Mitigating Persistent Client Dropout in Asynchronous Decentralized Federated Learning

Ignacy Stępka
istepka@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Nick Gisolfi
ngisolfi@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Kacper Trębacz
ktrebacz@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Artur Dubrawski
awd@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

ABSTRACT

We consider the problem of persistent client dropout in asynchronous Decentralized Federated Learning (DFL). Asynchronicity and decentralization obfuscate information about model updates among federation peers, making recovery from a client dropout difficult. Access to the number of learning epochs, data distributions, and all the information necessary to precisely reconstruct the missing neighbor's loss functions is limited. We show that obvious mitigations do not adequately address the problem and introduce adaptive strategies based on client reconstruction. We show that these strategies can effectively recover some performance loss caused by dropout. Our work focuses on asynchronous DFL with local regularization and differs substantially from that in the existing literature. We evaluate the proposed methods on tabular and image datasets, involve three DFL algorithms, and three data heterogeneity scenarios (iid, non-iid, class-focused non-iid). Our experiments show that the proposed adaptive strategies can be effective in maintaining robustness of federated learning, even if they do not reconstruct the missing client's data precisely. We also discuss the limitations and identify future avenues for tackling the problem of client dropout.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**.

KEYWORDS

Asynchronous Decentralized Federated Learning, Client Dropout, Gradient Inversion Attacks, Model Inversion Attacks

ACM Reference Format:
Ignacy Stępka, Nick Gisolfi, Kacper Trębacz, and Artur Dubrawski. 2025. Mitigating Persistent Client Dropout in Asynchronous Decentralized Federated Learning. In *FedKDD: International Joint Workshop on Federated Learning for Data Mining and Graph Analytics at 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining, August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 15 pages.

FedKDD @ KDD '25, August 3–7, 2025, Toronto, ON, Canada.

1 INTRODUCTION

The most widely used form of decentralized learning is Federated Learning (FL), where data remains distributed across clients. Each client performs local training and shares gradients with a central server, which aggregates them using, e.g., the FedAvg algorithm [7] and broadcasts the updated model back to the clients. In this work, we focus on a decentralized variant of this paradigm, Decentralized Federated Learning (DFL), where no central server exists. Instead, clients interact in a peer-to-peer, asynchronous manner [10].

One of the core challenges in such systems is unequal client participation. This is especially common in practical applications involving mobile devices or unreliable networks. Unequal participation and the presence of stragglers in synchronous settings have been shown to degrade overall system performance [16]. While prior work addresses transient client dropout in centralized setups [9, 11, 13], persistent dropout in asynchronous decentralized environments remains unexplored.

DFL environments are characterized by low observability, unknown local step counts, and high communication asynchronicity, rendering many existing solutions incompatible. To this end, we investigate adaptive mitigation strategies that respond to persistent client dropout and aim to recover performance that would otherwise be lost.

Our findings indicate that simply doing nothing after a dropout significantly reduces federation performance. Similarly, naïve strategies, such as removing the dropped client, lead to poor outcomes, especially under non-iid data distributions. Motivated by these observations, we propose adaptive strategies that reconstruct the missing client's data and instantiate a *virtual client* to continue participating in the optimization process.

We employ gradient inversion [3, 17] and model inversion attacks [5] to approximate the dropped client's data. Empirically, we demonstrate the effectiveness of these adaptive methods across three different DFL algorithms and data heterogeneity scenarios.

To sum up, in this work we make two key contributions. First, we identify the persistent client dropout problem in asynchronous DFL under low-information assumptions and introduce mitigation strategies that do not require modifying the core optimization algorithm. Second, we demonstrate that gradient and model inversion attacks can recover useful approximations of lost client data even when gradients reflect multiple local steps and data points, enabling

virtual clients to rejoin the optimization and significantly improve final performance.

2 RELATED WORK

Our focus is on decentralized learning, where the presence of a central server is explicitly disallowed. Recent advances, such as DFedAvgM, a decentralized FL algorithm employing momentum-based updates [10], have extended FL-style learning to fully decentralized settings. In another line of work, each client optimizes its local objective asynchronously and engages in peer-to-peer communication rounds, exchanging local models and incorporating neighbors' models into their optimization processes. Unlike traditional FL approaches that share gradients with a central server, they adopt a model-sharing paradigm with equivalent communication costs. Notable decentralized optimization approaches include ADMM-based methods [12], AIDE [8], DJAM [1], and FSR [4]. In this work, we use DFedAvgM, DJAM, and FSR as representative baselines to assess the effectiveness of our proposed adaptive strategies across a range of learning paradigms.

Client Dropout and Unequal Participation. The client dropout problem can be viewed as an extreme case of unequal participation in federated systems, and is closely related to the issue of stragglers [16]. Although it has been investigated in centralized FL settings, the assumptions and architectures in those works are fundamentally different from ours. For instance, MimiC [11] proposes a correction-based method where the server adjusts updates to account for absent clients. DReS-FL [9] introduces a secure mechanism for client replacement via Lagrange-coded data sharing, but it relies on specialized cryptographic primitives and restricts model design (e.g., requiring integer polynomial neural networks). Friends-to-Help [13] substitutes dropped client updates using neighboring clients with similar data distributions, but assumes transient dropout and the ability to exchange metadata about local data.

Our setting differs from that background in two important ways: (1) we assume persistent client dropout, where a lost client never returns to the federation, and (2) we operate in an asynchronous, decentralized environment without a central server. These make prior methods ill-suited for direct adoption, as they either depend on central orchestration, require modification of the underlying learning algorithms, or are infeasible in low-information regimes. In contrast, our approach works with arbitrary DFL methods and relies only on observing model updates. Additionally, our methods are compatible with setups involving local regularization/aggregation rather than centralized aggregation.

Data Extraction Attacks. A large body of work has studied data extraction in static centralized learning settings, often by exploiting the loss function of the attacked model [2]. These attacks typically initialize synthetic inputs and optimize them to minimize model loss, often including domain-specific regularizers such as total variation to enhance the realism of reconstructed samples. Haim et al. [5] propose a KKT-based method to reconstruct real training data, under the assumption that the model has converged to a stationary point satisfying optimality conditions. However, enforcing these constraints requires architectural restrictions, which substantially harm the model performance in practice [6].

Gradient inversion attacks offer an alternative by working directly with shared gradients rather than full models. These methods aim to match real and synthetic gradients via second-order optimization, and have been explored in the context of centralized FL [3, 14, 15, 17]. However, most prior work studies them in idealized scenarios where gradients are exchanged frequently and correspond to single batches or epochs. In contrast, DFL settings involve model updates that are the cumulative result of many local steps, making the inversion problem more difficult.

Further complicating matters, DFL algorithms often include private regularization terms, such as neighbor model alignment in DJAM [1] and FSR [4], introducing additional noise and obfuscation in the exchanged updates. Thus, applying gradient or model inversion in this context is more challenging and less explored. Our work takes a first step in this direction by empirically evaluating whether such inversion methods can still extract useful proxy data or biases for dropout mitigation in DFL.

3 BACKGROUND

3.1 System Model

We consider a network of m clients, where each client i holds private data (X_i, Y_i) and trains a local model with parameters θ_i. The collective goal is to minimize the average loss across clients while encouraging alignment between models:

$$\underset{\theta_1,\ldots,\theta_m}{\text{minimize}} \quad \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}_i(\theta_i, X_i, Y_i) \quad \text{s.t.} \quad \mathcal{R}(\theta_1, \ldots, \theta_m) = 0 \tag{1}$$

Here, $\mathcal{L}_i$ is the local data loss for client i, and $\mathcal{R}$ is a regularization term that encourages consistency between clients. Traditionally, $\mathcal{R}$ is defined in parameter space enforcing $\theta_0 + \theta_1 \ldots + \theta_{m-1} \approx 0$ as in DJAM or DFedAvgM. However, it can also operate in function space, as in FSR, where alignment is measured in the function space f_i, where $f_i := \mathcal{F}(\theta_i)$. $\mathcal{F}(\cdot)$ maps parameters to functions, e.g., a neural network forward pass.

Each client independently minimizes its local loss:

$$\underset{\theta_i}{\text{minimize}} \quad \mathcal{L}_i(\theta_i, X_i, Y_i) \tag{2}$$

but the choice of a particular optimization strategy varies by algorithm.

3.2 Algorithms

We consider three decentralized algorithms that represent distinct design choices in DFL: DJAM, FSR, and DFedAvgM.

3.2.1 DJAM. [1] is an asynchronous method that learns personal models through parameter-space regularization. Locally, each client minimizes the following objective:

$$\mathcal{L}_{\text{DJAM}} = \mathcal{L}_d + \|\theta_i^t - \theta_i^{t-1}\|_2 + \frac{1}{2}\sum_{j=1}^{N} g_{ij}\|\theta_i - \theta_{ij}^t\|_2 \tag{3}$$

where $\mathcal{L}_d$ is the standard data loss (e.g., cross-entropy). The second and third loss terms are, respectively, for self and neighbor regularization and are calulated as an L2 norm between model parameters. DJAM requires model design homogeneity across clients to enable the L2-based parameter space regularization.

3.2.2 DFedAvgM. [10] implements FedAvg-style updates in a decentralized and, in our case, asynchronous setting. In each round, a client first averages the latest received models from its neighbors according to the communication graph:

$$\theta_i^{t+1} = \sum_{j=1}^{N} g_{ij}\theta_j^t \tag{4}$$

Then, it performs local updates using the data loss $\mathcal{L}_d$, and finally broadcasts the updated model θ_i^{t+1} to its neighbors.

3.2.3 FSR. [4] supports heterogeneous models by regularizing in function space. Its loss function includes data loss, self-regularization acting as an adaptive learning rate, and neighbor regularization enforcing neighbor similarity in function space:

$$\mathcal{L}_{\text{FSR}} = \mathcal{L}_d + \frac{1-\omega}{\omega}\|f_i^t - f_i^{t-1}\|_2 + \lambda\frac{1}{N}\sum_{j=1}^{N} g_{ij}\|f_i - f_{ij}^t\|_2 \tag{5}$$

This allows FSR to operate under model heterogeneity while adapting learning rates.

3.3 Communication

We assume an asynchronous peer-to-peer setting, where clients communicate via a graph $G \in \mathbb{R}^{m\times m}$. Each entry $g_{ij} > 0$ indicates a communication link between clients i and j. The optimization proceeds in alternating local and communication rounds. In each communication round, a random pair of connected clients exchange their current models. The procedure is outlined in Algorithm 1.

4 METHODS

In this section, we introduce the concept of *client dropout* and present three strategies designed to mitigate its negative effects. A client dropout occurs when a client i becomes permanently unavailable and can no longer participate in the joint optimization process. This implies that i stops responding to peer-to-peer communications and is effectively excluded from further updates to the global objective eq. (1). Furthermore, the local data of the client (X_i, Y_i) are lost, making continued optimization more difficult. In settings with iid data, such dropouts could be tolerable, as every client contributes similar information. However, in more realistic scenarios with class imbalance or non-iid data distributions, the absence of even a single client may significantly harm global performance. We provide empirical support for this observation in Section 5.

4.1 Baseline Strategy: No Reaction

The first baseline comes down to taking no action when a client i drops out. Optimization continues under the assumption that i is still participating. In practice, this means that during any communication round involving i, no model exchange occurs, and the other clients retain and use the last known version of θ_i. While simple, this approach often significantly hurts the performance because that model θ_i will never change and thus convergence (in either parameter or function space) is infeasible.

4.2 Baseline Strategy: Forget the Dropped Client

The second baseline removes the dropped client completely from the federation. This is implemented in two steps: (1) client i is disconnected from the communication graph G, and (2) all clients that previously interacted with i delete their stored copy of θ_i and exclude it from future updates. For algorithms relying on neighbor information (e.g., DJAM, FSR), this also means that in their local objectives they will not consider the i-th neighbor in regularization. With this approach, the largest downside is that in non-iid settings, we simply lose all utility that could have otherwise been provided by the lost client.

4.3 Adaptive Strategies

Unlike the baselines, our adaptive strategies attempt to *reconstruct* the dropped client and restore its presence in the federation. Since the client's data is unavailable, we approximate it using synthetic data derived from the last known model. A new *virtual client* is instantiated using the recovered data and continues participating in training as if it were the original.

4.3.1 Random Images as a Sanity Check. As a simple adaptive technique, we simulate the lost client using its last known model θ_i and completely random data. We generate synthetic inputs $X' \sim \mathcal{U}[0,1]^d$ and assign labels $Y' \sim \mathcal{U}\{1, C\}$, where C is the number of classes. The virtual client then continues local training on (X', Y') as if it were real data.

4.3.2 Gradient Inversion. Gradient inversion aims to reconstruct a synthetic dataset whose gradients closely match those of the lost client. We initialize $X' \sim \mathcal{U}[0,1]^d$ and assign either random or trainable labels Y', then solve the following optimization:

$$\mathcal{L}_{\text{GI}} = d(\nabla W' - \nabla W)^2 + \lambda\mathcal{L}_{\text{prior}} \tag{6}$$

where $\nabla W' = \nabla_\theta \mathcal{L}_d(\theta, X', Y)$ and ∇W is the observed gradient from the lost client. The term $d(\cdot)$ denotes a distance metric (e.g., L2 norm [17] or cosine similarity [3]), and $\mathcal{L}_{\text{prior}}$ includes domain-specific priors (e.g., total variation for image smoothness [15]).

Label Strategies. While labels Y' can be sampled randomly, performance improves when labels are optimized jointly with X'. One approach [14] includes an additional loss term:

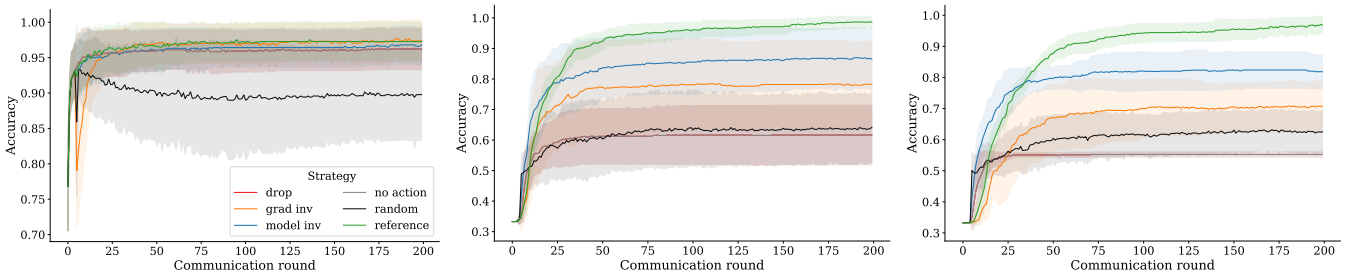
$$\mathcal{L}_{\text{labels}} = \|\theta(X') - Y'\|_2 \quad \text{where} \quad Y' = \text{softmax}(Y')$$

In our experiments, we use both L2 and cosine variants and adopt joint label optimization, which has proven effective in prior work [3, 14, 15].

Limitations. Gradient inversion becomes difficult when gradients reflect updates from multiple local epochs, mini-batches, and data points, denoted by *ENB* (Epochs, Number of samples, Batch size). In asynchronous DFL, *ENB* is often large, making ∇W less informative. For this reason, we also explore model inversion techniques, which do not rely on access to the client's gradients, but rather assume a static model.

4.3.3 Model Inversion. Model inversion assumes that the last available model θ_i is close to a stationary point for its local objective. Although we do not enforce strict optimality, we assume that:

$$\nabla_\theta \mathcal{L}_d(\theta, X, Y) \approx 0 \tag{7}$$



(a) iid (b) non-iid (clusters) (c) non-iid (classes)

Figure 1: Convergence plots for DFedAvgM algorithm on the wine dataset. Shaded regions represent mean ± standard deviation across 10 folds, over 200 communication rounds. A random client is dropped persistently after the 5th round. Colors: model inversion, gradient inversion, reference, random, drop, no action

To recover training data, we generate a synthetic dataset (X', Y') such that this approximate condition holds. The optimization problem becomes:

$$\underset{X'}{\text{minimize}} \quad \mathcal{L}_d(\theta, X', Y') \quad \text{s.t.} \quad \nabla_\theta \mathcal{L}_d(\theta, X', Y') \leq \epsilon \tag{8}$$

This can be solved by any first-order optimizer (e.g., SGD, Adam) performing iterative gradient descent, where the model θ is frozen and the gradients are propagated onto input data X'

$$X'_{t+1} = X'_t - \eta \nabla_{X'_t} \mathcal{L}_d(\theta, X', Y') \tag{9}$$

Here, X' is initialized from a uniform or normal distribution, and Y' from a uniform categorical distribution.

Domain Constraints. For both gradient and model inversion attacks we apply $\mathcal{L}_{\text{prior}} = \sum_{i=1}^{d} \text{ReLU}(x_i - 1) + \text{ReLU}(-x_i)$, a prior loss to ensure inputs remain within a valid range - in our case $x \in [0, 1]^d$. Additionally, we clamp X' to the valid domain after each optimization step.

Remarks. Compared to gradient inversion, model inversion may be less sensitive to large *ENB*, as it does not rely on access to ∇W. Instead, it exploits the structure of θ_i itself. We empirically assess the viability of this approach across different data distributions and DFL algorithms.

5 EXPERIMENTS

We evaluate the effectiveness of our dropout mitigation strategies across different DFL algorithms and data distributions. All experiments are conducted using logistic regression models implemented in PyTorch. We use a fully connected communication graph G, with $g_{ij} = 1$ for all $i \neq j$, and $g_{ii} = 0$ to avoid self-regularization. The federation consists of three clients, and we set the number of peer exchanges per communication round (parameter k in Algorithm 1) to 2. Client dropout occurs persistently at the 5th communication round. From that point onward, the dropped client ceases all participation, and its data becomes inaccessible.

Each experiment uses one of three datasets from the UCI repository[1]: *Wine*, *Iris*, and *Digits*. We apply 10-fold cross-validation and limit training to 200 communication rounds. Early stopping is triggered if all clients converge to the same accuracy on a holdout test set for 10 consecutive rounds. Full details on hyperparameters and implementation can be found in Appendix A.

We study three different data partitioning schemes: (1) **iid**, where data is uniformly distributed among clients; (2) **non-iid (clusters)**, where data is partitioned using k-means clustering; and (3) **non-iid (classes)**, where each client receives data from a distinct subset of classes.

5.1 Empirical Results

We first present convergence results (e.g., Figure 1, full set in Appendix B) that show mean test accuracy across all clients over time. Each curve represents a different dropout-handling strategy: the **Reference** strategy (green) represents the ideal scenario without client dropout; the **Drop** (forget) strategy (red) permanently removes the lost client; the **No Action** strategy (gray) retains a frozen copy of the dropped client's model; the **Random** strategy (black) reinstates the client with random data; the **Gradient Inversion** strategy (orange) reconstructs the client using matched gradients; and the **Model Inversion** strategy (blue) attempts reconstruction based solely on the final model weights.

Below, we first analyze the results for each method separately, and then provide a general summary of findings.

DFedAvgM. For all datasets and data partitions, model inversion consistently achieves the best performance, often matched closely by gradient inversion. The random strategy ranks third, with a negligible performance gap in the iid setting. In contrast, the drop strategy performs well only under iid assumptions, where clients still have reasonably representative data. In non-iid scenarios, removing the client significantly reduces global performance.

FSR. The trend is similar: model inversion typically performs best, with gradient inversion closely following. The random strategy again performs better than baseline strategies but falls short of the inversion-based methods.

DJAM. While model and gradient inversion still outperform the baselines, the performance gap relative to the random strategy is less pronounced. We hypothesize that this is due to the unobserved

[1] https://archive.ics.uci.edu

Dataset	Distribution	No action	Forget	Random	Grad inv	Model inv	Reference
wine	iid	0.96 ± 0.03	0.96 ± 0.03	0.90 ± 0.06	0.97 ± 0.03	0.97 ± 0.03	0.97 ± 0.03
	non-iid (clusters)	0.62 ± 0.10	0.62 ± 0.10	0.64 ± 0.11	0.78 ± 0.14	**0.86 ± 0.10**	0.99 ± 0.02
	non-iid (class)	0.55 ± 0.01	0.55 ± 0.01	0.63 ± 0.07	0.71 ± 0.08	**0.82 ± 0.05**	0.97 ± 0.03
iris	iid	0.90 ± 0.04	0.90 ± 0.04	0.89 ± 0.09	0.92 ± 0.09	**0.95 ±0.04**	0.97 ± 0.04
	non-iid (clusters)	0.64 ± 0.11	0.64 ± 0.11	0.70 ± 0.17	0.79 ± 0.17	**0.87 ± 0.12**	0.94 ± 0.05
	non-iid (class)	0.57 ± 0.04	0.57 ± 0.04	0.57 ± 0.13	0.62 ± 0.10	**0.73 ± 0.08**	0.84 ± 0.04
digits	iid	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.95 ± 0.02	0.94 ± 0.02	0.95 ± 0.01
	non-iid (clusters)	0.75 ± 0.04	0.75 ± 0.04	0.76 ± 0.04	0.84 ± 0.06	**0.86 ± 0.04**	0.95 ± 0.02
	non-iid (class)	0.55 ± 0.02	0.55 ± 0.02	0.63 ± 0.05	0.69 ± 0.04	**0.75 ± 0.04**	0.93 ± 0.02

Table 1: Mean (± std) accuracy of clients on a test set after 200 communication rounds for DFedAvgM algorithm over 10 folds.

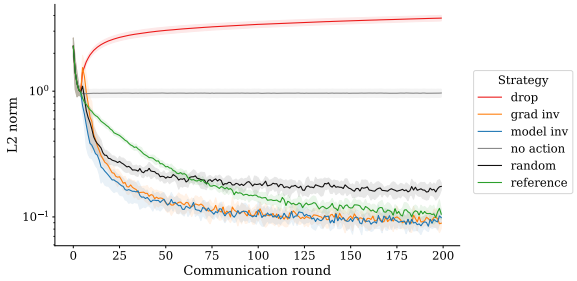


Figure 2: Model similarity over time (L2 norm between model parameters) for DJAM on the Digits dataset with non-iid (class) partitioning. Lower values indicate higher similarity. Note, that the y-axis is in logarithmic scale.

neighbor regularization loss in DJAM. Since we do not have access to this loss component for the dropped client, the data reconstruction becomes less precise, reducing the effectiveness of our adaptive attacks. This is supported by qualitative analysis of the recovered images from the Digits dataset (Appendix D), where samples extracted for DJAM resemble random noise, unlike the more coherent samples seen with DFedAvgM.

Summary. On average, adaptive strategies based on data reconstruction outperform both baselines and the random strategy. The performance gain is most pronounced in non-iid scenarios, highlighting the importance of recovering client-specific information in heterogeneous federations.

5.2 Model Similarity

To further investigate system dynamics post-dropout, we analyze how model similarity evolves across clients during training. Since the goal of DFL is to reach consensus, we expect inter-client model distances (in parameter space) to decrease over time.

In Figure 2, we plot the L2 norm between client models across communication rounds for DJAM on the Digits dataset. The reference strategy shows a steady decrease in distance, indicating convergence. In the drop scenario, similarity drops off as the remaining clients exclude the lost one, leading to model divergence. In the no action case, similarity may stagnate or fluctuate, depending on how aligned the stale model is with the evolving federation.

For all adaptive strategies, we observe a consistent reduction in distance, as the reconstructed (virtual) client re-engages in learning, pulling models toward a shared solution. These trends reinforce the conclusion that our adaptive strategies enable more cohesive behavior of the federation under communication loss or other client failures.

6 CONCLUSIONS

We introduced adaptive mitigation strategies for persistent client dropout in asynchronous decentralized federated learning, leveraging model and gradient inversion techniques to reconstruct lost clients. Our experiments demonstrate that these strategies consistently improve performance across various data heterogeneity conditions (especially in non-iid settings), optimization algorithms, and datasets, when compared to simpler alternatives.

In the future, we would like to expand this investigation to more complex scenarios, such as high-resolution image classification tasks, and conduct a more detailed analysis of the fidelity and privacy implications of the reconstructed data.

Although we conducted some preliminary experiments beyond the main results, this study does not yet systematically examine the impact of the size of the federation, network topology, or optimization hyperparameters. We provide select exploratory findings on these aspects in the Appendix, and we leave a more comprehensive investigation to future work.

ACKNOWLEDGMENTS

This work was partially supported by the NSF (awards 2406231 and 2427948), NIH (awards R01NS124642 and R01NR013912), DARPA (HR00112420329), and the US Army (W911NF-20-D0002).

REFERENCES

[1] Inês Almeida and João Xavier. 2018. DJAM: Distributed Jacobi Asynchronous Method for Learning Personal Models. *IEEE Signal Process. Lett.* 25, 9 (2018), 1389–1392.

[2] Hao Fang, Yixiang Qiu, Hongyao Yu, Wenbo Yu, Jiawei Kong, Baoli Chong, Bin Chen, Xuan Wang, Shu-Tao Xia, and Ke Xu. 2024. Privacy leakage on dnns: A survey of model inversion attacks and defenses. *arXiv preprint arXiv:2402.04013* (2024).

[3] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. 2020. Inverting Gradients – How easy is it to break privacy in federated learning? *Advances in Neural Information Processing Systems* 33 (2020), 16937–16937.

[4] Jack Good. 2024. *Trustworthy Learning using Uncertain Interpretation of Data.* Ph. D. Dissertation. Carnegie Mellon University.

[5] Niv Haim, Gal Vardi, Gilad Yehudai, Ohad Shamir, and Michal Irani. 2022. Reconstructing Training Data From Trained Neural Networks. *Advances in Neural*

Information Processing Systems 35 (Dec. 2022), 22911–22924.
[6] Ziwei Ji and Matus Telgarsky. 2020. Directional convergence and alignment in deep learning. *arXiv preprint arXiv:2006.06657* (2020).
[7] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*.
[8] Sashank J Reddi, Jakub Konečnỳ, Peter Richtárik, Barnabás Póczós, and Alex Smola. 2016. Aide: Fast and communication efficient distributed optimization. *arXiv preprint arXiv:1608.06879* (2016).
[9] Jiawei Shao, Yuchang Sun, Songze Li, and Jun Zhang. 2022. Dres-fl: Dropout-resilient secure federated learning for non-iid clients via secret data sharing. *Advances in Neural Information Processing Systems* 35 (2022), 10533–10545.
[10] Tao Sun, Dongsheng Li, and Bao Wang. 2022. Decentralized federated averaging. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 4 (2022).
[11] Yuchang Sun, Yuyi Mao, and Jun Zhang. 2024. MimiC: Combating Client Dropouts in Federated Learning by Mimicking Central Updates. *IEEE Transactions on Mobile Computing* 23, 7 (July 2024), 7572–7584.
[12] Paul Vanhaesebrouck, Aurélien Bellet, and Marc Tommasi. 2017. Decentralized collaborative learning of personalized models over networks. In *Artificial Intelligence and Statistics*. PMLR, 509–517.
[13] Heqiang Wang and Jie Xu. 2024. Friends to Help: Saving Federated Learning from Client Dropout. In *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 8896–8900.
[14] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. 2019. Beyond Inferring Class Representatives: User-Level Privacy Leakage From Federated Learning. In *IEEE INFOCOM 2019*. IEEE, 2512–2520.
[15] Hongxu Yin, Arun Mallya, Arash Vahdat, Jose M Alvarez, Jan Kautz, and Pavlo Molchanov. 2021. See through Gradients: Image Batch Recovery via GradInversion. *arXiv preprint arXiv:2104.07586* (2021).
[16] Hongbin Zhu, Junqian Kuang, Miao Yang, and Hua Qian. 2022. Client selection with staleness compensation in asynchronous federated learning. *IEEE Transactions on Vehicular Technology* 72, 3 (2022), 4124–4129.
[17] Ligeng Zhu, Zhijian Liu, and Song Han. 2019. Deep Leakage from Gradients. *Advances in Neural Information Processing Systems* 32 (Dec. 2019).

A EXPERIMENT SETUP DETAILS

A.1 Federation Parameters

We run all experiments with three participating clients, and cap each experiment at 200 communication rounds to ensure a uniform communication budget and fair comparison across methods. In every round, two client pairs are randomly selected to exchange their latest models. Regardless of participation in exchanges, each client performs a randomly chosen number of local optimization steps, uniformly sampled from the following categorical interval: $\mathcal{U}\{5, 10\}$. The outline of the global optimization procedure is presented in Algorithm 1.

Algorithm 1 Asynchronous Decentralized Federated Learning framework

Require: No of clients m, initial models $\{\theta_i\}_{i=1}^m$, comm. graph G
Initialize each client i with model θ_i
while not converged **do**
  for all clients i in parallel **do**
    Sample $E_i \sim \mathcal{U}\{5, 10\}$
    for $e = 1$ to E_i **do**
      $\theta_i \leftarrow \arg\min_{\theta_i} \mathcal{L}_i(\theta_i, X_i, Y_i)$
    end for
  end for
  Randomly select k pairs (i, j) such that $g_{ij} \neq 0$
  for all selected pairs (i, j) **do**
    Clients i and j exchange and update their models
  end for
end while
return $\{\theta_i\}_{i=1}^m$

A.2 Handling Data

Each data silo contains at most 200 samples. No datapoint is duplicated across clients. In the **iid** setting, data is randomly assigned to clients. In the **non-iid (clusters)** setting, we perform k-means clustering with k equal to the number of clients and assign one cluster per client. In the **non-iid (classes)** setting, each client is assigned a unique subset of the available classes. If the number of classes exceeds the number of clients, classes are distributed as evenly as possible, but preserving the constraint of each client having a unique subset of classes.

Each client splits its local dataset into training and validation subsets using an 80-20 split. All experiments are performed using 10-fold cross-validation, where 9 folds are used for client training and 1 held-out fold is used as a global test set, which we use to calculate metrics reported in experiments.

A.3 Algorithm Parameters

All three algorithms (FSR, DJAM, DFedAvgM) use the SGD optimizer for local training. For DFedAvgM, we set the learning rate to 0.01 and the momentum for self-regularization to 0.9. For FSR and DJAM, the learning rate is set to 0.1.

In FSR, we approximate the L2 distance in Hilbert space using random samples drawn from a uniform distribution. For each local data batch, we generate 500 synthetic points to estimate the function space regularization terms. While this approximation is admittedly suboptimal (particularly in high-dimensional input spaces) we consider it sufficient for the scope of this study. As FSR was only recently introduced by Good et al. [4], there is currently no established method in the literature for computing neighbor regularization coefficients more accurately. Additional hyperparameters include $\omega = 0.01$ and the neighbor regularization weight $\lambda = 50$.

A.4 Reconstruction Parameters

For all adaptive strategies, we reconstruct 50 synthetic data points per lost client. This number is arbitrarily fixed and does not assume any knowledge about the original client's dataset size or local training steps between model updates.

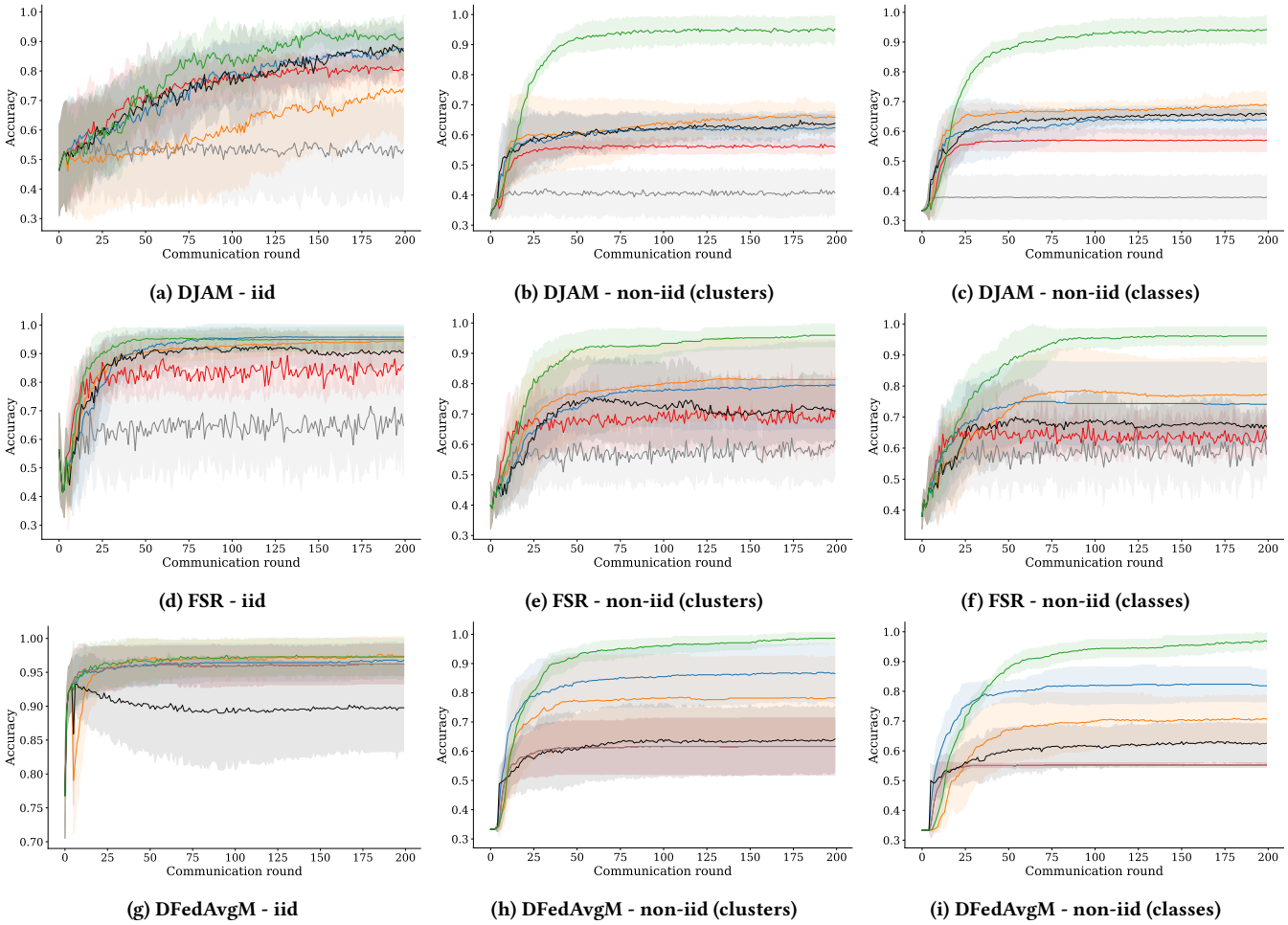
In the *random* baseline, the virtual client is trained for 10 epochs on randomly generated data before being introduced back to the federation. For both *gradient inversion* and *model inversion*, we use a batch size of 16, the Adam optimizer, and include auxiliary losses: Total Variation ($\lambda_{TV} = 0.01$) and Domain Constraint ($\lambda_{domain} = 0.1$).

For *model inversion*, we set the learning rate to 0.01, apply weight decay of 0.01, and train for 1000 epochs. The generated data is class-balanced.

For *gradient inversion*, the learning rate is set to 0.05 with 2000 training epochs. We run both L2 and cosine similarity variants and report results based on the best-performing variant.

B ALL CONVERGENCE PLOTS

In this section we present convergence plots for all combinations of dataset, algorithm and data heterogeneity type. They are grouped by dataset in figures of 9 subplots, where each row has a different algorithm and column, data heterogeneity type. We have plots for wine in Figure 3, iris in Figure 4, and digits in Figure 5.



(a) DJAM - iid (b) DJAM - non-iid (clusters) (c) DJAM - non-iid (classes)

(d) FSR - iid (e) FSR - non-iid (clusters) (f) FSR - non-iid (classes)

(g) DFedAvgM - iid (h) DFedAvgM - non-iid (clusters) (i) DFedAvgM - non-iid (classes)

Figure 3: Convergence plots for DJAM, FSR, and DFedAvgM algorithms on the wine dataset. Shaded regions represent mean ± standard deviation across 10 folds, over 200 communication rounds. A random client is dropped persistently after the 5th round. Colors: model inversion, gradient inversion, reference, random, drop, no action

.

C ALL TABLES

In this section, we present all results in one table (Table 2).

D WHAT DATA IS EXTRACTED

This section presents qualitative examples of images reconstructed using model inversion and gradient inversion attacks on the Digits dataset.

For model inversion (Figures 6 and 8), we show reconstructed samples after 1 epoch (left) and 800 epochs (right) of attack optimization. For gradient inversion (Figures 7 and 9), we present reconstructions at 1 epoch (left) and 1500 epochs (right).

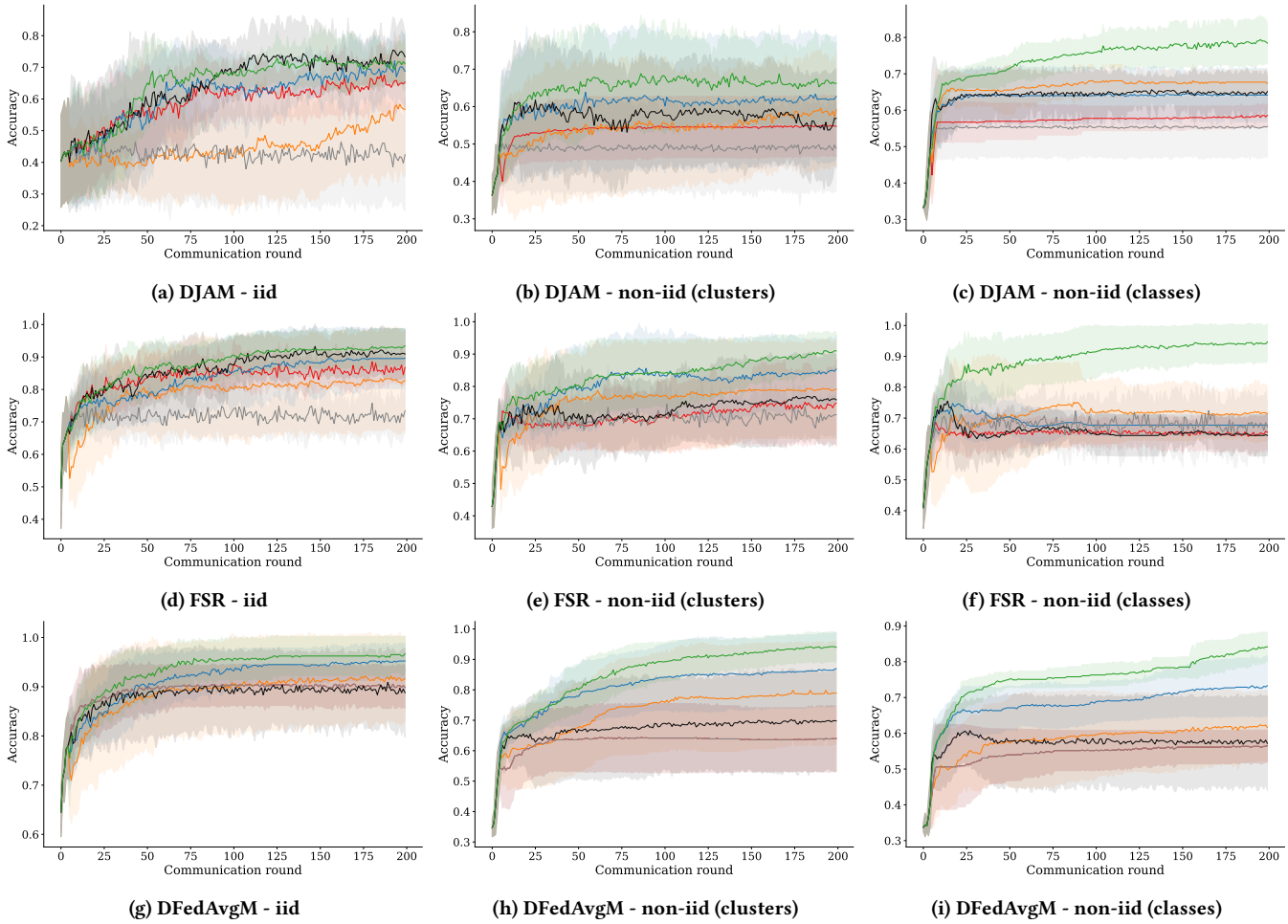
In general, model inversion appears to recover structural patterns resembling noisy or averaged digit shapes. While the reconstructions are imperfect, they are not random and reflect underlying data structure. Gradient inversion yields darker, lower-contrast images with less visible structure, though still non-random.

Based on both this qualitative inspection and the quantitative performance gains shown in Section 5, we hypothesize that the reconstructed images—though degraded—retain useful statistical biases, enabling the virtual client to partially preserve the contribution of the original client lost due to persistent dropout.

E SIMILARITY PLOTS

This section presents model similarity plots corresponding to the experiments and convergence plots shown in Figures 10 to 12 and detailed in Appendices B and C.

A key observation is that for the FSR algorithm, models do not converge in parameter space during optimization. This is not a surprise, as FSR explicitly optimizes for similarity in function space rather than parameter space. Aside from this, the similarity trends across experiments align with our findings in the main text and provide further evidence that the proposed adaptive strategies lead to more cohesive and aligned models across the federation.



(a) DJAM - iid (b) DJAM - non-iid (clusters) (c) DJAM - non-iid (classes)

(d) FSR - iid (e) FSR - non-iid (clusters) (f) FSR - non-iid (classes)

(g) DFedAvgM - iid (h) DFedAvgM - non-iid (clusters) (i) DFedAvgM - non-iid (classes)

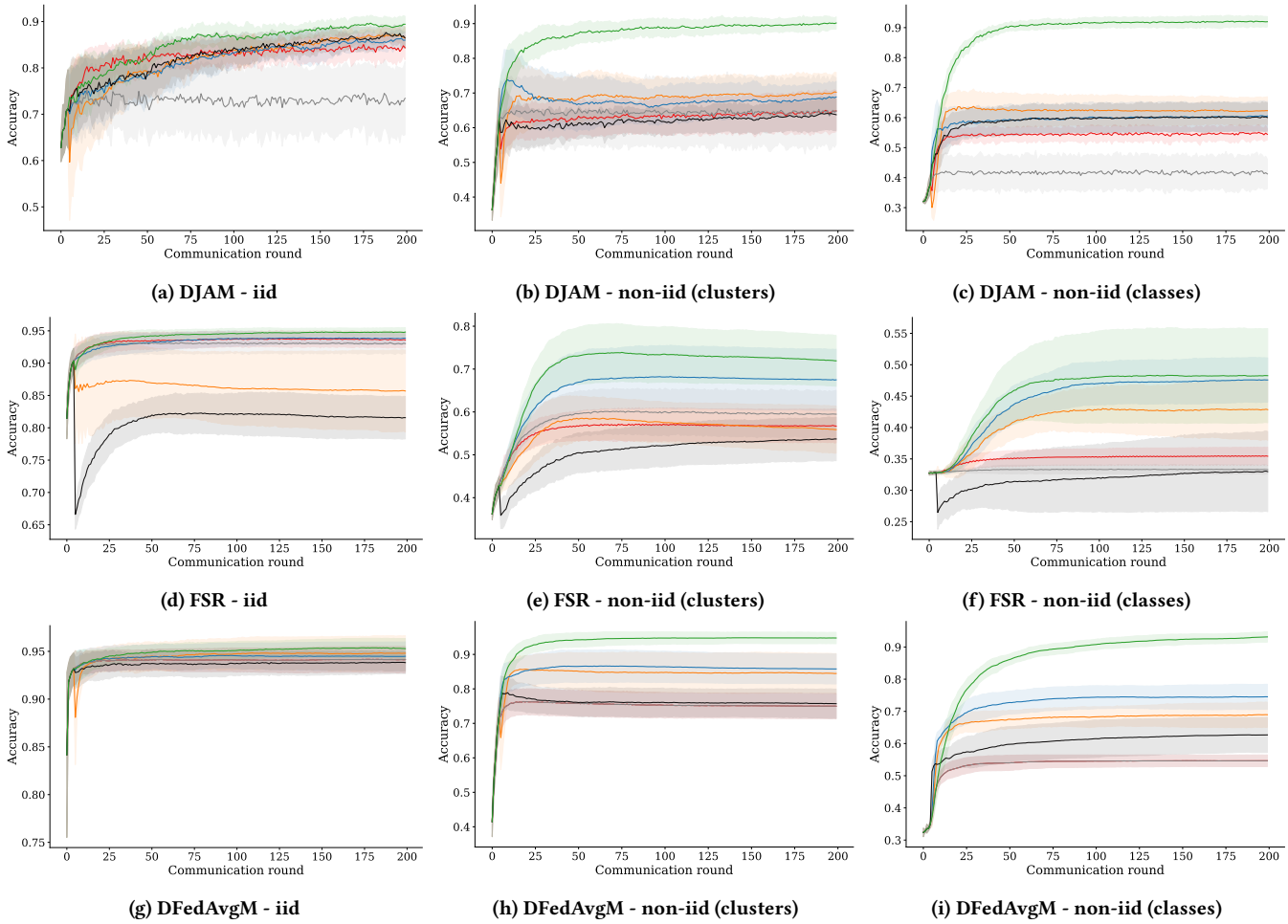
Figure 4: Convergence plots for DJAM, FSR, and DFedAvgM algorithms on the iris dataset. Shaded regions represent mean ± standard deviation across 10 folds, over 200 communication rounds. A random client is dropped persistently after the 5th round. Colors: model inversion, gradient inversion, reference, random, drop, no action

.

F RESULTS WITH NEURAL NETWORKS

This section presents a smaller set of experiments where the underlying model is a neural network instead of logistic regression. In Figure 13, we show results on the digits dataset using the DFedAvgM algorithm with a 3-layer multilayer perceptron (MLP), where each hidden layer has 128 units.

The key takeaway is that performance trends are consistent with those observed for simpler models. Adaptive strategies, particularly model and gradient inversion, remain effective even when using a neural network as the base model.



(a) DJAM - iid (b) DJAM - non-iid (clusters) (c) DJAM - non-iid (classes)

(d) FSR - iid (e) FSR - non-iid (clusters) (f) FSR - non-iid (classes)

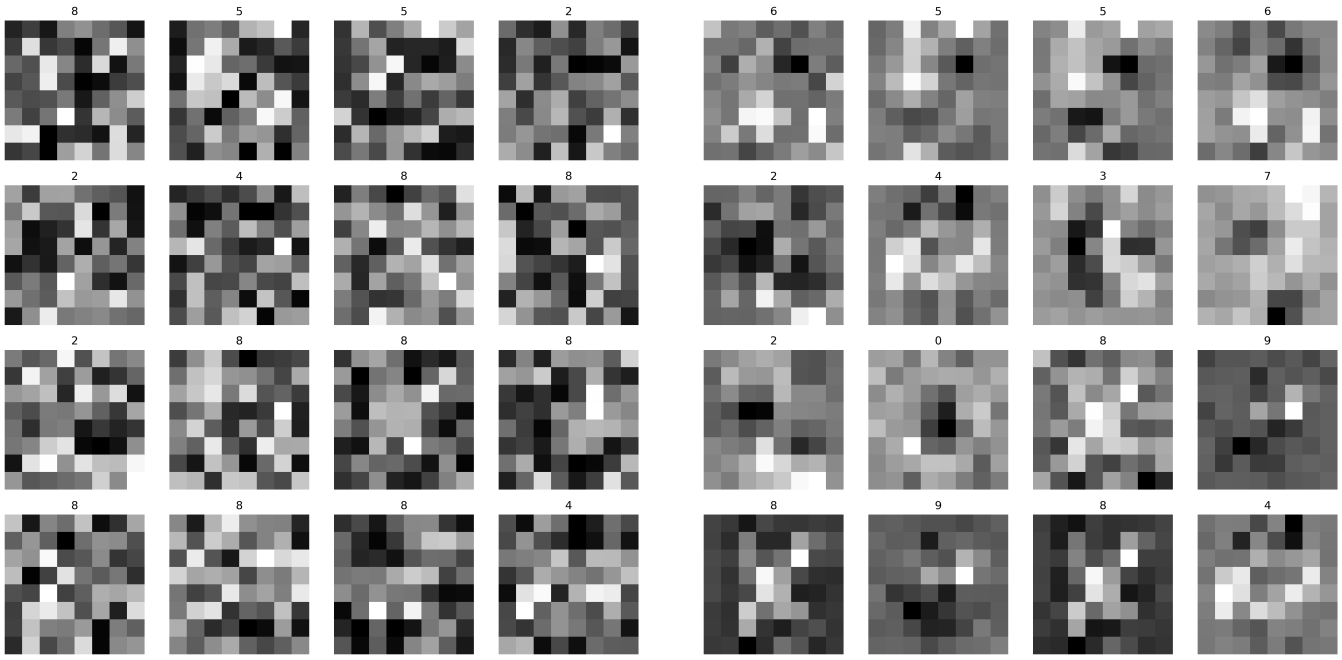
(g) DFedAvgM - iid (h) DFedAvgM - non-iid (clusters) (i) DFedAvgM - non-iid (classes)

Figure 5: Convergence plots for DJAM, FSR, and DFedAvgM algorithms on the digits dataset. Shaded regions represent mean ± standard deviation across 10 folds, over 200 communication rounds. A random client is dropped persistently after the 5th round. Colors: model inversion, gradient inversion, reference, random, drop, no action

.

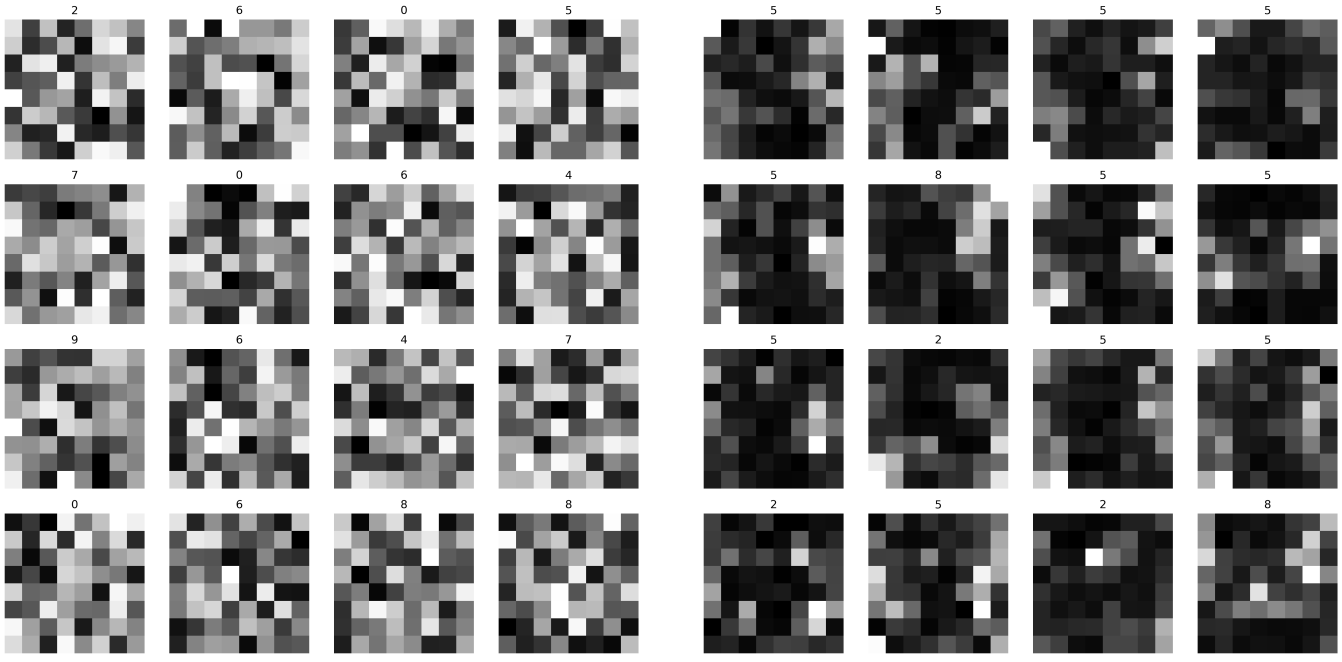
Algorithm	Dataset	Distribution	No action	Forget	Random	Grad inv	Model inv	Reference
DJAM	wine	iid	0.53 ± 0.14	0.80 ± 0.05	0.87 ± 0.08	0.74 ± 0.17	0.88 ± 0.05	0.91 ± 0.08
		non-iid (clusters)	0.41 ± 0.08	0.56 ± 0.02	0.64 ± 0.03	0.66 ± 0.05	0.62 ± 0.05	0.95 ± 0.04
		non-iid (class)	0.38 ± 0.08	0.57 ± 0.04	0.65 ± 0.02	0.69 ± 0.04	0.64 ± 0.05	0.94 ± 0.05
	iris	iid	0.42 ± 0.18	0.65 ± 0.08	0.73 ± 0.08	0.57 ± 0.21	0.69 ± 0.12	0.71 ± 0.08
		non-iid (clusters)	0.49 ± 0.12	0.55 ± 0.08	0.57 ± 0.13	0.59 ± 0.15	0.63 ± 0.16	0.66 ± 0.11
		non-iid (class)	0.56 ± 0.08	0.59 ± 0.03	0.65 ± 0.04	0.67 ± 0.02	0.64 ± 0.07	0.78 ± 0.06
	digits	iid	0.73 ± 0.08	0.84 ± 0.03	0.86 ± 0.02	0.87 ± 0.02	0.86 ± 0.02	0.89 ± 0.02
		non-iid (clusters)	0.65 ± 0.10	0.65 ± 0.06	0.64 ± 0.05	0.70 ± 0.06	0.69 ± 0.05	0.90 ± 0.02
		non-iid (class)	0.41 ± 0.05	0.54 ± 0.02	0.60 ± 0.05	0.62 ± 0.05	0.61 ± 0.05	0.92 ± 0.02
FSR	wine	iid	0.65 ± 0.17	0.86 ± 0.09	0.90 ± 0.05	0.94 ± 0.03	0.96 ± 0.04	0.95 ± 0.04
		non-iid (clusters)	0.61 ± 0.11	0.71 ± 0.12	0.71 ± 0.11	0.81 ± 0.13	0.80 ± 0.14	0.96 ± 0.04
		non-iid (class)	0.58 ± 0.08	0.63 ± 0.06	0.67 ± 0.05	0.77 ± 0.12	0.74 ± 0.13	0.96 ± 0.03
	iris	iid	0.73 ± 0.07	0.87 ± 0.05	0.91 ± 0.07	0.83 ± 0.15	0.90 ± 0.09	0.93 ± 0.05
		non-iid (clusters)	0.71 ± 0.07	0.75 ± 0.13	0.76 ± 0.14	0.79 ± 0.15	0.85 ± 0.11	0.91 ± 0.06
		non-iid (class)	0.67 ± 0.08	0.65 ± 0.06	0.64 ± 0.07	0.71 ± 0.08	0.68 ± 0.03	0.95 ± 0.06
	digits	iid	0.93 ± 0.02	0.94 ± 0.01	0.82 ± 0.03	0.86 ± 0.06	0.94 ± 0.01	0.95 ± 0.01
		non-iid (clusters)	0.59 ± 0.06	0.57 ± 0.04	0.54 ± 0.05	0.56 ± 0.05	0.67 ± 0.07	0.72 ± 0.06
		non-iid (class)	0.33 ± 0.01	0.35 ± 0.01	0.33 ± 0.06	0.43 ± 0.05	0.48 ± 0.04	0.48 ± 0.07
DFedAvgM	wine	iid	0.96 ± 0.03	0.96 ± 0.03	0.90 ± 0.06	0.97 ± 0.03	0.97 ± 0.03	0.97 ± 0.03
		non-iid (clusters)	0.62 ± 0.10	0.62 ± 0.10	0.64 ± 0.11	0.78 ± 0.14	0.86 ± 0.10	0.99 ± 0.02
		non-iid (class)	0.55 ± 0.01	0.55 ± 0.01	0.63 ± 0.07	0.71 ± 0.08	0.82 ± 0.05	0.97 ± 0.03
	iris	iid	0.90 ± 0.04	0.90 ± 0.04	0.89 ± 0.09	0.92 ± 0.09	0.95 ± 0.04	0.97 ± 0.04
		non-iid (clusters)	0.64 ± 0.11	0.64 ± 0.11	0.70 ± 0.17	0.79 ± 0.17	0.87 ± 0.12	0.94 ± 0.05
		non-iid (class)	0.57 ± 0.04	0.57 ± 0.04	0.57 ± 0.13	0.62 ± 0.10	0.73 ± 0.08	0.84 ± 0.04
	digits	iid	0.94 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.95 ± 0.02	0.94 ± 0.02	0.95 ± 0.01
		non-iid (clusters)	0.75 ± 0.04	0.75 ± 0.04	0.76 ± 0.04	0.84 ± 0.06	0.86 ± 0.04	0.95 ± 0.02
		non-iid (class)	0.55 ± 0.02	0.55 ± 0.02	0.63 ± 0.05	0.69 ± 0.04	0.75 ± 0.04	0.93 ± 0.02

Table 2: Mean accuracy of clients on a holdout test set after 200 communication rounds for all algorithms.



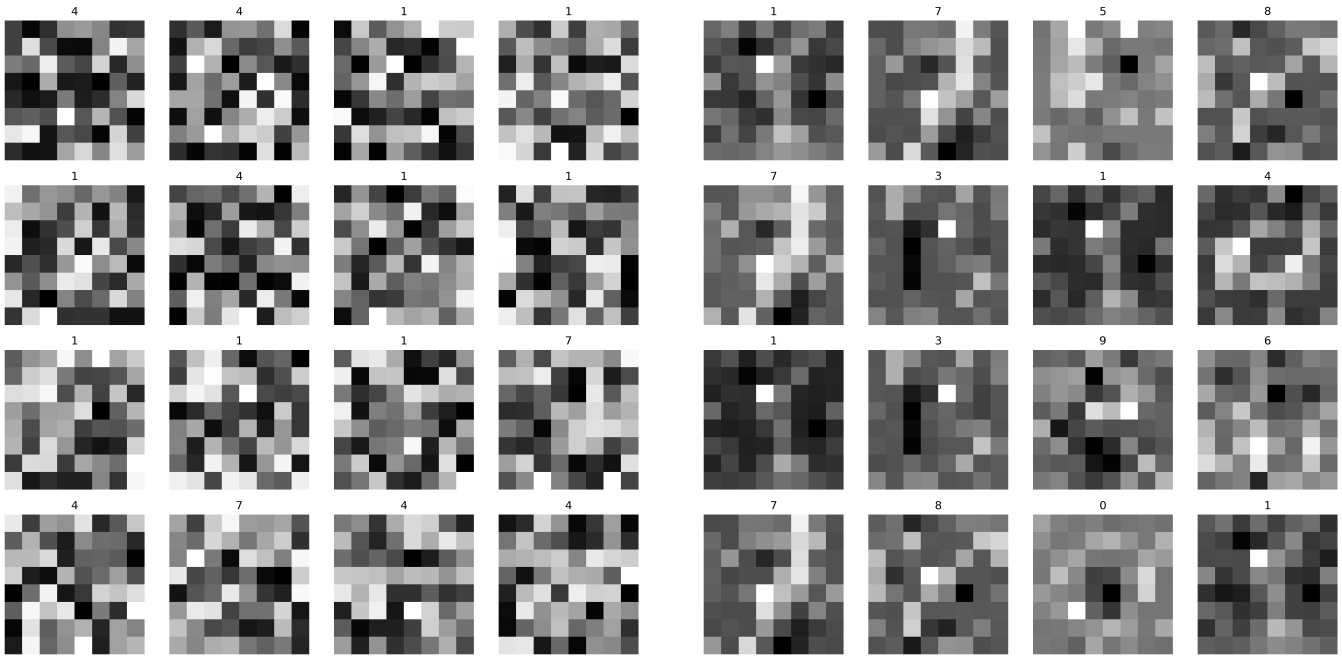
(a) After 1 epoch **(b) After 800 epochs**

Figure 6: Reconstructed images using model inversion on the Digits dataset (DFedAvgM, non-iid (class)).



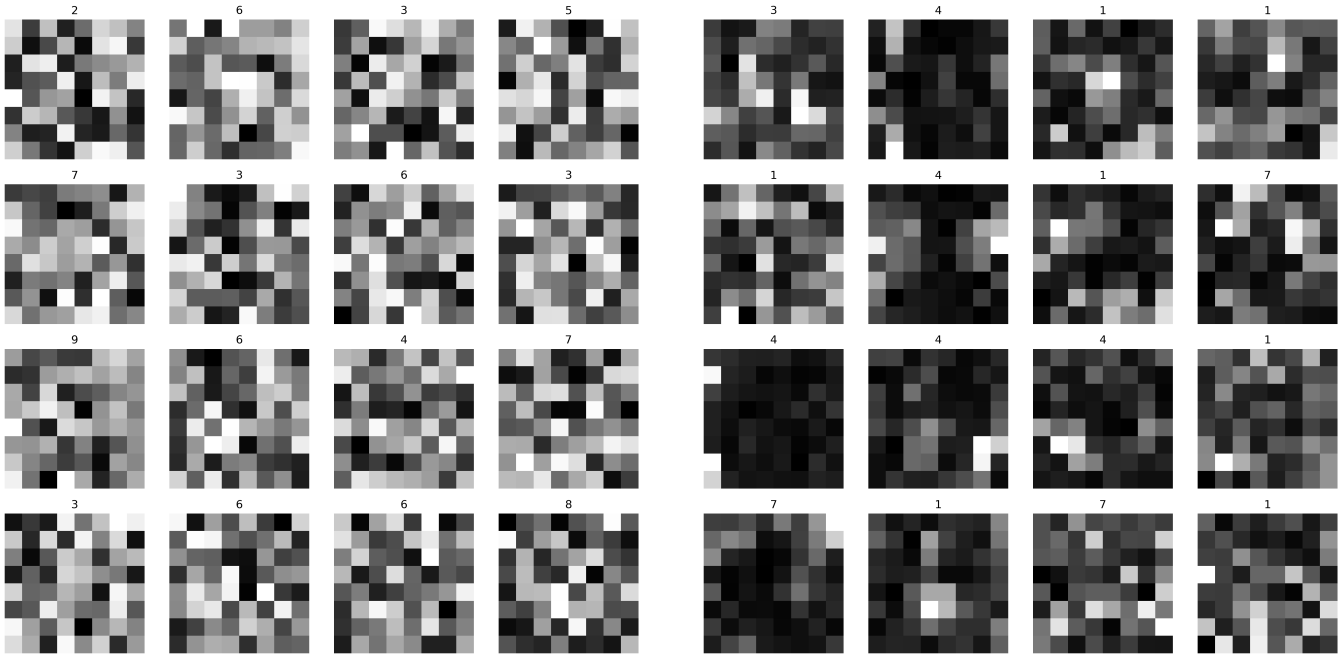
(a) After 1 epoch **(b) After 1500 epochs**

Figure 7: Reconstructed images using gradient inversion on the Digits dataset (DFedAvgM, non-iid (class)).



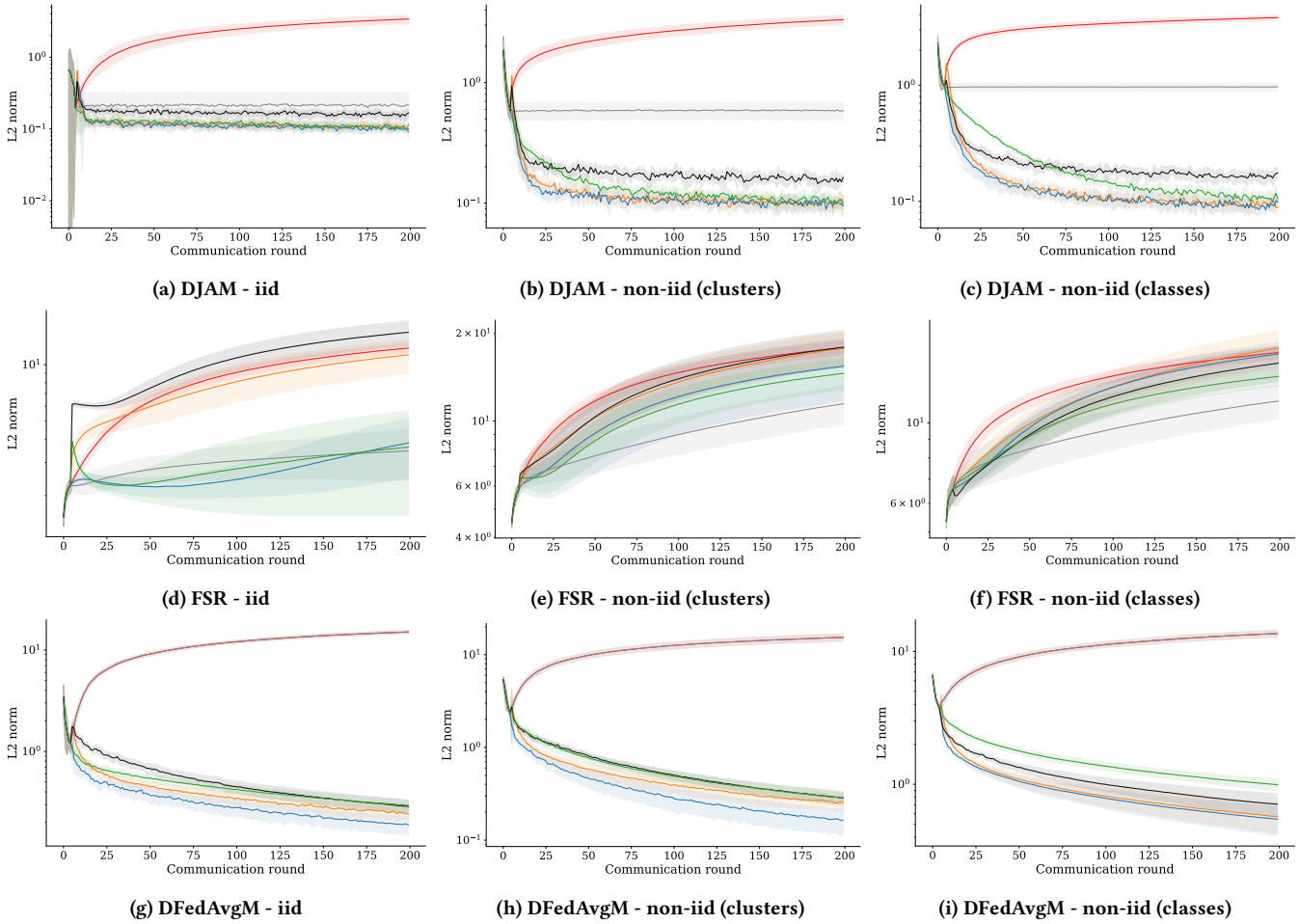
(a) After 1 epoch **(b) After 800 epochs**

Figure 8: Reconstructed images using model inversion on the Digits dataset (DJAM, non-iid (class)).



(a) After 1 epoch **(b) After 1500 epochs**

Figure 9: Reconstructed images using gradient inversion on the Digits dataset (DJAM, non-iid (class)).



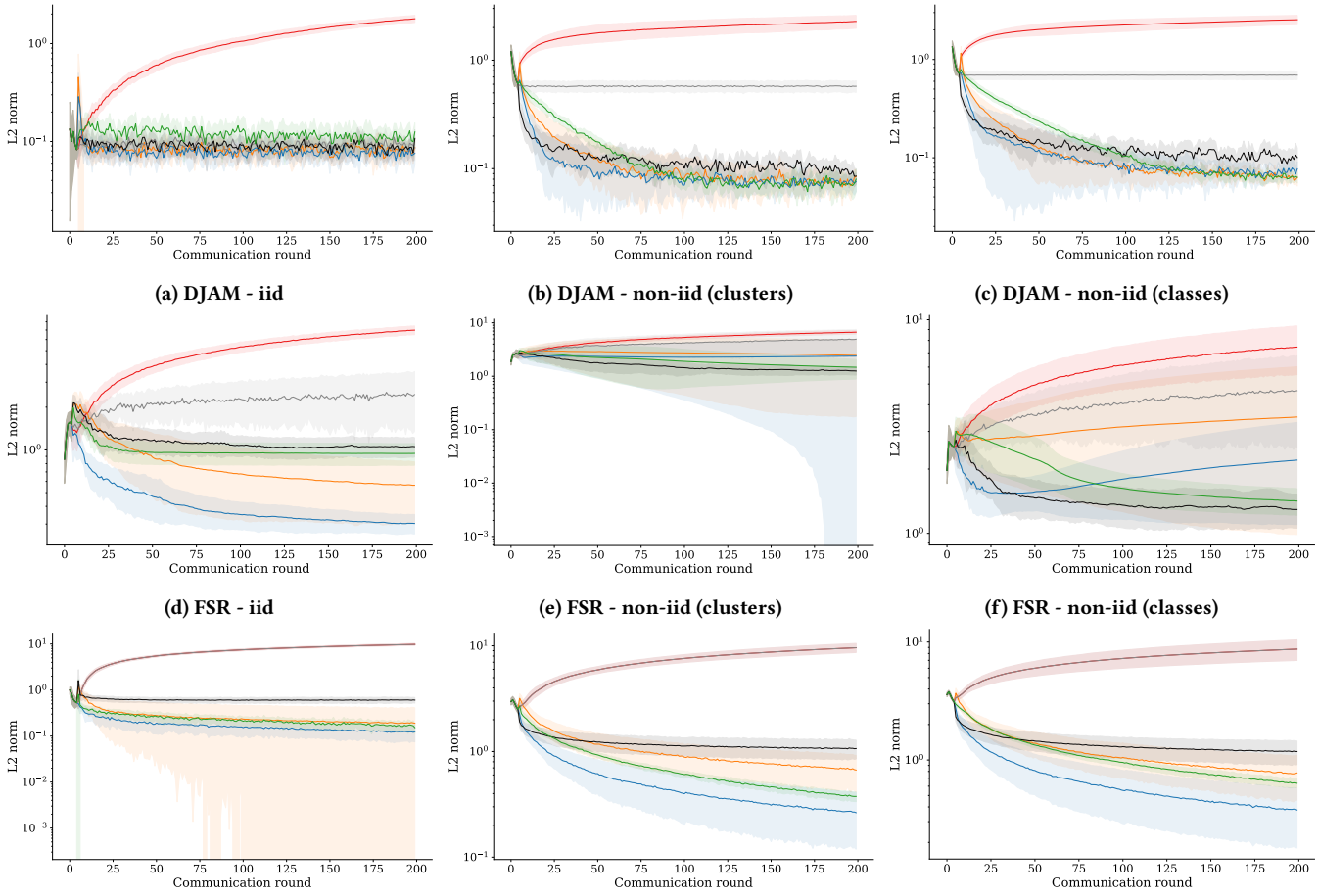
(a) DJAM - iid (b) DJAM - non-iid (clusters) (c) DJAM - non-iid (classes)

(d) FSR - iid (e) FSR - non-iid (clusters) (f) FSR - non-iid (classes)

(g) DFedAvgM - iid (h) DFedAvgM - non-iid (clusters) (i) DFedAvgM - non-iid (classes)

Figure 10: Similarity plots for DJAM, FSR, and DFedAvgM algorithms on the digits dataset. Shaded regions represent mean ± standard deviation across 10 folds, over 200 communication rounds. A random client is dropped persistently after the 5th round. Colors:model inversion, gradient inversion, reference, random, drop, no action

.



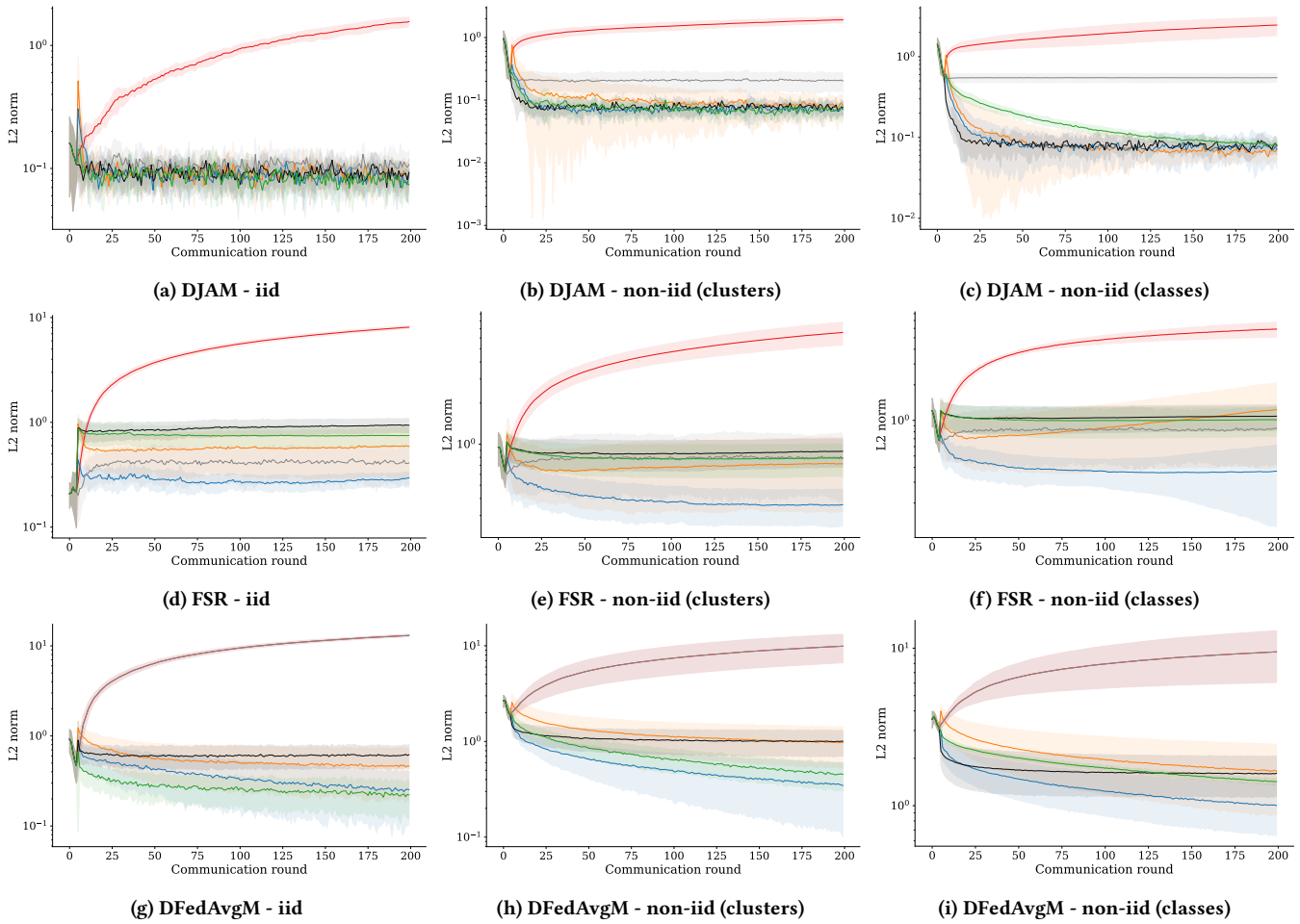
(a) DJAM - iid (b) DJAM - non-iid (clusters) (c) DJAM - non-iid (classes)

(d) FSR - iid (e) FSR - non-iid (clusters) (f) FSR - non-iid (classes)

(g) DFedAvgM - iid (h) DFedAvgM - non-iid (clusters) (i) DFedAvgM - non-iid (classes)

Figure 11: Similarity plots for DJAM, FSR, and DFedAvgM algorithms on the wine dataset. Shaded regions represent mean ± standard deviation across 10 folds, over 200 communication rounds. A random client is dropped persistently after the 5th round. Colors: model inversion, gradient inversion, reference, random, drop, no action

.



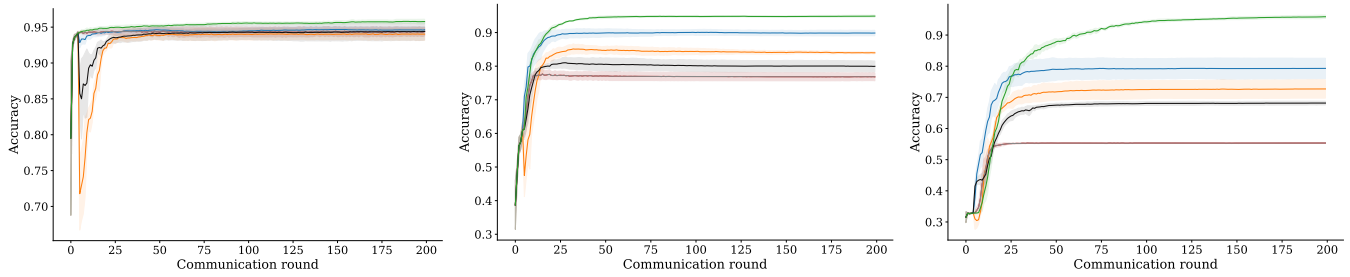
(a) DJAM - iid (b) DJAM - non-iid (clusters) (c) DJAM - non-iid (classes)

(d) FSR - iid (e) FSR - non-iid (clusters) (f) FSR - non-iid (classes)

(g) DFedAvgM - iid (h) DFedAvgM - non-iid (clusters) (i) DFedAvgM - non-iid (classes)

Figure 12: Similarity plots for DJAM, FSR, and DFedAvgM algorithms on the iris dataset. Shaded regions represent mean ± standard deviation across 10 folds, over 200 communication rounds. A random client is dropped persistently after the 5th round. Colors: model inversion, gradient inversion, reference, random, drop, no action

.



(a) DFedAvgM - iid (b) DFedAvgM (c) DFedAvgM - non-iid (classes)

Figure 13: Convergence plots for DFedAvgM on the digits dataset using a neural network model (3-layer MLP with 128 hidden units per layer). Shaded regions represent mean ± standard deviation across 3 folds, over 200 communication rounds. A random client is dropped persistently after the 5th round. Colors: model inversion, gradient inversion, reference, random, drop, no action.