Fourth Lab Report: Federated Learning

Université Côte d'Azur – MSc Data Science & Artificial Intelligence

Zeev Weizmann
zeev.weizmann@etu.univ-cotedazur.fr

November 2025

Fourth Lab Session — Federated Learning & Data Privacy

Topic of this lab is on malicious attacks in Federated Learning and the corresponding defenses.

Ex1: Implementation of Label-Flipping Attacks

First, I generated the data:

```
rm -r mnist/all_data

python generate_data.py \
  --dataset mnist \
  --n_clients 10 \
  --iid \
  --frac 0.1 \
  --save_dir mnist \
  --seed 1234
```

In `learner.py`, I created a new learner class `Label_Flipping_Learner`. This class overrides the training process so that all labels are shifted by one:

```
class Label_Flipping_Learner(Learner):

    def fit_epoch(self, loader):
        """
        Same as Learner.fit_epoch but flips labels by +1 (
            mod 10)
        """
        self.model.train()

        for x, y in loader:
```

```
            # Flip labels
            y = (y + 1) % 10   # ensures class stays in 0..9

            x = x.to(self.device).float()
            y = y.to(self.device).long()

            if self.is_binary_classification:
                y = y.float().unsqueeze(1)

            self.optimizer.zero_grad()

            outs = self.model(x)
            loss = self.criterion(outs, y)

            loss.backward()
            self.optimizer.step()
```

I added the parameter `prop` in `args.py` to represent the proportion of malicious clients:

```
parser.add_argument(
    "--prop",
    type=float,
    default=0.0,
    help="proportion of malicious clients"
)
```

Next, we modified the `init_client` function in `utils.py`. Clients whose IDs fall within $[0, \text{prop} \cdot 10)$ use the malicious learner:

```
def init_client(args, client_id, client_dir, logger):

    train_loader = get_loader(args.experiment, client_dir,
        args.bz, train=True)
    val_loader   = get_loader(args.experiment, client_dir,
        args.bz, train=False)
    test_loader  = get_loader(args.experiment, client_dir,
        args.bz, train=False)

    num_malicious = int(args.prop * args.n_clients)

    if client_id < num_malicious:
        print(f"[MALICIOUS] client {client_id} uses
            Label_Flipping_Learner")

        model = get_model(args.experiment, args.device)
        criterion = nn.CrossEntropyLoss().to(args.device)
        metric = accuracy
        optimizer = get_optimizer(
            optimizer_name=args.local_optimizer,
            model=model,
```

```
            lr=args.local_lr,
            mu=args.mu,
        )

        learner = Label_Flipping_Learner(
            model=model,
            criterion=criterion,
            metric=metric,
            device=args.device,
            optimizer=optimizer,
            is_binary_classification=False
        )

    else:
        learner = get_learner(
            experiment_name=args.experiment,
            device=args.device,
            optimizer_name=args.local_optimizer,
            lr=args.local_lr,
            mu=args.mu,
            seed=args.seed
        )

    return Client(
        client_id=client_id,
        learner=learner,
        train_loader=train_loader,
        val_loader=val_loader,
        test_loader=test_loader,
        local_steps=args.local_steps,
        logger=logger
    )
```

We then ran experiments with proportions $\{0, 0.1, 0.3, 0.5\}$ under i.i.d. settings:

```
for p in 0 0.1 0.3 0.5
do
    mkdir -p logs/mnist_prop_${p}

    python train.py \
        --experiment "mnist" \
        --n_rounds 25 \
        --local_steps 1 \
        --local_optimizer sgd \
        --local_lr 0.001 \
        --server_optimizer sgd \
        --server_lr 0.1 \
        --bz 128 \
        --device "cpu" \
```

```
        --log_freq 1 \
        --verbose 1 \
        --logs_dir "logs/mnist_prop_${p}/" \
        --prop $p \
        --seed 12 \
        --n_clients 10
done
```

The resulting accuracy values were plotted as shown below (Figure 1):

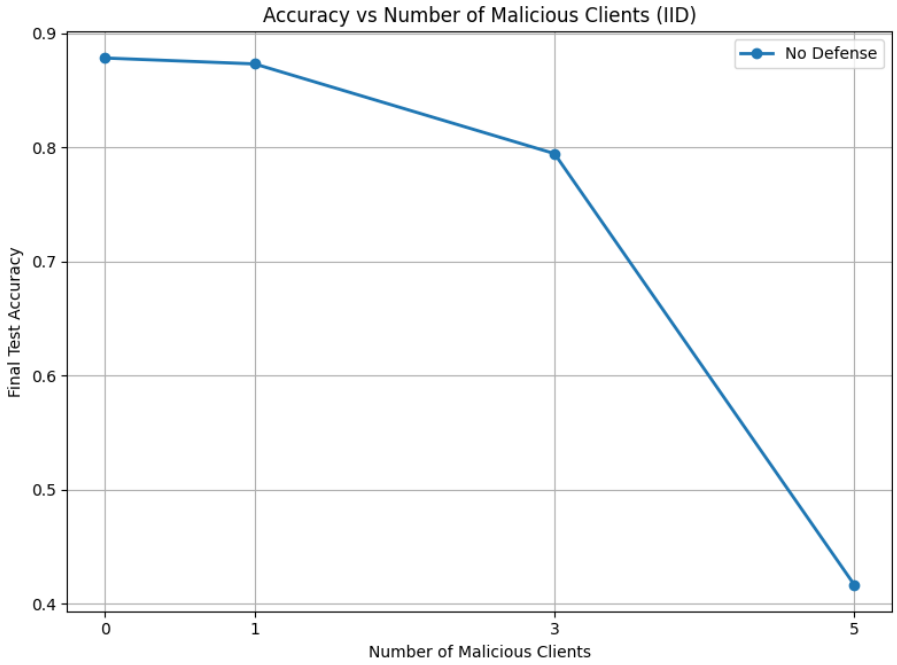

Figure 1: Effect of malicious clients on final accuracy (IID case).

Observations

From the plot, we observe that the accuracy remains high when only 0 or 1 malicious client is present. However, once 30% of the clients are malicious, the final accuracy starts to decline significantly. When 50% of the clients flip labels, the global model collapses and the accuracy drops to around 0.4.

Conclusion: Without any defense, FedAvg is highly vulnerable to label-flipping attacks. As the proportion of malicious clients increases, the global model rapidly degrades.

Exercise 2: Implementation of Defenses

In this exercise, I implemented a robust aggregation rule to defend Federated Learning against label-flipping attacks and compared the results with the baseline FedAvg experiment.

Median Aggregator

I added a new aggregation class `MedianAggregator` in `aggregator.py`:

```
class MedianAggregator(Aggregator):
    """Aggregator with coordinate-wise median (defense against attacks)."""

    def mix(self):

        self.sample_clients()

        # Local updates
        for client in self.sampled_clients:
            client.step()

        # Gather local models
        state_dicts = [client.learner.model.state_dict()
                       for client in self.sampled_clients]

        # Compute coordinate-wise median
        median_state = median_models(state_dicts)

        # Update global model
        self.global_learner.model.load_state_dict(median_state)

        # Broadcast to all clients
        for client in self.clients:
            copy_model(client.learner.model, self.global_learner.model)

        self.c_round += 1
```

Coordinate-Wise Median Function

I added the function `median_models` in `utils/torch_utils.py`, which computes the coordinate-wise median of all received client models:

```
def median_models(state_dicts):
    """
    Compute the coordinate-wise median of a list of state_dicts.
    """
    if len(state_dicts) == 0:
```

```
        raise ValueError("median_models received an empty list")

    keys = state_dicts[0].keys()
    median_state = {}

    for k in keys:
        tensors = [sd[k].detach().cpu().float() for sd in state_dicts]
        stacked = torch.stack(tensors, dim=0)
        median_tensor = torch.median(stacked, dim=0).values
        median_state[k] = median_tensor

    return median_state
```

Integration Into `get_aggregator`

I updated the `get_aggregator` function to support the new aggregator:

```
elif aggregator_type == "median":
    return MedianAggregator(
        clients=clients,
        clients_weights=clients_weights,
        global_learner=global_learner,
        logger=logger,
        sampling_rate=sampling_rate,
        sample_with_replacement=sample_with_replacement,
        verbose=verbose,
        seed=seed,
    )
```

Running the Experiments

I ran the experiments using:

$$\texttt{prop} \in \{0,\, 0.1,\, 0.3,\, 0.5\}, \qquad \texttt{aggregator\_type} = \text{"median"},$$

under the IID setting. For each configuration, I saved the final accuracy in a separate folder.

Displaying the Results on the Previous Graph

I plotted the median-based accuracies on the same graph $G1$ used in Exercise 1. The resulting Figure 2 is shown below:

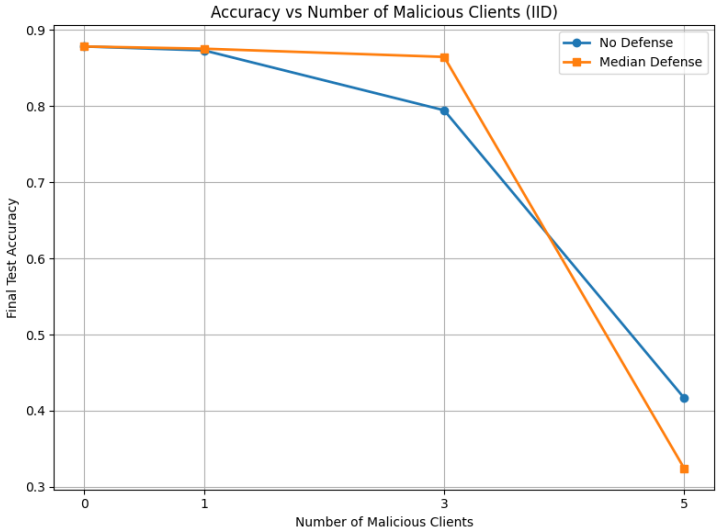

Figure 2: Comparison between FedAvg (no defense) and Median defense under varying attack intensities in the IID setting.

Observations

From the combined plot, I observe that the coordinate-wise median provides a clear advantage when the proportion of malicious clients is moderate.

- For `prop` $= 0$ and 0.1, the accuracy of the median aggregator is almost identical to the baseline FedAvg, which shows that the defense does not harm performance when no attack is present.
- For `prop` $= 0.3$, the median clearly outperforms FedAvg: while the average aggregation already begins to degrade, the coordinate-wise median preserves high accuracy. This demonstrates the robustness of the median against moderate label-flipping attacks.
- However, when the proportion of malicious clients reaches 50%, the behavior reverses. In this regime, the median aggregator performs *worse* than standard averaging. This is expected: once the adversarial majority dominates each coordinate, the median itself becomes corrupted and no longer reflects the honest updates. In this setting, even the simple mean performs better because it still retains the influence of the honest minority.

Overall, the median defense is highly effective when malicious clients remain a minority, but it breaks down once the adversaries constitute 50% or more of the population.

Ex3: Simulation of the Non-IID Case

I regenerated the dataset using the `--non_iid` option:

```
echo "=> Generate data.."

cd data/

rm -r mnist/all_data

python generate_data.py \
  --dataset mnist \
  --n_clients 10 \
  --non_iid \
  --frac 0.1 \
  --save_dir mnist \
  --seed 1234
```

I then ran the experiments for proportions of malicious clients equal to `0, 0.1, 0.3, 0.5`. Next, I repeated the experiments using `args.aggregator_type = "median"`. Both curves (with and without the defense) were plotted on the same graph G2, with the number of malicious clients on the x-axis and the final test accuracy on the y-axis. The resulting figure is shown in Figure 3.

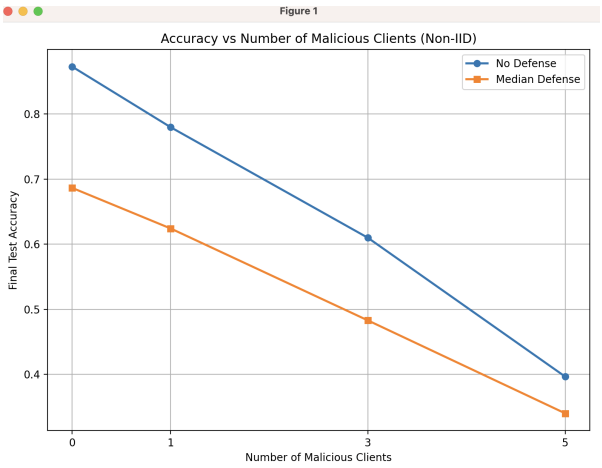

Figure 3: Final accuracy under label-flipping attack in the non-iid setting for different proportions of malicious clients.

Observations

In the non-IID scenario, the model accuracy degrades significantly faster compared to the IID case.

More interestingly, the median defense—which was effective in the IID setup—now performs *worse* than the standard FedAvg aggregation, and the gap becomes larger as the proportion of malicious clients increases. While this may appear counter-intuitive, the phenomenon is directly related to how the coordinate-wise median behaves under data heterogeneity.

Explanation

In the non-IID setting, honest clients train on heterogeneous data distributions, and thus their updates differ substantially from one another. The coordinate-wise median assumes that honest updates form a tight, coherent cluster. When this assumption is violated:

- honest updates become widely dispersed and may appear as outliers;
- the median operator suppresses many legitimate updates;
- this reduces the useful learning signal and leads to under-training of the global model.

In contrast, FedAvg is more tolerant to heterogeneous gradients: even when local updates disagree, their weighted average still captures meaningful global information. This explains why FedAvg outperforms the median defense in the non-IID case, despite not providing any robustness guarantees.

In summary, although the result may seem surprising, it follows directly from theoretical properties of robust aggregation: **median-based aggregation is reliable only when the honest majority is both large and coherent**, which does not hold under non-IID data distributions.