Federated Learning: A study on FedAvg and family

Ahmad Subhani Iqbal Muhammad Ibrahim Ahmad Jawwad

Abstract

Federated Learning (FL) offers a privacy-preserving paradigm for collaborative model training across decentralized clients, but its performance degrades significantly under statistical heterogeneity (non-IID data) and system constraints (local updates, partial participation). To systematically study these challenges and mitigation strategies, we developed a modular FL framework in PyTorch. Our empirical study first validates the equivalence between Federated SGD and centralized SGD in an IID setting, before exploring the impact of local epochs and client sampling on FedAvg's accuracy, stability, and client weight divergence under varying degrees of label skew generated via Dirichlet partitioning on CIFAR-10. Building on this baseline, we implement and compare four advanced heterogeneity-mitigation algorithms; FedProx (proximal regularization), SCAFFOLD (control variates), Gradient Harmonization (FedGH), and FedSAM (sharpness-aware local training); against FedAvg under fixed non-IID conditions. Our results provide a unified empirical view of how these distinct algorithmic ideas behave in practice concerning accuracy, convergence speed, stability, and communication/computation cost, highlighting practical trade-offs for designing robust FL systems in heterogeneous environments. Our Github repo can be found here.

1. Introduction

Federated Learning (FL) addresses a fundamental challenge: training a global model from decentralised data. Clients train locally and a server aggregates their updates. This process is undermined when client data distributions differ, a problem known as statistical heterogeneity. This "distribution shift" across clients is the primary obstacle to reliable model performance in real-world FL. Clients often have data that is not representative of the global distribution (e.g., label skew, where one client has mostly "Class A" and another has "Class B"). This causes local models to converge to different optima, a phenomenon called "client drift," which severely degrades the aggregated global model's accuracy.

This work conducts a rigorous, controlled empirical study of federated optimization methods designed to mitigate statistical heterogeneity. Our objectives are:

(1) Quantify the impact of heterogeneity: We first evaluate the performance degradation of vanilla FedAvg under varying degrees of controlled label-distribution skew (simulated via a Dirichlet distribution) to establish a baseline.

(2) Evaluate regularization and optimization methods: We implement and compare FedAvg against FedProx, which adds a local proximal regularization term to limit client drift, and FedSAM, which incorporates Sharpness-Aware Minimization into local training to find flatter, more generalizable minima.

(3) Analyze advanced drift-correction algorithms: We implement and evaluate state-of-the-art methods that explicitly correct for heterogeneity, including SCAFFOLD, which uses control variates to correct client gradient drift, and Gradient Harmonization (FedGH), a server-side technique that resolves conflicting client updates before aggregation.

2. Fed SGD vs Centralized SGD

We verified in a toy experiment that FedSGD exactly replicated the trajectory of centralised SGD. Table 1 shows the results of the experiment. We used a simple setup with 3 clients, 10 rounds, per client sample of 20000 images, lr 0.01. Results show that both follow nearly identical trajectories.

FedSGD averages the individual gradients:

$$\theta_g^{t+1} = \theta_g^t - \eta \sum_{i=1}^{M} \frac{N_i}{N} \nabla L_i(\theta_g^t)$$

which equals $\theta_g^t - \eta \nabla L(\theta_g^t)$ as the global loss is as follows and gradient is a linear operator.

$$L(\theta) = \sum_{i=1}^{M} \frac{N_i}{N} L_i(\theta)$$

Table 1. Comparison of centralized and FedSGD loss.

Round	Central. Loss	FedSGD Loss	Model Diff (L_2)
1	2.282982	2.282982	3.85×10^{-7}
2	2.265723	2.265723	6.21×10^{-7}
3	2.248669	2.248670	8.42×10^{-7}
4	2.231785	2.231785	1.09×10^{-6}
5	2.215039	2.215039	1.30×10^{-6}
6	2.198409	2.198409	1.51×10^{-6}
7	2.181875	2.181875	1.72×10^{-6}
8	2.165401	2.165401	1.94×10^{-6}
9	2.148972	2.148972	2.16×10^{-6}
10	2.132574	2.132573	2.40×10^{-6}

Figure 1. FedAvg Performance and Client Divergence on Cifar-10

3. Implementing FedAvg

In this task we implement the FedAvg algorithm. The first strategy is increasing local work per round by raising the number of local epochs/updates K between synchronizations. The second is client sampling or partial participation by aggregating updates from a random n out of N clients each round (vary n depending on available bandwidth). The goal is to measure performance (global test accuracy over communication rounds) and stability (convergence behaviour / oscillations) for both strategies. We present two primary plots: one comparing different K values, and one comparing different client sampling fractions n/N in Figure 1.

All experiments are conducted on the **CIFAR-10** dataset. We simulate a federated environment with **10 clients**, partitioning the dataset in an **IID** manner such that each client

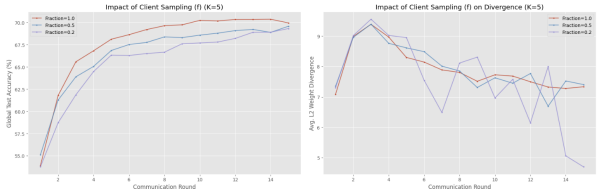

Figure 2. FedAvg Performance and Client Divergence on Cifar-10

Table 2. Final results after 15 rounds for varying local epochs (K).

METRIC	K=1	K=5	K=10	K=20
FINAL TEST ACC (%)	68.09	70.23	70.55	71.39
FINAL AVG. DIV.	2.95	7.46	6.94	6.60
TOTAL TIME (S)	∼191	∼847	∼1609	∼3201

holds 5000 samples. The model is a **SimpleCNN**, trained for a total of **15 communication rounds**. For local client updates, we use a local learning rate of **0.01** and a batch size of **32**.

3.1. Experiment 1

In this experiment, the client fraction f was fixed at 1.0 (full participation), while the number of local epochs K was varied ($K \in \{1, 5, 10, 20\}$).

3.2. Results

The final accuracy, divergence, and total training time for each run are summarized in Table 2. The convergence behavior is visualized in figure 1.

Increasing K from 1 to 20 resulted in a higher final test accuracy, from 68.09% to 71.39%. More significantly, higher values of K ($K \geq 5$) led to much faster convergence in terms of communication rounds. The $K = 1$ run starts at 38.38% accuracy, while the $K = 20$ run starts at 59.44% after just one round. This shows that performing more local work (higher K) is highly beneficial for reducing the number of communication rounds needed in this case.

As expected, higher K causes greater client drift. The initial divergence for $K = 20$ (18.06) is vastly higher than for $K = 1$ (2.21). This is because clients take more local steps, ”drifting” further from the initial global model. The average drift for all values of K are mentoined in Table 2.

We also notes that high K can cause instability. However, in this experiment, it leads to the best final accuracy. The reason could be that that the dataset is **IID (Independently and Identically Distributed)**. The ”instability” and performance degradation associated with high K (i.e., client drift) is a much more severe problem in **Non-IID** settings. In that case, high K would cause the client to overfit to its local data, harming the global model. Since our data is IID, the benefits of more local computation outweigh the costs of client drift.

The trade-off is clear but can vary based on the client's data distribution. $K = 20$ is most efficient in *communication rounds*. $K = 1$ is the fastest in time (∼191s) but the slowest to converge (in rounds) and achieves the lowest accuracy.

Table 3. Final results after 15 rounds for varying client fraction (f).

METRIC	F=1.0	F=0.5	F=0.2
FINAL TEST ACC (%)	69.94	69.57	69.31
FINAL AVG. DIV.	7.33	7.40	4.70
TOTAL TIME (S)	∼826	∼427	∼183

3.3. Experiment 2

In this experiment, the local epochs were fixed at $K = 5$, while the client participation fraction f was varied ($f \in \{1.0, 0.5, 0.2\}$).

3.4. Results

Final results are shown in Table 3, with convergence behavior in Figure 2.

3.5. Analysis

All three runs achieved a similar final accuracy (∼69-70%). However, the training stability was dramatically different. As seen in Figure 2, the $f = 1.0$ run is the smoothest with the quickest convergence. The $f = 0.5$ run is slightly more unstable. The $f = 0.2$ run is the most haphazard and slowest to converge. This experiment highlights the primary trade-off of client sampling.

- $f = 1.0$ **(Full Participation):** Highest stability, but slowest wall-clock time (∼826s) because the server must wait for *all 10 clients* to train and send updates.
- $f = 0.2$ **(Partial Participation):** Fastest wall-clock time (∼183s) because the server only waits for 2 randomly selected clients. This drastically reduces the computation and communication per round.

This shows the practical benefit of partial client participation: it offers a massive speedup in wall-clock time for a very small (in this case) drop in final accuracy.

When $f = 1.0$, the server performs an average over all 10 clients. The divergence, after an initial spike, shows a consistent, predictable decrease (from 9.39 down to 7.33).When $f < 1.0$, the server's aggregated model is an *estimate* of the true global average. Hence we see more variation for it in Figure 2. When $f = 0.2$, the server only averages 2 clients. The resulting update is highly dependent on which 2 clients were chosen. This introduces a large amount of **noise** into the training process. We can see from figure 2 that the average L2 weight divergence for this is highly volatile and jumps around. Though it ends up lower than the other two we expect it remain unstable if the training was allowed to run longer and hence jump back up.

4. Performance under varying Data Heterogeneity Conditions

Given everything we have seen so far, we notice FedAvg providing us with a relatively cost effective and privacy maintaining method to aggregate and update local models. However, it has been well reported that FedAvg is incredibly sensitive to label heterogeneity, and how skewing distributions to be non-IID we notice thatFedAvg begins performing weakly. To evaluate the sensitivity of FedAvg to label heterogeneity, we reused the implementation from Task 2 and systematically varied the non-IIDness of the client data using a Dirichlet-based label partitioning scheme. The key idea is to keep *all* optimization hyperparameters fixed and change only how the CIFAR-10 training set is split across clients, so that any performance differences can be attributed to label skew rather than to optimization noise.

4.1. Implementation and Experimental Setup

4.1.1. GLOBAL SET UP

All experiments were implemented in PyTorch. We consider $M = 10$ federated clients trained on CIFAR-10 (50 000 training images and 10 000 test images). The model on each client is a small CNN named `SimpleCNN`, suitable for CIFAR-10, the same one used in task 2.

The global training protocol is standard FedAvg. In each communication round r:

1. The server maintains a global model with parameters $w^{(r)}$.
2. All $M = 10$ clients participate (full participation, fraction $C = 1.0$). Each client i receives $w^{(r)}$ and performs $K = 5$ local epochs of SGD on its local dataset with:
   - Learning rate $\eta = 0.01$,
   - Momentum 0.9,
   - Weight decay 5×10^{-4},
   - Batch size $B = 32$.

   Concretely, each client instantiates a new `SimpleCNN`, loads $w^{(r)}$, and runs local training via the function `client_update`.
3. After local training, client i returns updated parameters $w_i^{(r+1)}$ to the server.
4. The server aggregates the client models via weighted averaging by local data sizes n_i:

$$w^{(r+1)} = \sum_{i=1}^{M} \frac{n_i}{\sum_{j=1}^{M} n_j} w_i^{(r+1)},$$

We run $R = 50$ communication rounds in all experiments. This was not necessary given trends settle down and plateau around $R = 20$.

4.1.2. DIRICHLET-BASED LABEL PARTITIONING.

To generate heterogeneous client datasets, we partition the CIFAR-10 training set *by class* using a symmetric Dirichlet distribution. Let $y \in \{0, \dots, 9\}$ denote class labels and $M = 10$ the number of clients. For each class c, we sample a vector

$$\pi^{(c)} \sim \mathrm{Dir}_M(\alpha),$$

where $\alpha > 0$ is a concentration parameter controlling the degree of label skew. The component $\pi_m^{(c)}$ specifies the fraction of class-c samples assigned to client m. We then:

1. Collect all indices of class c into a set I_c and shuffle them.
2. Compute counts $counts_m = \lfloor \pi_m^{(c)} \cdot |I_c| \rfloor$ for each client, and adjust the last count to fix rounding so that the total equals $|I_c|$.
3. Assign the appropriate slices of I_c to each client in a round-robin manner.

. We vary α over

$$\alpha \in \{100.0, 50.0, 10.0, 1.0, 0.5, 0.1\},$$

which spans regimes from nearly IID to highly skewed label distributions. For each α, we also log summary statistics of client dataset sizes (minimum, median, maximum, and sum), revealing that:

- For $\alpha = 100.0$, client sizes are almost uniform: $4910/5008.0/5207$ (min/median/max).
- For $\alpha = 1.0$, client sizes become imbalanced: $2367/5182.5/7072$.
- For $\alpha = 0.5$, the imbalance is stronger: $1510/4543.0/10867$.
- For $\alpha = 0.1$, we obtain extreme imbalance: $174/4063.5/18481$.

Thus decreasing α increases both label skew and client-level data imbalance.

4.2. Evaluation and drift metric.

After each aggregation round, we evaluate the global model $w^{(r)}$ on the standard CIFAR-10 test set. We report the top-1 test accuracy and we also measure a parameter-space drift between the global model and the set of locally updated client models. Let w denote the global model parameters and $\{w_i\}_{i=1}^M$ the corresponding client parameters after local training in a given round. We define

$$d_{\ell_2}(w, \{w_i\}) = \frac{1}{M} \sum_{i=1}^{M} \frac{\|w_i - w\|_2}{\|w\|_2}, \tag{1}$$

which is computed by flattening and concatenating all parameters into a single vector per model, and normalizing by the global ℓ_2 norm. This provides a scalar summary of how far, on average, client models move away from the global model in each round.

To isolate the effect of data distribution, we fix *all* hyperparameters across runs and reuse the same initial weights for every α. Concretely, we instantiate `SimpleCNN` once, save its state, and reload this state before starting each α-run. This ensures that differences in trajectories are not due to different random initializations.

4.3. Impact of Label Skew on Accuracy and Convergence

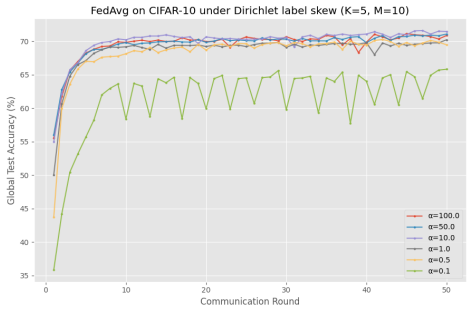

Figure 3. Global CIFAR-10 test accuracy vs. communication rounds for different Dirichlet concentration parameters α under FedAvg ($K = 5$, $M = 10$).

Figure 3 plots the global test accuracy as a function of communication rounds for each α. Numerically, the final accuracies after $R = 50$ rounds are:

- $\alpha = 100.0$: 71.16%
- $\alpha = 50.0$: 71.08%
- $\alpha = 10.0$: 71.46%
- $\alpha = 1.0$: 70.23%
- $\alpha = 0.5$: 69.48%
- $\alpha = 0.1$: 65.83%

Several trends emerge:

1. **Near-IID regimes behave similarly and match centralized performance.** For $\alpha \geq 10$, the curves are almost indistinguishable after the first few rounds. All three settings reach $\approx 65--66\%$ test accuracy by rounds 3–4 (e.g., 65.66% at round 3 for $\alpha = 100.0$ and 65.75% at round 3 for $\alpha = 10.0$) and then gradually climb to roughly 71% by round 50. Differences of $< 1\%$ between these runs fall within the usual stochastic variation of SGD.

2. **Moderate label skew yields a modest but consistent performance drop.** When α is reduced to 1.0 and 0.5, the final accuracy degrades slightly:

   - $\alpha = 1.0$ converges to 70.23%, about 1–1.5 percentage points below the best near-IID run.
   - $\alpha = 0.5$ converges to 69.48%, nearly 2 percentage points below $\alpha = 10.0$.

   Nevertheless, convergence remains relatively smooth: for $\alpha = 0.5$, accuracy reaches 67.01% by round 5 and then fluctuates in the 68–70% range for the remainder of training, stabilizing just under 70%.

3. **Extreme label skew causes a clear accuracy drop and instability.** The most non-IID setting, $\alpha = 0.1$, behaves notably differently:

   - The model starts from 35.86% at round 1 and only surpasses 60% accuracy after several rounds (e.g., 62.01% at round 7, 63.63% at round 9).
   - The curve exhibits pronounced oscillations: after reaching the mid-60s, accuracy repeatedly falls back into the high-50s or low-60s (e.g., 58.39% at round 10, 58.75% at round 13, 59.94% at round 20, 57.76% at round 38).
   - Ultimately, the run converges to 65.83% at round 50, roughly 4–5 percentage points lower than the near-IID runs.

   This behaviour matches the theoretical expectation that FedAvg becomes less stable and less accurate under strong label heterogeneity: local updates are driven by very client-specific objectives, and the global model oscillates as it tries to reconcile these conflicting signals.

4. **Convergence speed vs. final performance.** In all configurations, most of the performance is achieved early: for $\alpha \geq 1$, the model surpasses 65% test accuracy by rounds 4–5 and then refines to $\approx$ 70–71%. In contrast, for $\alpha = 0.1$, convergence is both slower and less reliable; the model takes longer to reach the mid-60s and spends many rounds bouncing between 60% and 65% before finally stabilizing near 66%.

Overall, the experiments confirm the qualitative statement that as label skew increases (smaller α), FedAvg performance decreases. The effect is modest for moderate skew ($\alpha = 1.0, 0.5$) but becomes substantial for the extreme case ($\alpha = 0.1$).

4.4. Client Drift Across Dirichlet Regimes

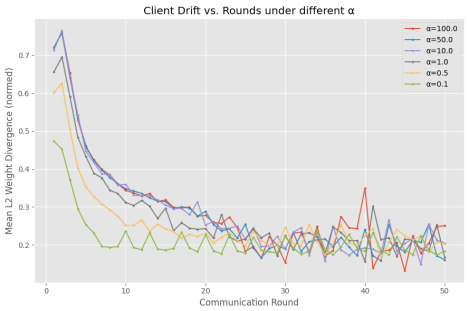

Figure 4. Mean normalized ℓ_2 weight divergence (drift) vs. communication rounds for different Dirichlet concentration parameters α.

Figure 4 shows the ℓ_2 drift as a function of communication rounds. Representative values for selected α are:

- $\alpha = 100.0$ (near-IID):
  - Round 1: drift ≈ 0.7202,
  - Round 10: drift ≈ 0.3433,
  - Round 50: drift ≈ 0.2508.
- $\alpha = 0.5$ (strong skew):
  - Round 1: drift ≈ 0.6014,
  - Round 10: drift ≈ 0.2511,
  - Round 50: drift ≈ 0.2032.
- $\alpha = 0.1$ (extreme skew):
  - Round 1: drift ≈ 0.4730,
  - Round 10: drift ≈ 0.2361,
  - Round 50: drift ≈ 0.1830.

Two observations stand out:

1. **Drift decreases over time for all α.** This indicates that the magnitude of local updates shrinks as training converges, regardless of how data is partitioned. Even in the highly skewed case ($\alpha = 0.1$), the drift metric decays from ≈ 0.47 at round 1 to ≈ 0.18 by round 50.

2. **Higher non-IIDness does not necessarily yield larger drift magnitude.** Somewhat counter-intuitively, the near-IID configuration ($\alpha = 100.0$) shows larger

initial drift (≈ 0.72) than the highly skewed configuration ($\alpha = 0.1$, ≈ 0.47). This can be explained by the interaction between client dataset sizes and step magnitudes:

- For near-IID splits, each client has a large and roughly equal dataset (≈ 5000 samples), enabling substantial local updates per round and hence large parameter changes.
- For $\alpha = 0.1$, some clients have very few samples (as few as 174), so their updates are small in magnitude. The average normalized ℓ_2 deviation can therefore be smaller even though the *directions* of the updates are more conflicting.

Thus, our scalar drift metric primarily captures the *size* of local updates, not necessarily the degree of disagreement among client objectives. Extreme non-IIDness still harms generalization (lower test accuracy and more oscillations), even though the reported drift magnitude is lower.

These results empirically validate the well-known limitation of FedAvg: it is robust to mild heterogeneity but degrades under strong label skew, both in terms of final performance and stability of the optimization trajectory.

5. Mitigating Heterogeneity

Given what we saw with respect to heterogeniety, In this section, we implemented four different federated learning algorithms: FEDPROX, FEDSCAFFOLD, FEDGH, and FEDSAM. We implemented the first three ourselves, while for FEDSAM we used the `sam` library (1).

For each model, we kept `batch_size` = 1024. Since the CPU is our bottleneck, a higher batch size speeds up training. A larger batch size also yields smoother optimization due to lower gradient noise, which can reduce the visible impact of algorithms like FEDGH. We tested all methods against a FEDAVG baseline.

Methodology

For all five models (FEDAVG, FEDPROX, FEDSCAFFOLD, FEDGH, FEDSAM), we used three values of α:

- high skew: $\alpha = 0.1$
- medium skew: $\alpha = 1$
- no skew: $\alpha = 100$

For every model, we used SGD with momentum 0.9 and weight decay 5×10^{-4}. We trained with 10 clients, 5 local epochs per round, and 15 total rounds. We chose 5 local epochs because this allows significant client drift; too few local steps would make improvements from specialized algorithms harder to observe. We selected 15 total rounds empirically; beyond this point the model begins to plateau, and additional rounds risk overfitting.

Hyperparameters

For FEDPROX, we experimented with multiple values of μ. At $\mu = 1$, the regularization was too strong and prevented proper learning, resulting in very low accuracy. At $\mu = 0.1$, drift was reduced to 0.11%; the lowest among all methods; but accuracy suffered significantly. For example, at $\alpha = 0.1$, accuracy dropped to 53%, below the FEDAVG baseline of 64%.

For FEDSAM, we set $\rho = 0.05$, which is the value commonly used in the literature.

Results

Table 4. Final Accuracy and Drift for Federated Learning Methods at Different Dirichlet α Values

Method	Skew Level	Final Accuracy (%)	Final Drift
FedAvg	High	63.49	0.183
	Medium	69.68	0.292
	Low	69.60	0.346
FedProx	High	54.86	0.110
	Medium	65.46	0.147
	Low	66.66	0.161
FedGH	High	61.61	0.213
	Medium	69.22	0.259
	Low	70.28	0.312
FedSAM	High	61.17	0.187
	Medium	69.65	0.267
	Low	70.72	0.304

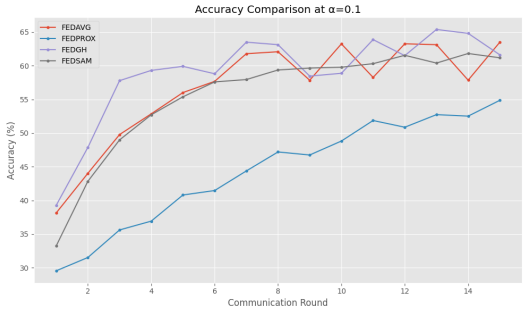

Figure 5. Accuracy comparison across FedAvg, FedProx, and FedSAM at $\alpha = 0.1$ (high skew).

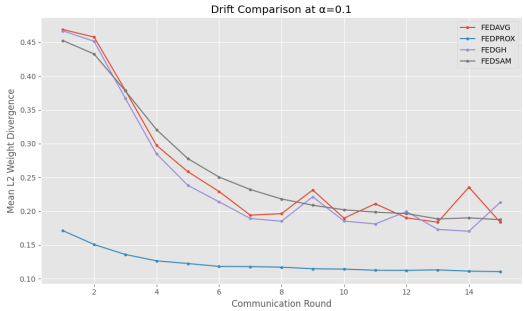

Figure 6. Drift comparison across FedAvg, FedProx, and FedSAM at $\alpha = 0.1$ (high skew).

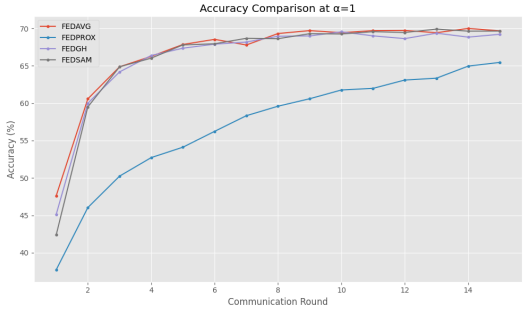

Figure 7. Accuracy comparison across FedAvg, FedProx, and FedSAM at $\alpha = 1$ (medium skew).

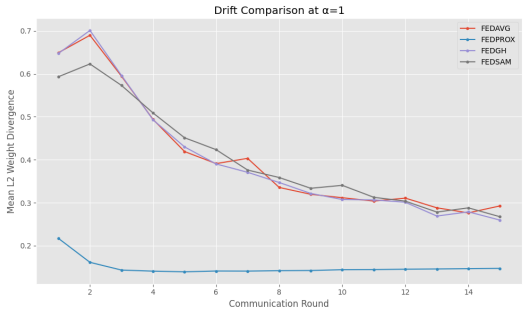

Figure 8. Drift comparison across FedAvg, FedProx, and FedSAM at $\alpha = 1$ (medium skew).

Discussion

Our experiments, especially for high skew, showcase the strengths of each method. We can see FedAvg and FedGH show a lot of oscillations, FedGH shows a strong start - reaching a higher accuracy faster than all other methods. FedProx is stable however underperforming. FedSAM is the most stable throughout. Below, we go through each method and speculate on the results:

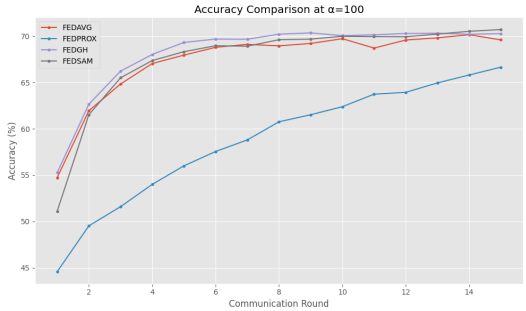

Figure 9. Accuracy comparison across FedAvg, FedProx, and FedSAM at $\alpha = 100$ (low/no skew).

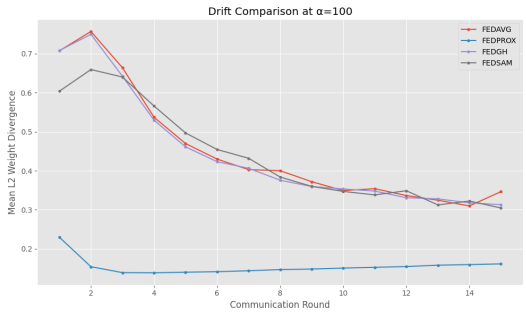

Figure 10. Drift comparison across FedAvg, FedProx, and FedSAM at $\alpha = 100$ (low/no skew).

FedAvg

FedAvg exhibits the highest client drift among all methods. However, reducing client drift does not necessarily improve accuracy. It is particularly unstable under high skew ($\alpha = 0.1$), showing significant oscillations as the model plateaus (see Figure 3). While FedAvg achieves reasonable final accuracy, its susceptibility to client heterogeneity limits its stability.

FedProx

FedProx reduces client drift effectively, especially under high skew, but this does not always translate to higher accuracy. With $\mu = 0.1$, the accuracy remains below FedAvg for high skew ($\alpha = 0.1$). Initially, using $\mu = 1$ led to extremely poor performance (accuracy = 37% for $\alpha = 0.1$), showing that FedProx is highly sensitive to μ. The suboptimal results can be attributed to:

1. Insufficient total communication rounds - FedProx can be seen still improving while all other methods plateau much earlier.

2. A relatively high μ emphasizing drift reduction over global minima convergence, which lowers accuracy under high skew.

Overall, FEDPROX is more stable than FEDAVG across all skew levels, however it is sensitive to the choice of μ and requires more communication rounds.

FEDGH

FEDGH shows faster early convergence in high skew settings. At $\alpha = 0.1$, it reaches 57% accuracy by round 3, while FEDAVG and others remain around 50% or lower. This occurs because FedGH aligns gradients across clients, effectively reducing "noise" in the global update direction. However, FedGH does not lead to more stable training in later rounds: oscillations persist as the model plateaus. This is likely because FedGH focuses on gradient alignment rather than searching for flatter minima like FEDSAM. Our relatively high learning rate (0.01) could also contribute to continued oscillations around the global minimum. Overall, FedGH is only marginally more stable than FEDAVG.

FEDSCAFFOLD

We were unable to successfully run FEDSCAFFOLD, encountering NaN errors and extremely low accuracy in our experiments, with values dropping as low as 4%. Our implementation followed the standard procedure: we maintained a global control vector at the server and local control vectors at each client. During training, the local model parameters were adjusted to account for the global control vector, after which the updated local control vectors were returned to the server for aggregation.

We also attempted gradient clipping to stabilize training, but this did not resolve the issues. Based on our observations, we speculate that FEDSCAFFOLD would not have outperformed FEDSAM. Similar to FEDPROX, its primary goal is to reduce client drift; however, as our results suggest, lower client drift does not necessarily translate to higher accuracy. Additionally, FEDSCAFFOLD doubles communication overhead due to the extra control vectors, further limiting its practicality in real-world scenarios.

FEDSAM

FEDSAM maintains a good balance between accuracy and stability across all skew levels. It consistently reduces client drift while achieving competitive accuracy. Its design, which searches for flatter global minima, makes it inherently more stable than FEDAVG and FedGH, particularly under high skew. We used $\rho = 0.05$ for these experiments, and while no extensive hyperparameter tuning was performed, there is potential for further improvement.

GENERAL OBSERVATIONS

The most significant effects of skew are seen at $\alpha = 0.1$ (high skew). For medium and low skew ($\alpha = 1$ and $\alpha = 100$), most methods exhibit similar performance, except FEDPROX, which generally underperforms. Overall, the most stable and effective method in terms of balancing accuracy and drift minimization is FEDSAM. As for communication and compute, FedSAM would require the most compute, while FedGH has the highest overhead as number of clients increases. Both FedProx and FedAvg are relatively simpler to implement and similar in their communication overhead and compute required - FedProx only adding an extra regularization term to FedProx.

6. Conclusion

In this work, we analyzed how classical and modern federated optimizers behave under realistic system and data constraints. By establishing the equivalence of FedSGD (with one local step and full participation) to centralized SGD, we isolated the effects of local computation, participation rate, and data heterogeneity. Our results show the core trade-offs: increasing local steps helps in IID settings but slows wall-clock time, while reducing participation accelerates training but increases variance—effects that become substantially more pronounced under heterogeneity. With Dirichlet-partitioned CIFAR-10, mild skew preserves IID-level accuracy, whereas severe skew ($\alpha = 0.1$) amplifies instability and slows convergence despite reduced drift, highlighting that drift alone is not a reliable predictor of performance.

Across methods, FedAvg and FedGH exhibit strong oscillations under high skew; FedGH improves quickest early on but fails to stabilize. FedProx is stable but typically underperforms and is highly sensitive to μ. FedScaffold did not train reliably in our setup and, given its overhead, is unlikely to outperform stronger baselines. FedSAM is consistently the most stable and competitive across skew levels, at the cost of additional local computation. The largest performance gaps emerge at high heterogeneity; under moderate skew, differences narrow, with FedSAM remaining the most reliable and FedProx generally lagging.

Overall, our findings show that while FedAvg remains effective in near-IID regimes, robust optimization under real-world heterogeneity requires methods that explicitly counteract client drift and instability. Techniques such as proximal regularization, gradient harmonization, and sharpness-aware updates offer complementary benefits, suggesting that future federated optimizers should integrate these principles to achieve both accuracy and practical efficiency at scale.

References

[1] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware Minimization for Efficiently Improving Generalization. In *Inter-*

national Conference on Learning Representations, 2021. https://openreview.net/forum?id=6Tm1mposlrM