FedSECA: Sign Election and Coordinate-wise Aggregation of Gradients for Byzantine Tolerant Federated Learning
—— Supplementary ——

A. Data Distributions

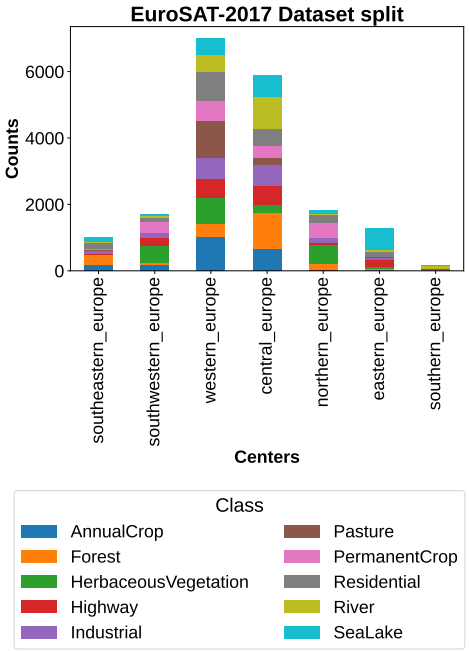

Figure A.1. We split EuroSAT[5] into 7 clients based on subregion grouping according to "The World Factbook". We use the geolocation information in GeoTIFF files to identify the countries of origin, based on which we assign images to specific clients.

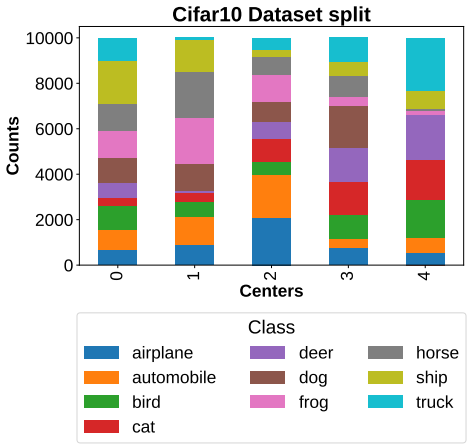

Figure A.2. We split dataset into 5 clients with class imbalance.

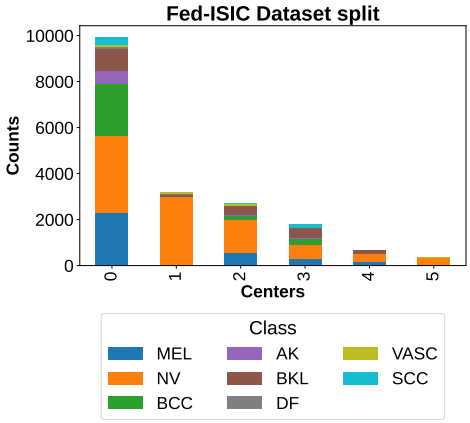

Figure A.3. We use exact clients split in FedISIC[8], which splits dataset based on devices and hospital of origin.

B. Training Setup

For all the experiments, we use F1-Score as a performance metric to account for class imbalances in test data. All the results are reported on a common pooled testset for the respective datasets. Collaboration happens at the end of every local training epoch wherein the clients would send all the trained models to the server. Global aggregation in the server (RAGGR) is carried out once all models are received, and after the aggregation, a single model is broadcast to all the clients. This will be used as starting weights for the next epoch locally. We call one single cycle of collaboration a round with a total of $\mathtt{T}$ rounds. Since we take a cross-silo setting, we assume all clients will be available for all rounds throughout collaboration and assume hardware capabilities are homogeneous to train for a given task with a given model.

FedSECA hyperparameters: Our proposed RAGGR has 2 hyperparameters corresponding to sparsification and server momentum, which are kept the same for all experiments, irrespective of the dataset. We use the Sparsification factor $\gamma = 0.9$, following it as a general rule of thumb from gradient compression and sparsification literature. We set aggregator momentum $\beta_{\mathtt{ra}} = 0.5$

Local Training Setup: Local (clients) training hyperparameters are constant across all experiments, defense/attack settings, and ablations unless specified explicitly. We use AdamW optimizer with weight decay$= 1 \times 10^{-6}$ and betas of $\beta_1 = 0.9, \beta_2 = 0.999$ and optimizer state is reset for every epoch. For Cifar10 $lr = 1 \times 10^{-3}$ ran for 50 epochs with a batch size of 64 and image size 224 and Kaiming weight initialization is used; for ISIC $lr = 1 \times 10^{-4}$ ran for 100 epochs with a batch size of 32 and image size of 200 and Imagenet-1K pretrained weights is used as initialization; for EuroSAT $lr = 1 \times 10^{-5}$ ran for 50 epochs with a batch size of 32 and image size of 224 and Imagenet-1K pretrained weights is used as initialization.

Comparison with Baselines: For experiments in *Paper-Fig.3*, the datasets are divided into cross-silo clients as shown in Sec. A. The Mean values of the last 5 communication rounds are tabulated in Tab. C.1 highlighting the trainings that collapsed or were severely impacted. We use F1-Score as a performance metric considering class imbalances in test data. Experiments were run on NVIDIA GeForce RTX 4090 24GB machines. For all three datasets, 2 clients in 4^{th} and 5^{th} rank are Byzantines. We compare several robust aggregators with our method, and the exact formulations and hyperparameters of the compared robust aggregators are discussed in Sec. E and attacks are discussed Sec. D.

FedSECA Component Ablation: In the experiments reported in *Paper-Tab.1*, we disable and enable certain components in order to check the criticality of components in defending against specific attacks. We use the CIFAR10 split shown in Fig. A.2. Experiments were run on NVIDIA GeForce RTX 4090 24GB machines. The *Paper-Tab.1* second column also shows the variable that we set to enable or disable certain components.

FedSECA Hyperparameter Sensitivity: In the experiments in *Paper-Fig.4*, we study the effect of varying the two hyperparameters in the FedSECA. We use the CIFAR10 split shown in Fig. A.2. Experiments were run on NVIDIA GeForce RTX 4090 24GB machines. For server momentum ($\beta_{\mathtt{ra}}$), we experiment with the values $\{0.9,\ 0.75,\ 0.5,\ 0.25,\ 0.1,\ 0.0\}$. And for the Sparsificaton ratio (γ), we experiment with values $\{0.99,\ 0.9,\ 0.82,\ 0.67,\ 0.51,\ 0.22\}$ with 50 rounds collaboration.

Proportion of Byzantines: For experiments in *Paper-Fig.5*, we use CIFAR10-IID split with 64 clients where class label distribution is homogeneous. The batch size of 16 is used for local training. The Byzantines counts are varied as $8(12\%),\ 17(25\%),\ 31(< 50\%),\ 34(> 50\%),\ 40(62\%)$ out of 64 clients. The experiment was run on NVIDIA A100 SXM4 40GB machines.

Increasing Number of Clients: For experiments in *Paper-Tab.2*, we use CIFAR10-IID split with varying number of clients as $5,\ 10,\ 25,\ 50,\ 100,\ 200$. The batch size of 64 and all local training hyperparameters are the same. The experiments were run on NVIDIA RTX A6000 49GB machines. Additionally, the time taken by each robust aggregator function with varying numbers of clients is benchmarked in Tab. C.2 using CPU computation alone on Intel(R) Xeon(R) Silver 4215 CPU @ 2.50GHz. The FedSECA is designed for cross-silo applications where only a dozen clients are expected in collaboration; for the sake of being exhaustive, we compare the time for client counts from 4 to 256.

C. FedSECA

Robust Aggregator	CIFAR10									EuroSAT									FedISIC								
	ALIE	IPM	Fang	LabelFlip	Mimic	Scale	MinMax	No Attack	*Mean*	ALIE	IPM	Fang	LabelFlip	Mimic	Scale	MinMax	No Attack	*Mean*	ALIE	IPM	Fang	LabelFlip	Mimic	Scale	MinMax	No Attack	*Mean*
FedAvg (No Defense)	0.10	0.02	0.02	0.69	0.82	0.02	0.02	0.88	*0.32*	0.97	0.90	0.02	0.89	0.97	0.02	0.02	0.98	*0.60*	0.71	0.53	0.01	0.59	0.77	0.04	0.08	0.77	*0.44*
Krum	0.02	0.02	0.04	0.76	0.62	0.76	0.02	0.76	*0.38*	0.98	0.41	0.54	0.54	0.93	0.54	0.54	0.54	*0.63*	0.24	0.06	0.22	0.09	0.67	0.22	0.18	0.22	*0.24*
RFA	0.14	0.50	0.02	0.80	0.71	0.65	0.02	0.88	*0.46*	0.97	0.70	0.01	0.93	0.96	0.14	0.01	0.97	*0.59*	0.62	0.29	0.08	0.51	0.51	0.04	0.08	0.67	*0.35*
CWTM	0.04	0.02	0.05	0.66	0.79	0.02	0.02	0.89	*0.31*	0.97	0.58	0.02	0.88	0.96	0.02	0.02	0.97	*0.55*	0.61	0.31	0.08	0.54	0.62	0.04	0.00	0.72	*0.36*
HuberLoss	0.14	0.72	0.04	0.85	0.62	0.77	0.02	0.89	0.51	0.97	0.69	0.78	0.93	0.95	0.96	0.78	0.97	*0.88*	0.61	0.29	0.35	0.50	0.66	0.66	0.36	0.67	0.51
CClipping	0.46	0.62	0.02	0.85	0.83	0.02	0.02	0.88	*0.46*	0.97	0.90	0.02	0.89	0.97	0.19	0.02	0.98	*0.62*	0.71	0.53	0.08	0.59	0.76	0.27	0.00	0.77	*0.46*
CC-RandBucket	0.06	0.02	0.02	0.72	0.83	0.02	0.02	0.88	*0.32*	0.97	0.88	0.02	0.79	0.97	0.15	0.02	0.98	*0.60*	0.71	0.53	0.01	0.58	0.78	0.16	0.08	0.76	*0.45*
CC-SeqBucket	0.79	0.54	0.02	0.84	0.80	0.02	0.02	0.88	*0.49*	0.95	0.88	0.02	0.50	0.96	0.15	0.02	0.96	*0.55*	0.71	0.47	0.02	0.45	0.73	0.24	0.00	0.72	0.42
COPOD-DOS	0.82	0.47	0.04	0.84	0.63	0.02	0.03	0.87	*0.46*	0.95	0.22	0.93	0.93	0.92	0.92	0.05	0.93	*0.73*	0.42	0.13	0.51	0.27	0.42	0.10	0.08	0.49	*0.30*
FL-Detector	0.02	0.52	0.46	0.06	0.44	0.81	0.70	0.88	*0.49*	0.96	0.72	0.93	0.93	0.97	0.94	0.93	0.97	0.92	0.37	0.24	0.57	0.50	0.44	0.57	0.57	0.66	*0.49*
TIES-Merging	0.85	0.02	0.73	0.72	0.81	0.02	0.03	0.82	*0.5*	0.98	0.92	0.02	0.91	0.97	0.02	0.01	0.98	*0.60*	0.73	0.00	0.05	0.60	0.72	0.04	0.00	0.75	*0.36*
FedSECA (Ours)	0.72	0.82	0.78	0.76	0.77	0.72	0.83	0.81	**0.78**	0.96	0.95	0.94	0.94	0.96	0.94	0.94	0.97	**0.95**	0.64	0.46	0.47	0.52	0.56	0.53	0.45	0.51	**0.52**

Table C.1. ***Main Results as Table:*** This corresponds to the main result in *Paper-Fig.3* showing convergence of various RAGGR uder attacks. Values shown are the mean F1-score of the last 5 comm-rounds computed on the test set. It can be clearly seen that all defenses collapse, at least for some of the attacks, whereas FedSECA is robust to all attacks. Only in the FedISIC dataset, the overall results are suboptimal, yet it does not collapse for any of the attacks, managing to converge.
Red indicates that the training collapsed due to the attack, cutoff at 0.2 for all three datasets.
Orange indicates that the training was impacted severely, cutoff at 0.5(CIFAR10), 0.55(EuroSAT) and 0.4(FedISIC).
Yellow indicates a noticeable drop in accuracy, cutoff at 0.7(CIFAR10), 0.9(EuroSAT) and 0.6(FedISIC).

Client Count	4	8	16	32	64	128	256
FedAvg	0.20±0.01	0.35 ±0.01	0.50 ±0.05	1.37 ±0.02	2.69 ±0.14	5.69 ±0.26	11.52 ±0.51
Krum	0.59±0.02	1.79 ±0.01	6.28 ±0.01	24.49 ±0.26	95.84 ±0.39	383.56 ±0.35	1606.68±3.75
RFA	0.86±0.02	1.48 ±0.01	2.61 ±0.02	5.69 ±0.04	11.54 ±0.21	23.80 ±0.40	48.55 ±0.26
CWTM	1.62±0.02	3.46 ±0.01	8.20 ±0.06	19.76 ±0.05	44.42 ±0.47	99.80 ±0.92	219.33 ±0.71
HuberLoss	1.47±0.01	2.04 ±0.01	2.73 ±0.01	5.81 ±0.01	11.50 ±0.21	23.60 ±0.01	35.64 ±0.46
CClipping	1.21±0.01	2.20 ±0.01	4.04 ±0.00	8.49 ±0.04	16.92 ±0.22	35.09 ±0.52	70.87 ±0.41
CC-RandBucket	0.97±0.01	1.77 ±0.02	2.63 ±0.02	6.51 ±0.03	12.58 ±0.57	26.87 ±0.22	53.06 ±0.13
CC-SeqBucket	1.11±0.01	2.14 ±0.02	2.91 ±0.13	8.39 ±0.06	16.97 ±0.20	34.84 ±0.11	70.39 ±0.08
COPOD-DOS	2.66±0.17	8.70 ±0.05	32.68±0.12	127.52±0.03	503.49±0.06	2036.33±4.08	8663.10±24.31
FL-Detector	1.55±0.37	2.81 ±0.56	5.89 ±1.06	12.51 ±2.58	23.49 ±4.88	46.40 ±9.26	94.14 ±19.73
TIES-Merging	1.89±0.04	3.36 ±0.07	6.25 ±0.10	12.40 ±0.21	24.71 ±0.89	51.46 ±1.65	107.06 ±1.40
FedSECA (Ours)	4.87±0.00	12.30±0.04	37.79±0.18	125.27±0.24	456.32±0.45	1791.72±3.72	8073.35±25.93

Table C.2. ***Time for single aggregation call of each* RAGGR**: The times shown are in seconds it took to complete aggregation operation. We report each aggregator function's mean and standard deviation by repeating the function 5 times. We use the ResNet18 model with a parameter size of 11.2M. Benchmarking is conducted with a CPU with 8 virtual cores node (AMD EPYC 9654) without GPU. We implement all the aggregator functions from scratch using torch tensors and numpy arrays using standard library functions without using any approximations or operation-specific optimizations. It can be observed that FedSECA is suitable for Cross-Silo settings, and it takes very long with a large number of clients but provides superior robustness to various attacks as shown in *Paper-Fig.3*. For Cross-Device applications, we recommend a client selection approach before robust aggregation.

Algorithm 1: **FEDSECA** Gradient Aggregation —*PyTorch pseudocode*

```
## Initialize
K     = number_of_clients
beta  = 0.5
gamma = 0.9
g     = torch.vstack([g1, g2, g3,...,gk])  # all clients gradients
g_tminus1 = g_sent_in_previously           # will be initialized to 0

def FedSECA_aggregator(g, g_tminus1, gamma, beta, K):
    ## Magnitude of Gradients
    magn_g = torch.abs(g)

    ## Sign of Gradients
    sign_g = torch.sign(g)

    ###### CRISE block ========================

    ## Concordance Ratio Computation
    rating_list = []
    for i in range(K):
        omega_c = torch.F.cosine_similarity(sign_g.view(K,-1) , sign_g[i].view(1,-1))
        rho_i = torch.sign(omega_c).mean()
        rho_list.append(rho_i)
    cr_rho = torch.vstack(rho_list).view(K,1)
    cr_rho = torch.clamp(cr_rho, min=0)

    ## Sign Voting
    voted_sign = (sign_g*repute).sum(dim=0).view(1,-1)

    ###### RoCA block ========================

    ## Gradient Clipping
    g_norms = torch.norm(g, dim=1)
    med_norm, _ = torch.median(norms, dim=0)
    norm_clip = torch.minimum(torch.tensor(1.0), med_norm/g_norms)
    g_clipped = g.view(K,-1) * norm_clip.view(K, 1)

    ## Gradient Clamping
    med_magn, _ = torch.median(torch.abs(g_clipped), dim=0)
    g_hat = torch.clamp(g, max=med_magn, min=-med_magn)

    ## Top-K Sparsification
    q = magn_g.quantile(gamma, dim=1)
    g_ddot = g_hat; g_ddot[magn_g<q] = 0.0

    ##  Coordinate-wise Gradient aggregation
    g_mask     = (0<(g_ddot * voted_sign)).int()
    g_selected = g_ddot * g_mask
    g_divisor  = g_mask.sum(dim=0)
    g_aggregate = safe_divide(g_selected.sum(dim=0), g_divisor)

    ## Server Momentum
    g_send = (1-beta)*g_aggregate + beta*g_tminus1
    g_tminus1 = g_send

    return gsend
```

C.1. Concordance Ratio in CRISE

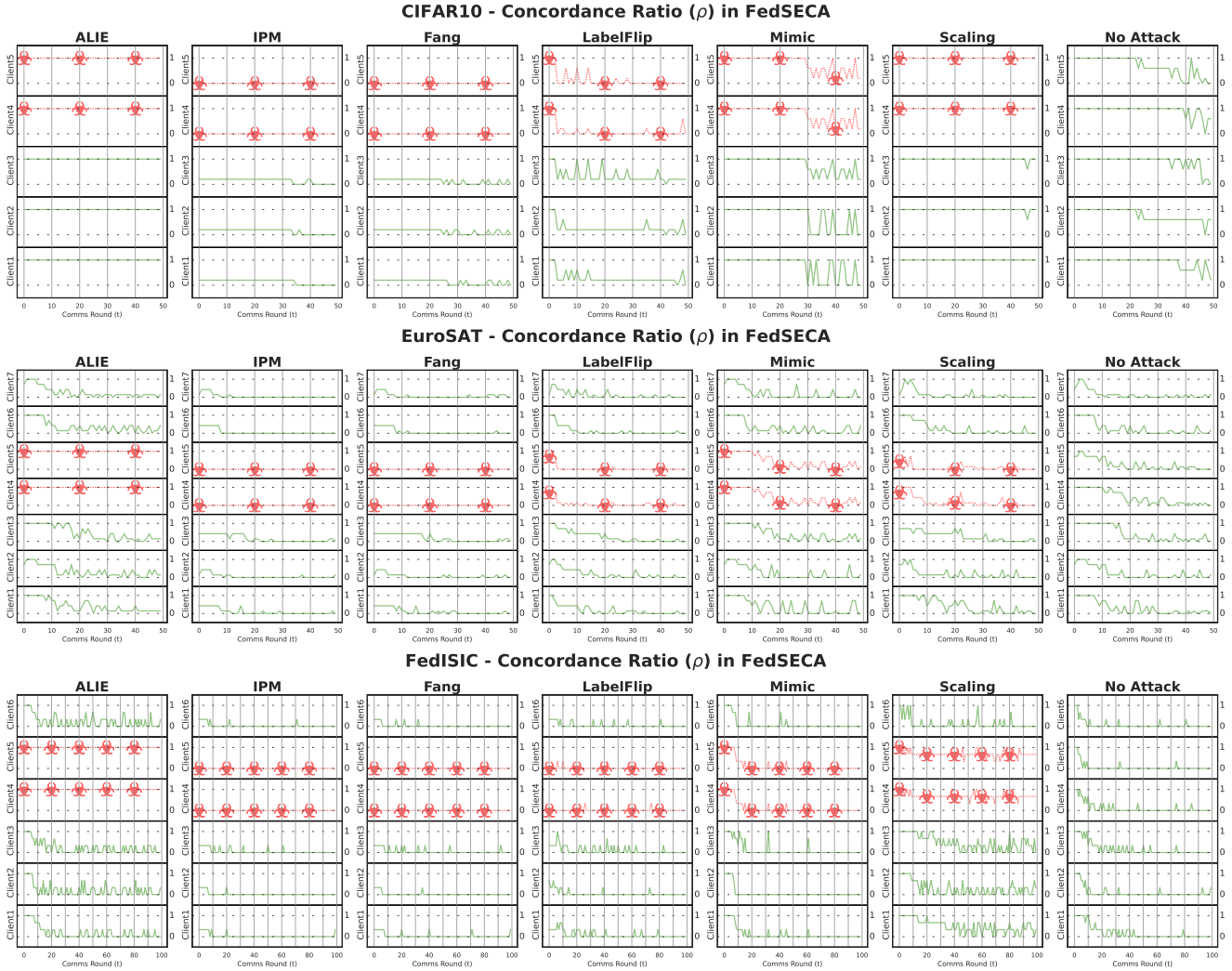

Figure C.4. **Gradients Concordance Ratio (ρ):** Evolution of ρ values for each client throughout the training computed in CRISE. Green lines correspond to honest clients, Red lines correspond to malicious clients. Client-level Concordance Ratio is used solely for voting weightage; once the optimal gradient signs are chosen for each parameter, the gradient filtering follows an egalitarian approach only considering the alignment of the signs but not the ρ. If we filter at the client level solely based on ρ, there is a high chance that an honest client might be ignored in its entirety and malicious clients might be included, especially as seen in the ALIE attack. Although ALIE evades the detection while computing ρ, it will be filtered out in the sparsification step, criticality of the sparsification step in neutralizing ALIE is also shown in *Paper-Tab.1*. Although it can be seen that some honest clients might have lower weightage, especially later in training, this has little effect on convergence.

Let $\mathcal{K}$ represent the set of all clients involved in the federation, $\mathtt{K} = |\mathcal{K}|$ total count of clients in the federation. Let $\mathcal{B}$ represent a set of Byzantine or malicious clients present in the consortium, $\mathtt{B} = |\mathcal{B}|$ total count of Byzantine clients. Let $\mathcal{H}$ represent a set of Honest clients, $\mathtt{H} = |\mathcal{H}| = (\mathtt{K} - \mathtt{B})$ total count of Honest clients. Here $\mathcal{B} \subset \{1, 2, .., k\}$ and $\mathcal{H} \subset \{1, 2, .., k\}$ such that set of all clients is given as $\mathcal{K} = \mathcal{B} \uplus \mathcal{H}$ and total count is $|\mathcal{K}| = |\mathcal{H}| + |\mathcal{B}|$. Let $\mathtt{D}$ represent the number of parameters in the model $\mathcal{M}$.

The rest of the variable names (and Greek letters) henceforth defined are for specific attack methods and robust aggregators. These variables are different from the ones used in FedSECA in main paper unless specified otherwise.

D. Attacks Formulation

$\overset{\triangledown}{\mathbf{w}}$ or $\overset{\triangledown}{\mathbf{g}}$ represents poisoned weights or gradients sent by byzantine clients. Let δ be small noise added to the attack hyperparameters to make the attack vector slightly different from each other to evade any heuristic checks, here $\delta \sim \mathsf{Uniform}(-0.05, 0.05)$

D.1. Crafting Parameter Direction (Fang)

Fang attack[4] shifts the global model parameter in a manner that counters its actual direction of change in the absence of any attacks. $\mathbf{s}$ indicates an increase or decrease (sign) of the global model parameter relative to the previous iteration. Thereby the objective is to perturb the weights in the opposite direction to $\mathbf{s}$

$$\begin{aligned} \overset{\triangledown}{\mathbf{w}}_b^{(t)} &= \widetilde{\mathbf{w}}^{(t-1)} - \lambda_{\mathtt{f}} \cdot \mathbf{s} \qquad \forall\, b \in \mathcal{B} \\ \mathbf{s} &= \mathsf{sgn}(\mu_{\mathbf{w}_\mathcal{H}}^{(t)} - \widetilde{\mathbf{w}}^{(t-1)}) \\ \text{where } \mu_{\mathbf{w}_\mathcal{H}}^{(t)} &= \frac{1}{\mathtt{H}} \sum_{k \in \mathcal{H}} \mathbf{w}_k^{(t)} \end{aligned} \tag{1}$$

Here $\lambda_{\mathtt{f}}$ is the strength of the attack, which can be adjusted as required. Here we use $\lambda = 0.1 + \delta$. sgn is the signum function.

D.2. A Little is Enough (ALIE)

In ALIE[2], the authors demonstrate that by consistently applying minor alterations to numerous parameters, a malicious client can disrupt the convergence of the global model. Given the normal distribution of data, if the attacker has access to a representative subset of the workers, only the corrupted workers' data is needed to estimate the mean and standard deviation of the distribution, thereby facilitating the manipulation of models.

$$\begin{aligned} \overset{\triangledown}{\mathbf{g}}_b^{(t)} &= \mu_{\mathbf{g}_\mathcal{H}}^{(t)} - z_{\mathtt{max}} \cdot \sigma_{\mathbf{g}_\mathcal{H}}^{(t)} \qquad \forall\, b \in \mathcal{B} \\ \text{where} \quad \mu_{\mathbf{g}_\mathcal{H}}^{(t)} &= \frac{1}{\mathtt{H}} \sum_{k \in \mathcal{H}} \mathbf{g}_k^{(t)} \quad , \\ \sigma_{\mathbf{g}_\mathcal{H}}^{(t)} &= \sqrt{\frac{1}{\mathtt{H}} \sum_{k \in \mathcal{H}} (\mathbf{g}_k^{(t)} - \mu_{\mathbf{g}_\mathcal{H}}^{(t)})^2} \end{aligned} \tag{2}$$

Here, $z_{\mathtt{max}}$ corresponds to the factor determining the standard deviation shift of the attack vector from actual honest updates. In literature this is computed based on the number of clients involved, which was ineffective under a cross-silo setting, thus we choose an empirical value based on the impact on no defense setting. We set $z_{\mathtt{max}} = 1.0 + \delta$.

D.3. Inner Product Manipulation (IPM)

The basis for the IPM[12] lies in the observation that as the gradient descent algorithm approaches convergence, the gradient $\mathbf{g}$ tends toward 0. Consequently, despite the bounded distance between the robust estimator and the true mean, it remains feasible to manipulate their inner product to yield a negative value.

$$\langle\, \nabla L(\mathbf{w}), \textsc{Aggr}(\{\mathbf{g}_k : k \in \mathcal{K}\})\, \rangle \geq 0 \tag{3}$$

In gradient descent algorithms, convergence in the loss is guaranteed only if the inner product between the true gradient $\nabla L(\mathbf{w})$ and the robust estimator remains non-negative as shown 3. Hence the potential attack vector can be formed as below:

$$\begin{aligned} \overset{\triangledown}{\mathbf{g}}_b^{(t)} &= -\epsilon \cdot \mu_{\mathbf{g}_\mathcal{H}}^{(t)} \qquad \forall\, b \in \mathcal{B} \\ \text{where} \quad \mu_{\mathbf{g}_\mathcal{H}}^{(t)} &= \frac{1}{\mathtt{H}} \sum_{k \in \mathcal{H}} \mathbf{g}_k^{(t)} \end{aligned} \tag{4}$$

Here in IPM, ϵ is the strength of the attack, which can be adjusted to increase or decrease attack intensity while remaining undetected. Here we use $\epsilon = 1.3 + \delta$.

D.4. Mimic

The objective of the Mimic attack proposed in [6] is to maximize the data imbalance perceived by aggregators, especially middle-seeking approaches, even in balanced datasets. This works by overemphasizing a specific set of clients in margins by mimicking their updates, thereby forming a majority. Because of this, clients with rich gradients will be excluded. The severity of the attack will increase with the increase in heterogeneity among actual distribution and type of defense mechanism used. For identifying the client to mimic $k^\star$, we compute the direction $\mathbf{z}$ of maximum variance of the honest workers' gradients. The client to mimic during the warmup steps are determined as follows

$$\begin{aligned}
\overset{\triangledown}{\mathbf{g}}_b^{(t+1)} &= \mathbf{g}_{k^\star}^{(t+1)} \\
k^\star &= \underset{k\in\mathcal{H}}{\operatorname{argmax}} \left[\mathbf{z}^{(t+1)^\top}\mathbf{g}_k^{(t+1)}\right] \qquad \forall\, b \in \mathcal{B} \\
\mathbf{z}^{(t+1)} &= \underset{\|\mathbf{z}\|=1}{\arg\max} \Bigg[\mathbf{z}^{(t)^\top}\Big(\sum_{t\in\mathcal{T}}\sum_{k\in\mathcal{H}}(\mathbf{g}_k^{(t)} - \mu_{\mathbf{g}_\mathcal{H}}^{(t)})\times \\
&\qquad\qquad (\mathbf{g}_k^{(t)} - \mu_{\mathbf{g}_\mathcal{H}}^{(t)})^\top\Big)\mathbf{z}^{(t)}\Bigg] \\
&\text{where} \quad \mu_{\mathbf{g}_\mathcal{H}}^{(t)} = \frac{1}{|\mathcal{H}|\,|\mathcal{T}|}\sum_{k\in\mathcal{H}}\sum_{t\in\mathcal{T}}\mathbf{g}_k^{(t)}
\end{aligned} \tag{5}$$

the computation of $\mathbf{z}$ can be approximated as below,

$$\begin{aligned}
\mathbf{z}^{(t+1)} &\approx \frac{t}{1+t}\mathbf{z}^{(t)} + \\
&\qquad \frac{1}{1+t}\sum_{k\in\mathcal{H}}(\mathbf{g}_k^{(t+1)} - \mu_{\mathbf{g}_\mathcal{H}}^{(t+1)})(\mathbf{g}_k^{(t+1)} - \mu_{\mathbf{g}_\mathcal{H}}^{(t+1)})^\top\mathbf{z}^{(t)} \\
&\text{where} \quad \mu_{\mathbf{g}_\mathcal{H}}^{(t+1)} = \frac{t}{1+t}\mu_{\mathbf{g}_\mathcal{H}}^{(t)} + \frac{1}{(1+t)|\mathcal{H}|}\sum_{k\in\mathcal{H}}\mathbf{g}_k^{(t+1)}
\end{aligned} \tag{6}$$

After the warmup steps, the mimicked client is kept constant $k^\star$ obtained at t for the rest of the training. $\mathbf{z}$ is initialized randomly at the beginning of the training and recomputed at each step during the warmup steps.

D.5. LabelFlip

Suppose, $\mathcal{C}$ set classes in the classification task, label flipping involves inverting labels such that prediction $\hat{y}_i$ each image maps to the wrong class y_i' instead of true y_i.

$$y_i' = \mathtt{C} - y_i \quad \text{where} \quad y \in \mathcal{C} \quad \text{and} \quad \mathtt{C} = |\mathcal{C}|$$

such that the categorical cross-entropy optimization of a Byzantine client becomes

$$\overset{\triangledown}{\mathbf{w}}_b^{(t)} = \widetilde{\mathbf{w}}^{(t+1)} - \eta\cdot\frac{1}{\mathtt{N}_k}\sum_{i=1}^{\mathtt{N}_k}\mathcal{L}(\mathbf{w}; x_i, y_i') \qquad \forall\, b \in \mathcal{B} \tag{7}$$

This would perturb the gradients, especially ones that are closer to the classification head, in the wrong direction; thus, the majority of the gradients will follow the actual direction, thereby remaining closer to other honest clients, while the critical gradients' specific task at hand will be poisoned.

D.6. Scaling

In Scaling attack, the original gradients are scaled by a factor ε, and this will derail the training. If the scaling is positive, then the direction of gradients will align with the true gradients, but the value will be higher than the regular values. This will cause the gradients to explode and destabilize the training. If the directions are flipped, this becomes equivalent to an IPM attack. In this work, we set the scaling factor as $\varepsilon = 10$.

$$\begin{aligned}
\overset{\triangledown}{\mathbf{g}}_b^{(t)} &= \varepsilon\cdot\mu_{\mathbf{g}_\mathcal{H}}^{(t)} \qquad \forall\, b \in \mathcal{B} \\
&\text{where} \quad \mu_{\mathbf{g}_\mathcal{H}}^{(t)} = \frac{1}{\mathtt{H}}\sum_{k\in\mathcal{H}}\mathbf{g}_k^{(t)}
\end{aligned} \tag{8}$$

D.7. Adaptive Attack

For optimization-based adaptive attacks, we use Min-Max[11] attack, with sign flipping (similar to [4]) where the attack strength is adaptively optimized for the defense employed based on the impact it can make on a given set of updates from honest clients. This version of the attack that we evaluated assumes full access to all benign gradients at the current step and robust aggregation strategy used in the server *(agr-updates)*.

E. Defenses Formulation

A robust aggregator would take all the clients' update vectors and is expected to return an update $\widetilde{\mathbf{g}}$ that is closer to true gradients and devoid of any potential corruptions.

$$\widetilde{\mathbf{g}}^{(t)} = \textsc{RAggr}(\{\mathbf{g}_k^{(t)} : k \in \mathcal{H} \uplus \mathcal{B}\}) \tag{9}$$

For the subsequent section, time step t is skipped for convenience, and all the aggregation is at timestep t gathering t^{th} parameters of local clients unless indicated otherwise.

E.1. Krum

Krum[3] is a middle-seeking aggregator that selects a gradient that is closest to the most client vectors. The method computes the distance score (ς) for each client based on its neighbors, and the client with a very minimal score is taken as a robust aggregate. In the Multi-Krum version, instead of taking a single client's update as aggregate, the average of top-m clients with the minimum distance is returned as aggregate.

$$\begin{aligned}
\widetilde{\mathbf{w}} &= \textsc{Krum}(\{\mathbf{w}_k : k \in \mathcal{K}\}) := \mathbf{w}_{i^\star} \\
i^\star &= \mathsf{argmin}_{i\in\mathcal{K}}(\varsigma_1, \ldots, \varsigma_\mathtt{K}) \\
\varsigma_i &= \sum_{i\to k}\|\mathbf{w}_i - \mathbf{w}_k\|^2 \qquad \forall i, k \in \mathcal{K}
\end{aligned} \tag{10}$$

Notation $i \to k$ indicates the nearest neighbours for i of count of which is $[\mathtt{K} - \mathtt{B} - 2]$. The $i^\star$ is specific client for which condition $\varsigma_{i^\star} \leq \varsigma_i$ holds for all $i \in \mathcal{K}$. Multi-Krum selects m weight vectors $\{\mathbf{w}_1^\star, \ldots, \mathbf{w}_m^\star\}$ that has minimal distance scores and takes the average of these vectors $\frac{1}{m}\sum_{i=0}^{m}\mathbf{w}_i^\star$ as aggregate. In our experiments, we take $m = 1$.

E.2. Coordinatewise Trimmed Mean (CWTM)

CWTM [14] is a middle-seeking approach on individual vector elements across clients, respectively, instead of the whole vector. The median version selects single median values of gradients for each coordinate, discarding the rest of the gradients. For a set of client gradient vectors $\mathbf{g}_k \in \mathbb{R}^\theta$, the coordinate-wise median is defined as follows:

$$\begin{aligned} \widetilde{\mathbf{g}} &= \textsc{CwMed}(\{\mathbf{g}_k : k \in \mathcal{K}\}) := [\widetilde{\mathbf{g}}^0, \widetilde{\mathbf{g}}^2, \ldots, \widetilde{\mathbf{g}}^{\mathtt{D}}] \\ \widetilde{\mathbf{g}}^\varphi &= \mathsf{median}(\{\mathbf{g}_k^\varphi : k \in \mathcal{K}\}) \quad \forall \varphi \in [\mathtt{D}] \end{aligned} \tag{11}$$

For each coordinate φ of $\widetilde{\mathbf{g}}$ is calculated by taking the median of the corresponding coordinates from all vectors $\mathbf{g}_k$, where median represents the median operation on a given set of numbers.

The same formulation can be extended to trimmed mean statistics, where at each coordinate β amount, gradients in both extremities are discarded, and the rest of the gradients are averaged. This is more advantageous than the median as richer gradients from other clients can also contribute. The motivation is that Byzantine clients will have their gradients far away from the honest clients' gradients, *i.e.* in the extremities. Here each coordinates φ is computed as the mean of $\{\mathbf{g}_k^\varphi; k \in \mathcal{U}^\varphi\}$ where $\mathcal{U}^\varphi$ is obtained by removing β fraction of largest and smallest elements from $\{\mathbf{g}_1^\varphi, \mathbf{g}_2^\varphi, ..., \mathbf{g}_{\mathtt{K}}^\varphi\}$ for a given coordinate φ

$$\begin{aligned} \widetilde{\mathbf{g}} &= \textsc{CwTrmean}(\{\mathbf{g}_k : k \in \mathcal{K}\}) := [\widetilde{\mathbf{g}}^1, \ldots, \widetilde{\mathbf{g}}^\theta] \\ \widetilde{\mathbf{g}}^\varphi &= \frac{1}{\mathtt{K} \cdot (1-2\beta)} \sum_{g \in \mathcal{U}^\varphi} g\,, \qquad \forall \varphi \in [\theta] \\ \mathcal{U}^\varphi &= \{\mathbf{a}_i^\varphi : \beta\mathtt{K} < i \leq (1-\beta)\mathtt{K}\} \quad \forall \varphi \in [\theta] \\ \mathbf{a}^\varphi &= \mathsf{sort}[\mathbf{g}_1^\varphi, \mathbf{g}_2^\varphi, \ldots, \mathbf{g}_{\mathtt{K}}^\varphi] \end{aligned} \tag{12}$$

Here $\beta \in [0, \frac{1}{2})$ indicates the proportion of possible Byzantines in collaboration, thus malicious gradients can be discarded. For our experiments, we set $\beta = 0.2$.

E.3. COPOD Distance-based Outlier Suppression (COPOD-DOS)

This method[1] detects and suppresses the outliers among the clients' parameters based on their pairwise distance. It uses both cosine similarity and Euclidean distance measures for identifying the outlier. This approach applies COPOD outlier detection method on the pairwise distance matricies $(\mathbf{M}^{\mathbf{S}}, \mathbf{M}^{\mathbf{E}})$ and produces an outlier score $\mathbf{r} = [r_1, \ldots, r_{\mathtt{K}}]$ where $r_i \in (0, \infty)$ for each client. The outlier score is used to create weightage (λ) for each client that sums to 1 by applying $\mathsf{softmax}$ for aggregation.

$$\begin{aligned} \widetilde{\mathbf{w}} &= \textsc{CopodDos}(\{\mathbf{w}_k : k \in \mathcal{K}\}) := \sum_{k \in \mathcal{K}} \lambda_k \mathbf{w}_k \\ [\lambda_0, \ldots, \lambda_k] &= \mathsf{softmax}(\mathbf{r}) \qquad \text{where} \quad \mathbf{r} = \frac{\mathbf{r}^{\mathbf{S}} + \mathbf{r}^{\mathbf{E}}}{2} \\ \mathbf{r}^{\mathbf{S}} &= \mathsf{COPOD}(\mathbf{M}^{\mathbf{S}}) \quad ; \quad \mathbf{M}^{\mathbf{S}} = \left[a_{ij}^{\mathbf{S}}\right] \\ \mathbf{r}^{\mathbf{E}} &= \mathsf{COPOD}(\mathbf{M}^{\mathbf{E}}) \quad ; \quad \mathbf{M}^{\mathbf{E}} = \left[a_{ij}^{\mathbf{E}}\right] \\ a_{ij}^{\mathbf{S}} &= 1 - \frac{\mathbf{w}_i^\top \cdot \mathbf{w}_j}{\|\mathbf{w}_i\| \, \|\mathbf{w}_j\|} \quad i,j \in [\mathtt{K}] \\ a_{ij}^{\mathbf{E}} &= \|\mathbf{w}_i - \mathbf{w}_j\| \quad i,j \in [\mathtt{K}] \end{aligned} \tag{13}$$

E.4. Robust Federated Aggregation - Geometric Median (RFA)

Geometric Median is statically more robust than other methods leveraging the centrality of parameter distribution in clients. But the critical problem is that achieving this minimization problem computationally is tricker and hinders the practicality. RFA[10] method handles this as a minimization problem and utilizes Smoothed Weiszfeild's algorithm to approximate the geometric median $\mathbf{v}$ in a faster and most stable manner.

$$\begin{aligned} \widetilde{\mathbf{w}} &= \textsc{GeoMedian}(\{\mathbf{w}_k : k \in \mathcal{K}\}) \\ &:= \operatorname*{argmin}_{\mathbf{v}} \sum_{k \in \mathcal{K}} \|\mathbf{v} - \mathbf{w}_k\| \\ \mathbf{v}^{(r+1)} &= \frac{\sum_{i=1}^{\mathtt{K}} \beta_i^{(r)} \cdot \mathbf{w}_i}{\sum_{i=1}^{\mathtt{K}} \beta_i^{(r)}}, \quad \forall r \in [1, \ldots, \mathtt{R}] \\ \text{where} \quad \beta_i^{(r)} &= \frac{\alpha_i}{\mathsf{max}[\varepsilon, \|\mathbf{v}^{(r)} - \mathbf{w}_i\|]} \quad , \quad \alpha_i = \frac{1}{\mathtt{K}} \end{aligned} \tag{14}$$

The median is computed iteratively with $\mathtt{R}$ steps. In our experiments, we set the Weiszfeld iterations $\mathtt{R} = 3$.

E.5. Centered Clipping (CClipping)

Centered Clipping [7] involves rescaling the gradient vector if it exceeds a predetermined radius from the reference vector and retaining it in the original scale if it is within the threshold. The underlying principle of this approach is that malicious gradients have to be far away from the reference vector $(\mathbf{v})$, leading to excessive Euclidean distance, which will trigger clipping. This method only scales back to a specific range and, thus does not exclude or suppress updates during aggregation from any clients, hence all diverse updates contribute to the aggregate vector. Clipping is carried out iteratively with $\mathtt{Q}$ steps during each aggreagation round.

This further improves the convergence of training with enhanced stability and robustness while also ensuring convergence guarantees. This work also explores momentum-based gradient updates from honest workers so that the variance of each global aggregation is bounded, hence further reducing the scope of the attack. For our evaluation, we reset the momentum in the optimizer during each global round to make the comparison fair with other approaches. Global aggregation at t-th round can be given as

$$\begin{aligned}
\widetilde{\mathbf{g}}^{(t)} &= \text{CCLIP}_{\mathtt{Q}}(\{\mathbf{g}_k^{(t)} : k \in \mathcal{K}\},\ \widetilde{\mathbf{g}}^{(t+1)}) := \mathbf{v}^{(\mathtt{Q})} \\
\mathbf{v}^{(q)} &= \mathbf{v}^{(q-1)} + \frac{1}{\mathtt{K}} \sum_{k=1}^{\mathtt{K}} \Big((\mathbf{g}_k^{(t)} - \mathbf{v}^{(q-1)}) \cdot \varkappa \Big), \qquad (15) \\
\varkappa &= \min\Big[1, \frac{\tau}{\|\mathbf{g}_k^{(t)} - \mathbf{v}^{(q-1)}\|}\Big] \quad \forall q \in [1, \ldots, \mathtt{Q}]
\end{aligned}$$

for the first clipping iteration gradient from the previous aggregation is used as reference estimate, $\mathbf{v}^0 = \widetilde{\mathbf{g}}^{(t+1)}$. We set clipping iterations as $\mathtt{Q} = 3$ and the clipping radius as $\tau = 100$

E.6. Randomized Bucketing with Centered Clipping (CC-RandBucket)

Randomized bucketing [6] approach splits the clients randomly into $\mathtt{L}$ buckets, with each bucket holding $\mathtt{S}$ client vectors. Each bucket is averaged into a single vector $\mathbf{u}_l$ before applying aggregation methods such as CCLIP. By randomly splitting clients into buckets (a) There is a higher chance of malicious updates grouped in different bins with the majority of regular clients; therefore reducing the influence of malicious clients would be reduced (b) As the clients in buckets are averaged, at least a few of the clients' gradients are mixed and contribute to the aggregate vector even if AGGR follows median approaches. This strategy also reduces the variance of gradients (among $\mathbf{u}_l$) that are passed to the aggregator. Also, because of the mixing of representative gradients, over multiple rounds, the aggregate vector will be closer to the true gradient, especially in the absence of malicious updates and with mild heterogeneity.

$$\begin{aligned}
\widetilde{\mathbf{g}}^{(t)} &= \text{CCRANDB}(\{\mathbf{g}_k^{(t)} : k \in \mathcal{K}\}) \\
&:= \text{CCLIP}_1([\mathbf{u}_1, \ldots, \mathbf{u}_{\mathtt{L}}],\ \widetilde{\mathbf{g}}^{(t+1)}) \\
\mathbf{u}_l &= \frac{1}{\mathtt{S}} \sum_{i=\mathtt{S}\cdot(l-1)+1}^{\min[\mathtt{K},\ \mathtt{S}\cdot l]} \mathsf{shuffle}(\mathbf{g}_1^{(t)}, \ldots, \mathbf{g}_{\mathtt{K}}^{(t)})_i \qquad (16) \\
&\qquad \forall\, l \in [1, \ldots, \mathtt{L}]; \quad \mathtt{L} = \lceil \mathtt{K}/\mathtt{S} \rceil
\end{aligned}$$

For our experiments we set the clipping iterations $\mathtt{Q} = 1$, clipping radius $\tau = 100$ and vectors per bucket $\mathtt{S} = 2$.

E.7. Sequential Bucketing with Centered Clipping (CC-SeqBucket)

Sequential Bucketing [9] takes a sequential aggregation procedure rather than a single shot approach like in CCRANDB. Total buckets is $\mathtt{R}$ with each bucket holding $\mathtt{S}$ client vectors. When Byzantines count is substantially higher, owing to randomness they can form a part in the majority of the bucket affecting the aggregation. For example, in CCLIP knowledge starting reference point (which will be the previous round aggregate) can make the aggregation vulnerable to attack. So CCSEQB provides additional guardrails to fix the above problems, (i) Instead of random bucketing, uses similarity measures and partitions the vectors into buckets ($\mathcal{G}_r$) such that each bucket will include the most dissimilar vectors, therefore Byzantines cannot choose a unanimous direction and still form a majority within buckets. (ii) AGGR is applied to each bucket instead of simple averaging thus neutralizing deceitful updates at the bucket level. (iii) When clipping instead of having the same reference vector for each bucket, use aggregate from the previous bucket $\mathbf{u}^{(r-1)}$ as a reference vector, and hence reference keeps shifting for each bucket, and its randomized, hence attacker cannot tailor anything specific to reference vector.

$$\begin{aligned}
\widetilde{\mathbf{g}} &= \text{CCSEQB}(\{\mathbf{g}_k : k \in \mathcal{K}\}) := \mathbf{u}^{(\mathtt{R})} \\
\mathbf{u}^{(r)} &= \text{CCLIP}_1(\{\mathbf{g} : \forall \mathbf{g} \in \mathcal{G}_r\},\ \mathbf{u}^{r-1}) \\
&\qquad \forall\, r \in [1, \ldots, \mathtt{R}] \quad \mathtt{R} = \lceil \mathtt{K}/\mathtt{S} \rceil \\
\mathcal{G}_r &= \{\mathbf{g} : 1 \le s \le \mathtt{S},\ \text{if } \exists\ \mathbf{g} \sim \mathcal{A}_s\} \\
&\qquad \forall\, r \in [1, \ldots, \mathtt{R}] \\
\mathcal{A}_s &= \big\{\mathsf{sort}_{\mathcal{Y}}(\mathbf{g}_1, \ldots, \mathbf{g}_{\mathtt{K}})_i : \qquad (17) \\
&\qquad [\mathtt{R}\cdot(s-1)+1] \le i \le \min[\mathtt{K},\ \mathtt{R}\cdot s]\big\} \\
&\qquad \forall\, s \in [1, \ldots, \mathtt{S}] \\
\text{where,} \quad \mathcal{Y}(\mathbf{a}) &\triangleq \frac{\mathbf{a} \cdot \widetilde{\mathbf{g}}^{(t+1)}}{\|\mathbf{a}\| \|\widetilde{\mathbf{g}}^{(t+1)}\|}
\end{aligned}$$

Here, $\mathcal{Y}(\mathbf{a})$ function is the score to sort the gradients. We use clipping iterations $\mathtt{Q} = 1$, clipping radius $\tau = 100$, vectors per bucket $\mathtt{S} = 2$.

E.8. TIES-Merging with Robustness

TIES-Merging [13] was introduced in the field of model merging as the solution to prevent performance drop due to information loss caused by interference of parameters. They identify deterioration mainly due to redundant parameter values and conflicting signs among the important parameters. The proposed solution does sign election based

on the magnitude of the task vectors. The task vector is the difference between the initial model vector and the trained model vector, which is very much equivalent to gradients computed in a single update round. When adapting it as a defense against model poisoning, we cannot use the magnitude-based election approach since malicious clients can scale attack gradients, making it dominant. Thus, we modified the TIES-Merging not to use magnitude but instead to give equal weight to all gradients and make it conducive to FL, which we define as R-TIESMERGE. We use this modified method for comparison with other RAGGR In our proposed method, we compute the weighting explicitly using the Concordance Ratio, which differs from the TIES-Merging.

$$\begin{aligned}\widetilde{\mathbf{g}} &= \text{R-TIESMERGE}(\{\mathbf{g}_k : k \in \mathcal{K}\}) := [\widetilde{\mathbf{g}}^1, \ldots, \widetilde{\mathbf{g}}^{\mathtt{D}}] \\ \widetilde{\mathbf{g}}^{\varphi} &= \frac{\sum_{k=0}^{\mathtt{K}} \delta_k^{\varphi} \cdot \mathbf{v}_k^{\varphi}}{\sum_{k=0}^{\mathtt{K}} \delta_k^{\varphi}} \qquad \text{where} \quad \delta_k^{\varphi} = I\left(\mathbf{u}^{\varphi} \cdot \mathbf{v}_k^{\varphi} > 0\right)\end{aligned} \tag{18}$$

Here $\delta \in \{0, 1\}$ is an indicator variable defined to represent the alignment of signs. $\mathbf{u}$ and $\mathbf{v}$ are defined as follows,

$$\begin{aligned}\mathbf{u} &= [\xi^1, \ldots, \xi^{\mathtt{D}}] = \mathsf{sgn}\Big[\sum_k \mathsf{sgn}(\mathbf{g}_k)\Big] \qquad k \in \mathcal{K} \\ \mathbf{v}_k &= \mathsf{sparsify}(\mathbf{g}_k, \gamma_{\mathtt{t}}) := [v^{\varphi} : \varphi = 1 \ldots \mathtt{D}] \\ &\text{where} \quad v^{\varphi} = \begin{cases} \mathbf{g}_k^{\varphi} & \text{if } \mathsf{abs}(\mathbf{g}_k^{\varphi}) > \mathsf{quantile}(\mathbf{g}_k, \gamma_{\mathtt{t}}) \\ 0 & \text{otherwise} \end{cases}\end{aligned} \tag{19}$$

Here $\gamma_{\mathtt{t}}$ is the sparsification factor which is set at 0.9, and vector $\mathbf{u}$ represents the elect sign for each parameter in the model.

E.9. Huber Loss-Based Weiszfeld Aggregation

This method minimizes multi-dimensional Huber loss (Φ) in the server aggregator. Actual implementation uses a modified Weizfield algorithm for the approximation indicated in Eq.21. It considers two requirements for good aggregator consistency without attack and robustness under attack through L2-loss minimization and geometric median.

$$\begin{aligned}\widetilde{\mathbf{w}} &= \text{HUBERLOSSAGG}(\{\mathbf{w}_k : k \in \mathcal{K}\}) \\ &:= \operatorname*{argmin}_{\mathbf{v}} \sum_{k \in \mathcal{K}} \Phi_k(\|\mathbf{v} - \mathbf{w}_k\|) \\ &\text{where } \Phi_k = \begin{cases} \frac{1}{2}\mathbf{u}^2 & \text{if } |\mathbf{u}| \le T_i \\ T_k\mathbf{u} - \frac{1}{2}T_k^2 & \text{if } |\mathbf{u}| > T_k \end{cases}\end{aligned} \tag{20}$$

$$\mathbf{c}^{(r+1)} = \frac{\sum_{k=1}^{\mathtt{K}} \min\left(1, \frac{\tau}{\|\mathbf{c}^{(r)} - \mathbf{w}_k\|}\right) \cdot \mathbf{w}_k}{\sum_{k=1}^{\mathtt{K}} \min\left(1, \frac{\tau}{\|\mathbf{c}^{(r)} - \mathbf{w}_k\|}\right)} \tag{21}$$

In this, we set the Huber loss threshold τ as 0.2, and set the maximum number of iterations (r) for Weizfield as 100.

E.10. FL-Detector

FL-Detector tries to explicitly detect malicious clients based on the history of previous updates. It uses GAP statistics on suspicion scores (shown in Eq.22) to identify malicious clients.

$$\begin{aligned}\mathbf{S}^{(t)} &= \{\mathbf{s}_1^{(t)}, \mathbf{s}_2^{(t)}, \ldots, \mathbf{s}_n^{(t)}\} \quad \text{where} \\ \mathbf{s}_i^{(t)} &= \frac{1}{\mathtt{T}} \sum_{r=0}^{N-1} \mathbf{d}_i^{(t-r)} \\ \mathbf{d}_i^{(t)} &= \|\hat{\mathbf{g}}_i^{(t)} - \mathbf{g}_i^{(t)}\|_2 \\ \hat{\mathbf{g}}_i^{(t)} &= \mathbf{g}_i^{(t-1)} + \mathbf{H}^{(t)} \cdot \mathbf{g}_i^{(t)}\end{aligned} \tag{22}$$

Here, $\mathbf{H}$ Hessian vector calculated using L-BFGS[15]. $\mathtt{T}$ is the window size that we set as 10. On suspicion score, we run the *GAP* statistics to obtain the number of clusters. If the cluster count is more than 1, then we assume that there is the possibility of malicious updates. If so, we then apply *K-means* clustering with a cluster size of 2 and take the clients in the cluster with an overall minimum suspicion score.

References

[1] Naif Alkhunaizi, Dmitry Kamzolov, Martin Takáč, and Karthik Nandakumar. Suppressing poisoning attacks on federated learning for medical imaging. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 673–683. Springer, 2022.

[2] Gilad Baruch, Moran Baruch, and Yoav Goldberg. A little is enough: Circumventing defenses for distributed learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[3] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. *Advances in neural information processing systems*, 30, 2017.

[4] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning attacks to {Byzantine-Robust} federated learning. In *29th USENIX security symposium (USENIX Security 20)*, pages 1605–1622, 2020.

[5] Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7):2217–2226, 2019.

[6] Sai Praneeth Karimireddy, Lie He, and Martin Jaggi. Byzantine-robust learning on heterogeneous datasets via bucketing. *arXiv preprint arXiv:2006.09365*, 2020.

[7] Sai Praneeth Karimireddy, Lie He, and Martin Jaggi. Learning from history for byzantine robust optimization. In *International Conference on Machine Learning*, pages 5311–5319. PMLR, 2021.

[8] Jean Ogier du Terrail, Samy-Safwan Ayed, Edwige Cyffers, Felix Grimberg, Chaoyang He, Regis Loeb, Paul Mangold, Tanguy Marchand, Othmane Marfoq, Erum Mushtaq, et al. Flamby: Datasets and benchmarks for cross-silo federated learning in realistic healthcare settings. *Advances in Neural Information Processing Systems*, 35:5315–5334, 2022.

[9] Kerem Özfatura, Emre Özfatura, Alptekin Küpçü, and Deniz Gündüz. Byzantines can also learn from history: Fall of centered clipping in federated learning. *IEEE Transactions on Information Forensics and Security*, 2023.

[10] Krishna Pillutla, Sham M Kakade, and Zaid Harchaoui. Robust aggregation for federated learning. *IEEE Transactions on Signal Processing*, 70:1142–1154, 2022.

[11] Virat Shejwalkar and Amir Houmansadr. Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning. In *NDSS*, 2021.

[12] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Fall of empires: Breaking byzantine-tolerant sgd by inner product manipulation. In *Uncertainty in Artificial Intelligence*, pages 261–270. PMLR, 2020.

[13] Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. Resolving interference when merging models. *arXiv preprint arXiv:2306.01708*, 2023.

[14] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In *International Conference on Machine Learning*, pages 5650–5659. Pmlr, 2018.

[15] Zaixi Zhang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Fldetector: Defending federated learning against model poisoning attacks via detecting malicious clients. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pages 2545–2555, 2022.