected as malicious/byzantine each had a 50% chance at each aggregation step of sending a sign-flipped gradient. Therefore the results for this scenario as marked by the right graph in Figure 6 shows how the clustering worked for number of malicious selected nodes from 0 to 14 in intervals of 2. The red-dashed line shows the expected value number of non-byzantine Nodes in the system. The graph ends up showing how the same behavior is once again exhibited in scenarios where there are no byzantine nodes selected in the system. In these cases the clustering algorithm still splits the 20 nodes into two different clusters roughly in half albeit with a finer grain of difference.

Through these graphs it can be seen that clustering significantly helps and is accurate when there are malicious nodes in the system but number do not exceed >50% of total nodes

4.7 Overall Discussion

Throughout the experimentation, it was consistently found that Trust-Krum seemed to provide marginal although still a benefit in scenarios where certain nodes were constantly acting byzantine. This make sense since in these cases previous aggregation round behavior does not have much use in determining overall behavior. However, there was a significant benefit in scenarios where certain node behavior changes across different iterations where Trust-Krum provided a significant performance gain. Another approach of utilizing clustering to estimate malicious nodes provided promising results, however, were only limited to when byzantine nodes took up <50% of the system. For a potential algorithm to be designed around this, it needs to be able to correctly identify if byzantine nodes are the majority or minority on a system and adjust accordingly.

4.8 Conclusion

We propose a novel addition to the Krum metric in the case that incorporates historical trust. The experiments showed that Trust-Krum, as an additive trust mechanism on top of the Krum algorithm provided a significant boost of performance in scenarios where previous round history can be better utilized for predicting behavior of a current round leading to significant model accuracy improvement and gradient selection. It was also determined through preliminary analysis that clustering could be used with some success to estimate number of byzantine nodes in a system as long as it was less than 50%. Further work will be doing more experimentation to determine if this trust algorithm can be incorporated with other types of byzantine-tolerant FL algorithms such as Zeno and provide similar improvements.

References

[1] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. 2017. Byzantine-tolerant machine learning. *arXiv preprint arXiv:1703.02757* (2017).

[2] L. Bottou, C. Cortes, J.S. Denker, H. Drucker, I. Guyon, L.D. Jackel, Y. LeCun, U.A. Muller, E. Sackinger, P. Simard, and V. Vapnik. 1994. Comparison of classifier methods: a case study in handwritten digit recognition. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3 - Conference C: Signal Processing (Cat. No.94CH3440-5)*, Vol. 2. 77–82 vol.2. https://doi.org/10.1109/ICPR.1994.576879

[3] Djamila Bouhata, Hamouma Moumen, Jocelyn Ahmed Mazari, and Ahcène Bounceur. 2022. Byzantine Fault Tolerance in Distributed Machine Learning : a Survey. arXiv:2205.02572 [cs.DC]

[4] Ángel Alexander Cabrera, Erica Fu, Donald Bertucci, Kenneth Holstein, Ameet Talwalkar, Jason I. Hong, and Adam Perer. 2023. Zeno: An Interactive Framework for Behavioral Evaluation of Machine Learning. (2023). https://doi.org/10.1145/3544548.3581268

[5] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. 2020. FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping. *CoRR* abs/2012.13995 (2020). arXiv:2012.13995 https://arxiv.org/abs/2012.13995

[6] Clement Fung, Chris J. M. Yoon, and Ivan Beschastnikh. 2020. Mitigating Sybils in Federated Learning Poisoning. (2020). arXiv:1808.04866 [cs.LG]

[7] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. 2007. PeerReview: practical accountability for distributed systems. *SIGOPS Oper. Syst. Rev.* 41, 6 (oct 2007), 175–188. https://doi.org/10.1145/1323293.1294279

[8] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. 2003. The Eigentrust algorithm for reputation management in P2P networks. *Proceedings of the 12th international conference on World Wide Web* (2003).

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

[10] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2023. Communication-Efficient Learning of Deep Networks from Decentralized Data. (2023). arXiv:1602.05629 [cs.LG]

[11] Hajar Moudoud, Soumaya Cherkaoui, and Lyes Khoukhi. 2022. Towards a Secure and Reliable Federated Learning using Blockchain. *CoRR* abs/2201.11311 (2022). arXiv:2201.11311 https://arxiv.org/abs/2201.11311

[12] Solmaz Niknam, Harpreet S Dhillon, and Jeffrey H Reed. 2020. Federated learning for wireless communications: Motivation, opportunities, and challenges. *IEEE Communications Magazine* 58, 6 (2020), 46–51.

[13] Oracle. 2024. What Is Big Data? https://www.oracle.com/big-data/what-is-big-data/. Accessed: 2024-02-25.

[14] Asadullah Tariq, Mohamed Adel Serhani, Farag Sallabi, Tariq Qayyum, Ezedin S. Barka, and Khaled A. Shuaib. 2023. Trustworthy Federated Learning: A Survey. arXiv:2305.11537 [cs.AI]

[15] Marc Vucovich, Devin Quinn, Kevin Choi, Christopher Redino, Abdul Rahman, and Edward Bowen. 2023. FedBayes: A Zero-Trust Federated Learning Aggregation to Defend Against Adversarial Attacks. arXiv:2312.04587 [cs.CR]