Scalable and Distributed Computing

BRACE:

Byzantine-Resilient Aggregation via Consensus Enforcement

A preliminary analysis

Marco Antonio Corallo

531466

Contents

Introduction

Overview

BRACE (*Byzantine-Resilient Aggregation via Consensus Enforcement*) is a proposed protocol designed to implement and evaluate a *consensus-based* approach for enhancing the robustness of **Federated Learning** (FL) systems against *Byzantine attacks*.

Various strategies have been proposed in the literature to address the challenges posed by Byzantine clients. These include *similarity-based methods*, *verification-based techniques*, and passive defenses such as *Byzantine-robust aggregation*. More recently, blockchain-inspired approaches have also been explored. While these methods offer partial mitigation, they often fail to provide a comprehensive solution. A consensus-oriented approach, such as the one proposed in BRACE, represents a novel paradigm that could potentially integrate several of these techniques into a unified and adaptive defense framework.

As a preliminary step in the development of BRACE, a baseline FL system was designed, implemented, and evaluated using two distinct models and datasets.
This report presents a summary of the initial experimental analysis conducted to assess the behavior and performance of these FL systems under a variety of conditions and assumptions.

Technologies

The entire project was developed in Python 3, utilizing *PyTorch* as the core machine learning framework, and *Ray* for distributed system orchestration.

Ray was instrumental in enabling the system to *scale out*, particularly during the training and hyperparameter tuning phases. It also provided a convenient and efficient way to emulate distributed system behavior by leveraging the *Actor Model* paradigm: a mathematical model of concurrent computation in which each actor encapsulates its own state and behavior

and communicates asynchronously with other actors via message passing.

This model aligns naturally with the architecture of federated learning systems, where multiple independent clients (actors) perform local computations and communicate updates to a central server. Using Ray's actor-based abstraction, each federated client could be implemented as an isolated, stateful component that trains locally and interacts with the coordinator node without shared memory, thus making the system more modular, scalable, and fault-tolerant.

Additionally, several Bash scripts were developed to automate the training pipeline across multiple configurations of system parameters and hyperparameters. These scripts also handled logging and collection of results, facilitating reproducibility and systematic analysis.

Design Choices

System Overview

The core idea behind this project is to implement a simple yet flexible Federated Learning (FL) system upon which to conduct a variety of experiments. The system is composed of the following components:

- an aggregator server;
- a variable number `n` of clients;
- a variable number (possibly zero) `b` of *Byzantine* clients.

The system operates over a configurable number `r` of communication rounds.

In each round, the following sequence of steps occurs:

1. the server randomly selects `k` clients, with $\texttt{k} \leq \texttt{n}$;
2. the server broadcasts the current global model to the selected `k` clients;
3. each client trains the model locally on its private dataset;
4. the clients send their updated model weights back to the server;
5. the server aggregates the received weights and updates the global model.

Datasets and Models

To evaluate the system's performance and behavior, we selected two widely-used image classification datasets with varying complexity: *MNIST* and *FashionMNIST*.

For each dataset, a dedicated model selection phase was conducted to identify a simple yet effective neural architecture suitable for federated training. This process employed an $80\% - 20\%$ hold-out validation split, and all models were trained using the *Adam* optimizer. Since Fashion-MNIST is inherently more complex than MNIST, it required a deeper convolutional model.

The final architectures and their corresponding hyperparameters are summarized in Table 2.1, along with the resulting classification accuracy.

Table 2.1: Model Architectures and Hyperparameters

Attribute	**MNIST**	**FashionMNIST**
Architecture	Two-layer MLP: - Input: 784 - Hidden: 512 units - Output: 10 classes ReLU + Softmax	CNN: - Conv1: 1→32, kernel=3 - MaxPool (2x2) - Conv2: 32→64, kernel=3 - MaxPool (2x2) - FC1: 2304→600 - Dropout (p=0.25) - FC2: 600→120 - FC3: 120→10
Learning Rate	0.00013292918943162168	0.0006251373574521746
Batch Size	50	128
Epochs	20	10
Accuracy	0.9779	0.9088

These two datasets were chosen because they strike a good balance between simplicity and representativeness. They are small enough to allow reasonably fast training and experimentation, yet large enough to be meaningfully partitioned across clients. However, this introduced certain constraints on the system configuration.

To ensure that each client receives a sufficiently large training set, the total number of clients `n` was limited to a maximum of 10. For example, with 60,000 training samples, dividing the dataset among more than 10 clients would result in local datasets that are too small to train reliable models.

For *single-round* experiments, the dataset was partitioned into `n` disjoint subsets, each assigned to one client. For multi-round scenarios, we explored two distinct configurations:

- **Real-time tasks:** In this scenario, the local training data changes at each round to simulate dynamic environments such as edge computing, IoT, or autonomous systems. The dataset is split into $\mathtt{n} \cdot \mathtt{r}$ parts, one per client per round. To maintain meaningful sample sizes, we limited the experiments to 6 clients over 2 rounds in this configuration.
- **Persistent tasks:** Here, clients train on static local datasets across all rounds, representing more stable environments where data updates are infrequent. This setup allows larger local training sets and supports a greater number of rounds and clients. We tested configurations with 5 rounds and 5, 7, and 10 clients.

To analyze the system's behavior under various conditions, we conducted experiments across multiple configurations:

- varying the number of clients `n`, up to a maximum of 10;
- adjusting the fraction of participating clients `k`, ranging from 70% to 100% of `n`;
- introducing a variable number of Byzantine clients $\mathtt{b} \in [0, 3]$;
- testing across different numbers of communication rounds, from 1–2 (for real-time scenarios) up to 5.

Model updates were aggregated using simple averaging — the default aggregation strategy in federated learning.

Byzantine Client Behavior

To evaluate the system's robustness, we introduced adversarial behavior in the form of a well-known *Dirty-Label* attack, which falls under the broader category of *data poisoning* techniques. In this attack, a Byzantine client maliciously swaps the labels of two or more classes during local training, injecting corrupted gradients into the global model update.

This method is historically one of the first demonstrated poisoning attacks in FL and remains a widely used baseline, despite not being the most sophisticated or effective.

In our experiments:

- For the MNIST dataset, Byzantine clients flipped labels between digits `1` and `7`.

- For FashionMNIST, they flipped labels between classes `7` (sneakers) and `9` (ankle boots).

Even this simple manipulation caused a significant drop in model performance — over 20% degradation in the centralized (non-federated) setting — demonstrating the potential severity of label-flipping attacks in FL environments.

Analysis and Results

As a baseline, we first evaluated the system under *benign* conditions (i.e., without any Byzantine clients). As expected, the system's performance gradually decreased as the size of the local training sets was reduced, demonstrating the importance of maintaining sufficient data per client.
We conducted analyses on different configurations of the system parameters, changing the total number of clients, the number of rounds and the percentage for client selection. Then we identified the most realistic settings, as well as the best trade-offs between accuracy and scalability, and in these scenarios we introduced an increasing number of byzantine nodes. This kind of scenarios are:

- `n` (total number of clients) $\in [4, 5, 6]$,
- `r` (number of rounds) $\in [1, 2]$,
- `k` (client selection percentage) $\in [70\%, 80\%, 100\%]$,
- `b` (number of byzantine clients) $\in [0, 1, 2, 3]$

under real-time constraints, and

- `n` (total number of clients) $\in [5, 7, 10]$,
- `r` (number of rounds) $= 5$,
- `k` (client selection percentage) $\in [70\%, 80\%, 100\%]$,
- `b` (number of byzantine clients) $\in [0, 1, 2, 3, 5]$

for the other tasks.
In the following we report the main results for MNIST and FashionMNIST, under both the Real-Time and common assumptions.

MNIST

We report here the performances variation at each new byzantine introduced into the system. The full table of data are reported in Appendix A.

3.1.1 Real-Time Assumption

To analyze the produced data, we need to separate them with respect to the context, that is, the values for parameters `n`, `r`, `k`. Here we separate the data in three tables (3.1, 3.2, 3.3), based on the client selection technique used.

In general, the introduction of a second Byzantine client appears to be the most critical tipping point for system performance—specifically when the proportion of Byzantine nodes lies between 33.33% and 50% of the total number of clients. Performance degradation becomes noticeably more severe in this range. However, configurations involving at least two communication rounds exhibit significantly greater stability.
By comparing the performance curves of single-round systems with those of two-round systems as shown in Figure 3.3, a clear trend emerges. In single-round setups, the curves are generally similar and exhibit sharp declines in performance as the number of Byzantine clients increases. In contrast, two-round systems show slightly more variation in benign settings (i.e., with no Byzantine clients), but performance declines in a more gradual, less abrupt manner as adversarial activity increases. This suggests that introducing a second round of training inherently enhances the system's robustness against Byzantine behavior.

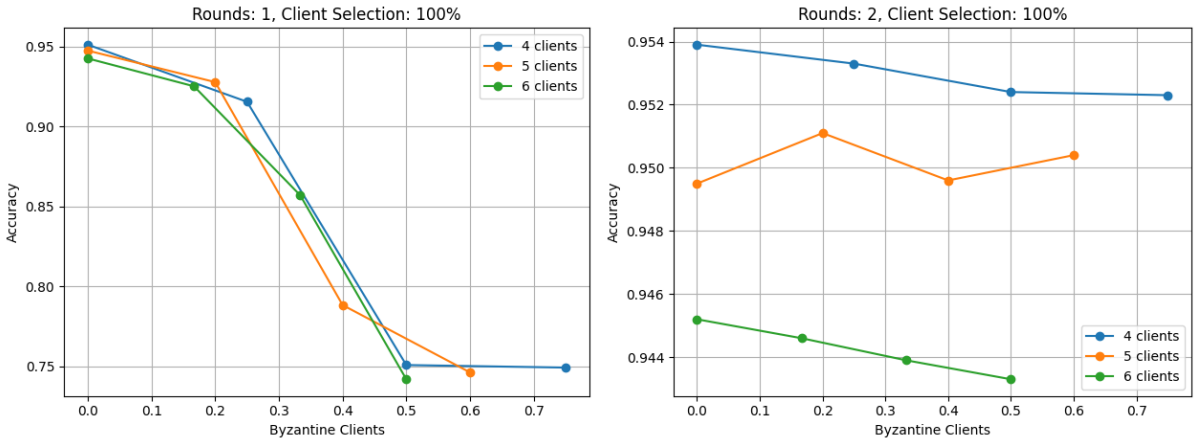

Figure 3.1: MNIST, comparison of number of rounds

Regarding client selection, the impact on performance appears to vary depending on the specific configuration. Notably, selecting 70% of the clients per round tends to yield better results in single-round systems with 5–6 clients, whereas this strategy does not appear to confer any significant advantage in two-round systems.

Table 3.1: MNIST, Real-Time - 100% client selection

N. Clients	**N. Rounds**	**Accuracy Sequence**
4	1	$0.9511 \rightarrow 0.9154 \rightarrow 0.7508 \rightarrow 0.7492$
4	2	$0.9539 \rightarrow 0.9533 \rightarrow 0.9524 \rightarrow 0.9523$
5	1	$0.9474 \rightarrow 0.9277 \rightarrow 0.7883 \rightarrow 0.7462$
5	2	$0.9495 \rightarrow 0.9511 \rightarrow 0.9496 \rightarrow 0.9504$
6	1	$0.9425 \rightarrow 0.9251 \rightarrow 0.8570 \rightarrow 0.7419$
6	2	$0.9452 \rightarrow 0.9446 \rightarrow 0.9439 \rightarrow 0.9433$

Table 3.2: MNIST, Real-Time - 80% client selection

N. Clients	**N. Rounds**	**Accuracy Sequence**
4	1	$0.9505 \rightarrow 0.8630 \rightarrow 0.7489 \rightarrow 0.7466$
4	2	$0.9515 \rightarrow 0.9528 \rightarrow 0.9514 \rightarrow 0.9521$
5	1	$0.9463 \rightarrow 0.9049 \rightarrow 0.7506 \rightarrow 0.7506$
5	2	$0.9484 \rightarrow 0.9496 \rightarrow 0.9483 \rightarrow 0.9481$
6	1	$0.9405 \rightarrow 0.8997 \rightarrow 0.8942 \rightarrow 0.7410$
6	2	$0.9439 \rightarrow 0.9428 \rightarrow 0.9437 \rightarrow 0.9420$

Table 3.3: MNIST, Real-Time - 70% client selection

N. Clients	**N. Rounds**	**Accuracy Sequence**
4	1	$0.9510 \rightarrow 0.7511 \rightarrow 0.7511 \rightarrow 0.7511$
4	2	$0.9527 \rightarrow 0.9528 \rightarrow 0.9517 \rightarrow 0.9517$
5	1	$0.9479 \rightarrow 0.9455 \rightarrow 0.8440 \rightarrow 0.8572$
5	2	$0.9493 \rightarrow 0.9480 \rightarrow 0.9488 \rightarrow 0.9474$
6	1	$0.9421 \rightarrow 0.9411 \rightarrow 0.9140 \rightarrow 0.7395$
6	2	$0.9427 \rightarrow 0.9428 \rightarrow 0.9434 \rightarrow 0.9432$

3.1.2 Persistent Training Set

These results further confirm the increased robustness of the system as the number of communication rounds grows. Not only do additional rounds lead to improved overall performance, but they also help mitigate the impact of Byzantine clients, preventing them from significantly degrading model accuracy.

It is also worth noting that, although overall performance tends to be slightly lower—an effect likely due to the reduced size of local training datasets—the system generally performs better when the total number of clients is higher. This is primarily because the proportion of Byzantine to benign clients is reduced in such configurations.

This observation is supported by additional experiments in which the number of Byzantine clients was increased to five. In these scenarios, where the proportion of Byzantine nodes ranges from 50% to 71.43% of the total clients, the system still performs well when the total number of clients is increased. Specifically, performance improves from approximately 77% with 5–7 clients to about 94% with 10 clients. These findings suggest that systems with a larger number of participating clients are generally more resilient to Byzantine attacks.

Finally, we observe that in this type of configuration, a narrower client selection policy—where only 70% of the clients are selected per round—tends to enhance performance in systems with 5–7 clients. However, this same strategy appears to be detrimental in systems with 10 clients, indicating a trade-off that depends on system scale.

Table 3.4: MNIST, 5 Rounds

N. Clients	Client Selection	Accuracy Sequence
5	100%	0.9758 → 0.9736 → 0.9691 → 0.9321
7	100%	0.9733 → 0.9718 → 0.9678 → 0.9589
10	100%	0.9677 → 0.9676 → 0.9656 → 0.9625
5	80%	0.9762 → 0.9738 → 0.9729 → 0.9579
7	80%	0.9737 → 0.9689 → 0.9620 → 0.9608
10	80%	0.9682 → 0.9657 → 0.9638 → 0.9610
5	70%	0.9788 → 0.9759 → 0.9750 → 0.9658
7	70%	0.9727 → 0.9688 → 0.9700 → 0.9701
10	70%	0.9681 → 0.9662 → 0.9615 → 0.9519

FashionMNIST

3.2.1 Real-Time Assumption

In this scenario, the impact of introducing a second Byzantine client appears to be significant only in systems with 4 or 5 clients. In contrast, its effect is negligible—or even slightly beneficial—in systems with 6 clients, where the proportion of Byzantine nodes decreases from 40% to 33.33%. This trend holds consistently across all three configurations, which differ in the proportion of clients selected at each round.
Once again, system performance proves to be substantially more stable in configurations involving at least two rounds, with performance degradation being nearly negligible across all settings.
As observed with the MNIST dataset, performance trends begin to diverge more clearly in multi-round configurations, whereas in single-round setups, system behavior tends to follow a more uniform pattern regardless of other parameters.

Table 3.5: FashionMNIST, Real-Time - 100% Client Selection

N. Clients	**Rounds**	**Accuracy Sequence**
4	1	$0.9027 \rightarrow 0.8982 \rightarrow 0.7323 \rightarrow 0.7110$
4	2	$0.9012 \rightarrow 0.9023 \rightarrow 0.9028 \rightarrow 0.9027$
5	1	$0.9005 \rightarrow 0.8987 \rightarrow 0.8057 \rightarrow 0.7099$
5	2	$0.8978 \rightarrow 0.9010 \rightarrow 0.8990 \rightarrow 0.8981$
6	1	$0.8976 \rightarrow 0.8949 \rightarrow 0.8666 \rightarrow 0.7117$
6	2	$0.8940 \rightarrow 0.8958 \rightarrow 0.8923 \rightarrow 0.8922$

Table 3.6: FashionMNIST, Real-Time - 80% Client Selection

N. Clients	**Rounds**	**Accuracy Sequence**
4	1	$0.9006 \rightarrow 0.8600 \rightarrow 0.7121 \rightarrow 0.7096$
4	2	$0.8997 \rightarrow 0.8966 \rightarrow 0.8963 \rightarrow 0.8974$
5	1	$0.8987 \rightarrow 0.8844 \rightarrow 0.7584 \rightarrow 0.7113$
5	2	$0.8974 \rightarrow 0.8982 \rightarrow 0.9002 \rightarrow 0.8996$
6	1	$0.8968 \rightarrow 0.8793 \rightarrow 0.8790 \rightarrow 0.7222$
6	2	$0.8922 \rightarrow 0.8932 \rightarrow 0.8953 \rightarrow 0.8909$

Table 3.7: FashionMNIST, Real-Time - 70% Client Selection

N. Clients	Rounds	Accuracy Sequence
4	1	0.8908 → 0.7664 → 0.7071 → 0.7090
4	2	0.8960 → 0.8975 → 0.8982 → 0.8948
5	1	0.8968 → 0.8978 → 0.8621 → 0.8298
5	2	0.8938 → 0.8972 → 0.8962 → 0.8957
6	1	0.8953 → 0.8790 → 0.8913 → 0.7069
6	2	0.8938 → 0.8981 → 0.8940 → 0.8881

3.2.2 Persistent Training Set

As for the *persistent* assumption in MNIST, these results confirm the increased robustness of the system as the number of communication rounds grows. More rounds not only improve overall performance but also prevent Byzantine clients from significantly degrading the model's accuracy.
In this scenario, we also notice that reducing the number of clients selected per round does not mitigate the impact of Byzantine clients; instead, it amplifies it. This effect may be attributed to the pseudo-random nature of the client selection process, which could occasionally lead to the repeated selection of the wrong subset—specifically, the Byzantine clients—thereby worsening the system's performance.

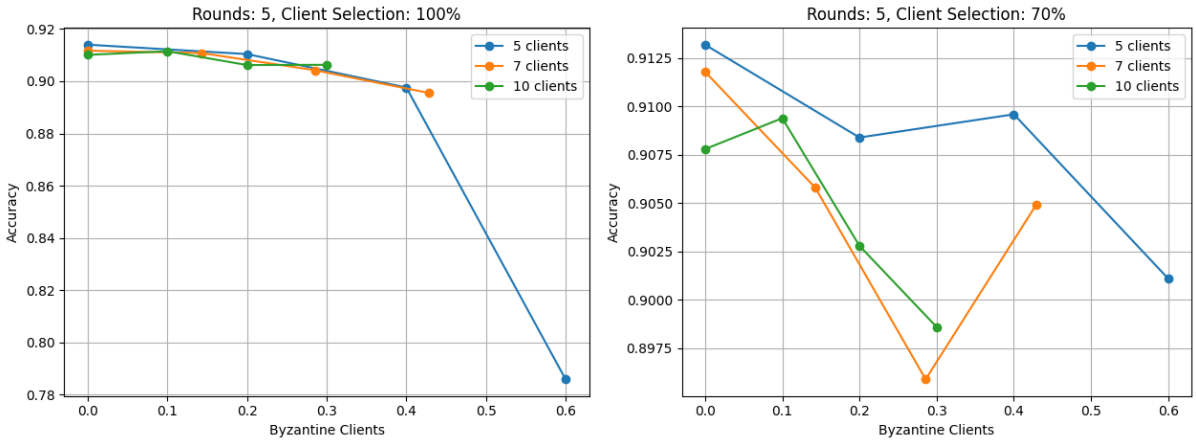

Figure 3.2: FashionMNIST, comparison of client selection

Aggregate Analysis

The analyses conducted on the two datasets were then aggregated to provide a more immediate and comprehensive view of system behavior as a

Table 3.8: FashionMNIST, 5 Rounds

N. Clients	Client Selection	Accuracy Sequence
5	100%	0.9140 → 0.9104 → 0.8976 → 0.7859
7	100%	0.9117 → 0.9108 → 0.9042 → 0.8955
10	100%	0.9101 → 0.9115 → 0.9062 → 0.9063
5	80%	0.9130 → 0.9092 → 0.9080 → 0.8727
7	80%	0.9100 → 0.9093 → 0.9039 → 0.9018
10	80%	0.9067 → 0.9072 → 0.9045 → 0.8932
5	70%	0.9132 → 0.9084 → 0.9096 → 0.9011
7	70%	0.9118 → 0.9058 → 0.8959 → 0.9049
10	70%	0.9078 → 0.9094 → 0.9028 → 0.8986

function of the number of Byzantine clients. These combined visualizations reinforce the individual findings observed for each model and, notably, highlight the critical role that client selection plays in determining system performance.

The aggregation of results across the two tasks supports the consistency of observed trends between the two datasets. This alignment suggests that the insights drawn from these experiments are not merely dataset-specific, but instead point to underlying systematic behaviors worthy of deeper investigation. These preliminary findings motivate further experimentation—on both existing and additional datasets—exploring larger client populations, more communication rounds, and alternative client selection strategies. The ultimate goal is to guide the design of federated learning systems that are both high-performing and resilient to adversarial conditions.

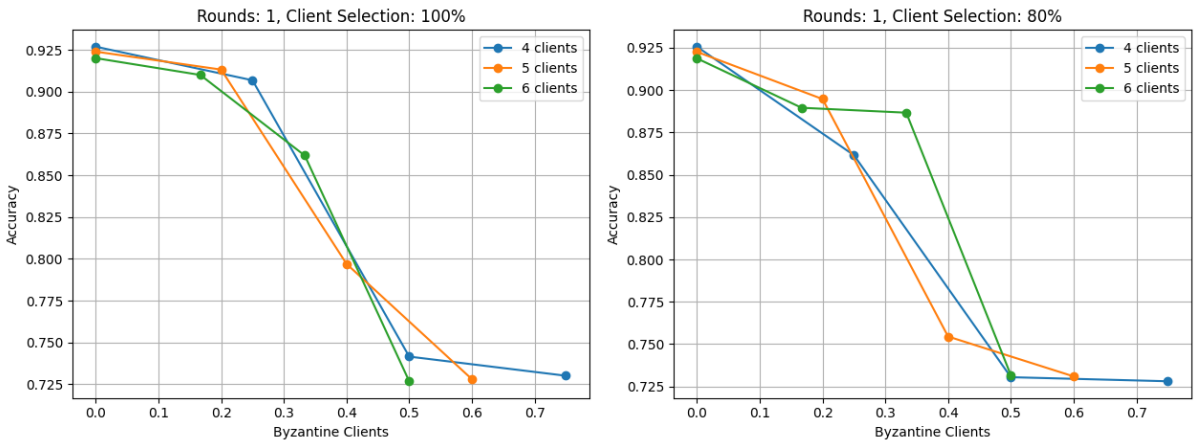

Figure 3.3: MNIST/FashionMNIST Aggregation, Real-Time Assumption

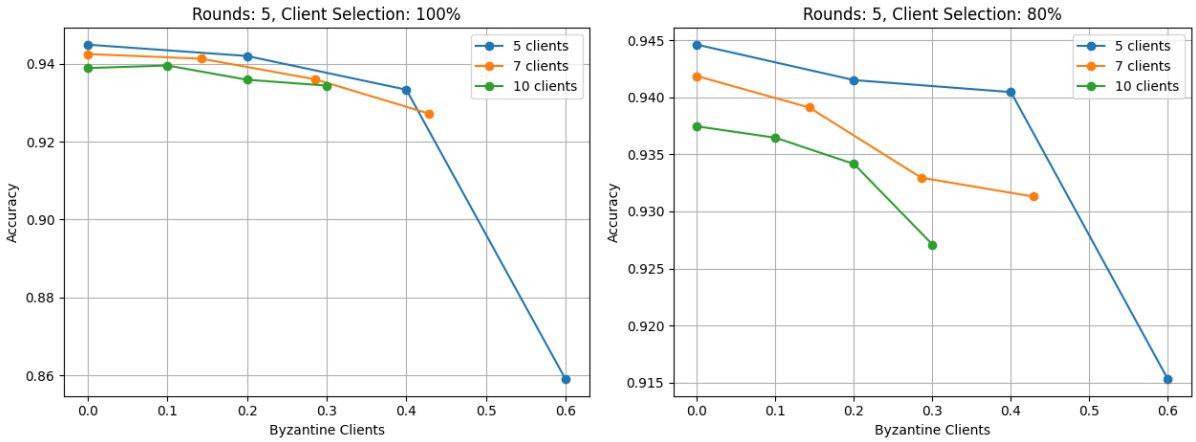

Figure 3.4: MNIST/FashionMNIST Aggregation, Persistent Assumption

Usage

4.0.1 Installation

To set up the environment, first install the required dependencies listed in the `requirements.txt` file. This can be done easily using `pip`:

```
pip install -r requirements.txt
```

The main components include Python 3, PyTorch, Ray, and other supporting libraries.

4.0.2 Running the Federated Learning System

The system is executed via the `federated_learning.py` script, which supports several configurable command-line arguments to customize the experiments:

- `-t` or `--task`: specifies the dataset/task to run. Options are `mnist` or `fashionmnist`.
- `-n` or `--n_clients`: sets the total number of clients participating in the simulation (default: 5).
- `-r` or `--rounds`: defines the number of federated learning rounds (default: 2).
- `-p` or `--percentage`: the fraction of clients selected to participate in each round (default: 1, i.e., all clients).
- `-b` or `--byzantine`: number of Byzantine (adversarial) clients in the system (default: 1).
- `-rtime` or `--realtime`: flag indicating that the local training dataset changes at each round, simulating real-time data distribution.

Example usage:

```
python federated_learning.py -n 10 -r 5 -p 0.5
```

This command runs the federated learning simulation with 10 clients, 5 rounds, and 50% of clients participating in each round.

4.0.3 Automation and Result Collection

To streamline experimentation across multiple configurations, two Bash scripts are provided:

- `collect.sh`: automates training runs and results collection for static datasets.
- `collect_rtime.sh`: automates experiments where the training set varies in each round (real-time scenario).

4.0.4 Analysis and Visualization

All experimental results are compiled and analyzed in a dedicated Python Jupyter notebook included in the project. This notebook contains scripts for loading results, generating performance plots, and facilitating further exploration of the system behavior under varying parameters.

Conclusion

The results of these analyses, along with the performance comparisons across different system configurations, offer valuable insights into the robustness of Federated Learning architectures in the presence of Byzantine clients. These findings serve as a foundation for identifying design principles that contribute to Byzantine-resilient FL systems. Based on the observed trends, the data suggest an optimal system architecture should include the following characteristics:

- A sufficiently large number of clients (≥ 6), which helps dilute the influence of any individual Byzantine node by lowering their relative proportion within the system;
- A number of communication rounds of at least 2, which significantly improves stability and mitigates performance degradation caused by adversarial behavior;
- A more nuanced and adaptive client selection strategy, dynamically defined in relation to other system parameters such as the total number of clients and the presence of potential adversaries. The client selection rate should be treated as a tunable hyperparameter of the distributed system itself. In general, overly narrow client selection (e.g., selecting only 70% of clients per round) may introduce instability and vulnerability, particularly in scenarios involving malicious participants.

Given the results, it is also worth considering a hybrid strategy even in real-time FL applications: rather than refreshing the local training sets at every round, the system could benefit from reusing the same datasets across multiple rounds. This approach may increase both learning effectiveness and resilience to data poisoning attacks, without significantly compromising the real-time nature of the task.

Overall, these findings form a preliminary yet meaningful basis for designing the first iteration of our Byzantine-resilient consensus protocol.

They provide both empirical evidence and strategic direction for the development of more secure and robust Federated Learning systems.

Future Work

Building on the results and observations of this preliminary study, several promising directions for future development and experimentation are identified. These will contribute to the continued refinement and validation of a robust, Byzantine-resilient Federated Learning framework. Key areas for future work include:

- **Exploring alternative aggregation strategies:** Future experiments will consider different model and weight aggregation methods beyond the simple average, including robust statistical techniques (e.g., median, trimmed mean, Krum, Bulyan) and adaptive approaches that account for anomalies or adversarial signals during training.
- **Evaluating diverse Byzantine attack strategies:** In addition to dirty-label data poisoning, other forms of attacks such as clean-label poisoning and model poisoning will be introduced to simulate more sophisticated adversarial behaviors. This will help assess the system's robustness across a broader threat landscape.
- **Scaling to larger datasets and client populations:** Testing the system on more complex and extensive datasets, along with higher numbers of participating clients, will be essential to validate the scalability and generalizability of the approach.
- **Experimenting with new client selection strategies:** Alternative client selection mechanisms will be explored, including trust-aware, performance-based, or history-driven selection policies. These methods may improve stability and enhance resilience in adversarial environments.
- **Design and implementation of a consensus protocol:** The ultimate goal of this project is the development of a dedicated consensus protocol tailored to Federated Learning. This protocol will aim to provide both robust aggregation and a principled way to handle trust and agreement across distributed nodes.

Together, these directions will guide the evolution of the system from a research prototype to a more mature, deployable solution that balances performance, robustness, and adaptability in real-world distributed learning scenarios.

Appendices

Accuracies

This appendix presents the detailed accuracy results obtained during the experimental phase of the project. The reported values refer to the performance of the trained models across different configurations, varying key parameters such as:

- Number of total clients (`n`);
- Number of communication rounds (`r`);
- Percentage of clients selected per round (`k`);
- Number of Byzantine clients (`b`);
- Dataset used (`MNIST` or `FashionMNIST`);
- Assumptions (`Real-Time` vs `Persistent`).

Each table in this section summarizes the final accuracy of the global model under the specified configuration. These results allow a comprehensive comparison of system behavior in both normal and adversarial settings, and support the analysis of robustness, generalization, and vulnerability to Byzantine failures.

The appendix serves as a reference for validating the discussion and conclusions drawn in the main body of the document, offering reproducibility and transparency for all tested scenarios.

N. Clients	N. Rounds	Client Selection	N. Byzantine	Accuracy
5	5	100	0	0.9758
5	5	80	0	0.9762
5	5	70	0	0.9788
7	5	100	0	0.9733
7	5	80	0	0.9737
7	5	70	0	0.9727
10	5	100	0	0.9677
10	5	80	0	0.9682
10	5	70	0	0.9681
5	5	100	1	0.9736
5	5	100	2	0.9691
5	5	100	3	0.9321
5	5	80	1	0.9738
5	5	80	2	0.9729
5	5	80	3	0.9579
5	5	70	1	0.9759
5	5	70	2	0.975
5	5	70	3	0.9658
7	5	100	1	0.9718
7	5	100	2	0.9678
7	5	100	3	0.9589
7	5	80	1	0.9689
7	5	80	2	0.962
7	5	80	3	0.9608
7	5	70	1	0.9688
7	5	70	2	0.97
7	5	70	3	0.9701
10	5	100	1	0.9676
10	5	100	2	0.9656
10	5	100	3	0.9625
10	5	80	1	0.9657
10	5	80	2	0.9638
10	5	80	3	0.961
10	5	70	1	0.9662
10	5	70	2	0.9615
10	5	70	3	0.9519

Table A.1: MNIST, Persistent Assumption

N. Clients	N. Rounds	Client Selection	N. Byzantine	Accuracy
4	1	100	0	0.9511
4	1	80	0	0.9505
4	1	70	0	0.951
5	1	100	0	0.9474
5	1	80	0	0.9463
5	1	70	0	0.9479
6	1	100	0	0.9425
6	1	80	0	0.9405
6	1	70	0	0.9421
4	1	100	1	0.9154
4	1	100	2	0.7508
4	1	100	3	0.7492
4	1	80	1	0.863
4	1	80	2	0.7489
4	1	80	3	0.7466
4	1	70	1	0.7511
4	1	70	2	0.7511
4	1	70	3	0.7511
5	1	100	1	0.9277
5	1	100	2	0.7883
5	1	100	3	0.7462
5	1	80	1	0.9049
5	1	80	2	0.7506
5	1	80	3	0.7506
5	1	70	1	0.9455
5	1	70	2	0.844
5	1	70	3	0.8572
6	1	100	1	0.9251
6	1	100	2	0.857
6	1	100	3	0.7419
6	1	80	1	0.8997
6	1	80	2	0.8942
6	1	80	3	0.741
6	1	70	1	0.9411
6	1	70	2	0.914
6	1	70	3	0.7395

Table A.2: MNIST, Real-Time Assumption, 1 Round

N. Clients	N. Rounds	Client Selection	N. Byzantine	Accuracy
4	2	100	0	0.9539
4	2	80	0	0.9515
4	2	70	0	0.9527
5	2	100	0	0.9495
5	2	80	0	0.9484
5	2	70	0	0.9493
6	2	100	0	0.9452
6	2	80	0	0.9439
6	2	70	0	0.9427
4	2	100	1	0.9533
4	2	100	2	0.9524
4	2	100	3	0.9523
4	2	80	1	0.9528
4	2	80	2	0.9514
4	2	80	3	0.9521
4	2	70	1	0.9528
4	2	70	2	0.9517
4	2	70	3	0.9517
5	2	100	1	0.9511
5	2	100	2	0.9496
5	2	100	3	0.9504
5	2	80	1	0.9496
5	2	80	2	0.9483
5	2	80	3	0.9481
5	2	70	1	0.948
5	2	70	2	0.9488
5	2	70	3	0.9474
6	2	100	1	0.9446
6	2	100	2	0.9439
6	2	100	3	0.9433
6	2	80	1	0.9428
6	2	80	2	0.9437
6	2	80	3	0.942
6	2	70	1	0.9428
6	2	70	2	0.9434
6	2	70	3	0.9432

Table A.3: MNIST, Real-Time Assumption, 2 Rounds

N. Clients	N. Rounds	Client Selection	N. Byzantine	Accuracy
5	5	100	0	0.914
5	5	80	0	0.913
5	5	70	0	0.9132
7	5	100	0	0.9117
7	5	80	0	0.91
7	5	70	0	0.9118
10	5	100	0	0.9101
10	5	80	0	0.9067
10	5	70	0	0.9078
5	5	100	1	0.9104
5	5	80	1	0.9092
5	5	70	1	0.9084
7	5	100	1	0.9108
7	5	80	1	0.9093
7	5	70	1	0.9058
10	5	100	1	0.9115
10	5	80	1	0.9072
10	5	70	1	0.9094
5	5	100	2	0.8976
5	5	80	2	0.908
5	5	70	2	0.9096
7	5	100	2	0.9042
7	5	80	2	0.9039
7	5	70	2	0.8959
10	5	100	2	0.9062
10	5	80	2	0.9045
10	5	70	2	0.9028
5	5	100	3	0.7859
5	5	80	3	0.8727
5	5	70	3	0.9011
7	5	100	3	0.8955
7	5	80	3	0.9018
7	5	70	3	0.9049
10	5	100	3	0.9063
10	5	80	3	0.8932
10	5	70	3	0.8986

Table A.4: FashionMNIST, Persistent Assumption

N. Clients	N. Rounds	Client Selection	N. Byzantine	Accuracy
4	1	100	0	0.9027
4	1	80	0	0.9006
4	1	70	0	0.8908
5	1	100	0	0.9005
5	1	80	0	0.8987
5	1	70	0	0.8968
6	1	100	0	0.8976
6	1	80	0	0.8968
6	1	70	0	0.8953
4	1	100	1	0.8982
4	1	80	1	0.86
4	1	70	1	0.7664
5	1	100	1	0.8987
5	1	80	1	0.8844
5	1	70	1	0.8978
6	1	100	1	0.8949
6	1	80	1	0.8793
6	1	70	1	0.879
6	2	100	1	0.8958
6	2	80	1	0.8932
6	2	70	1	0.8981
4	1	100	2	0.7323
4	1	80	2	0.7121
4	1	70	2	0.7071
5	1	100	2	0.8057
5	1	80	2	0.7584
5	1	70	2	0.8621
6	1	100	2	0.8666
6	1	80	2	0.879
6	1	70	2	0.8913
4	1	100	3	0.711
4	1	80	3	0.7096
4	1	70	3	0.709
5	1	100	3	0.7099
5	1	80	3	0.7113
5	1	70	3	0.8298
6	1	100	3	0.7117
6	1	80	3	0.7222
6	1	70	3	0.7069

Table A.5: FashionMNIST, Real-Time Assumption, 1 Round

N. Clients	N. Rounds	Client Selection	N. Byzantine	Accuracy
4	2	100	0	0.9012
4	2	80	0	0.8997
4	2	70	0	0.896
5	2	100	0	0.8978
5	2	80	0	0.8974
5	2	70	0	0.8938
6	2	100	0	0.894
6	2	80	0	0.8922
6	2	70	0	0.8938
4	2	100	1	0.9023
4	2	80	1	0.8966
4	2	70	1	0.8975
5	2	100	1	0.901
5	2	80	1	0.8982
5	2	70	1	0.8972
4	2	100	2	0.9028
4	2	80	2	0.8963
4	2	70	2	0.8982
5	2	100	2	0.899
5	2	80	2	0.9002
5	2	70	2	0.8962
6	2	100	2	0.8923
6	2	80	2	0.8953
6	2	70	2	0.894
4	2	100	3	0.9027
4	2	80	3	0.8974
4	2	70	3	0.8948
5	2	100	3	0.8981
5	2	80	3	0.8996
5	2	70	3	0.8957
6	2	100	3	0.8922
6	2	80	3	0.8909
6	2	70	3	0.8881

Table A.6: FashonMNIST, Real-Time Assumption, 2 Rounds