IMPLICIT FEDERATION BIAS: A STUDY OF EMERGENT GENERALIZATION IN FEDERATED LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Federated learning (FL) changes the optimization dynamics by dividing training across local client trajectories and periodic aggregation, yet when and how this affects the emergence of generalization is unknown. We examine this through grokking, a delayed generalization phenomenon in which models generalize only after a prolonged period of memorization, but whose dynamics have been studied almost entirely under centralized optimization, leaving their behavior under FL unclear. We systematically study this using multiple established grokking setups under FL and find that emergent generalization persists, but its timing and behavior can change substantially. Increasing the number of clients or local optimization steps often delays generalization, while alternative federated optimizers can accelerate it by up to an order of magnitude. Stronger federation effects can produce long plateaus at sub-perfect training accuracy or, in extreme cases, no clear memorization period. In both cases, training and test performance improve together only after a long delay. To explain these effects, we analyze how local functional changes survive aggregation. Local updates are dominated by client-specific features, while aggregation preferentially suppresses these components and retains a greater fraction of globally consistent features; the accumulation of this shared global signal is associated with the onset of generalization. We characterize this as an *implicit federation bias* over learned structure that can delay, accelerate, or qualitatively reshape emergent generalization.

1 INTRODUCTION

Grokking provides an example of how memorization and generalization can occur on very different timescales. In the classical setting, a neural network first memorizes its training data and only later transitions to strong held-out performance after prolonged optimization (Power et al., 2022). This separation between memorization and generalization makes grokking a useful setting for studying not only whether a model generalizes, but when and how structured, generalizing solutions emerge during training. However, grokking has predominantly been studied under centralized optimization, where every update is computed from a single global training process.

Federated learning (FL) (Kairouz et al., 2021; McMahan et al., 2017) changes this optimization process fundamentally since rather than taking successive updates on a global objective, clients perform multiple steps of local optimization before their resulting updates are periodically aggregated by a server. As a result, FL alters the optimization dynamics, which depend on factors such as the number of clients, the amount of local training between synchronization, client participation, data heterogeneity, and the aggregation algorithm. While literature has extensively studied convergence and final model generalization, considerably less is understood about how FL alters the temporal dynamics through which generalizable solutions emerge.

We therefore ask: **how does federated optimization change the emergence of generalization, and what properties of local training and aggregation determine any effects?** We study this question using federated variants of several established grokking setups spanning modular arithmetic, finite-group composition, and MNIST classification across transformer and MLP architectures. By using these setups where grokking is already known to occur under centralized training we can isolate the effects due to federation. we systematically vary the number of clients, amount of local train-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

ing between aggregation, client participation ratios, data partition structure and heterogeneity and federated optimization algorithm, whilst tracking learning behavior and model dynamics.

Across these settings we find that emergent generalization remains generally robust to federated optimization, although federation can substantially alter the timing and form. Increasing the effects of federation commonly delays generalization, whereas several optimizers accelerate it by up to an order of magnitude, relative to canonical FedAvg. Federation can also alter the learning trajectory and behavior commonly associated with classical grokking: rather than fully memorizing before later generalizing, models can plateau and experience a delay far below perfect training accuracy before undergoing a rapid, grokking-like training accuracy improvement, which occurs with a tandem improvement in test performance. To understand these effects we perform an analysis of model changes and find that whilst local changes are dominated by client-specific structures, averaging preferentially suppresses and averages out the local structures from across clients, whilst retaining and reinforcing the globally consistent structures present. The accumulation of this shared generalizable global structure is associated with the onset of generalization.

Overall, our contributions are as follows:

1. We provide a systematic study of emergent generalization under federated optimization, showing that federation changes the timing of emergent generalization and can preserve, delay, accelerate, or eliminate the classical post-memorization grokking regime.
2. We characterize how different key factors of federated optimization alter the learning dynamics and generalization performance.
3. We provide mechanistic evidence of how federation alters generalization dynamics and characterize this as *implicit federation bias*, whereby aggregation preferentially suppresses client-specific changes and retains globally consistent structure.

2 Related Work

Federated Learning. Because clients take multiple local steps between aggregations (McMahan et al., 2017; Stich, 2019), often on heterogeneous data, their local objectives can diverge from the global one. This local training dynamic and any heterogeneous data split across clients means that clients will often have different local optimization objectives from the global objective. A number of federated optimization methods have since been developed to control or compensate for these deviations both at the local and server levels, such as FedProx (Li et al., 2020), SCAFFOLD (Karimireddy et al., 2020), FedDyn (Acar et al., 2021) and FedDC (Gao et al., 2022), as well as adaptive server optimizers such as FedAdam and FedYogi (Reddi et al., 2021).

Beyond convergence, works have investigated how federated optimization affects generalization. Federated training under heterogeneity can converge to sharper solutions (Caldarola et al., 2022; Qu et al., 2022; Caldarola et al., 2025), which are associated with poorer generalization (Keskar et al., 2017); flat local solutions do not necessarily yield a flat global solution (Lee & Yoon, 2024). Other work has shown that distributed optimization can alter the implicit regularization effects present in centralized SGD, thereby affecting solutions favored by optimization (Dandi et al., 2022). Feature-learning analyses show that local optimization can alter which features are learned (Bao et al., 2024) and the balance between signal learning and memorization (Huang et al., 2024). Theoretical work has derived stability-based generalization bounds for common federated algorithms, showing them to depend on client heterogeneity and algorithm convergence behavior (Sun et al., 2024), with the number of local steps and server optimizer momentum potentially producing trade-offs between faster optimization and stability (Zeng et al., 2024). These results describe the solutions federated training reaches, or bound their generalization error; they do not address when during training generalization appears.

Grokking. Since Power et al. (2022) first reported grokking on small algorithmic datasets, it has been observed across a range of architectures and tasks (Liu et al., 2023; Humayun et al., 2024; Gromov, 2023). A major line of work has connected this transition to the emergence of internal representations and regime phase transitions during training (Liu et al., 2022; Nanda et al., 2023; Rubin et al., 2024). Other work has focused on why generalizing solutions emerge only after a prolonged delay, with a series of competing and complementary findings relating to weight norms,

loss landscapes, neural network circuits, feature learning dynamics and implicit bias (Liu et al., 2023; Varma et al., 2023; Nanda et al., 2023; Kumar et al., 2024; Lyu et al., 2024). Despite differing in their proposed mechanisms, these accounts broadly emphasize that continued optimization after memorization can qualitatively alter the representations or solutions learned, which can eventually favor more structured generalizing solutions. To the best of our knowledge no work studies whether grokking persists and how it is affected.

3 Experimental Setup

We study how federated optimization affects the emergence of generalization using five established grokking setups adapted from prior work (Gromov, 2023; Liu et al., 2023; Chughtai et al., 2023; Nanda et al., 2023). These consist of both multi-layer perceptron and transformer-based architectures, with different task datasets, and optimization settings in order to provide generalizable findings across federated settings. The setups used are summarized as follows:

- **A**: Two layer quadratic MLP on modular arithmetic tasks.
- **B**: Two layer quadratic MLP on S_5 group composition.
- **C**: One layer transformer model on modular arithmetic tasks.
- **D**: One layer transformer model on S_5 group composition.
- **E** Three layer ReLU MLP with large initialization on MNIST classification.

Full experimental details are provided in Appendix A. Using each setup, we go through and ablate a range of federated learning factors and explore how this affects generalization. All experimental runs are repeated and tested across three different seeds to prevent any outlier seeds from skewing results. To perform federated training, we use the Flower library (Beutel et al., 2020) to simulate training across multiple devices. Time is measured in centralized-equivalent gradient steps, so centralized and federated runs are compared at equal gradient work. We define generalization as the point at which a model achieves an improved test performance comparable to training performance. Grokking refers to the standard increase of test performance, often suddenly, as training at zero training loss continues. Let T_{memo} denote the first point at which the model reaches the specified training-performance threshold, and T_{gen} the first point at which it reaches the corresponding held-out generalization threshold. Exact thresholds are defined for each setup in Appendix A.

4 Findings

Across all five setups, emergent generalization remains generally robust to federated optimization, however federation can change *when* generalization occurs, *how* the transition unfolds and in more extreme cases, the final generalizing performance reached. We first show that federation can qualitatively reshape the trajectory associated with classical grokking, before studying the effect of individual federated learning factors on learning dynamics and generalization.

4.1 Federation can preserve or reshape emergent generalization

Federated models continue to exhibit delayed emergent generalization but federation can qualitatively alter the trajectory preceding it. Figure 1 illustrates this for the transformer setups as we increase the effects of federation by varying the number of clients; the same patterns are demonstrated in the following sections as we increase federation through other factors. When the effects of federation are low, learning closely resembles classical grokking behavior, whereby training accuracy reaches near-perfect performance before a delayed increase in test performance. As we increase the effects due to federation, this separation decreases. training accuracy can plateau well below complete memorization and its final value, even failing to increase at all, with it later rapidly increasing in tandem with test accuracy when emergent generalization occurs. Therefore, federation does not simply shift the timing of classical grokking behavior, but rather it can alter the relationship between memorization and generalization itself, producing delayed transitions in which training accuracy only reaches high levels at the same time as test accuracy. We analyze the mechanisms underlying these altered learning trajectories in Section 5, examining how local client learning and aggregation change which and how structures accumulate in the global model.

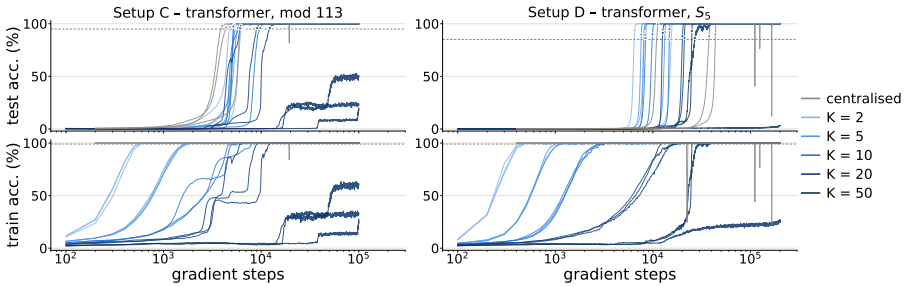

Figure 1: **Training and test performance curves as number of clients varies.** The figure shows for setups C and D increasing the number of clients affects memorization and generalization onset timings. Grokking-like transitions also emerging in the training curves with greater numbers of clients.

4.2 Effects of Federation

We next isolate different components of federated optimization and examine how they each affect resulting learning and generalization dynamics. In this section we focus on the empirical effects of each factor; in Section 5, we analyze the mechanisms underlying the observed differences and link them to these observed results. Across our experiments, federation affects learning in three ways: (1) a *temporal shift*, affecting the timing of memorization or generalization, (2) a *trajectory shift*, affecting the relationship and dynamics between training and test performance during training, and (3) an *outcome shift*, affecting the final test performance reached, often only in more extreme cases.

4.2.1 Number of Clients

Splitting data across increasing amounts of clients K generally produces larger delays to both memorization and generalization, and progressively changes the trajectory between them (Figure 1). Low-client settings ($K = 2, 5$) closely resemble centralized grokking behavior. At intermediate $K = 10, 20$ training accuracy often plateaus substantially below perfect memorization (e.g., 60-40%) before jumping in tandem with test accuracy when emergent generalization occurs. At $K = 50$, training and test performance become highly coupled, with both behaviors matching each other and both undergoing sharp changes as similar points in training. Increasing the number of clients can therefore affect not only generalization time but also the separation between memorization and generalization that defines the classical grokking regime. It is worth noting that for most cases, the solutions produced are all of the same caliber, achieving close to 100% performance.

4.2.2 Local epochs

Increasing the amount of local training between aggregation rounds produces a clear temporal shift under FedAvg. Across most setups, increasing the number of local epochs E delays both memorization and generalization, with the effect becoming more pronounced at larger numbers of local epochs, and under non-IID data distributions. Increasing E from 5 to 25 delays the median time to generalize T_{gen} in almost all runs, with shifts ranging up to $3.5\times$ (Figure 3). Larger E is also seen to alter the form of transition and learning dynamics like with the number of clients. As shown in Figure 2, training and test performance again become more closely coupled as the effects of federation increase, with both rising sharply together, rather than generalization following a long post-memorization delay, in settings with high numbers of local epochs.

4.2.3 Partial participation

In contrast to the number of clients and amount of local training, reducing the participation fraction has comparatively little effect on learning behavior and generalization timing. Across the tested setups, the behaviors and timings for both memorization and emergent generalization remain close to the full-participation timings, even when only 25% of clients participate per aggregation round, and with non-IID data. (Figure 4). Partial participation therefore does not appear to have any effect on delays observed due to federation.

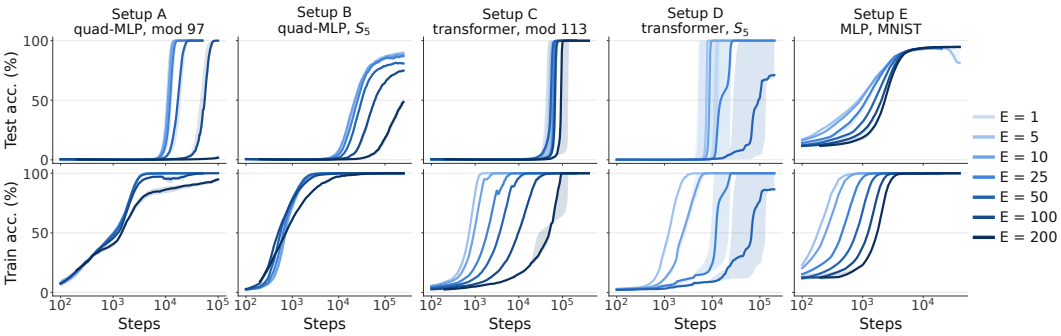

Figure 2: **Training and test performance curves as number of local epochs varies.** Increasing the amount of local training between aggregation rounds generally shifts both memorization and generalization later in training. At larger E, the separation between training and test dynamics can also decrease, with some settings exhibiting sharp improvements in training and test accuracy at similar points in training.

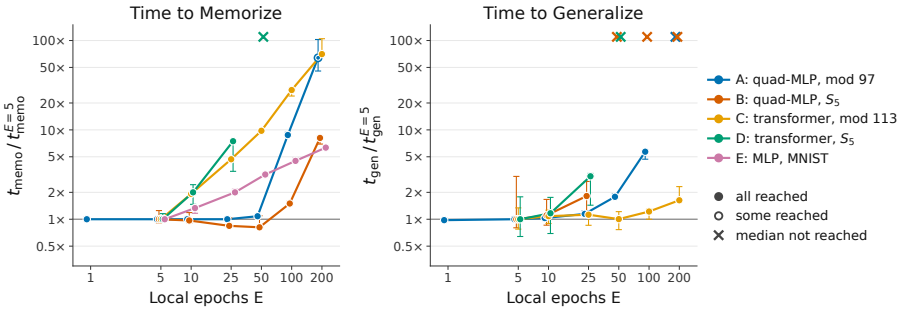

Figure 3: **Relative memorization and generalization timing ratios as the number of local epochs varies.** The ratio shown is relative to $E = 5$ local epochs. Increasing local epochs often delays memorization, as well as generalization.

4.2.4 Data partitioning and heterogeneity

Generalization timing is comparatively robust to moderate levels of data heterogeneity. across most setups, decreasing the Dirichlet concentration produces relatively small changes in generalization timing in most setups, until the most heterogeneous regime we test ($\alpha_{\text{dir}} = 0.01$), whereby runs often fail to generalize within training budget (Figure 5). The structure of data partitioning can also have noticeable effects. Operand-based partitions have ranging effects and can sometimes produce classical grokking behavior, sometimes with delay, or in other cases display the moderate training accuracy plateau-like delay and generalization. Output label-based partitions on the other hand frequently prevent generalization within budget, even when comparable unstructured non-IID partitions are able to (Figure 6). Overall the effect of data partitioning depends not only on heterogeneity magnitude but also on how task structure is distributed across clients. In addition, data partitioning often has more visible outcome shifts compare to other factors, producing more generalizations that are below comparable counterparts.

4.2.5 Federated optimization algorithm

The choice of federated optimizer produces the largest changes in generalization timing that we observe. We compare FedAvg with FedAvgM, FedAdam, FedYogi, SCAFFOLD and FedProx across chosen federated settings. We see that FedAvgM, FedAdam, FedYogi, and SCAFFOLD all generalize faster, typically reducing memorization time by more than $2\times$ and generalization time by $5 - 10\times$ (Figure 7). The only exception is FedProx, which memorizes and generalizes more slowly than FedAvg. These differences show that changing how client updates are processed can substantially accelerate the emergence of generalization. Hyperparameter sweeps for each algorithm are in Appendix C.

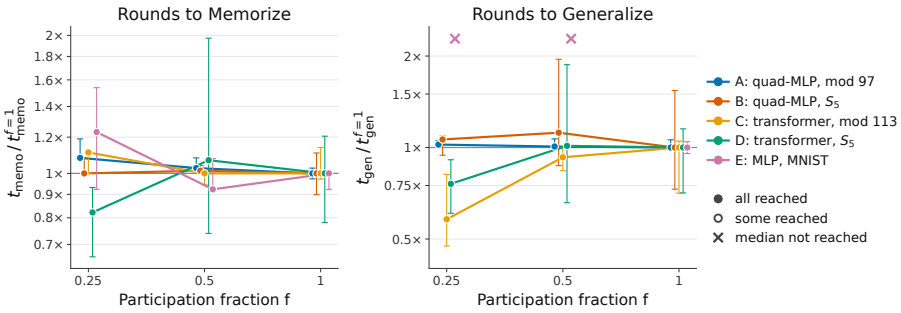

Figure 4: **Slowdown ratio of memorization and generalization as participation fraction varies.** The ratio is relative to full participation. Reducing the participation ratio has minimal negative effects on generalization onset timing, and can have positive effects on transformer-based models.

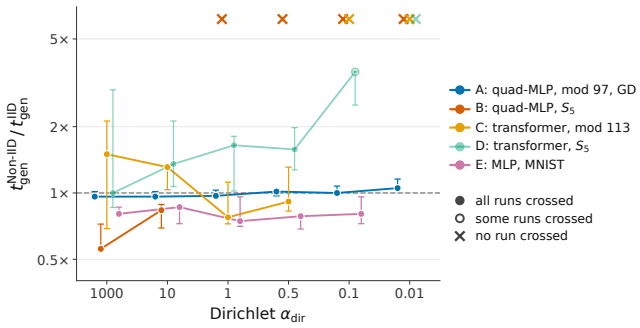

Figure 5: **Effect of data heterogeneity on federated generalization.** We plot the generalization time T_{gen} versus the Dirichlet concentration parameter α_{dir} for $K = 10$ clients, under different training fractions α. Smaller α_{dir} corresponds to stronger non-IID partitioning, while large α_{dir} approaches IID. Across most settings, generalization time is relatively stable under moderate heterogeneity, but extreme heterogeneity ($\alpha_{\text{dir}} = 0.01$) noticeably prevents generalization.

5 Analysis

Section 4 showed how different aspects of federated learning affect emergent generalization and learning behavior. We saw that increasing the number of clients or amount of local training can delay generalization and alter learning behavior, whilst other factors such as partial participation have comparably little effect, and that alternative federated optimizers can substantially accelerate generalization. Our analysis focuses on two areas: *how much, and which parts of local learning survive aggregation?*, and *how does this affect generalization?* To demonstrate the intuition, we study the first primarily in Setup A, where the symmetry of modular addition permits an exact functional decomposition, and use controlled server-update experiments to study the second. We provide additional plots and findings for the other setups in Appendix Section D.2.

Mechanistic hypothesis: implicit federation bias. Before working through the evidence we first layout the idea. We hypothesize that repeated local optimization and aggregation induce an implicit bias over the functional structure learned by the global model, with this specific to forms of decentralized learning. Clients optimize for their local objectives and therefore develop client-specific features, that are not useful globally in general solutions. Aggregation combines these local features and by nature preferentially retains changes that are consistent across clients (i.e., generalizable features), whilst suppressing the client-specific components that disagree. This process can affect the accumulation of globally useful features required for both memorization and generalization. We refer to this federation-induced selection over the learned structure as *implicit federation bias*. Under this view, federation can alter learning behavior and generalization through two avenues. The first is through altering the composition of updates provided to the server (e.g., more client updates, different learnt updates), and correspondingly how that affects which functional components survive

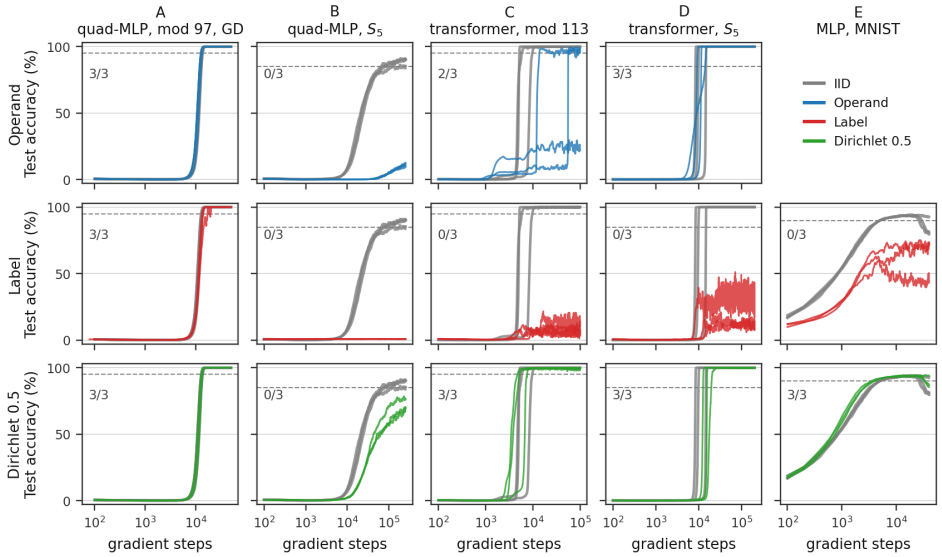

Figure 6: **Test performance during training for different data partitioning schemes.** Splitting by output label performs the worst, followed by operand splits, with both often generalizing slower or not at all compared with non-structured splits. Setup E lacks an operand partition due to MNIST consisting of images, and so lacking an appropriate way to split considering input data only.

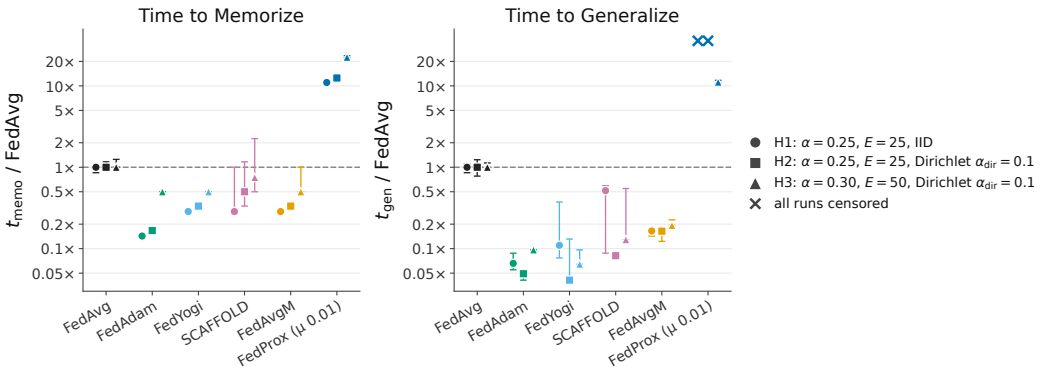

Figure 7: **Slowdown ratio of generalization with different federated learning algorithms relative to FedAvg.** All algorithms tested, except FedProx, learn faster than FedAvg, on average memorizing twice as fast and generalization approximately an order of magnitude sooner.

aggregation; the second is through the server optimization rule and how the surviving functional signal is used server-side (e.g., server-side optimizers).

We first ask which components of local learning survive aggregation. In modular addition, changes in model outputs can be decomposed into components associated with individual operands, memorization of particular input pairs, structure associated with the sum and answer separately, and a task-symmetric component determined only by $(a+b-c) \bmod 97$. We denote this final component by G (see Appendix A.9 for measurement details). Since G applies the same output pattern across the complete addition table, a correctly directed G component can express a global addition rule. Local client changes are dominated by operand-specific and pair-lookup structure. Across the six example configurations shown in Figure 8a, operand-only terms account for approximately 57–86% of functional-change energy and pair lookup for 12–38%, whereas G accounts for less than 0.1%. Local optimization therefore predominantly changes functions associated with the client's observed data, while the globally symmetric component initially forms only a small fraction of the local signal. Aggregation does not preserve these components equally. Pair-lookup changes are strongly

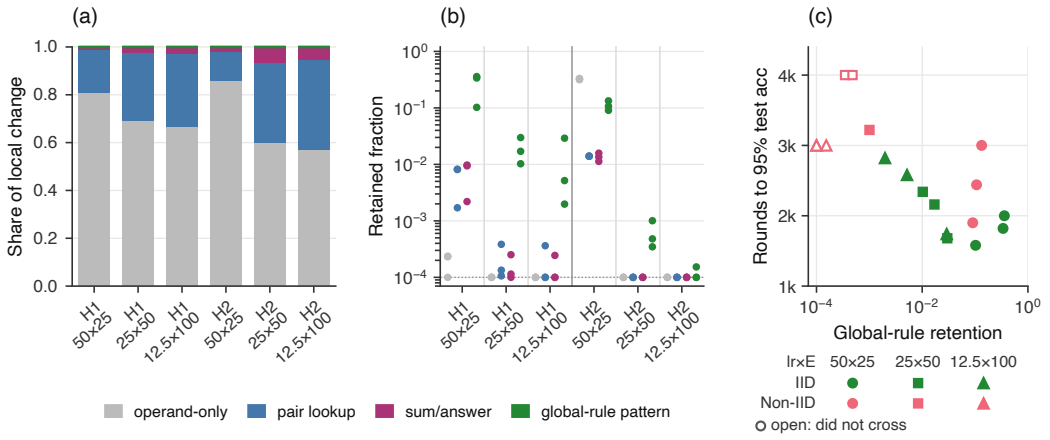

Figure 8: **Averaging favors the generalized rule over memorization.** H1: IID clients; H2: Dirichlet partition ($\alpha_{\text{dir}} = 0.1$). Groups in (a) and (b) are client learning rate $\times$ E, at a fixed product. (a) Clients' local changes are mostly operand-only, with some pair lookup structure, and very little signal in the global rule pattern. (b) Averaging keeps more of G than a pair lookup. (c) Runs that keep more of G generalize faster. Crosses show runs that never generalized, plotted at their budget.

suppressed, whereas approximately 9–36% of G change energy is retained for learning rate 50 and $E = 25$ (Figure 8b). Although aggregation also attenuates the signal for G, it retains a substantially greater fraction of the globally consistent component than of local lookup structure. We interpret this selective retention as direct evidence of implicit federation bias. The amount of global-rule structure retained is also associated with generalization timing: runs retaining more G tend to reach the generalization threshold earlier (Figure 8c). This relationship does not establish retention as the sole cause of the delay, but it connects the functional filtering produced by aggregation to the effects observed in Section 4.

Server step gain biases learning and generalization. The filtering analysis explains which components reach the server, but not how quickly those components accumulate. We therefore separately examine the scale of the applied optimization step. In centralized training, the delay between memorization and generalization approximately follows inverse learning-rate scaling over the tested range (Figure 9a), indicating that the timescale of the generalization transition depends strongly on optimization step size. We observe the same effect directly under FedAvg. Increasing the server gain γ accelerates generalization while leaving the client-side learning procedure unchanged: doubling the applied server update to $\gamma = 2$ approximately halves the steps required to reach the generalization threshold across three seeds under both IID and non-IID partitioning (Figure 9b). This provides a second component of the explanation for the optimizer results in Section 4. Once globally consistent structure survives aggregation, larger effective server updates allow that signal to accumulate more rapidly. Consequently, the acceleration produced by adaptive server optimizers need not imply that they discover a qualitatively different generalizing direction; part of their benefit can arise from amplification of the aggregated update. Across the federated algorithms we study, effective server-step gain follows this pattern closely: methods applying larger effective updates generally generalise sooner (Figure 9b). SCAFFOLD is the notable exception, achieving substantial acceleration without the same increase in effective step gain. This suggests that update scale explains much of the improvement from the server-side adaptive optimizers, while SCAFFOLD additionally benefits from changing the direction or quality of client updates through its control-variate correction.

Explaining the effects of federated factors. This federation selection bias provides a common interpretation of the factor sweeps and phenomena observed in Section 4. Increasing the number of clients divides the global task across a greater number of local objectives, increasing the opportunity for clients to develop structures that are not consistently shared across the population. Increasing local training has a related effect: clients travel further along their local trajectories before synchronization, allowing client-specific structures to become more pronounced. In both cases,

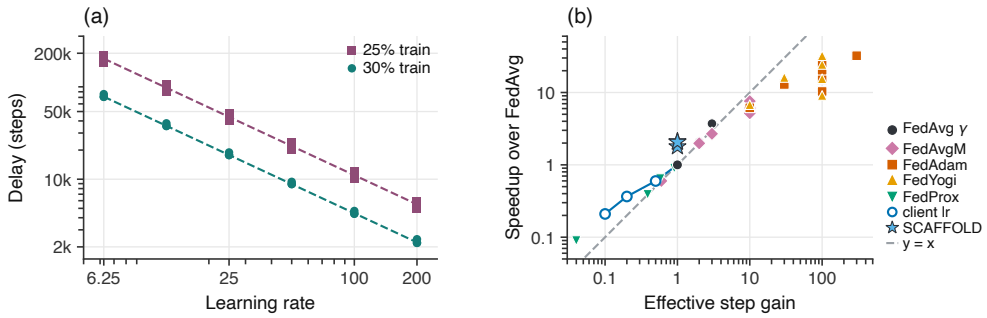

Figure 9: **Larger optimisation steps lead to earlier generalization.** (a) In centralized training, we plot the number of steps between memorization and generalization against the learning rate. Doubling the learning rate roughly halves the delay. (b) For each federated algorithm and different client learning rates, we plot how much faster it generalizes compared to FedAvg against how much larger its step size is compared to FedAvg. Further details in Appendix B.2. Algorithms that take larger steps generalise faster.

globally consistent structure can therefore accumulate more slowly, consistent with the delayed generalization and altered training trajectories observed as K and E increase. Data heterogeneity further changes the structure available to each local objective. Notably, the non-IID runs in Figure 19 show greater parameter-space coherence while nevertheless generalizing later, demonstrating that stronger client agreement is not necessarily beneficial. For longer local trajectories, heterogeneous clients can retain less G structure, providing a functional explanation for why sufficiently strong or structured heterogeneity can impede generalization. Partial participation has much smaller effects since changing which subset of clients contributes to a given aggregation round does not systematically alter generalization, as the balance between client-specific and global structure within updates remains similar on average. The same filtering mechanism also helps to explain the altered memorization–generalization relationship observed. If memorizing structure is repeatedly suppressed during aggregation while globally consistent generalizing structure accumulates across rounds, the global model need not establish a strong memorizing solution before the shared solution becomes dominant, allowing training and test performance to rise together after the seen prolonged delay.

6 Conclusion

We systematically study emergent generalization under federated learning and show that federated learning can alter both when and how generalization emerges. More clients, local training, and data partitioning in cases, delays generalization and reduces the separation between memorization and generalization. Conversely partial participation has little effect on training and alternative federated optimizers are able to accelerate generalization by up to an order of magnitude.

Our analysis provides evidence for *implicit federation bias* as a mechanism underlying these effects, whereby local optimization produces largely client-specific changes, while aggregation retains the remaining globally consistent structure across client updates. Our results suggest that generalization dynamics therefore depend on both what functional signals survive aggregation and how quickly they accumulate. This also helps explain why client drift and divergence alone is insufficient to predict generalization behavior, and why changes to the server update scale can substantially alter the generalization timescale. More broadly, federated optimization should be viewed as having its own implicit bias over learned structure, and not merely as a distributed approximation to centralized training; this distinction is important when interpreting differences between federated algorithms: some apparent gains can arise from differences in effective update scale, while others may reflect changes to the direction or quality of the aggregated update.

REFERENCES

Durmus Alp Emre Acar, Yue Zhao, Ramon Matas Navarro, Matthew Mattina, Paul Whatmough, and Venkatesh Saligrama. Federated learning based on dynamic regularization. In *International Conference on Learning Representations*, 2021.

Yajie Bao, Michael Crawshaw, and Mingrui Liu. Provable benefits of local steps in heterogeneous federated learning for neural networks: A feature learning perspective. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 2857–2902. PMLR, 2024.

Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, and Nicholas D. Lane. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

Debora Caldarola, Barbara Caputo, and Marco Ciccone. Improving generalization in federated learning by seeking flat minima. In *European Conference on Computer Vision*, pp. 654–672. Springer, 2022. doi: 10.1007/978-3-031-20050-2_38.

Debora Caldarola, Pietro Cagnasso, Barbara Caputo, and Marco Ciccone. Beyond local sharpness: Communication-efficient global sharpness-aware minimization for federated learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 25187–25197, 2025.

Bilal Chughtai, Lawrence Chan, and Neel Nanda. A toy model of universality: Reverse engineering how networks learn group operations. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 6243–6267. PMLR, 2023.

Yatin Dandi, Luis Barba, and Martin Jaggi. Implicit gradient alignment in distributed and federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 6454–6462, 2022. doi: 10.1609/aaai.v36i6.20597.

Liang Gao, Huazhu Fu, Li Li, Yingwen Chen, Ming Xu, and Cheng-Zhong Xu. FedDC: Federated learning with Non-IID data via local drift decoupling and correction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10112–10121, 2022.

Andrey Gromov. Grokking modular arithmetic. *arXiv preprint arXiv:2301.02679*, 2023.

Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.

Wei Huang, Ye Shi, Zhongyi Cai, and Taiji Suzuki. Understanding convergence and generalization in federated learning through feature learning theory. In *International Conference on Learning Representations*, 2024.

Ahmed Imtiaz Humayun, Randall Balestriero, and Richard G. Baraniuk. Deep networks always grok and here is why. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 20722–20745. PMLR, 2024.

Peter Kairouz, H. Brendan McMahan, et al. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1–2):1–210, 2021. doi: 10.1561/2200000083.

Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. SCAFFOLD: Stochastic controlled averaging for federated learning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5132–5143. PMLR, 2020. URL `https://proceedings.mlr.press/v119/karimireddy20a.html`.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*, 2017.

Tanishq Kumar, Blake Bordelon, Samuel J. Gershman, and Cengiz Pehlevan. Grokking as the transition from lazy to rich training dynamics. In *International Conference on Learning Representations*, 2024.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Taehwan Lee and Sung Whan Yoon. Rethinking the flat minima searching in federated learning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 27037–27071. PMLR, 2024. URL https://proceedings.mlr.press/v235/lee24aa.html.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In *Proceedings of Machine Learning and Systems*, volume 2, pp. 429–450, 2020. URL https://proceedings.mlsys.org/paper/2020/hash/1f5fe83998a09396ebe6477d9475ba0c-Abstract.html.

Ziming Liu, Ouail Kitouni, Niklas S. Nolte, Eric J. Michaud, Max Tegmark, and Mike Williams. Towards understanding grokking: An effective theory of representation learning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 34651–34663, 2022.

Ziming Liu, Eric J. Michaud, and Max Tegmark. Omnigrok: Grokking beyond algorithmic data. In *International Conference on Learning Representations*, 2023.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Kaifeng Lyu, Jikai Jin, Zhiyuan Li, Simon S. Du, Jason D. Lee, and Wei Hu. Dichotomy of early and late phase implicit biases can provably induce grokking. In *International Conference on Learning Representations*, 2024.

H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pp. 1273–1282. PMLR, 2017. URL https://proceedings.mlr.press/v54/mcmahan17a.html.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *International Conference on Learning Representations*, 2023.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

Zhe Qu, Xingyu Li, Rui Duan, Yao Liu, Bo Tang, and Zhuo Lu. Generalized federated learning via sharpness aware minimization. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 18250–18280. PMLR, 2022.

Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Adaptive federated optimization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=LkFG3lB13U5.

Noa Rubin, Inbar Seroussi, and Zohar Ringel. Grokking as a first order phase transition in two layer networks. In *International Conference on Learning Representations*, 2024.

Sebastian U. Stich. Local SGD converges fast and communicates little. In *International Conference on Learning Representations*, 2019.

Zhenyu Sun, Xiaochun Niu, and Ermin Wei. Understanding generalization of federated learning via stability: Heterogeneity matters. In *Proceedings of the 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *Proceedings of Machine Learning Research*, pp. 676–684. PMLR, 2024. URL https://proceedings.mlr.press/v238/sun24a.html.

Vikrant Varma, Rohin Shah, Zachary Kenton, János Kramár, and Ramana Kumar. Explaining grokking through circuit efficiency. *arXiv preprint arXiv:2309.02390*, 2023.

Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Nghia Hoang, and Yasaman Khazaeni. Bayesian nonparametric federated learning of neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 7252–7261. PMLR, 2019.

Dun Zeng, Zheshun Wu, Shiyu Liu, Yu Pan, Xiaoying Tang, and Zenglin Xu. Understanding generalization of federated learning: The trade-off between model stability and optimization. *arXiv preprint arXiv:2411.16303*, 2024.

A Experimental Details

Unless otherwise specified, experimental conditions are evaluated across three seeds to reduce sensitivity to individual runs. If configuration details are not mentioned explicitly then the factors used are $K = 10$ clients, $E = 5$ local epochs, full client participation ($f = 1$), and FedAvg aggregation. Deviations from these defaults are stated in the corresponding sections. Also unless otherwise stated, experiments use seeds $\{42, 123, 456\}$, with selected experiments also using additional seeds, 789 and 1011.

Failure to generalize or grok within the training budget should not be interpreted as evidence that the model would never generalize. Rather, it indicates that the relevant generalization threshold was not reached within the compute budget used. Such runs are treated as right-censored.

For the finite group and modular-arithmetic tasks, a fraction α of the complete dataset is sampled uniformly without replacement for training, with the remaining examples forming the test set. This is used as a means to artificially alter training difficulty. The same train–test split is used between corresponding centralized and federated runs with the same seed. Federated partitioning is applied only after the global training set has been selected, and all evaluation is performed on the global held-out test set.

A.1 Setup A – Quadratic MLP and modular arithmetic

Setup A is adapted from the quadratic network studied by Gromov (2023). We use a two-layer MLP with hidden width $N = 256$, no biases, and an element-wise quadratic activation,

$$f(x) = W_2\phi(W_1x), \qquad \phi(h) = h^2.$$

Following the mean-field parameterization of the original setup, the forward-pass scaling is absorbed into the initialization:

$$(W_1)_{ij} \sim \mathcal{N}\left(0, \frac{1}{2n}\right), \qquad (W_2)_{ij} \sim \mathcal{N}\left(0, \frac{1}{256^2}\right).$$

Training uses full-batch gradient descent, with learning rate 50 and no weight decay. The objective is mean-squared error (MSE) against one-hot target vectors.

A.2 Setup B – Quadratic MLP and S_5 group composition

Unlike Setup A, Setup B is trained with AdamW (Loshchilov & Hutter, 2019) rather than gradient descent. We use learning rate 10^{-3}, weight decay 1.0, full-batch local training, and cross-entropy loss.

A.3 Setup C – Transformer and modular arithmetic

Setup C is adapted from the one-layer transformer used by Nanda et al. (2023). The task is modular addition.

The input sequence consists of exactly two tokens corresponding to (a, b). A shared token embedding is used for both positions together with learned positional embeddings. The model contains one transformer block with four attention heads, residual width $d = 128$, no causal mask, no LayerNorm, and no biases. The attention block is followed by a ReLU MLP of width 512 with a residual connection. Output logits are read from the second token position. Token embeddings and the output unembedding matrix are not tied.

The embedding, positional embedding, unembedding, attention matrices and MLP input matrix are initialized from $\mathcal{N}(0, 1/d)$, while the MLP output matrix is initialized from $\mathcal{N}(0, 1/512)$.

Training uses AdamW with learning rate 10^{-3} and cross-entropy loss. Weight decay is 0.1

A.4 Setup D – Transformer and S_5 group composition

Setup D combines the transformer architecture with the S_5 group-composition task studied in previous work on mechanistic interpretability of group operations (Chughtai et al., 2023).

The dataset contains every ordered pair $(g, h) \in S_5 \times S_5$. Since $|S_5| = 120$, this gives $120^2 = 14{,}400$ examples. The target is the composition

$$(g \circ h)(i) = g(h(i)).$$

Each input consists of two group-element tokens and there are 120 output classes.

The architecture follows Setup B but uses a larger residual width $d = 256$. It therefore has four attention heads with head dimension 64, together with the same width-512 ReLU MLP.

Training uses AdamW with learning rate 10^{-3}, weight decay 1.0, full-batch local updates, and cross-entropy loss.

A.5 Setup E – Omnigrok MLP and MNIST classification

Setup E adapts the small-data MNIST (LeCun et al., 1998) setting used by Omnigrok (Liu et al., 2023). The model is a fully connected ReLU network with architecture

$$784 \to 200 \to 200 \to 10,$$

including biases. Parameters are initialized and then multiplied by 9. This enlarged initialization encourages the delayed-generalization regime studied in Omnigrok.

Training and test sets are disjoint subsets sampled from the standard MNIST training split; the canonical MNIST test split is not used. Images are flattened to 784 dimensions and scaled to $[0, 1]$, without additional normalization. The model is trained using AdamW with learning rate 10^{-3}, weight decay 0.1, mini-batches of 100 examples, and MSE loss against one-hot targets.

A.6 Client Data Partitions

The default IID partition randomly permutes the training set before splitting it approximately equally between clients. Client dataset sizes therefore differ by at most one example.

We additionally consider several heterogeneous partitioning regimes. In the *operand* partition, examples are assigned according to their first input operand. In the *label* partition, assignment is determined by the target class.

For continuous control over statistical heterogeneity, we use Dirichlet label-skew partitions (Yurochkin et al., 2019; Hsu et al., 2019). For each class c, client proportions are sampled as

$$q_c \sim \mathrm{Dirichlet}(\alpha_{\mathrm{dir}} \mathbf{1}_K),$$

where smaller α_{dir} corresponds to greater heterogeneity; the subscript distinguishes it from the global training-data fraction α.

A.7 Training-Time Normalization

To compare experiments with different numbers of local epochs and different client-participation rates, federated training time is expressed in centralized-equivalent steps. After each communication round we increment

$$t \leftarrow t + E\frac{\sum_{k\in S_t} n_k}{n_{\text{train}}}.$$

At full participation this corresponds to E centralized-equivalent steps per communication round. With equal client sizes and participation fraction f, it is approximately Ef steps per round. For the full-batch tasks, one unit corresponds to processing the equivalent of the entire centralized training set once in a gradient update. For MNIST, one unit corresponds to one pass over n_{train} examples.

A.8 Grokking and Generalization Metrics

We define the memorization time as the first logged point at which training accuracy reaches 99%,

$$t_{\text{memo}} = \min\{t : \text{Acc}_{\text{train}}(t) \geq 0.99\}.$$

Generalization thresholds are set to 95% test accuracy for Setups A, and C, 85% for Setups B and D, and 90% for Setup E. The thresholds differ because the attainable test-accuracy ceiling differs between tasks.

A.9 Fraction Retention Measurements

We measure the retained fraction of each component X,

$$\kappa_X = \frac{\left\|X[\Delta\bar{f}]\right\|^2}{\sum_i p_i \left\|X[\Delta f_i]\right\|^2}$$

The energy of X in the averaged model's change over one round, divided by the sample-weighted mean of that energy in the clients' own changes from the same incoming model. For each seed we report the median of κ_X over the rounds before test accuracy first reaches 95%.

A.10 Implementation Details

The experiment seed controls the train–test split, client partitioning, model initialization, MNIST subset sampling, and minibatch ordering. Runs are not guaranteed to be bitwise deterministic because client-update aggregation order can vary, deterministic GPU kernels are not enforced, and client sampling under partial participation is not fully seeded. Experiments use Python 3.10, PyTorch 2.10 (Paszke et al., 2019), torchvision 0.25, Flower 1.27 with its Ray simulation backend, and NumPy. Experiments were run on a mix of NVIDIA L4, A40 and 3080 GPUs. Overall this took approximately 2,500 L4 GPU hours.

B Federated Training Information

Federated experiments are implemented using Flower's synchronous simulation framework. In each communication round, the server selects $\lfloor fK \rfloor$ clients from the K available clients. Each selected client begins from the current global parameters and constructs a fresh local optimizer. As such, optimizer state such as Adam moment estimates and momentum buffers is correctly not retained between communication rounds.

Each client performs E local epochs before returning its updated parameters and number of local training examples to the server. For Setups A–D, training is full-batch, meaning that one local epoch corresponds to one gradient update on the client's complete local dataset. For Setup E, one local epoch corresponds to a shuffled pass through the client's data using minibatches of size 100.

Unless otherwise stated, aggregation uses FedAvg,

$$w^{(t+1)} = \sum_{k\in S_t} \frac{n_k}{\sum_{j\in S_t} n_j} w_k^{(t+1)},$$

where S_t is the set of participating clients and n_k is the number of examples held by client k.

B.1 Federated Optimizers

In addition to FedAvg, we evaluate several federated optimization algorithms. Sweeps of each method's hyperparameter are in Appendix C.

FedProx. FedProx (Li et al., 2020) retains FedAvg aggregation but adds a proximal term to each client's local objective,

$$\mathcal{L}_k^{\text{prox}} = \mathcal{L}_k + \frac{\mu}{2}\|w - w_{\text{global}}\|_2^2,$$

with $\mu = 0.01$.

FedAvgM. FedAvgM (Hsu et al., 2019) applies heavy-ball momentum to the aggregated server update. Following server-hyperparameter calibration, we use server learning rate $\eta_s = 1.0$ and momentum $\beta_s = 0.9$.

FedAdam and FedYogi. FedAdam and FedYogi apply adaptive server-side optimization to the aggregated client pseudo-gradient (Reddi et al., 2021). We use

$$\beta_1 = 0.9, \qquad \beta_2 = 0.99, \qquad \tau = 10^{-3},$$

with calibrated server learning rate $\eta_s = 0.1$.

SCAFFOLD. SCAFFOLD corrects each local gradient using client and global control variates,

$$g \leftarrow g - c_k + c.$$

We use the Option-II control-variate update from Karimireddy et al. (2020).

B.2 Effective Step-Size Gain

Figure 9b compares federated rules by their effective step-size gain g: the factor by which a rule scales FedAvg's per-round update $\overline{\Delta}_t$, the example-weighted mean of the client displacements, so that FedAvg has $g = 1$. All values below are for Setup A, whose clients use gradient descent with learning rate $\eta_c = 50$ by default.

Server learning rate. FedAvg with server step $w_{t+1} = w_t + \gamma\overline{\Delta}_t$ has $g = \gamma$.

FedAvgM. The momentum buffer $m_{t+1} = \beta_s m_t + \overline{\Delta}_t$ is applied as $w_{t+1} = w_t + \eta_s m_{t+1}$. For a slowly varying $\overline{\Delta}_t$ the buffer settles at $\overline{\Delta}_t/(1-\beta_s)$, so $g = \eta_s/(1-\beta_s)$.

FedAdam and FedYogi. The server step is $\hat{\eta}_s\, m_t/(\sqrt{v_t} + \tau)$, where m_t and v_t are moving averages of $\overline{\Delta}_t$ and $\overline{\Delta}_t^2$, and the bias-corrected step size $\hat{\eta}_s$ approaches η_s after the first rounds. In Setup A every coordinate of $\overline{\Delta}_t$ is roughly six orders of magnitude below $\tau = 10^{-3}$, so $\sqrt{v_t} \ll \tau$ and the step reduces to $(\eta_s/\tau)\,\overline{\Delta}_t$: FedAvg with $g = \eta_s/\tau$. The preconditioner is inactive in this regime, so these rules rescale the update without changing its direction.

FedProx. Writing $u = w - w_{\text{global}}$, a local step is $u \leftarrow (1 - \eta_c\mu)\, u - \eta_c \nabla\mathcal{L}_k$, a leaky integrator. Over E local steps with a slowly varying gradient, the client's accumulated update is scaled relative to plain local gradient descent by

$$g = \frac{1 - (1 - \eta_c\mu)^E}{\eta_c\mu E},$$

and the local loop diverges when $\eta_c\mu > 2$.

Reduced client learning rate. $\overline{\Delta}_t$ scales with the client learning rate, so $g = \eta_c/50$.

SCAFFOLD. The control variates change the direction of each local gradient but not its scale, so $g = 1$.

Besides the calibrated settings in Appendix B.1, Figure 9b includes the server-learning-rate and μ calibration ladders ($K = 10$, $E = 5$, IID; Setup A in Figure 10) and FedAvg with $\gamma \in \{2, 3\}$. Each point is compared with FedAvg on the same configuration.

C HYPERPARAMETER SWEEPS

To avoid comparing federated algorithms at arbitrarily chosen hyperparameters, we calibrated each algorithm over its primary free hyperparameter. For FedAdam and FedYogi, we swept the server learning rate $\eta_s \in \{0.01, 0.03, 0.1, 0.3\}$; for FedProx, the proximal strength $\mu \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$; and for FedAvgM, the 2×2 grid of $\eta_s \in \{0.3, 1.0\}$ and momentum $\in \{0.5, 0.9\}$. SCAFFOLD has no corresponding free hyperparameter and so is not included here.

Figure 10 reports the time to memorize and the time to first reach the generalization threshold for each setting, relative to FedAvg on the same configuration. Crosses indicate configurations for which no seed reached the event within budget. For the per-method calibration plots (Figure 11), color encodes the ordered hyperparameter magnitude. FedAvgM varies two parameters jointly, so its settings are ordered by the effective gain $\eta_s/(1-\beta_s)$ (Appendix B.2).

The hyperparameters used and described in the papers are chosen as those which generalize fastest across the majority of setups, and are the best overall values, as demonstrated in Figure 11.

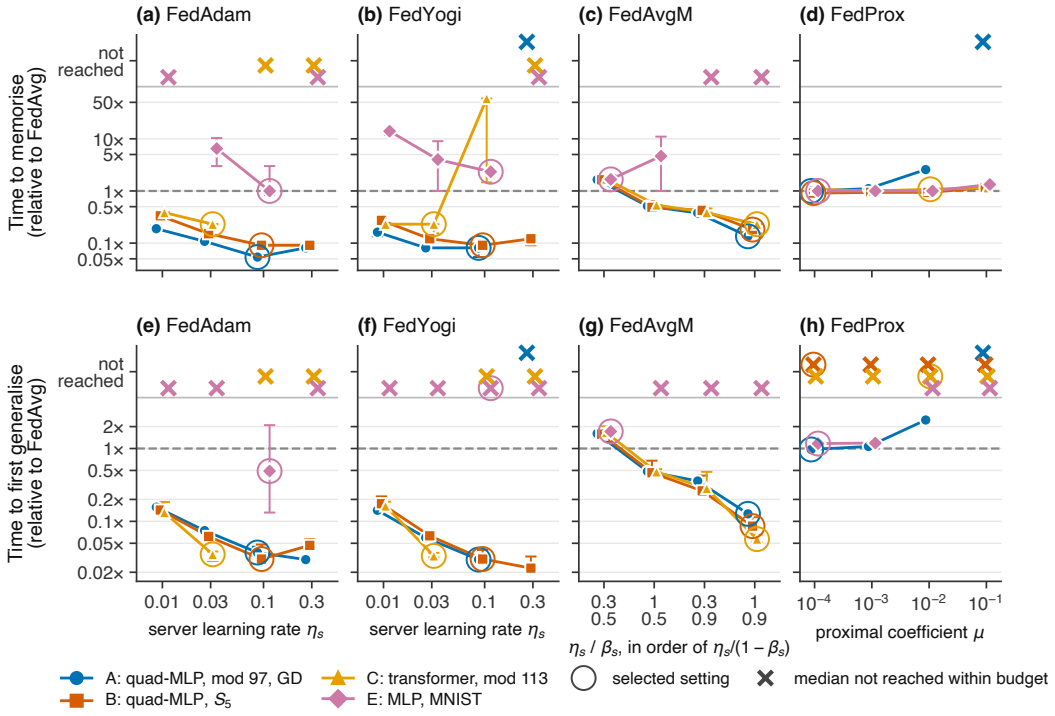

Figure 10: **Federated optimizer hyperparameter calibration.** (a-b) shows the time to memorize and (e-h) shows time to generalize for each FL algorithm variant, relative to FedAvg on the same configuration ($K = 10$, $E = 5$, IID).

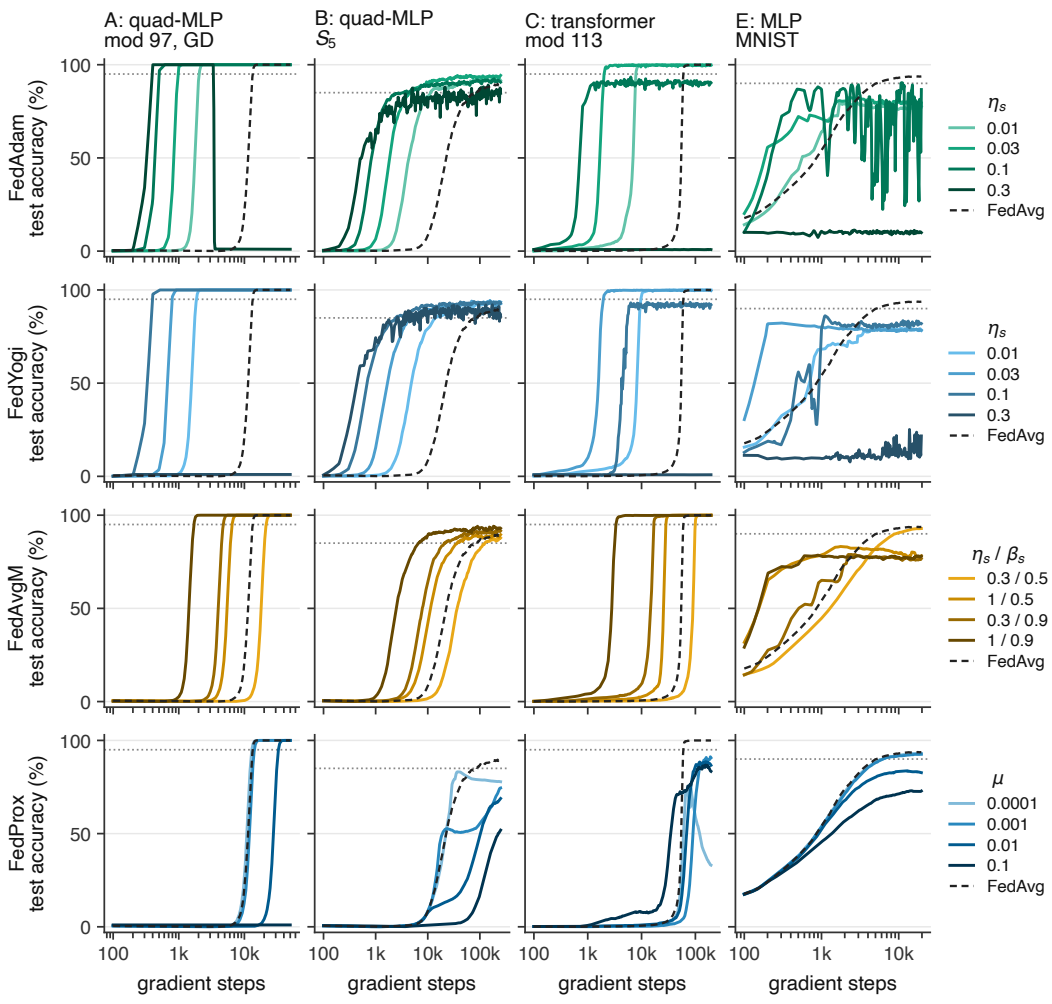

Figure 11: **Test accuracy for each federated optimizer setting.** ($K = 10$, $E = 5$, IID). Darker lines are larger values (for FedAvgM, larger $\eta_s/(1-\beta_s)$). Dashed: FedAvg on the same configuration. Horizontal dotted line: the setup's generalization threshold.

D SUPPLEMENTARY PLOTS FOR FINDINGS

In this section we provide additional plots as evidence for phenomenon discussed in Section 4.

D.1 TRAINING CURVES

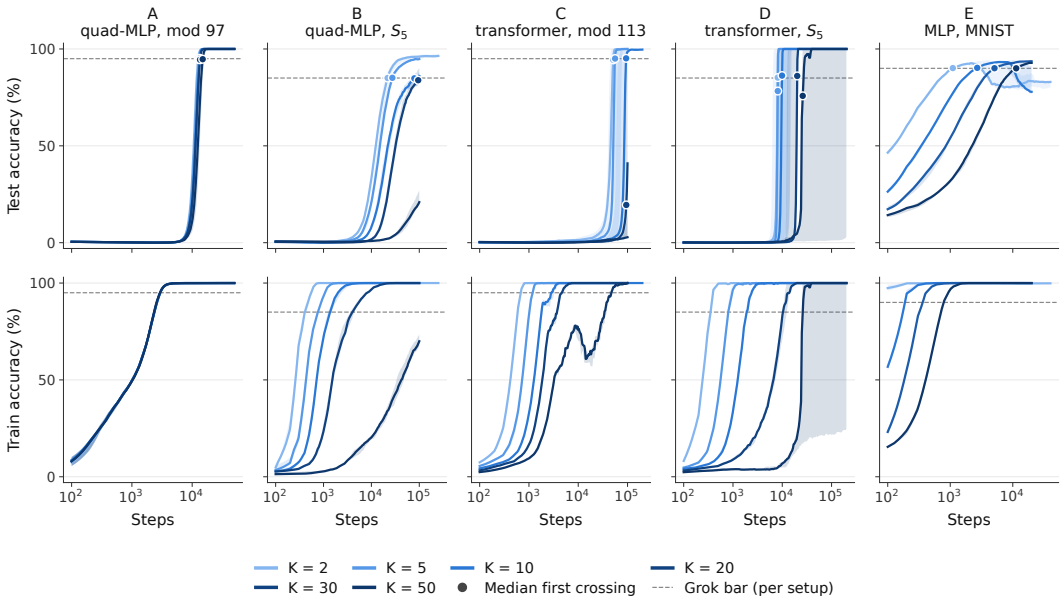

Figure 12: **Training and test accuracy throughout optimization as number of clients varies.**

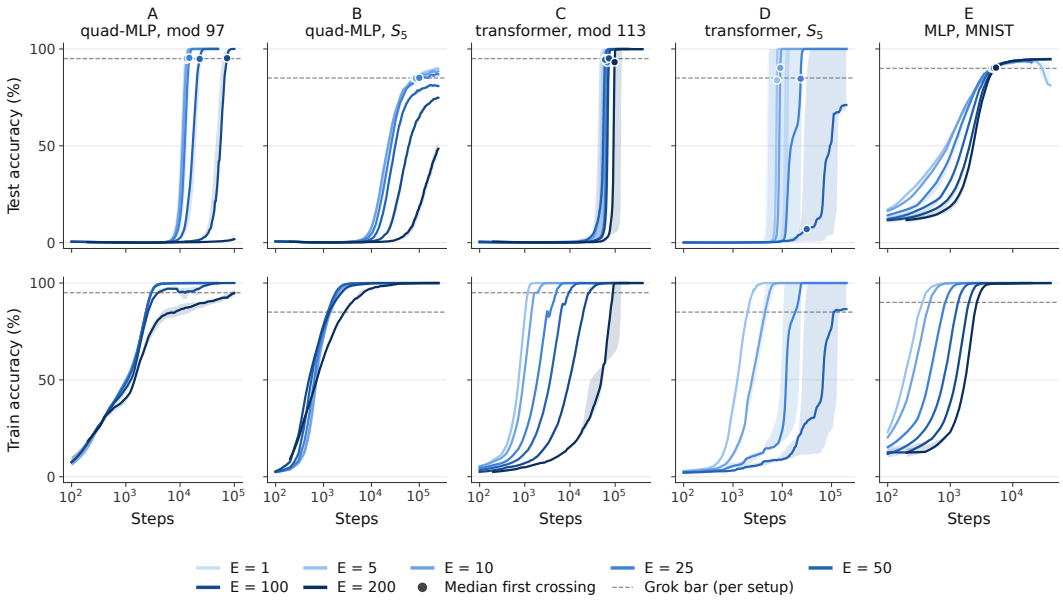

Figure 13: **Training and test accuracy throughout optimization as number of local epochs varies.**

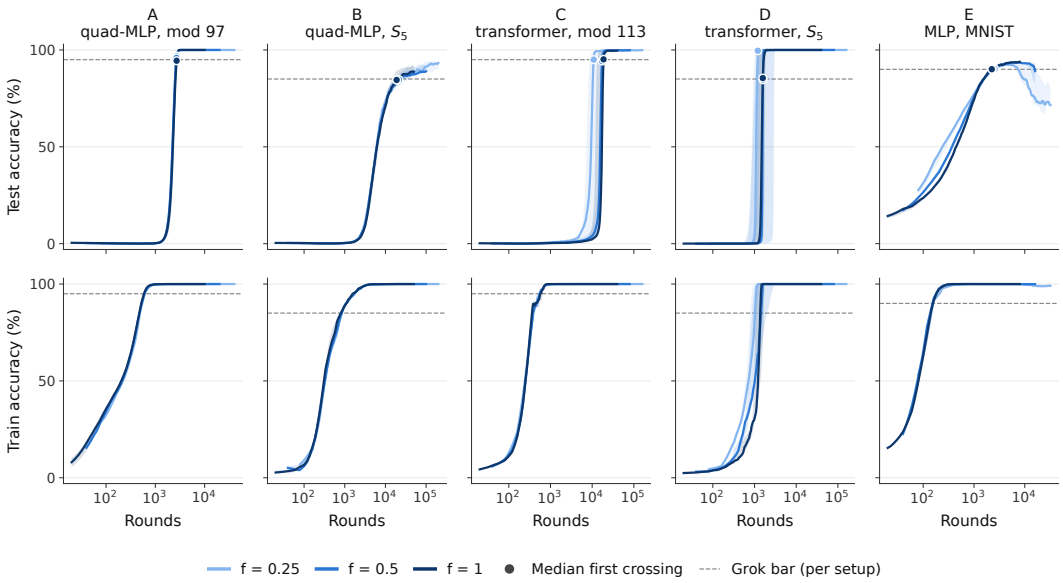

Figure 14: **Training and test accuracy throughout optimization as participation fraction varies.**

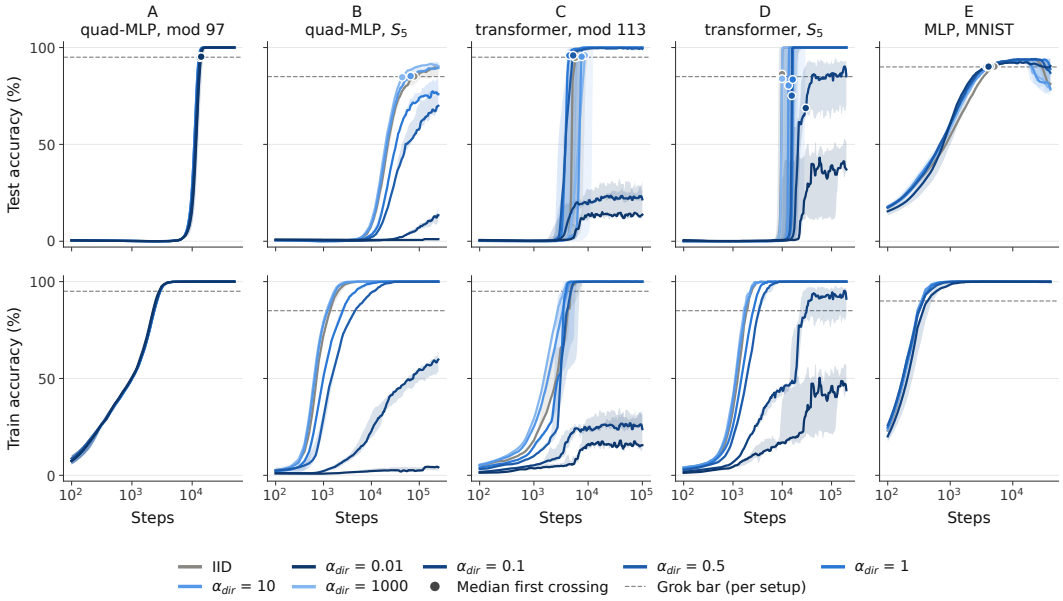

Figure 15: **Training and test accuracy throughout optimization as data heterogeneity varies.**

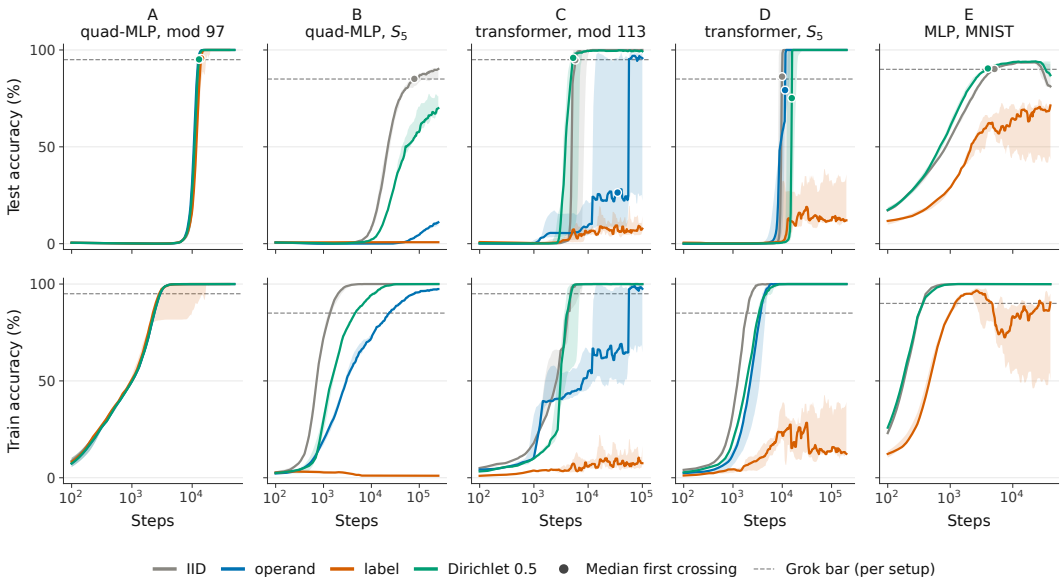

Figure 16: **Training and test accuracy throughout optimization as data split structure differs.**

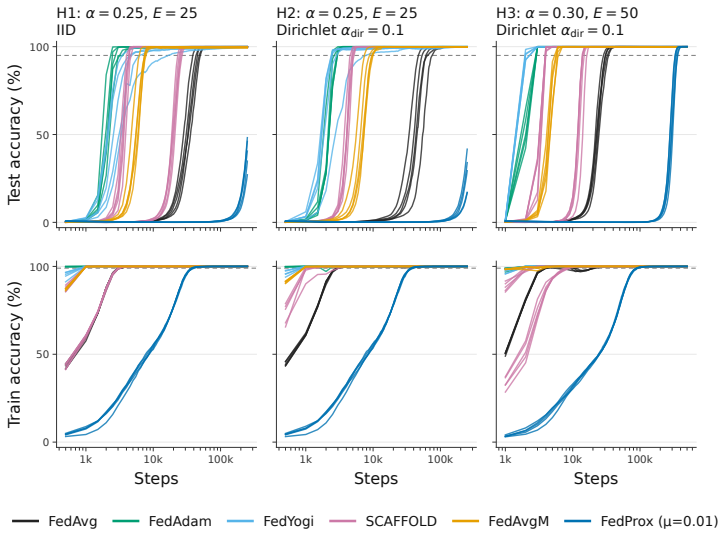

Figure 17: **Training and test accuracy throughout optimization for different federation algorithms.**

D.2 Retention across setups

Figure 18 repeats Figure 8 on setups A–D (FedAvg, $K = 10$, three seeds), for IID clients at $E \in \{10, \ldots, 200\}$ and Dirichlet skew $\alpha_{\text{dir}} \in \{0.01, \ldots, 10^3\}$ at $E = 5$. Setup E has no operand grid to decompose.

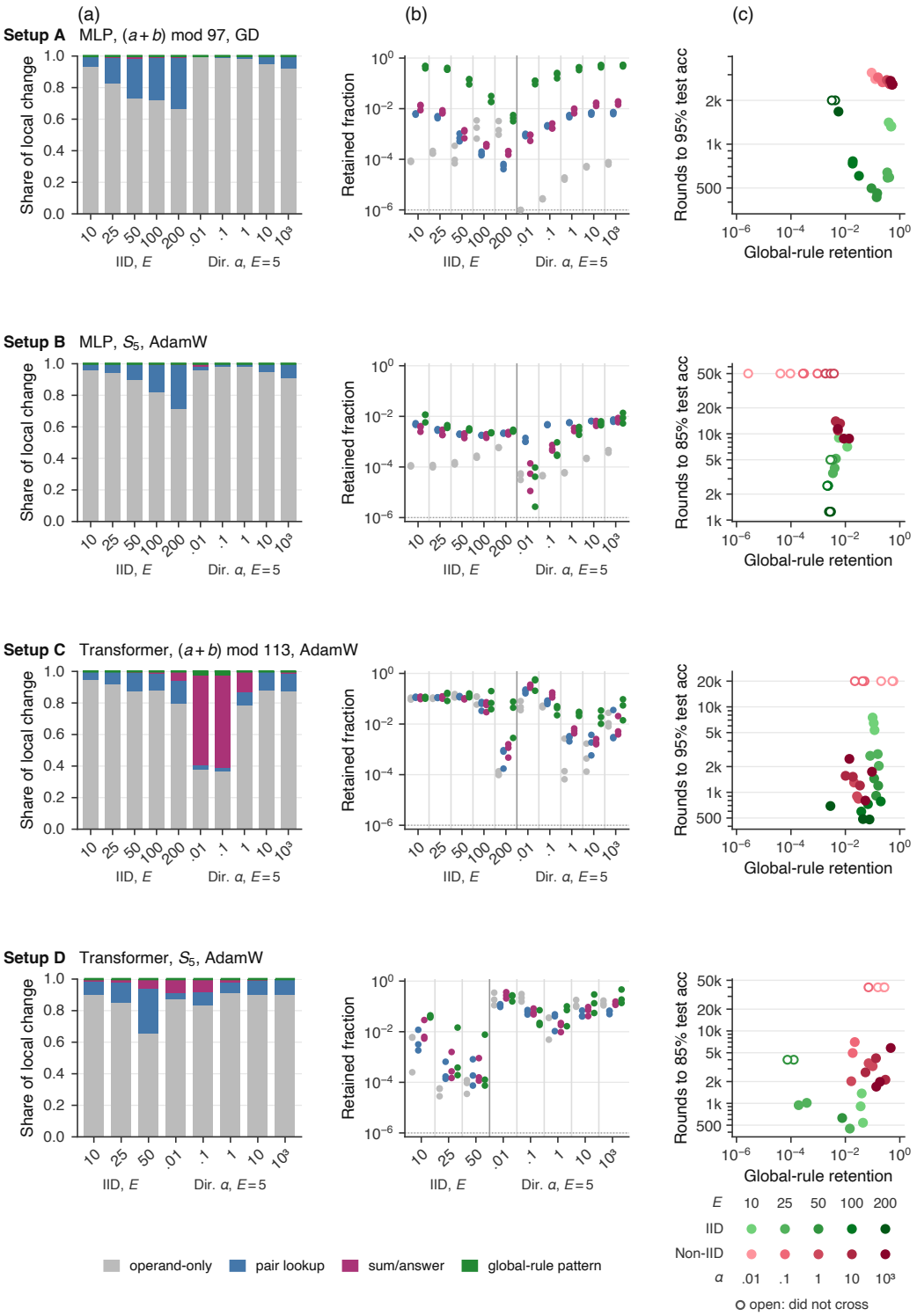

Figure 18: **Server aggregation favors the generalized rule over memorization.** A replication and extension of Figure 8 across setups A-D.

E ADDITIONAL ANALYSIS

In this section we extend the analysis we did in 5 by covering implementation details of the experiments run and provide additional smaller investigations relevant to the discussion.

E.1 CLIENT DRIFT AND DISAGREEMENT DOES NOT EXPLAIN GENERALIZATION.

For client i, let $\Delta_{i,t} = w_{i,t}^{\text{local}} - w_t$ denote its displacement after E local steps from the incoming server model. FedAvg forms the sample-weighted average over K clients:

$$\overline{\Delta}_t = \sum_i p_i \Delta_{i,t}, \qquad w_{t+1} = w_t + \gamma \overline{\Delta}_t$$

where p_i is the client's share of the participating training examples and γ is the server gain (a server step size), with $\gamma = 1$ for standard FedAvg.

Although clients can move substantially while optimizing their local objectives, these movements may cancel when averaged. We measure how much client movement survives averaging using client-update coherence,

$$C_t = \frac{\|\overline{\Delta}_t\|^2}{\sum_i p_i \|\Delta_{i,t}\|^2},$$

which is 1 when clients agree, about $1/K$ for uncorrelated equally weighted updates, and below $1/K$ when clients oppose one another.

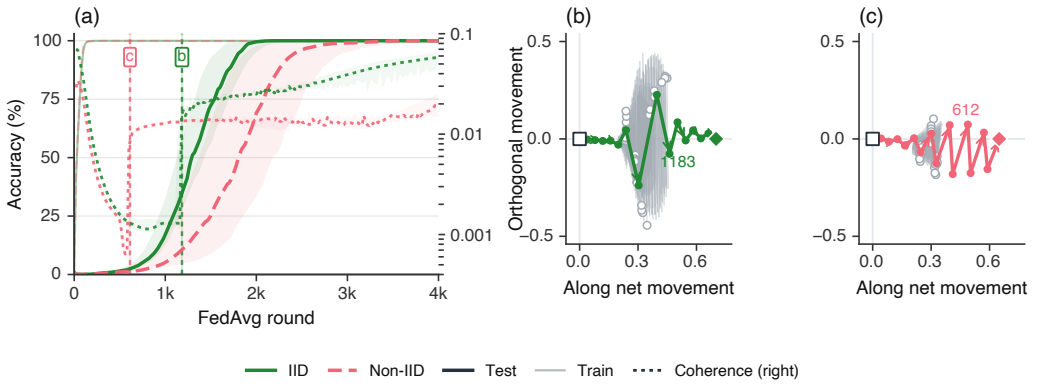

Figure 19: **Client updates largely cancel under averaging.** (Setup A, $K = 10$, learning rate 50, $E = 25$). (a) Coherence stays far below $1/K = 0.1$ throughout training, and lower for IID than non-IID clients even though IID clients generalize first. (b, c) In the marked windows, clients (gray) spread widely around the server, whose path zig-zags as successive updates reverse. Planes are spanned by the net server movement and the leading orthogonal direction, which capture most server movement but little client movement.

Figure 19 shows substantial cancellation under both IID and non-IID partitioning. However, the amount of cancellation does not explain generalization timing. At round 700, coherence is approximately 0.0012 for IID clients and 0.011 for non-IID clients, despite the non-IID runs generalizing later. We also observe increases in coherence and reversals in the server trajectory without corresponding improvements in test performance.

Thus, client disagreement describes how strongly local updates combine, but not whether the surviving movement supports generalization. Explaining the delays observed in Section 4 therefore requires examining the structure of what survives aggregation rather than its magnitude alone, which we also explore in greater detail in Appendix E.3.

E.2 CLIENT TRAJECTORIES AND PROJECTION CHOICE

We extend Figure 19 with local trajectories from Setup A using ten clients, learning rate 50, 25 local steps, full participation, and standard FedAvg. Each snapshot replays one local round from a saved server checkpoint. These are diagnostic replays, not bitwise reproductions of the original accelerator trajectories. Round r denotes the incoming checkpoint; the paths show round $r + 1$.

We select four checkpoints per run: (1) after the first recorded 99% training accuracy, (2) before and (3) after the largest recorded increase in server-step norm between memorization and the first 95% test accuracy, and (4) after that test threshold. Selection is retrospective and uses the available checkpoints at intervals of 100 rounds. Figure 22 locates the snapshots in each run.

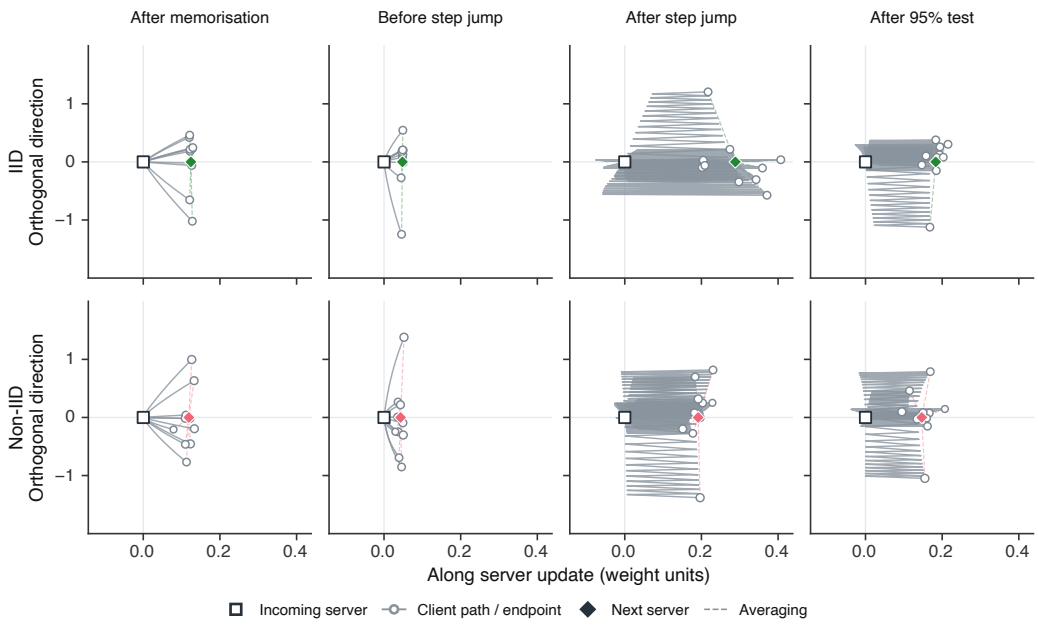

Figure 20: **Client paths in planes aligned with the server update.** Each panel shows the 25 local steps from the incoming server; circle area scales with client sample weight. The horizontal axis follows the averaged update and the vertical axis the leading orthogonal endpoint disagreement, fitted per panel.

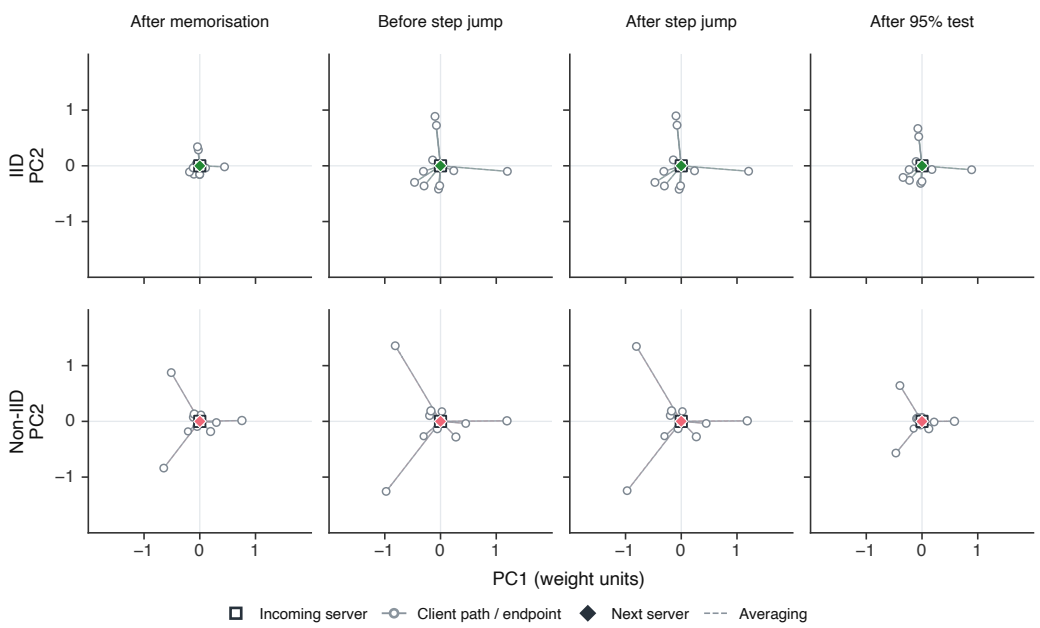

Figure 21: **Client paths in fixed PCA planes.** The replays of Figure 20, projected into one fixed client PCA plane per row.

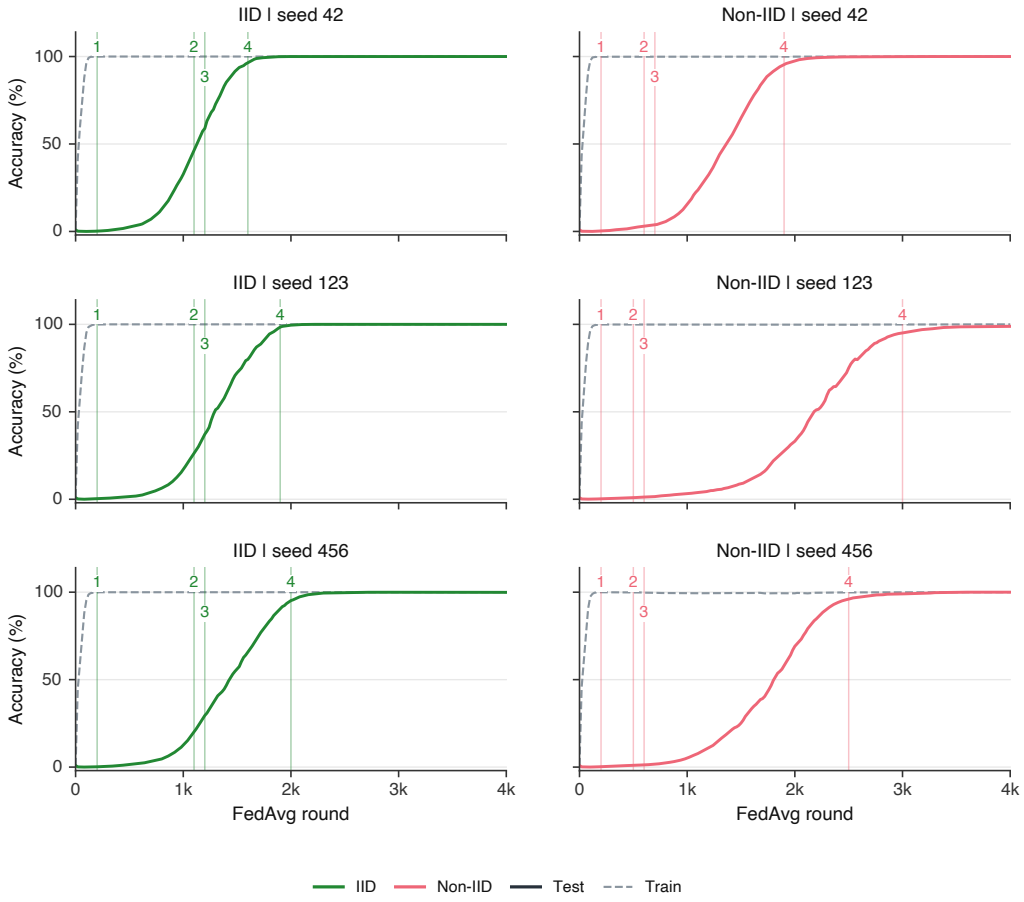

Figure 22: **Training context for the trajectory snapshots.** Server train and test accuracy for IID and non-IID runs (seeds 42, 123, 456). Vertical guides mark the four incoming checkpoints used in the local replays.

We project client paths onto two planes that answer different questions.

The *server-aligned plane* (Figure 20) is built from the server update. Its horizontal axis is the averaged update; its vertical axis is the direction, perpendicular to that, in which client endpoints disagree most. It is refitted at every checkpoint and always contains the full server step.

The *client PCA plane* (Figure 21) is built from the clients. It spans the two directions in which clients move most, pooled over all local steps and all four checkpoints of a run.[1] Each row uses one fixed plane, so positions can be compared along a row but not between rows.

Both planes show only part of the motion: the server-aligned plane keeps 11–17% of the squared client displacement and the PCA plane 3–24% (weighted by client size, summed over local steps). Paths that look alike in two dimensions, such as the before- and after-jump panels, can still differ in the full parameter space.

The PCA plane almost hides the server. It keeps at most 0.016% of the squared server step, so in Figure 21 the next server sits on top of the incoming one even though the server moved. Put differently, in the directions where clients move most, their movements cancel when averaged.

[1] Uncentered SVD of client displacements from the incoming server, weighted by client sample count, with checkpoints and local steps weighted equally and no per-layer normalization.

E.3 Local steps, aggregation, and function

The trajectory views show where clients move, but not whether that movement helps. To test this, we average the client weights after every local step e, as if the round ended there, and evaluate the result; at $e = 25$ this is the actual FedAvg update. As a centralized reference, we run e steps of full-batch gradient descent on the pooled training set from the same incoming server. The two coincide at $e = 1$. Losses are reported relative to their value at the incoming server.

Clients improve on their own shards but not on the full training set, and averaging cancels almost all of their change (Figure 23; ranges over 24 snapshots from three seeds, figure shows seed 42). By $e = 25$, clients lower their own-shard loss to 0.33–0.77, while their mean loss on the full training set rises to 1.02–1.47. The averaged model stays at 0.990–1.006 at every step, and its held-out loss at 0.995–1.002. Pooled gradient descent does better in every snapshot, reaching 0.896–0.998.

Averaging the client outputs instead of the weights gives the same training loss to within 0.1%, so this is not caused by the nonlinearity of weight averaging: the clients' changes cancel each other. The lower panels show the same in weight space. By $e = 25$, the coherence of Appendix E.1 is below 0.05: less than 5% of the squared client movement survives averaging. After the step jump, coherence alternates between odd and even steps because consecutive local steps reverse direction, as they do in pooled gradient descent from the same point.

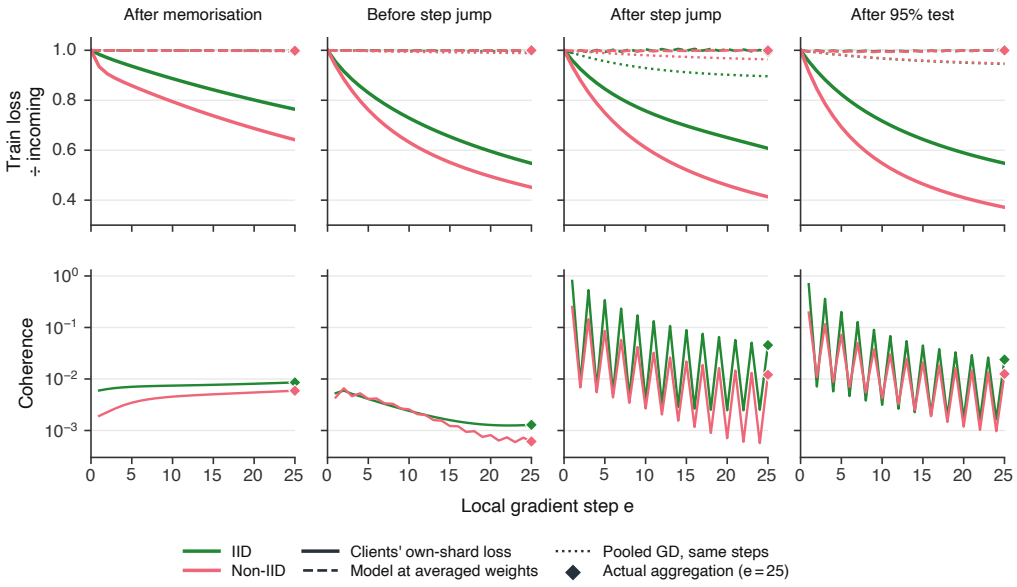

Figure 23: **Local progress and the model obtained by averaging at each local step.** Top: training loss relative to the incoming server. Bottom: coherence of the client displacements.

Figure 24 shows which functional changes survive. We summarize the class-centered output grid over all p^2 operand pairs in two ways. The *task-symmetric margin* is $g(0)$, the correct-class value of the component that depends only on $(a + b - c) \bmod p$. The *lookup energy* is the mean square of the pair-lookup component, the operand interaction not explained by $a + b$. Both summaries use the known task symmetry; neither measures generalization directly. In every snapshot, every client raises lookup energy by $+1.2$ to $+10.0\%$ by $e = 25$, while the averaged model changes it by -0.24 to $+0.72\%$. Client changes in the margin often differ in sign, and the averaged model keeps their weighted mean. For the margin, the sample-weighted average of client outputs is exactly this weighted client mean; where it rises, averaging the weights instead keeps 95–99% of the gain in IID runs and 38–92% in non-IID runs. This is the within-round counterpart of Figure 8.

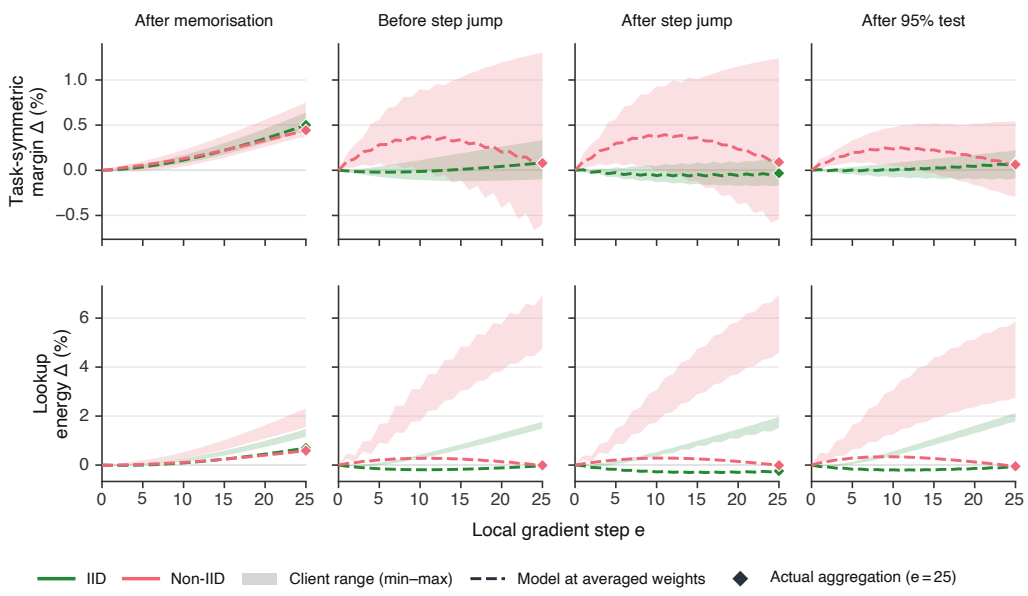

Figure 24: **What functional change survives averaging.** Change from the incoming server in the task-symmetric margin (top) and lookup energy (bottom), as a percentage of the incoming value. Bands span the ten clients; dashed lines are the model at averaged weights.

F Limitations

Our experiments use relatively small models and controlled grokking tasks, future work should investigate this phenomenon in larger models and more challenging tasks. Most configurations are evaluated over three seeds; more seeds could help to increase reliability. Our federated settings, although appropriate for the tasks and research, use relatively small client populations and controlled data partitions, and therefore do not capture the full complexity of real-world FL.

G AI Usage Statement

We used AI tools to assist with the work for this paper, all in line with AI usage policies. For retrieval and discovery, generative AI was used to find sources for our literature review although sources were verified to ensure they matched what they were being referenced for. For research ideation and execution, generative AI tools were used to assist with experiment development. For drafting and polishing writing, generative AI tools were used to draft certain sections of this work, and were used to help polish and refine aspects of writing and presentation for the paper.