StitchFL: Heterogeneous Submodel Training in Federated Learning for Membership Inference Attacks Mitigation

Anonymous authors
Paper under double-blind review

Abstract

Federated Learning (FL) enables organizations to collaboratively train models without sharing raw data, yet exchanged models can remain vulnerable to membership inference attacks (MIAs). This vulnerability often increases with model size, creating a privacy–utility trade-off, since bigger models tend to perform better. Submodel training in FL offers a natural way to limit model exposure by allowing each client to train only a fraction of the global architecture. However, many existing approaches impose structural constraints such as nested submodels, shared bases, full-model clients, coverage of every layer, or auxiliary data at the server. We present StitchFL, a heterogeneous submodel training method that allows clients to train arbitrary subsets of neurons across layers (including skipping layers entirely) and reconstructs the global model without requiring private or auxiliary data at the aggregator. StitchFL combines count-normalised averaging of overlapping slices, pseudo-inverse stitching of skipped layers, and anchor-based output recalibration using synthetic inputs. We instantiate the method for dense, convolutional, and recurrent architectures. Using a dense neural network, StitchFL reaches 0.89 accuracy compared to 0.83 achieved by the second best submodel training baseline. It also outperforms competing submodel methods on the convolutional case and forecasting task. Across six MIAs, restricting clients to 40% of each hidden layer reduces the strongest MIA effect by nearly 68.3%. These results show that flexible submodel allocation can substantially reduce client-side model exposure while preserving most of the utility of full-model federated training.

1 Introduction

Federated Learning (FL) allows several parties to jointly train a model when the training data are partitioned across silos that cannot be grouped for legal, commercial, or privacy reasons (McMahan et al., 2017). Each client trains on its own private dataset and sends only the resulting parameters to an aggregator, which merges them (canonically by weighted averaging, as in FedAvg) and broadcasts the global model back for the next round. Since raw records never leave their owner, FL suits domains such as healthcare, finance, and sensing (Kairouz et al., 2021).

However, although FL provides greater privacy, it remains vulnerable to privacy leakage: inference attacks recover sensitive information directly from the exchanged weights (Hu et al., 2022; Niu et al., 2024), and gradients alone can be inverted to reconstruct samples (Zhu et al., 2019). We focus on *membership inference attacks* (MIAs), which decide whether a given record belonged to the training set (Shokri et al., 2017); membership alone can be disclosive, and conventional FL is especially exposed, as every participant receives the global model each round and is thus a white-box adversary by construction (Nasr et al., 2019; Gupta et al., 2021). In centralised learning, a simple mitigation strategy is to reduce model capacity: smaller networks overfit less and leave a weaker membership footprint (Yeom et al., 2018; Carlini et al., 2022), while larger ones are more accurate but memorise more aggressively. This creates a fundamental privacy–utility trade-off (Németh et al., 2025).

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.

In FL, one simple approach to addressing this problem is to employ smaller models too. However, its inherently decentralised architecture also enables an alternative that combines the advantages of both approaches: *submodel training*. In submodel training, the aggregator partitions the global architecture, each client trains one slice locally, and the aggregator reconstructs the full model (Németh et al., 2025). Reconstruction, however, is not a matter of pasting neurons back into place, and the reconstruction (or fusion) rule determines how much accuracy survives. Existing submodel schemes impose requirements that are restrictive or unrealistic here: that one client train the complete network, that submodels be nested or share a common base, that every client cover every layer, or that the aggregator hold public or partially shared client data (Horváth et al., 2021; Kang et al., 2023; Liang et al., 2025; Chen et al., 2026).

These limitations motivate the need for a more flexible submodel-training framework. We therefore propose STITCHFL, a method for training and fusing submodels under a far freer partitioning while preserving as much full-model accuracy as possible. Before presenting the method, we emphasise a key point of this work: Although the motivation is to limit membership leakage, the central challenge in this field is the fusion step, how to reconstruct the original model from the submodels, losing as little accuracy accuracy as possible. We regard STITCHFL as complementary to orthogonal defences such as differential privacy (Abadi et al., 2016) or secure aggregation (Bonawitz et al., 2017), which act on the exchanged updates rather than on the model capacity exposed to each client. We therefore do not benchmark against these approaches. Hence, we make the following statement.

Scope of this work. We focus on submodel training and, specifically, on the challenge of reconstructing the global model from arbitrarily partitioned client submodels while minimising the resulting loss in predictive performance.

Within this scope, STITCHFL lets each client freely choose which neurons of which layers to train (without requiring nested submodels, a shared base, any client to hold the full architecture, or having at least one neuron per layer) and reconstructs the global model through three data-free primitives: count-normalised averaging of overlapping slices, pseudo-inverse stitching of skipped layers, and anchor-based recalibration of the output layer using synthetic inputs regenerated from a public seed. No client shares raw data, activations, or auxiliary datasets.

The rest of the paper is organised as follows. Section 2 reviews related work on submodel training, Section 3 details STITCHFL methodology, Section 4 reports the experimental evaluation, and Section 5 concludes and outlines future work. Next, the appendix is structured in the following way: Appendix A describes the preliminaries. Appendix B extends the previous methodology. Appendix C details all the parameters for Section 4. And finally, Appendix D provides additional results that complement those reported in the main text.

2 RELATED WORK

As stated above, the motivation of this work is submodel training as a defence against MIA. That methodology, however, is predominantly studied in the context of device heterogeneity, where the goal is to accommodate clients with limited computation, memory or energy. Both motivations lead to the same technical problem: *how to partition a global model and, especially, how to fuse the resulting submodels*. In fact, they are connected, since the transition from one perspective to the other is straightforward as the model capacity assigned to a client determines both the computational cost it must sustain and the amount of information potentially exposed to an attacker, so a low-powered device and a client for which we demand stronger privacy call for the same treatment; a resource-aware allocation rule and a privacy-aware one differ in what determines the budget, not in what is done with it. Privacy-motivated submodel training is scarce, Németh et al. (2025) being the closest antecedent to our work, and it addresses only a specific part of Convolutional Neural Networks (CNNs). We therefore review and compare against the heterogeneity-driven literature, where the fusion problem has been studied in depth. Existing methods fall into two families, according to whether the submodels are derived from client capacity alone or from an estimate of parameter importance.

The first family builds submodels by scaling the global architecture to the resources of each client. **HETEROFL** (Diao et al., 2020) reduces the width of every hidden layer by a fixed shrinkage ratio and assigns each client the upper-left slice of each weight tensor, so that each submodel is a subset

of the next larger one; each parameter is averaged only over the clients whose submodel contains it. **FJORD** (Horváth et al., 2021) generalizes this ordered structure through Ordered Dropout, training subnetworks made of ordered prefixes of neurons or channels, and **NEFL** (Kang et al., 2023) scales both width and depth, combining channel reduction with block skipping in ResNets and ViTs. All three are efficient and broadly applicable, but they are nested by construction and require some client to train the complete network, which constrains how the submodels can be distributed and makes the largest client bear the whole exposure.

To avoid tying coverage to a single large client, **FEDROLEX** (Alam et al., 2022) rotates the selected neurons across rounds, so that every part of the model is trained at some point. **FEDECOVER** (Liang et al., 2025) pursues the same objective by distributing the neurons of every layer among the clients according to their capabilities, although some neurons may still be left untrained in a given round and only gain priority in the next one, which has already been lost for them. **FEDRAC** (Wang et al., 2026) estimates the contribution each client can make, builds nested submodels accordingly, and rebuilds them in later rounds while prioritising the least frequently trained neurons. The limitation shared by this group is that the partitioning is fixed and merely reshaped during training: coverage is achieved because every client eventually trains parameters across the entire model, which is precisely what a privacy-oriented scheme must avoid.

The second family selects which parameters a client trains according to an estimate of their importance. **FIARSE** (Wu et al., 2024) can be read as an importance-aware counterpart of FJORD, since each client sets a threshold determined by its capacity and trains the parameters whose magnitude exceeds it. **FEDLOAD** (Martins et al., 2026) measures importance at the level of units rather than connections, scoring a neuron by the accumulated magnitude of the weights attached to it. Both lower the client load, but their submodels remain nested and one client still carries the full weight of the model. **FEDGSE** (Chen et al., 2026) breaks the nesting by building, for each client, an auxiliary dataset resembling its own data, evaluating the model on it and selecting the parameters that are most influential for that client in particular. **HALOFL** (Lian et al., 2025) scores complete units instead of individual weights, combining their activation magnitude and frequency on local data; the aggregator merges these local importances into a global one and dispatches a submodel to each client according to its capacity, latency and stability. **SUBFLOT** (Jiang et al., 2026) derives a personalised submodel using optimal transport to measure the correspondence between the global model and the client's historical models, recomputing the assignment at every round. Finally, **FEDGRAFT** (Liu et al., 2025) does not prune a single global model at all: it assembles a tree with one shared root and several branches from a model zoo, gives each client the root plus the branch that fits in its memory, and aggregates each branch only across the clients that trained it.

Table 1 provides a summary of the partitioning properties of the different methods. For each method, we always consider the most favourable case allowed by its constraints. The column *Max. client fraction* indicates the largest submodel a client can be assigned, expressed as a fraction of the global model. An entry of β means that the capacity of each submodel can be chosen freely, while 1 means the whole model must be assigned. *Min. achiev. exposure* is the minimum number of times a neuron is trained. *Training Gini* measures how unevenly training is distributed between the most and the least trained neurons with the Gini coefficient Gini (1912). A value of 0 means perfectly balanced training, and a value of 1 means maximal imbalance. The remaining columns report whether the client can choose its own submodel or is dynamically chosen (*Client Chosen*), whether the assignment depends on the submodels of other clients (*Depends on other clients*) or on results from previous rounds (*Depends on previous results*), whether a client can skip entire layers (*Skip layers*), and whether any extra information is required from the clients, such as anchors, an auxiliary dataset or historical models (*Extra client metadata required*).

All these methods address the same problem and trade simplicity, coverage and personalisation in different ways. The great majority, however, require either that one client train the complete network or that every neuron be touched at some point across the rounds. Both conditions are restrictive for our purposes, and they arguably run against the very meaning of submodel training, in which no participant should train the whole model and yet the whole model should be trained collectively. Two further differences are worth noting from our method. First, we instantiate our method for dense, recurrent, convolutional and attention-based architectures, whereas the works above are designed and evaluated almost exclusively on CNNs and, in some cases, Multilayer Perceptrons (MLPs). Second, all of them assume that each client holds at least one neuron in every layer, whereas we impose no such restriction: a client may skip a layer entirely, and the resulting jump is reconnected without

any auxiliary data. Only FEDGSE, HALOFL and SUBFLOT admit a genuinely free allocation, and they are therefore the methods against which we compare in Section 4.

Method	Max. client fraction	Min. achiev. exposure	Training Gini	Client Chosen	Depends on other clients	Depends on previous results	Skip layers	Extra client metadata required
StitchFL	β	1	0	Yes	No	No	Yes	Anchors
HeteroFL	1	K	$\frac{K-1}{3K}$	Yes	Yes	No	No	No
FjORD	1	K	$\frac{K-1}{3K}$	Yes	Yes	No	No	No
NeFL	1	K	$\frac{K-1}{3K}$	Yes	Yes	No	No	No
FedRolex	β	K	0	No	No	Yes	No	No
FedEcover	β	K	0	No	Yes	Yes	No	No
FedRAC	1	K	1	No	Yes	Yes	No	No
FIARSE	1	K	$\frac{K-1}{3K}$	No	No	No	No	No
FedLoad	1	K	$\frac{K-1}{3K}$	No	No	No	No	No
FedGSE	β	1	0	No	No	No	No	Aux. Dat.
HaloFL	β	1	0	No	No	No	No	No
SubFLOT	β	1	0	No	No	No	No	Hist. models
FedGraft	1	1	$\frac{K-1}{3K}$	No	Yes	No	No	No

Table 1: Comparison of the partitioning properties of existing submodel-training methods.

3 METHODOLOGY

We split the aggregator model into smaller client models and merge the trained clients back into one. For every architecture this reduces to two routines (since stitching may not be necessary): *averaging* the overlapping slices clients train (Section 3.1.1), and *readout repair*, since after averaging the output layer no longer matches the features feeding it (Section 3.1.2). We write both once; an architecture then only supplies its slices, its default for untrained cells and its synthetic inputs. We detail Multilayer Perceptrons (MLPs) and Long Short-Term Memory networks (LSTMs). For CNNs, we follow a similar idea to MLPs, so the method is described in Appendix B.3. All architectures are described in Appendix A.1. In this work we do not consider the case where the network is a transformer, but in Appendix B.4.1 we give the idea of how it would be. Additionally, in Appendix B we give more details of our methodology.

3.1 GENERAL FUSION PRIMITIVES

Linear layers are $\mathbf{x} \mapsto \phi(\mathbf{x}W + b)$ with row-vector inputs. Given K clients, client k owns in each hidden layer ℓ a contiguous, possibly empty range $\mathcal{I}_\ell^k = \{s_\ell^k, \dots, e_\ell^k\}$ (s of start and e of end), and *skips* the layer when it is empty; input and output layers are kept whole. Contiguity costs nothing, since renumbering a layer's neurons never changes the network's output Ganju et al. (2018). After training, client k returns the blocks it owned.

3.1.1 AVERAGING THE CLIENT SLICES

For the weight array Θ we keep a running sum S and count C of the same shape, both starting at zero, and for each client owning a slice of Θ,

$$\begin{aligned} S\big[\mathcal{R}^k, \mathcal{C}^k\big] \mathrel{+}= \Theta^k, \qquad C\big[\mathcal{R}^k, \mathcal{C}^k\big] \mathrel{+}= \mathbf{1}, \\ \Theta_w = S_w / C_w \ \text{ if } C_w > 0, \qquad \Theta_w = \Pi_w \ \text{ otherwise,} \end{aligned} \tag{1}$$

where the rows and columns $\mathcal{R}^k, \mathcal{C}^k$ the client's k block occupies are the only architecture-dependent ingredient, w is a connection between neurons, and Π is a default for cells no client trained; in code the recovery is a sum over count with a $\max(C, 1)$ guard. This generalises the HeteroFL rule Németh et al. (2025), which forces every slice to start at index 0, to slices starting anywhere. The only remaining choice is Π: the dense case takes $\Pi = 0$ and repairs skipped layers by "stitching" (Section 3.2.1), the recurrent and transformer cases a shared random template (Section 3.3).

3.1.2 REPAIRING THE OUTPUT LAYER

Each client trained its output layer to read *its own* last hidden layer, so the readout of the fused model is misaligned; we must refit it using no private data. This needs a public seed ξ and a recipe for synthetic *anchors* X^{anc} that every party regenerates locally, so only the seed is transmitted, plus the map g from an input to the last hidden features of the merged model. Clients return $f_k(X^{\mathrm{anc}}) \in \mathbb{R}^{N\times n_{L+1}}$; the aggregator averages them into a *target* and refits $\Theta = [W^{(L+1)}; b^{(L+1)}]$ against the merged features $H = g(X^{\mathrm{anc}})$, bias absorbed by a column of ones in $\widetilde{H} = [H \mid \mathbf{1}]$, by ridge regression,

$$Y^{\mathrm{tea}} = \frac{1}{K}\sum_{k=1}^{K} f_k\big(X^{\mathrm{anc}}\big), \tag{2}$$

$$\Theta^{\star} = \arg\min_{\Theta} \big\|\widetilde{H}\Theta - Y^{\mathrm{tea}}\big\|_F^2 + \lambda\|\Theta\|_F^2 = \big(\widetilde{H}^{\top}\widetilde{H} + \lambda I\big)^{-1}\widetilde{H}^{\top}\, Y^{\mathrm{tea}}. \tag{3}$$

The target is an ensemble average evaluated on synthetic anchors and therefore does not require transmitting raw data, gradients, or intermediate activations. Since the returned predictions are nevertheless data-dependent model outputs, we do not assume them to be leakage-free. If stronger privacy guarantees are required, clients may add calibrated noise to $f_k(X^{\mathrm{anc}})$ to make equation 2 differentially private at no extra cost. Each architecture below only specifies its anchors and its g.

3.2 MULTILAYER PERCEPTRON

The aggregator model in this section is an MLP with L hidden layers and n_i neurons in layer l_i. A weight between consecutive layers u and $u+1$ is accumulated by equation 1 on rows $\mathcal{I}_u^k$ and columns $\mathcal{I}_{u+1}^k$—buffers $(S^{(u)}, C^{(u)})$ for the weight, $(\sigma^{(u)}, \kappa^{(u)})$ for the bias—and recovered at $\Pi = 0$. A client that gets no neuron in some layer instead has a *skip-edge*: one weight jumping over the missing layer, which connects the wrong pair of layers and so cannot be averaged this way. The cells it leaves empty are filled by stitching, not by the $\Pi = 0$ default.

3.2.1 STITCHING SKIP-EDGES

We *push* a skip-edge $u \to v$ with $v > u+1$ and weights (W_A, b_A) forward one layer at a time, borrowing the next normal block $u \to u{+}1$ that other clients have already filled in,

$$W_B = S^{(u)}[\mathcal{R},\mathcal{C}]/C^{(u)}[\mathcal{R},\mathcal{C}], \qquad b_B = \sigma^{(u)}[\mathcal{C}]/\kappa^{(u)}[\mathcal{C}], \tag{4}$$

with $\mathcal{R} \subseteq \mathcal{I}_u^k$ the usable rows and $\mathcal{C}$ the columns that already carry data. Requiring the jump to be reproduced through that borrowed block, and treating the activation as roughly linear over the region of interest Frankle et al. (2020), determines the replacement (W', b') in closed form (Appendix B.1.1),

$$\boxed{W' = W_B^{+}\, W_A[\mathcal{R},:], \qquad b' = b_A - b_B\, W',} \tag{5}$$

with W_B^{+} the pseudo-inverse, in code a least-squares solve. The jump now goes $u{+}1 \to v$: if $u{+}1 = v$ it is a normal edge and is averaged via equation 1, otherwise it is pushed again, the common case $v = u+2$ being a single push. Each new client fills more of $C^{(u)}$ and may unlock jumps blocked before, so the queue is re-scanned after every client, up to $L+2$ rounds (Algorithm 1). Crucially, equation 5 uses only the aggregator's partial average and the client's own jump weights—no inputs, activations or labels—unlike methods that need real samples.

3.2.2 REPAIRING THE OUTPUT LAYER

The repair of Section 3.1.2 runs with clipped Gaussian anchors: for $z_{ij} \overset{\mathrm{iid}}{\sim} \mathcal{N}(0,1)$ drawn from the public seed ξ and a clipping level $\tau > 0$, the anchors are $X_{ij}^{\mathrm{anc}} = \mathrm{clip}(z_{ij}, -\tau, \tau)$, and with g the merged hidden stack they give $H \in \mathbb{R}^{N\times n_L}$; Equations 2–3 then give the repaired output layer.

3.3 LONG-SHORT TERM MEMORY

The aggregator model stacks recurrent layers of widths $n_1, \ldots, n_L$ over sequences of length T with n_0 features per step, ending in a linear output layer of width n_{L+1}. Each layer is a gated cell with

G gates ($G = 4$ for an LSTM, 3 for a GRU, 1 for a plain RNN), with gate pre-activations $\mathbf{z}_t^{(\ell)} = \mathbf{h}_t^{(\ell-1)} W^{(\ell)} + \mathbf{h}_{t-1}^{(\ell)} U^{(\ell)} + b^{(\ell)}$ and state $\mathbf{h}_t^{(\ell)} = \Phi^{(\ell)}(\mathbf{z}_t^{(\ell)}, \mathbf{c}_{t-1}^{(\ell)})$, where $\Phi^{(\ell)}$ is the gate logic plus cell-state update, $W^{(\ell)} \in \mathbb{R}^{n_{\ell-1} \times G n_\ell}$, $U^{(\ell)} \in \mathbb{R}^{n_\ell \times G n_\ell}$, $b^{(\ell)} \in \mathbb{R}^{G n_\ell}$ and $\hat{y} = \mathbf{h}_T^{(L)} W^{(L+1)} + b^{(L+1)}$; unit j owns the G columns $\{j + g n_\ell\}_{g=0}^{G-1}$ and $\mu(c) = c \bmod n_\ell$ is the unit owning column c. Clients again get contiguous unit slices $\mathcal{I}_\ell^k$; the one new twist versus the MLP is that picking a unit picks all G of its gate columns, so with $\mathcal{G}_\ell(\mathcal{I})$ the gate-column expansion of a unit set (Appendix B.1.2) a client's weights map into the aggregator arrays as the slices

$$W_k^{(\ell)} \hookrightarrow W^{(\ell)}\big[\mathcal{I}_{\ell-1}^k,\, \mathcal{G}_\ell(\mathcal{I}_\ell^k)\big], \quad U_k^{(\ell)} \hookrightarrow U^{(\ell)}\big[\mathcal{I}_\ell^k,\, \mathcal{G}_\ell(\mathcal{I}_\ell^k)\big], \quad b_k^{(\ell)} \hookrightarrow b^{(\ell)}\big[\mathcal{G}_\ell(\mathcal{I}_\ell^k)\big]. \tag{6}$$

Averaging recurrent units requires consistent alignment across clients, which is not guaranteed under independent initialisations. We avoid explicit unit matching by initialising all clients from corresponding slices of a shared random template ($\bar{\theta}$) generated from the public seed (ξ). The aggregator therefore draws one random template $\bar{\theta}$ from the public seed ξ and hands each client the matching slice of equation 6 as its *initial weights*. Every array is then accumulated by equation 1 on those slices (the output layer on rows $\mathcal{I}_L^k$) and recovered at $\Pi = \bar{\theta}$, so a layer that every client skips simply keeps its template values; unlike zeros, the template does not switch those gates off and wipe the cell's memory. Which cells receive real data is decided by the slices alone—a recurrent link needs one client holding both of its endpoints—so overlap between clients buys coverage, and a few clients with large overlap are preferable to many tiny disjoint ones (Appendix B.1.4).

3.3.1 Repairing the output layer

Here the anchors of Section 3.1.2 look like real sequences: for a correlation level $\rho \in [0, 1)$, $X^{\text{anc}} \in \mathbb{R}^{N \times T \times n_0}$ follows an AR(1) process drawn from the public seed ξ,

$$X_{:,0,:}^{\text{anc}} \sim \mathcal{N}(0, I), \qquad X_{:,t,:}^{\text{anc}} = \rho\, X_{:,t-1,:}^{\text{anc}} + \sqrt{1 - \rho^2}\, \boldsymbol{\epsilon}_t, \quad \boldsymbol{\epsilon}_t \overset{\text{iid}}{\sim} \mathcal{N}(0, I), \tag{7}$$

where $\sqrt{1 - \rho^2}$ keeps each step at unit variance, matching standardised inputs, and $\rho > 0$ supplies the step-to-step correlation needed to exercise the cell's memory. With g returning the final top-layer state $H = \mathbf{h}_T^{(L)}$, equation 2–equation 3 repair $(W^{(L+1)}, b^{(L+1)})$ and leave the recurrent weights unchanged.

4 Results

In this section, we present and analyse the main experimental results.

4.1 Evaluation Overview and Baselines

Appendix C provides the complete experimental configuration of our experiments. The main body is organised in three parts. Section 4.2.1 quantifies the privacy gained by submodel training, comparing the membership leakage of the client submodels against that of the complete server model. Section 4.2.2 compares the utility of the fused model obtained by each method across datasets and architectures. Finally, Section 4.2.3 reports the privacy–utility trade-off of each method as the submodel size varies. Additional results, including the study on MIA on generic model sizes, one-shot fusion and an ablation study, are deferred to Appendix D. Throughout this section we compare StitchFL against three representative state-of-the-art submodel-training baselines: FedGSE, HaloFL, and SubFLOT. SubFLOT was implemented using the code provided in `https://github.com/jiangz20/SubFLOT`, whereas FedGSE and HaloFL are implemented by us, as no official implementation is publicly available.

StitchFL recalibrates the output layer using 20000 synthetic anchors. HaloFL receives the per-layer widths as its budget and scores units by importance ($\alpha = 0.5$, $\theta = 0$, retention in $[0, 0.85]$). FedGSE builds its public set from 260 synthetic samples per client, drawn from a Gaussian fitted to the local features. SubFLOT aligns submodels by exact-EMD optimal transport with mixing $\alpha = 0.5$ and subspace-anchored regularisation $\lambda = 0.01$.

Finally, to make the comparison fair and interpretable, every method follows exactly the same evaluation protocol. In the case of StitchFL, the submodel division is the one made in Appendix C, and

for the remaining methods, clients are assigned the same capacity budgets as in STITCHFL. Then, this fused model is evaluated in the testing dataset of each client, and we take the average. Specifically, each client owns an $80\%/20\%$ train/test split of its own data. We consider federated settings with three or five clients. This deliberately small scale makes the submodel allocation easier to control, visualise, and interpret, allowing a clearer analysis of the partitioning and fusion behaviour. Finally, this process is repeated 10 times, and the figures show the statistics of these 10 runs.

4.2 EXPERIMENTS

4.2.1 PRIVACY GAINED

This section quantifies how much membership leakage is removed by training submodels instead of the complete model. All methods are evaluated under the same per-client capacity budgets, so, in this case, what an external adversary with white-box access can extract depends only on the submodel it receives, not on the fusion rule. We therefore report a single comparison between the complete server model and the five client submodels, obtained for the MLP on Letter Recognition. The six membership inference attacks and the evaluation metrics are described in detail in Appendix D.1.1. In short, the AUC measures how well the attacker separates members from non-members across all decision thresholds, with 0.5 corresponding to random guessing. The TPR at a fixed low FPR (1%, 5%, 10%) measures the fraction of members the attacker identifies with high confidence, which captures the worst-case privacy breach.

Figure 1 shows that every client submodel leaks less than the complete model under the four strongest attacks. RMIA is the most effective attack against the complete model, with an AUC of 0.69, which drops to 0.56–0.60 on the submodels. LiRA falls from 0.67 to 0.57–0.60, TMIA from 0.65 to 0.55–0.59 and Attack-R from 0.63 to 0.57–0.61. The gain is more pronounced in the low-FPR regime. At 1% FPR, RMIA recovers 9.8% of the members of the complete model but only 1.3%–3.5% of those of the submodels, and LiRA drops from 8.4% to 2.2%–3.8%, a three- to seven-fold reduction that brings several submodels close to the random baseline. The same trend holds at 5% and 10% FPR, where the TPR of RMIA falls from 0.20 to 0.07–0.10 and from 0.28 to 0.12–0.17, respectively. The Quantile and Yeom attacks are close to chance already on the complete model (AUC of 0.57 and 0.56, and TPR approximately equal to the FPR), so submodel training has little leakage left to remove against them. Finally, the residual leakage differs across clients, with Clients 2 and 4 retaining the most, which indicates that the privacy of the scheme is determined by the most exposed submodel.

4.2.2 UTILITY OF THE DIFFERENT METHODS

This section compares the utility of the fused global model obtained by each method across the three architectures. For each task we report its most representative metric: accuracy for the MLP and the CNN, and the coefficient of determination R^2 for the LSTM. The remaining metrics are deferred to Appendix D.2. As a reference we include FEDAVG trained on the complete model, whose performance a submodel-training method aspires to recover, so the distance to it measures the utility lost by partitioning. Each boxplot in Figure 2 summarises 10 independent runs, marking the median.

On the **MLP**, before presenting the comparison, we first establish the two main reference bounds: FEDAVG trained with the complete global model, which represents the upper utility bound, and FEDAVG trained with the smallest considered submodel, which provides the corresponding lower-capacity reference (but more private). These two settings achieve 0.95 and 0.85, respectively. Looking at the other methods, STITCHFL achieves 0.89, the highest among the submodel training methods. FEDGSE follows closely at 0.83, whereas SUBFLOT (0.69) and HALOFL (0.62) fall considerably behind and also show the widest boxplots, with runs ranging from 0.57 to 0.70 for HALOFL. On the **CNN**, the gap to the complete model widens for every method, as expected from the larger capacity required by CIFAR-10. FEDAVG attains 0.71, the lower-capacity bound reaches 0.54, and STITCHFL again leads the submodel-training methods with 0.58, ahead of HALOFL (0.54), FEDGSE (0.51) and SUBFLOT (0.32). The ranking of the baselines changes between the two architectures, since FEDGSE is second on the MLP but third on the CNN, while STITCHFL is consistently first. On the **LSTM**, only STITCHFL and FEDAVG are shown, since none of the baselines defines an extraction or fusion rule for recurrent networks. STITCHFL reaches a median R^2 of

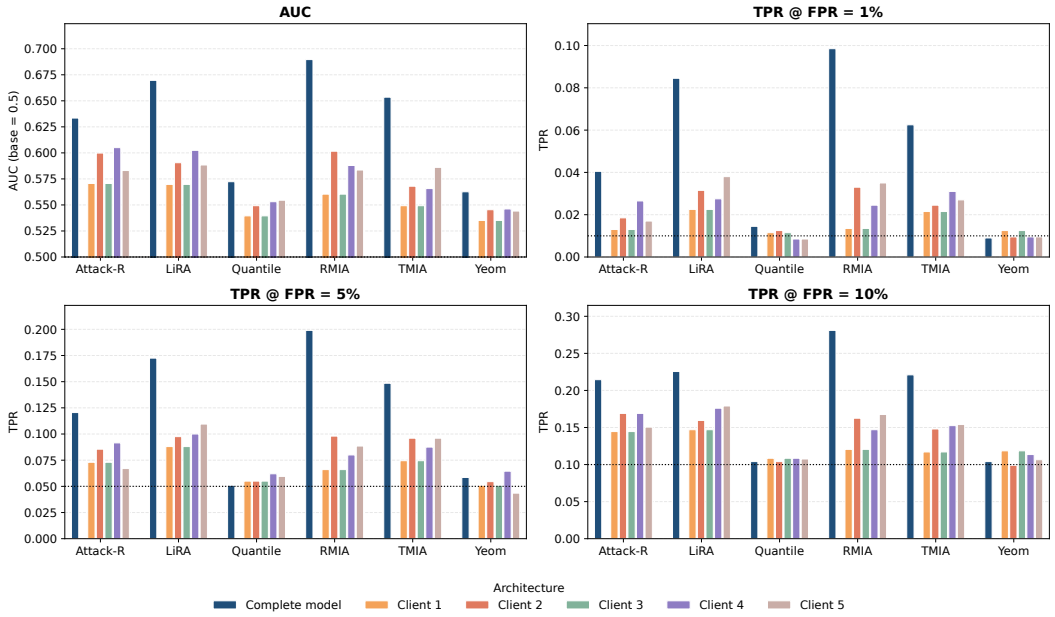


Figure 1: Membership leakage of the complete server model and of the five client submodels.

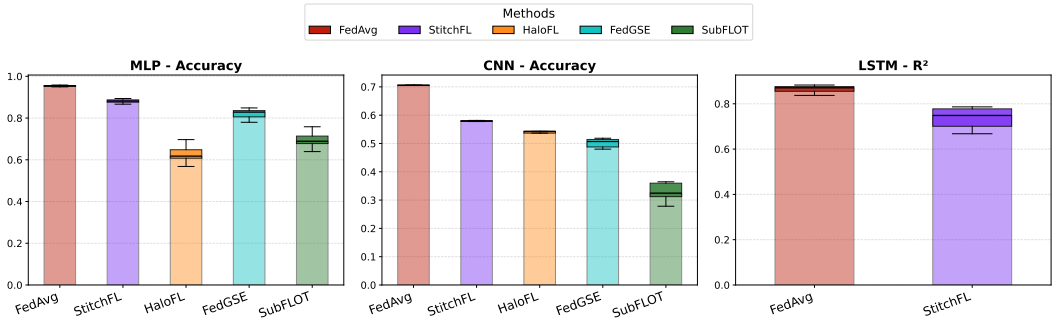


Figure 2: Utility of the fused global model for each method.

0.75 against 0.87 for FEDAVG, retaining about 87% of the explained variance even though no client trains more than roughly half of each recurrent layer, at the cost of a larger variance across runs (0.67–0.79). The corresponding lower bound for FEDAVG is 0.71.

4.2.3 TRADE-OFF PRIVACY-UTILITY OF THE DIFFERENT METHODS

The previous experiments used a configuration mixing submodels of different sizes and overlaps. Here we study how privacy and accuracy evolve as client submodels grow. We restrict the analysis to an MLP with three hidden layers of 60 neurons each and consider $[12, 12, 12]$, $[24, 24, 24]$, $[36, 36, 36]$, and $[60, 60, 60]$ submodels, the last being equivalent to FEDAVG. Privacy is measured as the average of six membership inference attacks. We use submodel capacity as a common proxy for exposure, so methods with the same capacity share the same privacy level and differ only in accuracy (Figure 3).

The four panels show the same trend: as submodels grow, more neurons are shared, the setting approaches FEDAVG, and the accuracies naturally converge. The most informative regime is therefore the low-capacity one. With $[12, 12, 12]$ submodels, STITCHFL reaches 0.784 accuracy, compared with 0.727 for FEDGSE, 0.692 for SUBFLOT, and 0.345 for HALOFL, while operating at the same privacy level. As capacity increases, the differences shrink: at $[36, 36, 36]$ FEDGSE is only

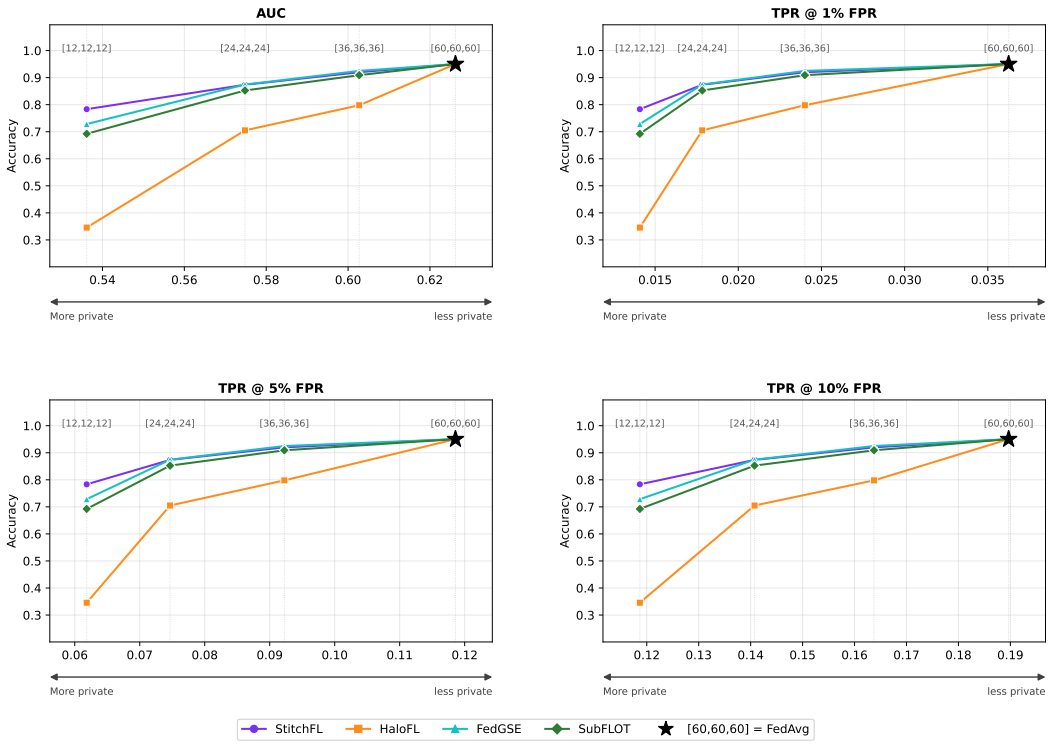


Figure 3: Privacy vs. accuracy of the different methods increasing the model size.

marginally ahead (0.922 vs. 0.913), and at $[60, 60, 60]$ all methods coincide with FEDAVG by construction.

5 CONCLUSIONS

In this work, we presented STITCHFL, a submodel-training method for FL that splits a full server model into heterogeneous client submodels and reconstructs it at the aggregator. By enabling each client to operate on only a small fraction of the global architecture, STITCHFL facilitates the use of low-capacity client submodels, which our experiments show to be substantially less vulnerable to MIA than the complete model, while preserving much of the global predictive utility. Crucially, and in keeping with the core premise of FL, this is achieved without any client sharing sensitive raw data: the fusion relies solely on count-normalised averaging, data-free stitching through the Moore–Penrose pseudo-inverse, and anchor-based readout recalibration, all of which operate on model parameters and synthetic anchors rather than on private samples. Our experiments confirm that STITCHFL achieves this privacy–utility trade-off while allowing unrestricted network partitioning, and consistently outperforms state-of-the-art submodel training approaches across different neural architectures. As future work, we aim to further reduce the performance gap induced by partitioning by exploring new fusion and recalibration techniques that better preserve the behaviour of the FEDAVG algorithm, especially in non-IID environments. In addition, we intend to deepen the study of submodel training in the context of resource-constrained devices, building on the neuron-importance principles that underpin several of the methods reviewed in this paper, so as to combine their device-aware efficiency with the reduced model exposure and architectural generality enabled by STITCHFL.

AI DISCLOSURE

In this work, we used generative AI tools primarily to improve writing quality, readability, and grammatical correctness. Generative AI was also used to assist with software implementation, including

scripts for running experiments and generating figures from the obtained results. In addition, AI-assisted coding was used during the implementation of our proposed STITCHFL method, as well as the HaloFL, FedGSE, and SubFLOT baselines, to facilitate implementation and improve code correctness and efficiency. All AI-assisted code was manually reviewed, tested, and validated by the authors.

Generative AI tools were not used to conduct the literature review, formulate the research problem, design the structure of the paper, or develop the underlying methodology and conceptual design of STITCHFL, which were carried out by the authors. The authors have reviewed all AI-assisted content and take full responsibility for the final text, implementations, experimental results, and claims presented in this work.

REPRODUCIBILITY STATEMENT

To promote transparent and reproducible research, and to facilitate future work building upon our results, we make the complete implementation used in this study publicly available in an anonymized repository at `https://anonymous.4open.science/r/StitchFL-4FB8/`. The repository contains the source code required to reproduce the proposed method and the experimental evaluation presented in this work.

REFERENCES

Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep Learning with Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 308–318. ACM, 2016. doi: 10.1145/2976749.2978318.

Samiul Alam, Luyang Liu, Ming Yan, and Mi Zhang. FedRolex: Model-Heterogeneous Federated Learning with Rolling Sub-Model Extraction. *Advances in Neural Information Processing Systems*, 35:29677–29690, December 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/hash/bf5311df07f3efce97471921e6d2f159-Abstract-Conference.html`.

Martin Bertran, Shuai Tang, Aaron Roth, Michael Kearns, Jamie H. Morgenstern, and Steven Wu. Scalable Membership Inference Attacks via Quantile Regression. *Advances in Neural Information Processing Systems*, 36:314–330, December 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/hash/01328d0767830e73a612f9073e9ff15f-Abstract-Conference.html`.

Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191. ACM, 2017. doi: 10.1145/3133956.3133982.

Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramèr. Membership inference attacks from first principles. In *2022 IEEE Symposium on Security and Privacy (SP)*, pp. 1897–1914. IEEE, 2022. doi: 10.1109/SP46214.2022.9833649.

Dingfan Chen, Ning Yu, Yang Zhang, and Mario Fritz. Gan-leaks: A taxonomy of membership inference attacks against generative models. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 343–362, 2020. doi: 10.1145/3372297.3417238.

Genlang Chen, Yabo Jia, Haozhao Wang, Chaoyi Pang, and Wenchao Xu. FedGSE: Gradient-Based Submodel Extraction for Resource-Constrained Federated Learning. *IEEE Transactions on Mobile Computing*, 25(6):7735–7747, June 2026. ISSN 1536-1233, 1558-0660, 2161-9875. doi: 10.1109/TMC.2025.3647623. URL `https://ieeexplore.ieee.org/document/11314781/`.

Enmao Diao, Jie Ding, and Vahid Tarokh. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. *arXiv preprint arXiv:2010.01264*, 2020.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International conference on machine learning*, pp. 3259–3269. PMLR, 2020.

Peter W. Frey and David J. Slate. Letter recognition using Holland-style adaptive classifiers. *Machine Learning*, 6(2):161–182, 1991.

Karan Ganju, Qi Wang, Wei Yang, Carl A Gunter, and Nikita Borisov. Property inference attacks on fully connected neural networks using permutation invariant representations. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pp. 619–633, 2018.

Corrado Gini. *Variabilità e mutabilità: contributo allo studio delle distribuzioni e delle relazioni statistiche.[Fasc. I.]*. Tipogr. di P. Cuppini, 1912.

Umang Gupta, Dimitris Stripelis, Pradeep K. Lam, Paul Thompson, Jose Luis Ambite, and Greg Ver Steeg. Membership Inference Attacks on Deep Regression Models for Neuroimaging. In *Proceedings of the Fourth Conference on Medical Imaging with Deep Learning*, pp. 228–251. PMLR, August 2021. URL https://proceedings.mlr.press/v143/gupta21a.html.

Jamie Hayes, Luca Melis, George Danezis, and Emiliano De Cristofaro. Logan: Membership inference attacks against generative models. *arXiv preprint arXiv:1705.07663*, 2017.

Benjamin Hilprecht, Martin Härterich, and Daniel Bernau. Monte carlo and reconstruction membership inference attacks against generative models. *Proceedings on Privacy Enhancing Technologies*, 2019(4):232–249, 2019. doi: 10.2478/popets-2019-0067.

Samuel Horváth, Stefanos Laskaridis, Mario Almeida, Ilias Leontiadis, Stylianos Venieris, and Nicholas Lane. FjORD: Fair and Accurate Federated Learning under heterogeneous targets with Ordered Dropout. In *Advances in Neural Information Processing Systems*, volume 34, pp. 12876–12889. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/hash/6aed000af86a084f9cb0264161e29dd3-Abstract.html.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, Philip S. Yu, and Xuyun Zhang. Membership Inference Attacks on Machine Learning: A Survey. *ACM Computing Surveys*, 54(11s):1–37, January 2022. ISSN 0360-0300, 1557-7341. doi: 10.1145/3523273. URL https://dl.acm.org/doi/10.1145/3523273.

Antonio Martínez Ibarra, Aurora González-Vidal, and Antonio Skarmeta. Pleiadata: consumption, hvac, temperature, weather and motion sensor data for smart buildings applications. *Scientific Data*, 10(1):118, 2023.

Zheng Jiang, Nan He, Yiming Chen, and Lifeng Sun. Submodel Extraction for Efficient and Personalized Federated Learning via Optimal Transport. pp. 3369–3378, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Jiang_Submodel_Extraction_for_Efficient_and_Personalized_Federated_Learning_via_Optimal_CVPR_2026_paper.html.

Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, 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.

Honggu Kang, Seohyeon Cha, Jinwoo Shin, Jongmyeong Lee, and Joonhyuk Kang. NeFL: Nested Federated Learning for Heterogeneous Clients. October 2023. URL https://openreview.net/forum?id=Hm6maU150b.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Zirui Lian, Qianyue Cao, Chao Liang, Jing Cao, Zongwei Zhu, Zhi Yang, Cheng Ji, Changlong Li, and Xuehai Zhou. HaloFL: Efficient Heterogeneity-Aware Federated Learning Through Optimal Submodel Extraction and Dynamic Sparse Adjustment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 44(9):3518–3531, September 2025. ISSN 0278-0070, 1937-4151. doi: 10.1109/TCAD.2025.3548003. URL https://ieeexplore.ieee.org/document/10909716/.

Juntao Liang, Lan Zhang, Xiangmou Qu, and Jun Wang. FedEcover: Fast and Stable Converging Model-Heterogeneous Federated Learning with Efficient-Coverage Submodel Extraction. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, pp. 2575–2587, Hong Kong, Hong Kong, May 2025. IEEE. ISBN 9798331536039. doi: 10.1109/ICDE65448.2025.00194. URL https://ieeexplore.ieee.org/document/11113012/.

Ruixuan Liu, Ming Hu, Zeke Xia, Xiaofei Xie, Jun Xia, Pengyu Zhang, Yihao Huang, and Mingsong Chen. FedGraft: Memory-Aware Heterogeneous Federated Learning via Model Grafting. *IEEE Transactions on Mobile Computing*, 24(12):13506–13519, December 2025. ISSN 1536-1233, 1558-0660, 2161-9875. doi: 10.1109/TMC.2025.3591537. URL https://ieeexplore.ieee.org/document/11088243/.

Yiyong Liu, Zhengyu Zhao, Michael Backes, and Yang Zhang. Membership Inference Attacks by Exploiting Loss Trajectory. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2085–2098, Los Angeles CA USA, November 2022. ACM. ISBN 9781450394505. doi: 10.1145/3548606.3560684. URL https://dl.acm.org/doi/10.1145/3548606.3560684.

Bruno Martins, Eric Samikwa, Torsten Braun, Denis Rosário, Eduardo Cerqueira, and Leandro Villas. FedLoad: Adaptive Partial Training for Model Heterogeneous Federated Learning. In *2026 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, Kuala Lumpur, Malaysia, April 2026. IEEE. ISBN 9798331577292. doi: 10.1109/WCNC65185.2026.11555362. URL https://ieeexplore.ieee.org/document/11555362/.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 54 of *Proceedings of Machine Learning Research*, pp. 1273–1282. PMLR, April 2017. URL https://proceedings.mlr.press/v54/mcmahan17a.html.

Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 739–753. IEEE, 2019. doi: 10.1109/SP.2019.00065.

Jun Niu, Peng Liu, Xiaoyan Zhu, Kuo Shen, Yuecong Wang, Haotian Chi, Yulong Shen, Xiaohong Jiang, Jianfeng Ma, and Yuqing Zhang. A survey on membership inference attacks and defenses in machine learning. *Journal of Information and Intelligence*, 2(5):404–454, September 2024. ISSN 29497159. doi: 10.1016/j.jiixd.2024.02.001. URL https://linkinghub.elsevier.com/retrieve/pii/S2949715924000064.

Gergely D. Németh, Miguel Ángel Lozano, Novi Quadrianto, and Nuria Oliver. Privacy and Accuracy Implications of Model Complexity and Integration in Heterogeneous Federated Learning. *IEEE Access*, 13:40258–40274, 2025. ISSN 2169-3536. doi: 10.1109/ACCESS.2025.3546478. URL https://ieeexplore.ieee.org/document/10906485/.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership Inference Attacks Against Machine Learning Models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 3–18, San Jose, CA, USA, May 2017. IEEE. ISBN 9781509055333. doi: 10.1109/SP.2017.41. URL http://ieeexplore.ieee.org/document/7958568/.

Boris Van Breugel, Hao Sun, Zhaozhi Qian, and Mihaela van der Schaar. Membership inference attacks against synthetic data through overfitting detection. *arXiv preprint arXiv:2302.12580*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Zihui Wang, Yuhang Fu, Mengmeng Du, Zhimin Yuan, Yachen Liu, Weisheng Liao, Kaiyu Wang, and Zheng Wang. FedRAC: Rolling Submodel Allocation for Collaborative Fairness in Federated Learning. pp. 31802–31811, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Wang_FedRAC_Rolling_Submodel_Allocation_for_Collaborative_Fairness_in_Federated_Learning_CVPR_2026_paper.html.

Joshua Ward, Chi-Hua Wang, and Guang Cheng. Data plagiarism index: Characterizing the privacy risk of data-copying in tabular generative models. *arXiv preprint arXiv:2406.13012*, 2024.

Joshua Ward, Chi-Hua Wang, and Guang Cheng. Privacy auditing synthetic data release through local likelihood attacks. *arXiv preprint arXiv:2508.21146*, 2025a.

Joshua Ward, Yuxuan Yang, Chi-Hua Wang, and Guang Cheng. Ensembling Membership Inference Attacks Against Tabular Generative Models. In *Proceedings of the 18th ACM Workshop on Artificial Intelligence and Security*, pp. 182–193, Taipei , Taiwan, October 2025b. ACM. ISBN 9798400718953. doi: 10.1145/3733799.3762977. URL https://dl.acm.org/doi/10.1145/3733799.3762977.

Feijie Wu, Xingchen Wang, Yaqing Wang, Tianci Liu, Lu Su, and Jing Gao. FIARSE: Model-Heterogeneous Federated Learning via Importance-Aware Submodel Extraction. *Advances in Neural Information Processing Systems*, 37:115615–115651, December 2024. doi: 10.52202/079017-3672. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/d17477a5ff8b7ddb49e53f9c04305da5-Abstract-Conference.html.

Jiayuan Ye, Aadyaa Maddi, Sasi Kumar Murakonda, Vincent Bindschaedler, and Reza Shokri. Enhanced membership inference attacks against machine learning models. In *Proceedings of the 2022 ACM SIGSAC conference on computer and communications security*, pp. 3093–3106, 2022.

Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. Privacy Risk in Machine Learning: Analyzing the Connection to Overfitting. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pp. 268–282, Oxford, July 2018. IEEE. ISBN 9781538666807. doi: 10.1109/CSF.2018.00027. URL https://ieeexplore.ieee.org/document/8429311/.

Sajjad Zarifzadeh, Philippe Liu, and Reza Shokri. Low-Cost High-Power Membership Inference Attacks, December 2023. URL https://arxiv.org/abs/2312.03262v3.

Ligeng Zhu, Zhijian Liu, and Song Han. Deep Leakage from Gradients. In *Advances in Neural Information Processing Systems*, volume 32, pp. 14747–14756. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/hash/60a6c4002cc7b29142def8871531281a-Abstract.html.

A Preeliminaries

A.1 Neural Network Structures

To understand how we split the main model into the small ones, we define the structures of the DL models considered in this work.

A.1.1 The multilayer perceptron

The MLP is a dense neural architecture composed of a sequence of fully connected layers. Given an input vector $\mathbf{x} \in \mathbb{R}^d$, each layer applies an affine transformation followed by a nonlinear activation function:

$$\mathbf{h}^{(l)} = \sigma\left(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}\right),$$

where $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ denote the weight matrix and bias vector of layer l, respectively, and $\sigma(\cdot)$ represents an activation function such as ReLU, sigmoid, or tanh.

A.1.2 Convolutional neural networks

CNNs are designed to exploit local structures and spatial patterns through shared convolutional filters. In a convolutional layer, a set of learnable kernels is applied across the input to extract local features, reducing the number of trainable parameters compared with fully connected architectures and promoting translation equivariance. Formally, a convolutional operation can be expressed as:

$$\mathbf{H}_{i,j,k} = \sigma\left(\sum_{u,v,c} \mathbf{X}_{i+u,j+v,c}\mathbf{K}_{u,v,c,k} + b_k\right),$$

where $\mathbf{X}$ denotes the input tensor, $\mathbf{K}$ the learnable filters, and $\mathbf{H}$ the resulting feature map. CNNs usually combine convolutional layers with normalization operations, nonlinear activation functions, and dimensionality reduction mechanisms such as pooling. This enables the construction of hierarchical representations, ranging from low-level local patterns to more abstract semantic features. Moreover, CNN architectures commonly include an MLP-based classification head after the convolutional feature extractor. In this configuration, the convolutional layers act as hierarchical feature extractors, while the fully connected layers map the learned representation into the target output space.

A.1.3 Long Short-Term Memory networks

Long Short-Term Memory networks (LSTMs) are recurrent architectures specifically designed to capture long-range temporal dependencies while mitigating the vanishing gradient problem. Their structure relies on an internal memory cell $\mathbf{c}_t$ and three gating mechanisms: forget, input, and output gates. These gates regulate which information is discarded, incorporated, and exposed at each time step:

$$\begin{aligned}
\mathbf{f}_t &= \sigma\left(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f\right), \\
\mathbf{i}_t &= \sigma\left(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i\right), \\
\tilde{\mathbf{c}}_t &= \tanh\left(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c\right), \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t, \\
\mathbf{o}_t &= \sigma\left(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o\right), \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t).
\end{aligned} \tag{8}$$

Through this gated formulation, LSTMs are particularly suitable for time series, sequential signals, and event streams, where predictions may depend not only on recent observations but also on information accumulated over extended temporal horizons.

A.2 Membership Inference Attacks

Machine Learning models retain information about their training data, opening a channel through which private information leaks even when the raw data never leaves its owner. Privacy attacks target two assets: the model itself and the training data Niu et al. (2024). Against training-data privacy, model inversion attacks reconstruct sensitive attributes of a record, and property inference attacks recover global properties of the training distribution Hu et al. (2022). MIAs, the focus of this work, ask the most elementary question: given a data point x and access to a target model $\mathbb{F}_T$, was x part of the training set $\mathcal{D}_{\text{train}}$? Formally, an MIA is a binary decision rule

$$\mathbb{F}_a : \{\mathbb{F}_T(x_T),\, \mathcal{K}\} \Rightarrow \{0, 1\}, \tag{9}$$

where $\mathcal{K}$ is the adversary's auxiliary knowledge and the output 1 indicates membership Niu et al. (2024). Despite emitting a single bit, MIA is the canonical measure of privacy leakage: membership alone can be disclosive—establishing that a clinical record trained a disease-specific model reveals the diagnosis—and it is a precondition for stronger reconstruction attacks. Regulatory frameworks such as NIST and GDPR regard membership disclosure as a confidentiality violation Shokri et al. (2017); Hu et al. (2022).

Adversarial knowledge is conventionally split into three threat models Niu et al. (2024): *black-box*, where only model outputs are observed; *gray-box*, which adds knowledge of the data distribution, enabling shadow datasets disjoint from $\mathcal{D}_{\text{train}}$ Shokri et al. (2017); and *white-box*, where the attacker holds the parameters and hence per-layer activations and loss gradients Gupta et al. (2021). Conventional full-model FL naturally exposes clients to a white-box setting: each participant receives the complete global model at every round and may observe its evolution across rounds, while a curious server may additionally observe individual client updates. Submodel training changes this exposure by restricting each client to the parameters contained in its assigned submodel.

Membership is decidable because models behave differently on data they memorised: this *generalisation gap* from overfitting is a sufficient condition for vulnerability Yeom et al. (2018), though leakage also arises from unintended memorisation of rare records and from the gradient footprint of each sample Niu et al. (2024). Attacks exploit these signals either as *binary-classifier* methods—the shadow-training technique of Shokri et al. Shokri et al. (2017) trains k shadow models and fits a classifier on their member/non-member outputs—or as *metric-based* methods, which compute a scalar score (loss, confidence, prediction entropy) and threshold it Yeom et al. (2018). The recent lineage refines *what* score to compute and *against what* to compare it. A single global threshold wrongly assumes identical member and non-member distributions for every record, so a hard non-member may score worse than an easy member Carlini et al. (2022). Calibrating per record against reference models trained *without* the target reframes MIA as a likelihood-ratio test between the worlds *the model saw* x and *it did not* Carlini et al. (2022); Ye et al. (2022); later work cuts this cost via pairwise population comparisons Zarifzadeh et al. (2023), a single quantile regressor Bertran et al. (2023), or the loss trajectory across training Liu et al. (2022). This also shifted the metric toward the true positive rate at low false positive rates, the regime that constitutes a real breach Carlini et al. (2022); Ward et al. (2025b). The attacks evaluated in Section 4—Yeom Yeom et al. (2018), LiRA Carlini et al. (2022), RMIA Zarifzadeh et al. (2023), TMIA Liu et al. (2022), the quantile attack Bertran et al. (2023), and Attack-R Ye et al. (2022).

A.2.1 Attacks considered

We evaluate six MIAs: Yeom, LiRA, TMIA, RMIA, Attack-R, and quantile regression. Let f_θ be the target model, (x, y) a candidate sample, and $\mathcal{L}(f_\theta(x), y)$ its prediction loss.

Our goal is to measure how membership vulnerability changes with model capacity. Therefore, an informative attack must depend on the behavior of the target model:

$$s(x, y) = \Phi\big(\mathcal{L}(f_\theta(x), y)\big), \tag{10}$$

possibly after calibration with reference statistics. All selected attacks derive their membership evidence from the target loss or a monotone transformation of it.

We exclude MIAs designed for synthetic data that operate only in the input space. These include nearest-neighbor and distance scores, such as DCR (Chen et al., 2020), DPI (Ward et al., 2024), and MC (Hilprecht et al., 2019); density-based methods, such as DOMIAS (Van Breugel et al., 2023) and Gen-LRA (Ward et al., 2025a); and discriminator-based attacks, such as LOGAN (Hayes et al., 2017). Their scores have the form

$$s(x, y) = \Psi\big(x, \mathcal{D}_{\text{ref}}, \mathcal{D}_{\text{synth}}\big), \tag{11}$$

and do not query f_θ. These methods are unsuitable for two reasons. First, they require samples from a generative model, whereas our targets are discriminative regressors. Second, their scores are invariant to the target architecture when the data remain fixed. They would therefore produce nearly identical results for small and large networks, preventing them from measuring the effect of model capacity. We consequently restrict the evaluation to the six model-dependent attacks above.

Loss thresholding (Yeom). The loss-based attack of Yeom et al. Yeom et al. (2018) exploits the generalization gap between training and held-out samples. The adversary queries f_θ, computes $\mathcal{L}(f_\theta(x), y)$, and predicts membership whenever the loss falls below a global threshold τ, typically set to the average training loss or estimated from a reference population. The attack requires neither shadow models nor an auxiliary attack classifier and is therefore computationally negligible. Its principal limitation is that a single global τ ignores the intrinsic difficulty of individual samples: easy non-members incur low loss and are systematically misclassified as members, which caps performance in the low false-positive-rate (FPR) regime.

Likelihood Ratio Attack (LiRA). LiRA Carlini et al. (2022) casts membership inference as a per-sample hypothesis test. For each target sample, the adversary trains shadow models on datasets that either contain the sample (IN models) or exclude it (OUT models). The confidence assigned to the correct label is mapped through a logit transform, and Gaussian distributions $\mathcal{N}(\mu_{\text{in}}, \sigma^2_{\text{in}})$ and $\mathcal{N}(\mu_{\text{out}}, \sigma^2_{\text{out}})$ are fitted to the resulting IN and OUT scores. Membership is decided by the likelihood ratio

$$\Lambda(x, y) = \frac{p\big(s(x, y) \mid \mathcal{N}(\mu_{\text{in}}, \sigma^2_{\text{in}})\big)}{p\big(s(x, y) \mid \mathcal{N}(\mu_{\text{out}}, \sigma^2_{\text{out}})\big)}, \tag{12}$$

with the sample declared a member when $\Lambda(x, y) > \tau$. This per-sample calibration yields state-of-the-art true-positive rates at low FPR, at the cost of training a large number of shadow models and assuming accurate knowledge of the target training pipeline.

Robust Membership Inference Attack (RMIA). RMIA Zarifzadeh et al. (2023) refines likelihood-ratio inference by explicitly modelling the alternative worlds admitted by the non-membership hypothesis. Rather than contrasting the target sample against a single averaged OUT distribution, RMIA contrasts it pairwise against population samples $z \sim \pi$. For each pair (x, z) it estimates whether the observed target model is more probable under the world in which x was trained on than the world in which z was, and the membership score is the fraction of population samples dominated by x:

$$\text{Score}_{\text{RMIA}}(x) = \Pr_{z \sim \pi}\left[\frac{\Pr(x \mid \theta)/\Pr(x)}{\Pr(z \mid \theta)/\Pr(z)} \geq \gamma\right]. \tag{13}$$

Combining reference models with population-level calibration makes RMIA effective with only a handful of reference models and markedly less sensitive to architecture and data-distribution mismatch than prior likelihood-based attacks.

TrajectoryMIA. TrajectoryMIA Liu et al. (2022) builds on the observation that members and non-members may attain comparable losses on the *final* target model while having followed distinct optimization trajectories. Because intermediate checkpoints of f_θ are typically inaccessible, the adversary distils the target model into a student and retains the student's checkpoints across distillation epochs. Evaluating (x, y) on this sequence yields a *distilled loss trajectory* $\big(\mathcal{L}_1, \ldots, \mathcal{L}_T\big)$, which is concatenated with the loss on the original target model and fed to a binary attack classifier. The attack thus exploits *how* a sample is progressively learned rather than its terminal loss alone.

Quantile Regression attack. The quantile regression attack Bertran et al. (2023) replaces the global threshold of loss-based attacks with a sample-conditional one. Using an auxiliary dataset of known non-members, the adversary queries the target model and fits a regression model $q_\alpha(x)$ that predicts a high conditional quantile of the target's confidence score, optimizing the pinball loss at a chosen FPR level α. A sample is declared a member when $s(x, y) > q_\alpha(x)$. Unlike LiRA, the attack trains a single regression model and requires no knowledge of the target architecture; its effectiveness, however, hinges on the size and distributional fidelity of the auxiliary set.

Attack-R. Attack-R Ye et al. (2022) performs difficulty calibration against a pool of pre-trained reference models $\Theta' = \{\theta'_1, \ldots, \theta'_k\}$. For a target sample x, it compares the probability $\Pr(x \mid \theta)$ assigned by the target model with that assigned by each reference model, and defines

$$\text{Score}_{\text{R}}(x) = \frac{1}{k}\sum_{i=1}^{k} \mathbf{1}\left[\frac{\Pr(x \mid \theta)}{\Pr(x \mid \theta'_i)} \geq 1\right]. \tag{14}$$

A high score indicates that the target model fits x more strongly than models that did not train on it. Because the same reference models are reused across all queries, Attack-R operates fully offline and is considerably cheaper than online LiRA. It nonetheless calibrates only against reference models, without contrasting the target sample with other population samples—a limitation that RMIA addresses directly.

B Extended methodology

B.1 Deferred derivations for the fusion rules

This appendix expands the three arguments that Section 3 states without derivation: the linearisation behind the stitching solution equation 5, the gate-column bookkeeping and the shared template that give recurrent units a common identity, and the coverage conditions of the recurrent arrays. Notation is that of Section 3.

B.1.1 Stitching: from local linearity to the closed form

Let a client own a skip-edge $u \to v$ with $v > u + 1$ and weights (W_A, b_A), and let (W_B, b_B) be the partial server average of the next normal block $u \to u{+}1$ given by equation 4, restricted to the usable rows $\mathcal{R} \subseteq \mathcal{I}_u^k$ and to the columns $\mathcal{C}$ that already carry data. We want a replacement edge (W', b') that, placed after the borrowed block, reproduces what the jump did to the hidden state h_u. Treating the activation as roughly linear over the region of interest Frankle et al. (2020)—the linear-mode connectivity regime in which fused models remain functional—this is the requirement

$$h_u\, W_A[\mathcal{R}, :] + b_A \;\approx\; \big(h_u\, W_B + b_B\big)\, W' + b'. \tag{15}$$

Both sides are affine in h_u, so equation 15 holds for every h_u exactly when the linear and the constant terms match separately,

$$W_B\, W' = W_A[\mathcal{R}, :], \qquad b_B\, W' + b' = b_A, \tag{16}$$

and solving the first equation in the least-squares sense and substituting into the second gives the closed form equation 5 used in the main text. The system equation 16 is over-determined when $\mathcal{R}$ has more usable rows than the borrowed block has columns and under-determined otherwise; the pseudo-inverse returns the minimum-residual solution in the first case and the minimum-norm one in the second, which is why no explicit case distinction appears in equation 5.

A jump that still crosses more than one layer after a push is pushed again, so carrying a jump all the way from u to v chains the solves

$$\big(W_B^{(v-2)}\big)^{+} \cdots \big(W_B^{(u)}\big)^{+}\, W_A, \tag{17}$$

which do not collapse into a single pseudo-inverse, since $(AB)^{+} \neq B^{+}A^{+}$ in general; each push therefore adds its own approximation error, and the common case $v = u + 2$ is the benign one with a single solve. Pushing is also progressive in the clients: the support $\mathcal{C}$ in equation 4 is non-empty only once some client has filled those columns of $C^{(u)}$, so a jump that cannot be pushed after one client may become pushable after the next. The waiting queue is therefore re-scanned after every client, for at most $L + 2$ rounds (Algorithm 1); a scan that resolves nothing ends the loop, and any jump left unresolved simply never contributes, its cells keeping the default $\Pi = 0$.

B.1.2 Gate-column bookkeeping

In a gated cell each unit j of layer ℓ owns G columns of the gate pre-activation of Section 3.3, one per gate, at positions $j + g n_\ell$. Selecting a set of units therefore means selecting the union of their gate columns,

$$\mathcal{G}_\ell(\mathcal{I}) = \bigcup_{g=0}^{G-1} \big\{\, j + g n_\ell : j \in \mathcal{I} \,\big\}, \qquad |\mathcal{G}_\ell(\mathcal{I})| = G\,|\mathcal{I}|, \tag{18}$$

which is the expansion used in the embedding equation 6: a contiguous slice of units becomes G equally spaced blocks of columns, one per gate, while the row indices stay in unit space. Splitting a unit across clients is never allowed, since a gate block is only meaningful together with the other $G - 1$ blocks of the same unit.

B.1.3 Shared random template

Averaging unit by unit presupposes that "unit j of layer ℓ" denotes the same feature in every client. For an MLP this can always be arranged after the fact, because permuting the neurons of a layer and

the corresponding rows and columns of the adjacent weights leaves the function unchanged Ganju et al. (2018). A recurrent layer offers no such freedom: the same permutation would have to be applied simultaneously to the rows and to the G gate blocks of $U^{(\ell)}$, which are tied through the cell state across time, so units trained from independent initialisations end up in incomparable coordinate systems and averaging them mixes unrelated features.

We remove the problem at the source. The server draws a single random template $\bar{\theta} = \{(\bar{W}^{(\ell)}, \bar{U}^{(\ell)}, \bar{b}^{(\ell)})\}_{\ell=1}^{L+1}$ from the public seed ξ and gives each client the slice of it selected by equation 6 as the client's initial weights. Every client then starts from the same point in the same coordinate system and stays near it during local training, which is the regime in which averaging is meaningful. The same template doubles as the default $\Pi = \bar{\theta}$ in equation 1: it requires no data, it is reproducible from the seed by every party, and it is the recurrent counterpart of MLP stitching. Filling with zeros instead would be actively harmful, since a zeroed gate block drives the forget and input gates to $\sigma(0) = \frac{1}{2}$ and the candidate to 0, silently decaying the cell state over the affected units rather than leaving them untrained.

A skipped recurrent layer could alternatively be rerouted analytically, by linearising the gate logic $\Phi^{(\ell)}$ into an affine time-invariant recurrence and applying the identity equation 5 in the recurrent state space. We use template completion throughout instead: it avoids that second approximation and is exact whenever any client covers the cell.

B.1.4 Coverage of the recurrent arrays

Which cells of the merged model carry real client data, and which keep the template, follows from the unit slices alone. A recurrent link $U_{ic}^{(\ell)}$ is trained only if some client holds *both* of its endpoints, the source unit i and the target unit $\mu(c)$, in layer ℓ:

$$\big(C_U^{(\ell)}\big)_{ic} > 0 \iff \exists\, k :\ i \in \mathcal{I}_\ell^k \text{ and } \mu(c) \in \mathcal{I}_\ell^k, \tag{19}$$

while an input link $W_{ic}^{(\ell)}$ needs one client holding i in layer $\ell - 1$ and $\mu(c)$ in layer ℓ,

$$\big(C_W^{(\ell)}\big)_{ic} > 0 \iff \exists\, k :\ i \in \mathcal{I}_{\ell-1}^k \text{ and } \mu(c) \in \mathcal{I}_\ell^k. \tag{20}$$

Everything else keeps its template value. These are pair conditions, so coverage is bought by overlap rather than by headcount: K clients with disjoint slices cover only the block-diagonal part of $U^{(\ell)}$, whereas a few clients whose slices overlap substantially cover the off-diagonal links as well. This is the recurrent analogue of the column-support condition that governs stitching (Section 3.2.1) and the reason the ablation of Appendix D.4 favours few wide clients over many narrow ones. The bias $b^{(\ell)}$ and the first-layer input weight $W^{(1)}$ carry no pair condition—the input layer is never split—and are fully covered as soon as $\bigcup_k \mathcal{I}_\ell^k = \{1, \dots, n_\ell\}$.

B.2 Algorithms for MLP and LSTM

Algorithms 1 and 2 collect the fusion procedures of Sections 3.2 and 3.3 in executable form. Both share the same skeleton—accumulate every client's blocks into a pair of sum and count buffers, then recover each array by count-normalised averaging—and differ in a single respect, which is exactly the difference between the two architectures: what to do with a layer that a client skipped. Comparing the two listings side by side is therefore the shortest way to see what the shared random template buys.

Algorithm 1 is progressive in two senses. Clients are processed one at a time, and within each client the edges are classified on arrival: those that connect consecutive layers are aggregated immediately, while the jumps that cross a missing layer are placed in a queue $\mathcal{Q}$. The inner loop then repeatedly scans that queue and pushes forward every jump whose support $(\mathcal{R}, \mathcal{C})$ is non-empty, i.e. every jump for which the next block already contains averaged data to borrow from. This is why the loop must be re-run after each client rather than once at the end: a jump that cannot be pushed now may become pushable once a later client has filled the missing columns, so the amount of stitching that can be resolved grows as clients are consumed. A jump that lands on $u + 1 = v$ has become an ordinary edge and is aggregated; otherwise it is re-queued for another push. The *progress* flag guards the only failure mode of the procedure: if a full scan resolves nothing, no later scan over the same queue

Algorithm 1 Progressive heterogeneous fusion with data-free stitching.

Require: Client models $\{W_k\}_{k=1}^{K}$, Layer ranges $\{(s_\ell^k, e_\ell^k)\}_{k,\ell}$, Server widths $\{n_\ell\}_\ell$

1: $S^{(\ell)}, C^{(\ell)}, \sigma^{(\ell)}, \kappa^{(\ell)} \leftarrow \mathbf{0}$
2: $\mathcal{Q} \leftarrow \emptyset$
3: **for** $k \leftarrow 1$ **to** K **do**
4: **for all** edges (u, v, W_A, b_A) in client k **do**
5: **if** $v = u + 1$ **then**
6: Aggregate (W_A, b_A) using equation 1
7: **else**
8: $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(u, v, W_A, b_A)\}$
9: **end if**
10: **end for**
11: **repeat**
12: $\mathcal{Q}_{\text{new}} \leftarrow \emptyset$
13: *progress* $\leftarrow$ FALSE
14: **for all** $(u, v, W_A, b_A) \in \mathcal{Q}$ **do**
15: Determine its support $(\mathcal{R}, \mathcal{C})$
16: **if** $\mathcal{R} \neq \emptyset \wedge \mathcal{C} \neq \emptyset$ **then**
17: Compute (W_B, b_B) using equation 4
18: Compute (W', b') using equation 5
19: **if** $u + 1 = v$ **then**
20: Aggregate (W', b') at layer $u + 1$
21: **else**
22: $\mathcal{Q}_{\text{new}} \leftarrow \mathcal{Q}_{\text{new}} \cup \{(u + 1, v, W', b')\}$
23: **end if**
24: *progress* $\leftarrow$ TRUE
25: **end if**
26: **end for**
27: $\mathcal{Q} \leftarrow \mathcal{Q}_{\text{new}}$
28: **until** $\mathcal{Q} = \emptyset \vee \neg$*progress*
29: **end for**
30: Recover $W^{(\ell)}, b^{(\ell)}$ using equation 1

would either, and the loop exits rather than spinning. Jumps left unresolved at that point simply never contribute, and the corresponding cells keep the default $\Pi = 0$. Each push costs one least-squares solve, and a jump crossing d layers needs d of them. The final recovery is performed once, after all clients have been processed, and the readout repair of Section 3.1.2 is applied afterwards; it is omitted from the listing because it is identical to lines 9–11 of Algorithm 2.

Algorithm 2 has no queue at all. Because every uncovered cell falls back to the shared template $\bar{\theta}$, a skipped layer is not a special case to be repaired but the ordinary behaviour of the recovery step, so the accumulation reduces to a single pass over clients and layers with no iteration, no least-squares solve and no local-linearity approximation. The only element specific to the recurrent case appears in line 4, where selecting a unit expands into its G gate columns before the buffers are indexed. The listing then closes with the anchor pipeline: the AR(1) anchors are regenerated from the public seed, the clients' predictions on them are averaged into a teacher target, and one ridge solve overwrites the output layer, leaving the recurrent weights untrained.

The contrast between the two makes the trade-off explicit. The dense procedure pays an iterative, approximation-based repair in exchange for allowing clients to omit layers entirely; the recurrent one avoids that cost by fixing a common coordinate system in advance, which is necessary anyway because recurrent units, unlike dense neurons, cannot be renumbered to make a client's selection contiguous. Neither procedure reads any client data: the only information crossing the network beyond the trained blocks is the clients' predictions on anchors that every party can regenerate from the seed.

Algorithm 2 Progressive heterogeneous fusion of stacked recurrent networks.

Require: Client models $\{W_k^{(\ell)}, U_k^{(\ell)}, b_k^{(\ell)}\}$, ranges $\{(s_\ell^k, e_\ell^k)\}$, widths $\{n_\ell\}$, gates G, template $\bar{\theta}$ (seed ξ), anchors N, autocorrelation ρ, ridge λ.

```
Initialise all sum/count buffers to 0.
for k = 1, ..., K do
    for each present layer ℓ of client k do
        G_ℓ^k ← ⋃_{g=0}^{G-1} (I_ℓ^k + g n_ℓ)                ▷ Eq. equation 18
        Accumulate W_k^(ℓ), U_k^(ℓ), b_k^(ℓ) into the buffers via Eq. equation 1.
    end for
end for
Recover {W^(ℓ), U^(ℓ), b^(ℓ)} by count-normalised averaging with template fallback, Eq. equation 1 at Π = θ̄.
Generate AR(1) anchors X^anc from ξ, Eq. equation 7.
Collect predictions and form Y^tea = (1/K) Σ_k f_k(X^anc), Eq. equation 2.
Extract fused features H = g(X^anc), solve Eq. equation 3, overwrite (W^(L+1), b^(L+1)).
return fused server model.
```

B.3 Fusion strategy in convolutional networks

We describe here the split-and-merge recipe for convolutional architectures. It is presented in the appendix because it requires no new machinery: once a channel is recognised as playing the role of a neuron, the convolutional case is the dense case of Section 3.2 with two extra axes carried along, and the three steps of that section—averaging the normal weights, stitching the skip-edges, and repairing the output layer—apply in the same order and with the same equations. Only the normalisation layers, whose running statistics are not parameters, need a treatment of their own.

B.3.1 Splitting the network

The server model is a stack of B convolutional blocks followed by global average pooling and a dense head with L hidden layers. Block ℓ holds n_ℓ channels and a kernel $W^{(\ell)} \in \mathbb{R}^{h_\ell \times w_\ell \times n_{\ell-1} \times n_\ell}$, with n_0 the number of input channels of the image. Exactly as in Section 3.2, client k receives a contiguous slice of channels $I_\ell^k = \{s_\ell^k, \dots, e_\ell^k\}$ in each block, possibly empty, and only the input channels and the output classes are never split. The client instantiates a narrower network with $|I_\ell^k|$ channels per block and trains it end to end on its own data; after training it returns the blocks it owned, which map into the server arrays by slicing the two channel axes and keeping the spatial extent whole,

$$W_k^{(\ell)} \hookrightarrow W^{(\ell)}\big[:, :, I_{\ell-1}^k, I_\ell^k\big]\,. \tag{21}$$

A client therefore sees the kernel connecting channel i of block $\ell-1$ to channel j of block ℓ only if it holds both endpoints, which is the two-sided condition of the dense case; the spatial axes are never sliced, since a filter is shared across positions by construction and owning a channel means owning its whole $h_\ell \times w_\ell$ kernel. The contiguity assumption is as harmless here as it was for the MLP: channels within a block may be renumbered freely without changing the function computed by the network, so any selection can be made contiguous by relabelling.

Global average pooling joins the two halves of the model. It collapses the spatial axes and leaves the channel axis untrained, so the input rows of the first dense layer are indexed by the channels of the last convolutional block and the client's head reads exactly the features its own extractor produces. From that point on the head is an MLP: it is split into neuron slices and merged with Section 3.2 unchanged, output layer kept whole.

B.3.2 Averaging the normal kernels

A normal kernel connects block u to the next block $u+1$, and is averaged with the generic rule of Eq. 1 using the slice of Eq. 21: the server keeps a sum array $S^{(\ell)}$ and a count array $C^{(\ell)}$ of the same shape as $W^{(\ell)}$, adds each client's block into the cells it occupies, and finishes with Eq. 1 using the

dense default $\Pi = 0$,

$$W^{(\ell)}_{\cdot\cdot ij} = \frac{S^{(\ell)}_{\cdot\cdot ij}}{\max\Big(C^{(\ell)}_{\cdot\cdot ij},\, 1\Big)}. \tag{22}$$

Contributions may be accumulated with unit weight or, when clients hold training sets of very different sizes, with weight equal to the number of local samples, in which case the rule reduces to FEDAVG on any cell that every client covers. A kernel cell receives real data only when some client holds both of its endpoint channels in adjacent blocks,

$$\exists\, k : i \in I^k_u \ \wedge\ j \in I^k_{u+1}, \tag{23}$$

so the fraction of covered cells is governed, as in the dense case, by how much the client slices overlap. Cells left genuinely empty belong to blocks that some client skipped and are handled by stitching next, not by $\Pi = 0$. Where no client covers a pair of channels that all clients kept, the zero default is benign: a zero kernel means the two channels are simply not connected, which is a valid convolutional layer.

B.3.3 STITCHING SKIP-EDGES

When a client is assigned no channel in a block, it has a skip-edge: a single kernel that jumps from block u to block v with $v > u+1$, and that cannot be averaged with the normal step-by-step kernels because it connects the wrong pair of blocks. We push it forward one block at a time, exactly as in Section 3.2.1. The only adaptation is that the least-squares solve is performed on the channel axis: folding the spatial and input-channel axes of a kernel into rows, $\widehat{W} \in \mathbb{R}^{(h\,w\,n_u)\times n_{\text{out}}}$, the jump (W_A, b_A) and the next normal block (W_B, b_B) borrowed from Eq. 5 give

$$W' = \widehat{W}_B^{+}\,\widehat{W}_A[R,:], \qquad b' = b_A - b_B W', \tag{24}$$

where W' is a channel-mixing matrix, realised as a 1×1 convolution or folded directly into the next block's kernel along its output-channel axis. The jump now goes $u+1 \to v$; if $u+1 = v$ it has become a normal kernel and is averaged with Eq. 1, and otherwise it is pushed again, re-scanning the waiting queue after every client as in Algorithm 1. Two approximations are involved rather than one: the local linearity of the activation already assumed in the dense case, and the fact that Eq. 24 composes the two layers at the channel level only, which is exact for 1×1 kernels and approximate otherwise, since chaining two spatial convolutions enlarges the receptive field in a way a channel map cannot reproduce. This is a further reason to avoid long chains of stitches, already discouraged by the ablation of Appendix D.4. In our experiments every client keeps every convolutional block, only narrower, so no stitching is needed in the feature extractor and the mechanism is used only in the dense head.

B.3.4 RECALIBRATING THE NORMALISATION STATISTICS

The normalisation layers are the only component that does not fit the generic rule. Their scale and shift are ordinary per-channel parameters and are averaged with Eq. 1 on the index set I^k_ℓ, except that a channel covered by no client keeps the identity transformation $(\gamma, \beta) = (1, 0)$ rather than zero, since a zero scale would silence it. The running mean and variance, by contrast, are statistics rather than parameters: each client's describe the activations of its own narrow network, and the fused model produces different activations, so averaging them would calibrate the normalisation to a distribution that no longer exists—which alone is enough to destroy the fused classifier, as every subsequent block would receive badly standardised inputs. We therefore re-estimate them from statistics the clients report. Each client computes, for every normalisation layer and every channel it owns, the mean and variance of the corresponding pre-normalisation activations over its local data and transmits only those two scalars per channel; the server combines them by the law of total variance,

$$\mu^{(\ell)}_c = \frac{\sum_{k\in\mathcal{K}_c} m_k\,\hat{\mu}_{k,c}}{\sum_{k\in\mathcal{K}_c} m_k}, \qquad \Big(\sigma^{(\ell)}_c\Big)^2 = \frac{\sum_{k\in\mathcal{K}_c} m_k\left(\hat{\sigma}^2_{k,c} + \hat{\mu}^2_{k,c}\right)}{\sum_{k\in\mathcal{K}_c} m_k} - \Big(\mu^{(\ell)}_c\Big)^2, \tag{25}$$

with $\mathcal{K}_c = \{k : c \in I^k_\ell\}$ the clients holding channel c and m_k their sample counts. No raw sample, activation map or gradient is transmitted, only two scalars per channel and layer, which is

comparable to the anchor predictions of Eq. 2. Uncovered channels keep the statistics of the freshly initialised server model. The pooling is a first-order correction rather than an exact recalibration, since it combines statistics measured on each client's own submodel; an alternative that transmits nothing at all is to estimate (μ, σ^2) by pushing the anchors of the next subsection through the fused network, at the cost of calibrating on inputs whose statistics differ from real images.

B.3.5 REPAIRING THE OUTPUT LAYER

We close with the generic repair of Section 3.2.2, which applies unchanged. The anchors are the image analogue of the clipped Gaussian noise used there: given the public seed ξ and a clipping level τ, $X^{\mathrm{anc}} \in \mathbb{R}^{N \times H_0 \times W_0 \times n_0}$ has i.i.d. clipped standard normal entries in the standardised input space of the dataset, so that every party can regenerate them locally from the seed alone and nothing but the seed is transmitted. The feature map g runs an anchor through all the merged convolutional blocks and the pooling layer, returning $H = g(X^{\mathrm{anc}}) \in \mathbb{R}^{N \times n_B}$ if the head has no hidden layers, or through the merged hidden layers of the head as well otherwise. Feeding X^{anc} and H into Eqs. 2–3 refits the classifier, and nothing else in the network changes. The motivation is the one given in Section 3.1.2 and is, if anything, stronger here: after a feature extractor has been reassembled from partially disjoint channel slices, the classifier is the layer that is guaranteed to be reading features other than the ones it was trained on.

B.4 FUSION STRATEGY IN TRANSFORMERS AND LARGE LANGUAGE MODELS

In our work, we focus on MLPs, CNNs, and LSTMs. However, for the sake of completeness, we detail the algorithm for attention-based architectures.

B.4.1 TRANSFORMERS STRUCTURE

Transformer-based models replace recurrence with self-attention mechanisms, enabling the modelling of global dependencies among all elements of a sequence in a highly parallelizable manner. Their core component is the scaled dot-product attention mechanism, defined as:

$$\mathrm{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V},$$

where $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are the query, key, and value matrices, respectively, and d_k is the dimensionality of the keys. In practice, Transformers employ multi-head attention, residual connections, layer normalization, and position-wise feed-forward networks. The latter can be interpreted as an MLP applied independently to each token representation, providing nonlinear feature transformation after the attention operation. Positional encodings are also incorporated to preserve information about the order of the sequence. Therefore, Transformer architectures combine attention-based global dependency modeling with MLP-based nonlinear transformations, which contributes to their strong representational capacity.

A transformer block has two parts, each wrapped in a residual connection and a layer normalisation: multi-head self-attention (MHA) and a position-wise feed-forward network (FFN). Both run on a shared *residual stream* of width d Vaswani et al. (2017). One block's weights are: the attention projections $W_Q, W_K, W_V, W_O \in \mathbb{R}^{d \times d}$, split into h heads of size $d_h = d/h$; the feed-forward weights $W_1 \in \mathbb{R}^{d \times d_{ff}}$ and $W_2 \in \mathbb{R}^{d_{ff} \times d}$ with an activation between; and the embedding $E \in \mathbb{R}^{|\mathcal{V}| \times d}$ (plus the usually-tied unembedding and the layer-norm parameters, all of size d). The residual stream forces the *same* width d on every block, every norm and both embeddings. It plays the role that the fixed input/output sets $\mathcal{I}_0$ and $\mathcal{I}_{L+1}$ played for the MLP (Section 3.2)—but pinned at every layer, not only at the two ends.

B.4.2 WHAT WE VARY, AND WHAT WE KEEP FIXED

We vary the width of one thing only: the inner FFN size d_{ff} of each block. Attention, the layer norms and the embeddings are taken from the shared template unchanged. Two reasons. First, shrinking the residual width d (the output of W_O, the rows of W_2, the rows of E, every LayerNorm) would propagate through the residual path into every later block and break the shared-d rule; d is

fixed by the architecture, not a free knob. Second, we *could* split attention by heads—each head is independent and a group of heads is a contiguous block of (W_Q, W_K, W_V, W_O)—but that reintroduces an alignment problem the template alone does not solve. The template argument (Section 3.3) relies on linear-mode connectivity Frankle et al. (2020), which holds well for feed-forward weights but is shakier for attention, where heads interact through a softmax over token pairs rather than through plain weights. So we leave per-head splitting for future work, split only the FFN, and take attention from the template.

B.4.3 This reduces to the MLP case of Section 3.2

With that choice, the problem splits block by block. In block ℓ, client k gets a contiguous range $\mathcal{I}_\ell^k \subseteq \{0, \dots, d_{ff} - 1\}$ of FFN units, builds the slim FFN from the matching template slice (Section 3.3), and trains it end-to-end with attention and norms frozen (or fine-tuned the same way for everyone). Merging the FFN reuses the row/column slicing of equation 1: W_1 is averaged on its output columns $\mathcal{I}_\ell^k$, W_2 on its input rows $\mathcal{I}_\ell^k$, and the middle bias on $\mathcal{I}_\ell^k$, finished with equation 1 and the template default $\Pi = \bar{\theta}$ exactly as in Section 3.3. Skip-edges (Section 3.2.1) never occur: every client keeps every block, just with a narrower d_{ff}, so no stitching is needed—merging is plain per-block averaging. Gated FFN variants (GLU, SwiGLU Shazeer (2020)) use three weights instead of two; the same d_{ff} slicing applies to all three and nothing else changes.

B.4.4 Output repair, and LLMs

The output repair of Section 3.1.2 carries over to the unembedding (the head that turns features into token scores). The server draws anchor token sequences—either i.i.d. from a public token-frequency table or a snippet of public unlabelled text—collects the averaged teacher logits via equation 2, runs the anchors through the merged transformer to get final features $H \in \mathbb{R}^{\tilde{N} \times d}$, and solves the ridge system equation 3 for the new unembedding. Two practical tweaks at LLM scale. First, the vocabulary $|\mathcal{V}|$ is huge (10^4–10^5), so we solve only for a frequent-token subset, or use a low-rank form $E^\top = AB$ with $A \in \mathbb{R}^{d \times r}$ fixed from the template and only $B \in \mathbb{R}^{r \times |\mathcal{V}|}$ refitted. Second, training full FFNs per client is too costly for billion-parameter models; a natural fit is to slice the d_{ff} axis at the level of LoRA adapters Hu et al. (2021), each client training a LoRA module of width $|\mathcal{I}_\ell^k|$ from the matching slice of a full-rank template adapter. Averaging and ridge repair apply to the adapter weights unchanged, and the base model never crosses the wire. We see this LoRA version as the practical route to large pretrained LLMs and leave its experiments to future work.

C Settings of the experiments

For the evaluation of our submodel training method, we consider several federated scenarios running in a KVM-based virtual machine with an x86_64 architecture, equipped with an AMD EPYC 9554 64-Core Processor, 20 virtual CPUs (vCPUs), and 125 GiB of RAM. We use a vanilla Python code to implement the FL setup.

In this work, we evaluate our approach on three datasets that jointly cover tabular, image, and time-series modalities. The *Letter Recognition* dataset Frey & Slate (1991), available from the UCI Machine Learning Repository, is a tabular classification benchmark of 20000 instances, each corresponding to one of the 26 capital letters of the English alphabet. The instances were generated from black-and-white rectangular pixel renderings of the letters in 20 different fonts, which were randomly distorted, and each resulting instance is described by 16 integer-valued attributes (statistical moments and edge counts) scaled to the range $[0, 15]$. Following the standard protocol, 16000 instances are used for training and 4000 for testing. The *CIFAR-10* dataset Krizhevsky et al. (2009) is a widely used image-classification benchmark consisting of 60000 colour images of size 32×32 pixels, evenly distributed across 10 mutually exclusive classes (such as airplane, automobile, bird, and cat), with 50000 images for training and 10000 for testing. Finally, the *PLEIAData* dataset Ibarra et al. (2023) provides consumption, HVAC, indoor temperature, weather, and motion-sensor data collected over almost one year in the Pleiades building of the University of Murcia; its time-series nature makes it well suited to evaluate the recurrent variants of our method. Together, these datasets allow us to assess our framework across heterogeneous data types.

We consider an external adversary, i.e., a third party that does not take part in the federation but obtains the model held by one of the clients, for instance by compromising the client's device or because the client deploys or releases its local model. The adversary therefore has white-box access to a single client submodel and, as is standard for calibrated membership inference attacks, access to auxiliary data drawn from the same distribution, which it uses to train shadow or reference models. Its goal is to decide whether a given record belonged to the training set of that client. Clients never receive the complete global model: at every round the aggregator sends each client only the slice of parameters corresponding to its own submodel, and the fused model is never redistributed in its complete form. Consequently, the capacity exposed to the adversary is bounded by the size of the submodel of the compromised client, which is the quantity that STITCHFL controls. The aggregator is assumed to be honest: it follows the protocol, keeps the fused global model private, and does not attempt to infer membership from the client updates or from the predictions on the synthetic anchors that clients return for the readout repair. Since each client only ever observes its own slice, the same analysis also bounds the leakage towards a curious client. Attacks by a curious aggregator are orthogonal to this work and can be addressed by combining STITCHFL with secure aggregation or differential privacy.

The training data of each dataset is distributed among 5 clients for Letter Recognition, 5 clients for CIFAR-10 and 3 clients for PLEIAData. We deliberately keep the number of clients small. Splitting these datasets among more clients in Letter Recognition and CIFAR-10 would leave each one with very few training samples, and the resulting drop in performance would be caused by data scarcity rather than by the method under study. For the same reason, the data is partitioned among clients as close to IID as possible. In this way, the gap with respect to the complete model is mainly attributable to the division of the model into submodels, which is the effect we aim to isolate. The influence of data heterogeneity is analysed separately in the ablation study of Appendix D.4. On the other hand, the PLEIAData partition is defined this way because the dataset is structured in 3 subclients (one per building), so we wanted to respect this division.

Submodel partitioning. The division of the experiments made in Section 4.2.2 follows this structure. The MLP used on Letter Recognition has three hidden layers of 128, 64 and 32 neurons. The five clients train the neuron ranges $[0, 44)$–$[0, 12)$–$[0, 8)$, $[12, 72)$–$[10, 27)$–$[4, 15)$, $[40, 84)$–$[25, 37)$–$[11, 19)$, $[52, 104)$–$[36, 50)$–$[16, 25)$ and $[73, 128)$–$[49, 64)$–$[22, 32)$, that is, submodels of widths 44–12–8, 60–17–11, 44–12–8, 52–14–9 and 55–15–10. These correspond to between 28.6% and 39.3% of the hidden neurons of the complete model. The CNN used on CIFAR-10 has three convolutional blocks of 64, 128 and 256 channels followed by a dense layer of 256 units. Clients 1 to 3 receive 24–48–96 channels (37.5% of each block), shifted along the channel axis as $[0, 24)$, $[12, 36)$ and $[24, 48)$ in the first block, and Clients 4 and 5 receive 20–40–80 channels (31.25%). In the dense layer, the clients train $[0, 96)$, $[48, 160)$, $[96, 192)$, $[144, 224)$ and $[160, 256)$, that is, 96, 112, 96, 80 and 96 units, so each submodel holds between 31.3% and 39.8% of the units of the complete network. Finally, the LSTM used on PLEIAData stacks two recurrent layers of 64 and 32 units. The three clients receive the windows $[0, 36)$, $[14, 50)$, $[28, 64)$ in the first layer and $[0, 18)$, $[7, 25)$, $[14, 32)$ in the second, i.e., 56.25% of the units of every layer. This larger and more overlapping share is deliberate: recurrent units only admit cell-by-cell averaging where several clients cover the same unit (Section 3.3), so neighbouring clients share 22 units of the first layer and 11 of the second. In every case, no client comes close to holding the complete architecture, which is the regime the method is designed for.

This division is applied literally only to STITCHFL, whose clients train exactly the ranges above. The competing methods are resource-driven: their contribution is precisely the rule that decides which parameters a client should train given a budget, namely importance scores in HALOFL, gradient approximation in FEDGSE and optimal-transport matching in SUBFLOT. Imposing our positions on them would replace the very component being evaluated. We therefore transfer from the base division only the size of each submodel. Each client is given the capacity implied by its range, computed as the fraction of units it retains in every layer, and from that point on each method selects the parameters of the client as its own algorithm prescribes. In this way, all the methods are compared under the same heterogeneity budget, and they differ only in how each submodel is placed within the server model and how the submodels are fused.

D EXTENDED RESULTS

In this section, we provide the additional experimental results that could not be included in the main body. These results complement the main evaluation and provide further evidence supporting the findings discussed in the paper.

D.1 MIA RESULTS

D.1.1 EVALUATION METRICS

Membership inference is a binary decision problem, so every attack of Appendix A.2.1 ultimately produces a membership score $s(x, y)$, and a decision is obtained by thresholding it, $\hat{m}_\tau(x, y) = \mathbf{1}[\, s(x, y) \geq \tau \,]$. Sweeping τ over its whole range traces the ROC curve of the attack, which relates the true positive rate, i.e., the fraction of members correctly identified as such,

$$\mathrm{TPR}(\tau) = \Pr\left[\, s(x, y) \geq \tau \;\middle|\; (x, y) \in D_{\text{train}} \,\right], \tag{26}$$

to the false positive rate, i.e., the fraction of non-members wrongly declared members,

$$\mathrm{FPR}(\tau) = \Pr\left[\, s(x, y) \geq \tau \;\middle|\; (x, y) \notin D_{\text{train}} \,\right]. \tag{27}$$

The evaluation set is balanced, containing the same number of members and non-members, so an attacker that ignores the model entirely obtains $\mathrm{TPR}(\tau) = \mathrm{FPR}(\tau)$ for every τ, which is the diagonal of the ROC plane and the reference against which any leakage must be measured. We summarise this curve through two complementary views, both reported for all six attacks.

The first is the area under the ROC curve (**AUC**),

$$\mathrm{AUC} = \int_0^1 \mathrm{TPR}\big(\mathrm{FPR}^{-1}(u)\big)\, \mathrm{d}u, \tag{28}$$

which equals the probability that a randomly chosen member receives a higher score than a randomly chosen non-member. It is threshold-free and therefore does not depend on how an attacker would set its operating point, it takes the value 0.5 under random guessing and 1 for a perfect attack, and it is comparable across architectures, which makes it a convenient single-number summary. What it measures, however, is the average-case leakage of a model: it weights every operating point equally, including regimes in which the attacker accuses most of the population.

The second is the true positive rate at a fixed low false positive rate, **TPR@FPR** $\in \{1\%, 5\%, 10\%\}$, which reads the ROC curve at its leftmost end. As argued by Carlini et al. (2022), aggregate metrics such as AUC or attack accuracy can be misleading as privacy measures: an attack that is only marginally better than chance on average, but that identifies a small number of training records with high confidence, constitutes a real breach, whereas an attack that never makes a confident call does not, however favourable its AUC. The low-FPR regime isolates precisely this behaviour, since it measures how many members can be recovered while keeping the number of false accusations bounded. It is also the regime that matches the threat model of this work, where a client that receives a submodel is interested in the few records it can identify reliably, and not in a population-level statistic. Consequently, a value of $\mathrm{TPR@1\%\,FPR}$ close to 0.01 indicates the absence of a confident breach, whereas a TPR several times the corresponding FPR budget indicates a model from which individual records can be singled out.

Reporting both views is necessary because the attacks differ in how they calibrate the target loss, and consequently in the regime where each of them is strongest: uncalibrated thresholding is competitive on AUC but weak at low FPR, while per-sample calibrated tests behave in the opposite way. An evaluation based on a single metric would therefore favour one family of attacks over the other, and would misrepresent the privacy of a model as the strength of whichever attack that metric happens to reward.

D.1.2 MEMBERSHIP LEAKAGE DEPENDING ON MODEL CAPACITY

The premise on which STITCHFL is built is that the amount of membership information carried by a model grows with its capacity: larger networks fit their training partition more tightly, leave a stronger memorisation footprint, and are therefore easier to attack (Yeom et al., 2018; Carlini et al.,

2022; Németh et al., 2025). We verify this premise empirically more generally than in Figure 1. Four multilayer perceptrons of increasing capacity are trained on the same data partition of Letter Recognition, spanning more than one order of magnitude in parameter count: L16 (714 parameters), L32-16 (1514), L64-32-16 (4138) and L128-64-32-16 (13482). Each model is attacked with the six attacks of Section A.2.1 under identical conditions, and Figure 4 reports AUC and TPR at 1%, 5% and 10% FPR.

The **AUC** grows monotonically with capacity for all six attacks, but the effect is far from uniform. The reference-model-based attacks scale the most steeply: RMIA rises from 0.534 on L16 to 0.708 on L128-64-32-16, LiRA from 0.539 to 0.687 and TrajectoryMIA from 0.513 to 0.679. Attack-R starts as the strongest attack on the smallest model (0.553) but saturates early, gaining only 0.013 between the last two capacities, which suggests that comparing against a reference model exploits the memorisation footprint of the smallest networks but does not benefit from the extra capacity of the largest one. The two uncalibrated attacks remain close to the baseline throughout, reaching only 0.589 (Quantile) and 0.581 (Yeom) on the largest model.

The trend is considerably sharper in the low-**FPR** regime, which is the operationally relevant notion of a privacy breach (Carlini et al., 2022). On L16 no attack exceeds a TPR of 0.016 at 1% FPR, so all of them are essentially indistinguishable from random guessing: a model of 714 parameters simply does not retain enough information about individual records to be attacked. As capacity grows, the calibrated attacks separate from the diagonal, and on L128-64-32-16 RMIA reaches a TPR of 0.099, roughly ten times its false-positive budget and a six-fold increase over the smallest model, followed by TrajectoryMIA (0.065) and LiRA (0.062). The same ordering holds at 5% and 10% FPR, where the largest model is attacked at 0.204 and 0.298 TPR by RMIA, against 0.054 and 0.116 for L16. The Quantile and Yeom attacks behave in the opposite way and even fall below the diagonal on the largest model (0.042 and 0.043 at 5% FPR), which confirms that exploiting capacity requires calibrating the score of a sample against reference models or population samples, and that an uncalibrated loss threshold does not profit from a larger target.

Two consequences follow for our setting. First, the relationship between capacity and leakage is monotone and steep in exactly the regime that matters, so reducing the model exposed to a client is a meaningful privacy mechanism rather than an incidental side effect of partitioning. Second, the strongest attack against every capacity is one of the calibrated ones, so the comparison of Section 4.2.1 is driven by the attacks that are most sensitive to the size of the exposed submodel, which is a conservative choice.

D.2 Complete utility results

Section 4.2.2 reported, for each architecture, the metric that best summarises its task: accuracy for the two classification problems and R^2 for the regression problem. For completeness, this section reports the remaining metrics. For the classification tasks we add macro-averaged recall, precision and F1-score, together with the Matthews correlation coefficient (MCC), which is informative when the errors are unevenly distributed across classes. For the regression task we add RMSE, MAE, and the two normalised indicators used in building-energy practice, the normalised mean bias error (NMBE) and the coefficient of variation of the RMSE (CV-RMSE). As in the main body, every boxplot summarises 10 independent runs and FedAvg on the complete model is included as the reference that a submodel-training method aspires to recover.

Figure 5 reports the five metrics on **Letter Recognition**. The ranking of the methods is identical under every metric, with StitchFL ahead of FedGSE, SubFLOT and HaloFL, so the conclusion of the main body does not depend on the metric chosen. Two observations are added by the extra panels. First, recall and accuracy coincide for all the methods, which is expected because the 26 classes of the dataset are close to balanced, and the F1-score follows the same values closely. Second, precision is systematically higher than recall for the weakest methods, by 7.6 points for HaloFL (0.693 against 0.617) and 6.9 points for SubFLOT (0.757 against 0.688), whereas the gap is only 1.0 point for StitchFL (0.872 against 0.862) and vanishes for FedAvg. A precision clearly above recall indicates that the fused model concentrates its predictions on a subset of the classes and leaves the remaining ones under-predicted, that is, the damage caused by an incoherent fusion is not spread uniformly but falls on entire classes. The MCC, which penalises exactly

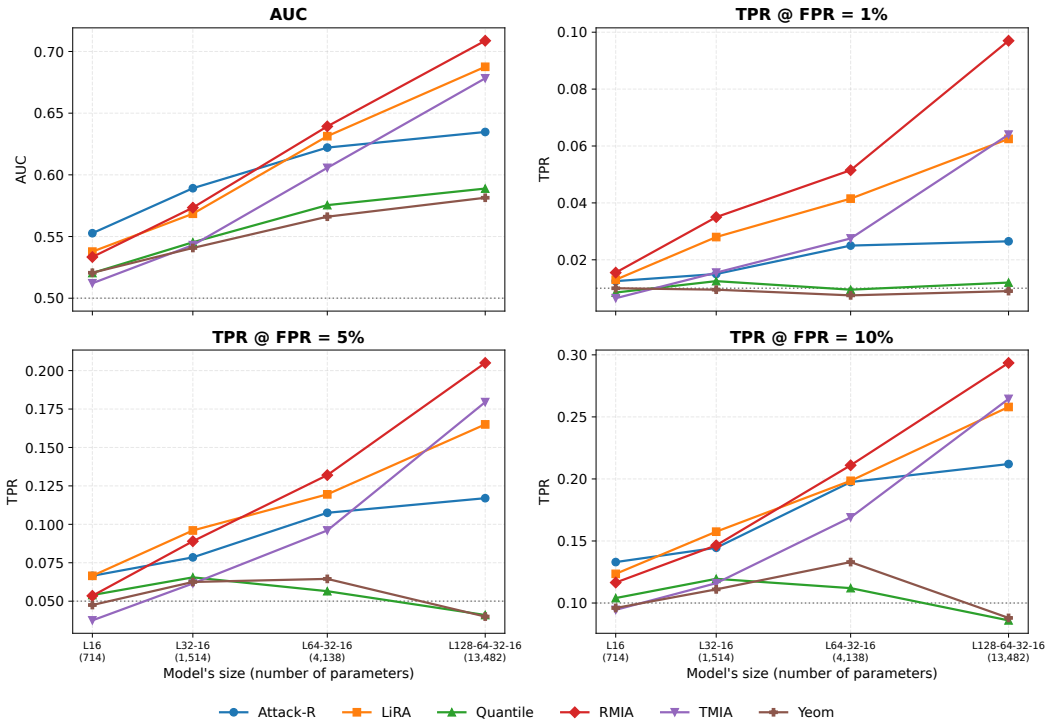


Figure 4: Membership leakage as a function of model capacity on the Letter-recognition dataset.

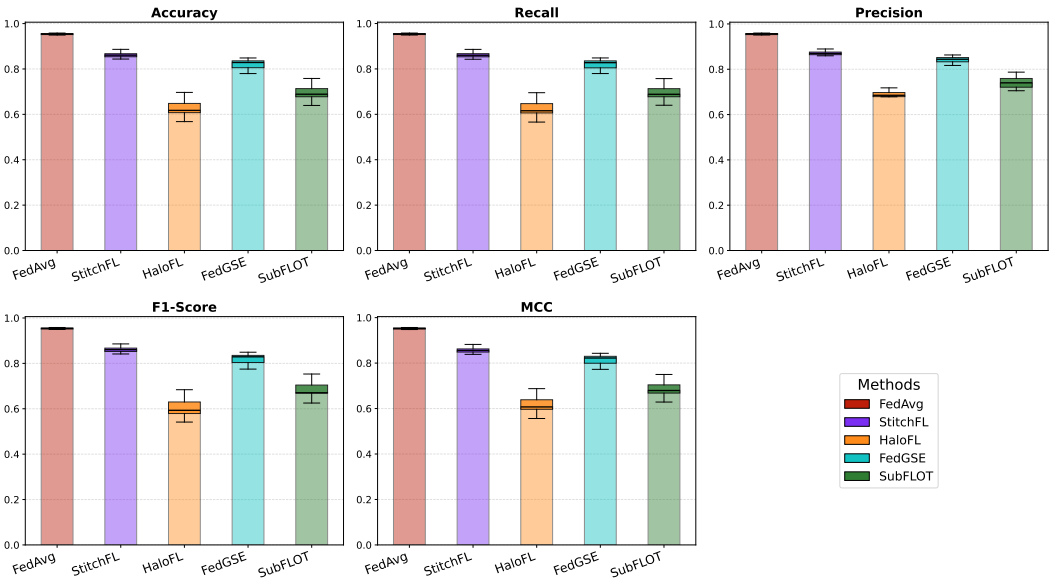


Figure 5: Utility of the fused global model on Letter Recognition under all the classification metrics: accuracy, recall, precision, F1-score and MCC. Boxplots summarise 10 independent runs.

this behaviour, confirms the picture: StitchFL attains 0.857 against the 0.953 of FedAvg, while HaloFL drops to 0.605.

Figure 6 reports the same metrics on **CIFAR-10**, where the effect is more pronounced because the task demands more capacity from each submodel. The ordering of the methods is again preserved across metrics, with one exception: HaloFL attains the second-best precision (0.605 against the

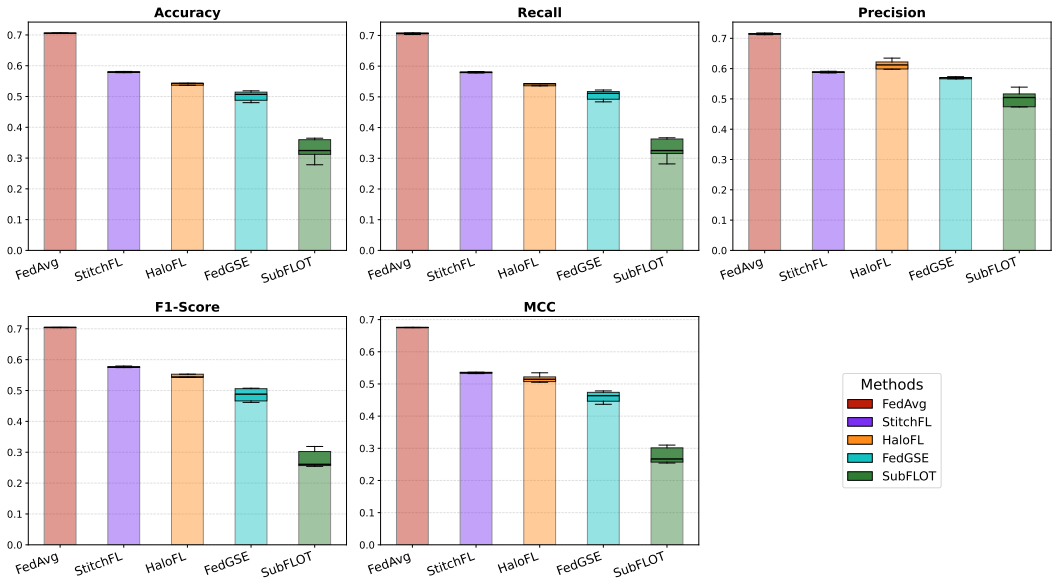


Figure 6: Utility of the fused global model on CIFAR-10 under all the classification metrics: accuracy, recall, precision, F1-score and MCC.

0.588 of StitchFL), yet its recall is 3.7 points lower (0.542 against 0.579), so it falls behind on F1-score (0.546 against 0.577) and on MCC (0.520 against 0.536). This is the same asymmetry observed on the MLP, and it shows that a high precision alone does not indicate a better fused model, but rather a model that abstains from part of the label space. The behaviour of SubFLOT is the extreme case: its accuracy of 0.322 falls to an F1-score of 0.260 and an MCC of 0.265, which means that its predictions are concentrated on a few classes. StitchFL is the method whose F1-score stays closest to its accuracy (0.577 against 0.579), so the utility it recovers is distributed evenly over the classes rather than concentrated.

Figure 7 reports the regression metrics on **PLEIAData**. Beyond the R^2 of 0.75 against 0.87 discussed in the main body, the fused model raises the RMSE from 0.67 to 0.84 and the MAE from 0.41 to 0.62. The MAE therefore degrades relatively more than the RMSE (+51% against +25%), which indicates that the loss of utility comes from a moderate error spread over many predictions rather than from a few large deviations, since the RMSE would penalise the latter more heavily. This is consistent with the two data-free primitives of the method: pseudo-inverse stitching and anchor-based readout repair restore a functioning model without reproducing the fine-grained calibration that local data would provide. The NMBE makes this explicit, rising from a median of 1.8% for the reference to 9.3% for the fused model, with a box spanning roughly 2.4% to 12.2% across runs, so the fused model tends to overestimate consumption. Finally, the CV-RMSE increases from 25.3% to 32.0%.

D.3 One-shot results

The experiments of Section 4 evaluate each method in a fully federated regime, with several communication rounds in which clients train, the aggregator fuses, and the fused model is re-split and redistributed. That regime conflates two distinct abilities: how faithfully a method reassembles a global model from heterogeneous submodels in a single fusion, and how well the repeated round structure corrects whatever that fusion gets wrong. A method whose fusion operator is poor may still reach acceptable accuracy if given enough rounds to re-synchronise the clients, and conversely a method with an excellent fusion rule may show only a modest advantage once every competitor has been allowed to iterate. In this section we therefore isolate the first ability. Each client trains its assigned submodel locally to convergence, without any intermediate communication; the submodels are then fused once, following the rule of each method, and the resulting global model is evaluated with the same per-client protocol used throughout. There is no second round, so no error introduced

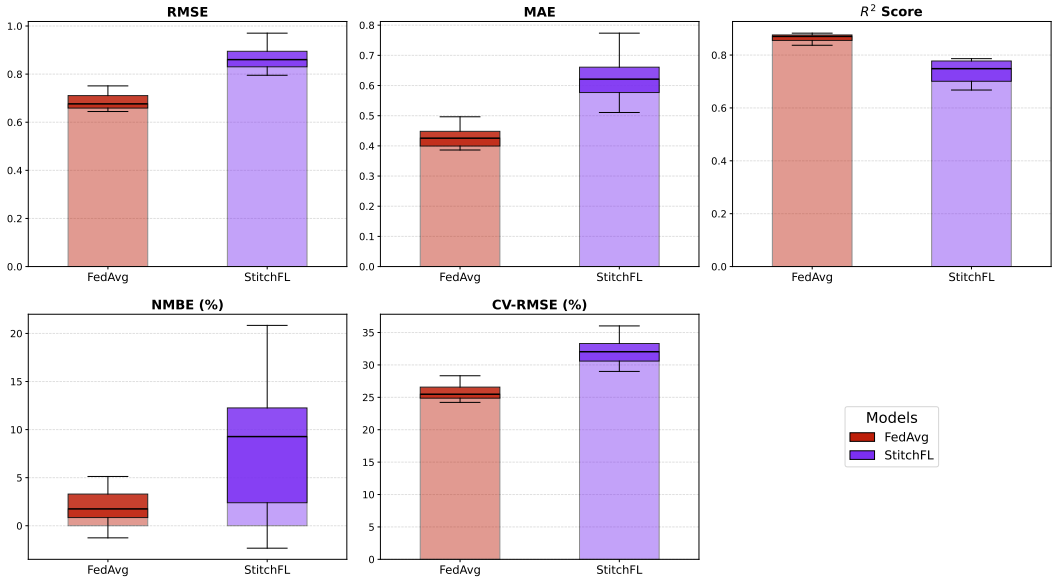


Figure 7: Utility of the fused global model on PLEIAData under all the regression metrics: RMSE, MAE, R^2, NMBE and CV-RMSE. Boxplots summarise 10 independent runs.

by the fusion can be repaired afterwards, and the metrics measure the quality of the fusion operator alone.

D.3.1 DENSE ARCHITECTURE.

Figure 8 reports the one-shot fusion of the MLP on Letter Recognition, under the same free, partially disjoint partitioning across five clients used in Section 4.2.2. STITCHFL attains the best value of every classification metric among the submodel-training methods: an accuracy of 0.862 and a recall of 0.862, ahead of FEDGSE (0.827), SUBFLOT (0.683) and HALOFL (0.637). The ordering is preserved on F1-score (0.862 against 0.822, 0.660 and 0.605) and on MCC (0.856 against 0.819, 0.678 and 0.627), so no metric reverses the conclusion. Since a single fusion admits no subsequent round in which an incoherent reassembly could be corrected, this ranking measures the quality of the fusion operator itself rather than the ability of the federation to recover from a poor one.

Two aspects of the figure are worth noting beyond the ranking. First, the gap between precision and recall again separates the methods by how evenly the damage of the fusion is distributed across classes. STITCHFL loses less than one point between the two (0.871 against 0.862), whereas HALOFL and SUBFLOT show gaps of 5.3 and 7.9 points (0.690 against 0.637, and 0.762 against 0.683), which indicates that their fused models concentrate on part of the label space and under-predict the rest. The F1-score, which combines both, is where these two methods fall furthest behind. Second, STITCHFL and FEDGSE show visibly tighter boxes than SUBFLOT and HALOFL, whose whiskers span up to ten points on F1-score, so the quality of their single fusion depends appreciably on the run.

Finally, the comparison with the iterative setting is informative. The accuracy that STITCHFL reaches after a single fusion (0.862) is essentially the one it attains after ten communication rounds in Section 4.2.2, which indicates that on this task the fusion operator reassembles a coherent model in one step, and that the subsequent rounds refine it rather than repair it. This is the intended behaviour of a data-free fusion built on normalised averaging of overlapping slices, pseudo-inverse stitching and anchor-based readout recalibration, and it is what makes the method applicable in settings where repeated communication is not available.

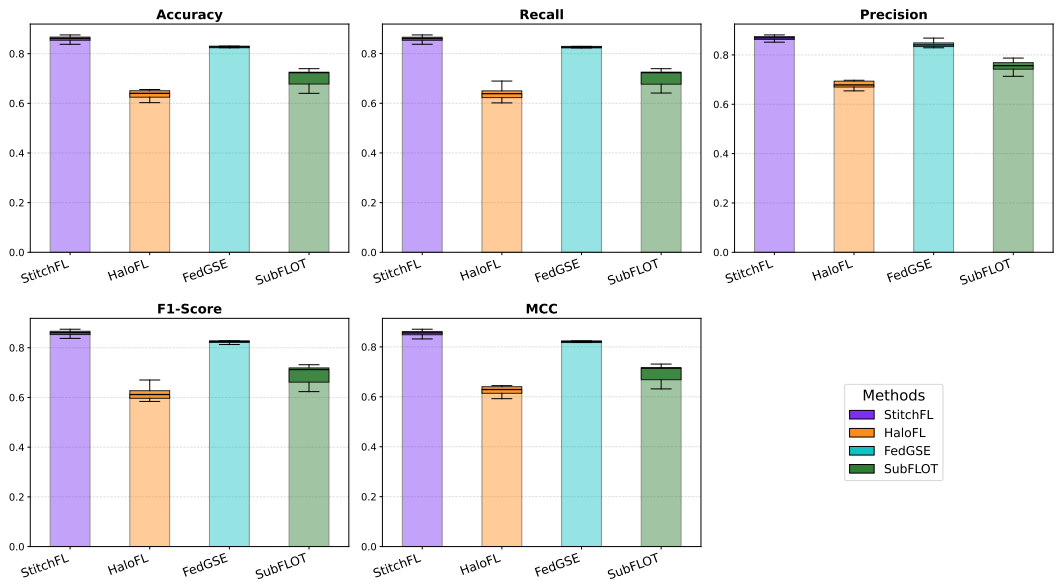


Figure 8: One-shot fusion of the MLP on Letter Recognition.

D.3.2 Convolutional architecture.

Figure 9 reports the corresponding experiment on CIFAR-10. StitchFL again yields the best reconstruction on the metrics that capture overall classification quality, with the highest accuracy (0.4857), recall (0.4861), F1-score (0.4890) and MCC (0.4388), ahead of SubFLOT (0.3919 accuracy, 0.3900 F1, 0.3841 MCC) and HaloFL (0.3473, 0.3412, 0.3294). The gap is starkest against FedGSE, whose single fusion essentially fails to classify: with an accuracy of 0.0993, a recall of 0.1000 and an MCC of exactly 0.0000 it performs no better than random assignment over the ten classes, and its precision of 0.0099 indicates that it collapses onto a single label. This collapse stems from the aggregation rule rather than from the neuron-selection criterion: FedGSE blends each client update into the previous global model instead of replacing it, so that under a single fusion a large share of the weights still carries the random initialisation. The rule is designed to be applied over many communication rounds, and Section 4.2.2 examines its behaviour under that regime; the one-shot figure should therefore be read as a lower bound for delta-based aggregation. The only metric on which StitchFL does not rank first is precision, where SubFLOT (0.5999) and HaloFL (0.5341) edge ahead of our 0.5134; this is the familiar pattern of models that buy precision with recall, as confirmed by their recall values (0.391 and 0.347) falling well below ours (0.486), and both the F1-score and the MCC resolve the trade-off in our favour.

D.3.3 Recurrent architecture.

Figure 10 reports the one-shot fusion of the two-layer LSTM on the PLEIAData forecasting task, against the complete model trained locally by each client, since no competing submodel-training method is applicable to recurrent architectures. Here the fused model tracks the reference remarkably closely across all five metrics despite never having been trained as a whole, and on the coefficient of determination it even slightly surpasses it, reaching $R^2 = 0.680$ against 0.651. This inversion is a property of the reference rather than a gain of our method. With a single round, the complete-model baseline averages parameters of recurrent networks that each client trained independently, and that average is taken over *every* unit of the model before the test is run, with no subsequent round in which the clients could re-synchronise around it. StitchFL, in contrast, averages only where client slices actually overlap: most cells are contributed by a single client or left at the shared template, so the great majority of the fused parameters are never mixed at all, and the fused model inherits far less of the damage that averaging misaligned recurrent units causes. The absolute error metrics show only a marginal degradation (RMSE of 1.016 versus 0.970, MAE of 0.720 versus 0.627) and the relative dispersion is almost unchanged (CV-RMSE of 37.59% against

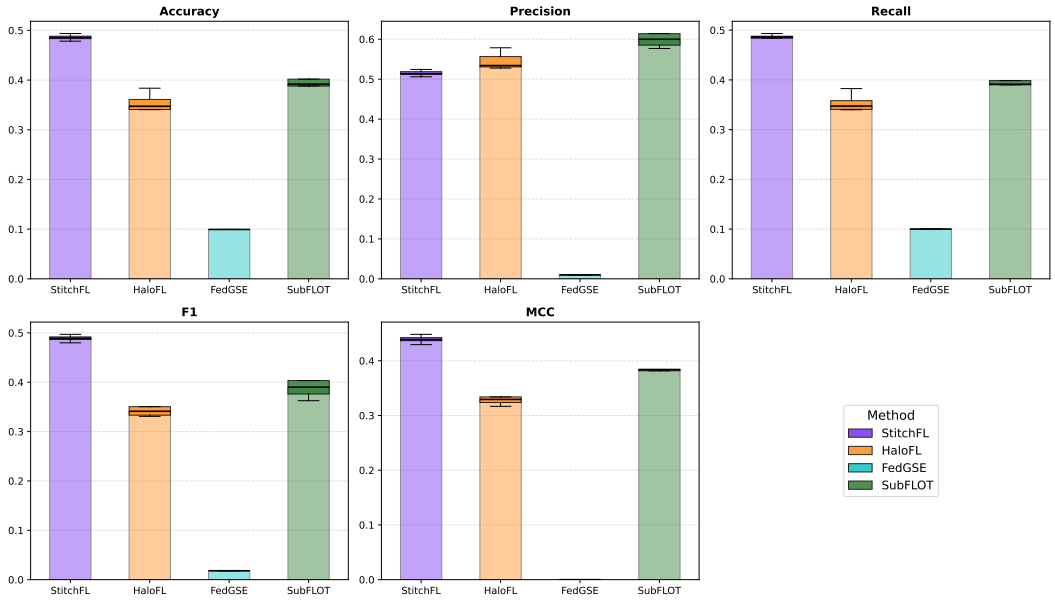


Figure 9: One-shot fusion of the global CNN on the CIFAR-10 classification task.

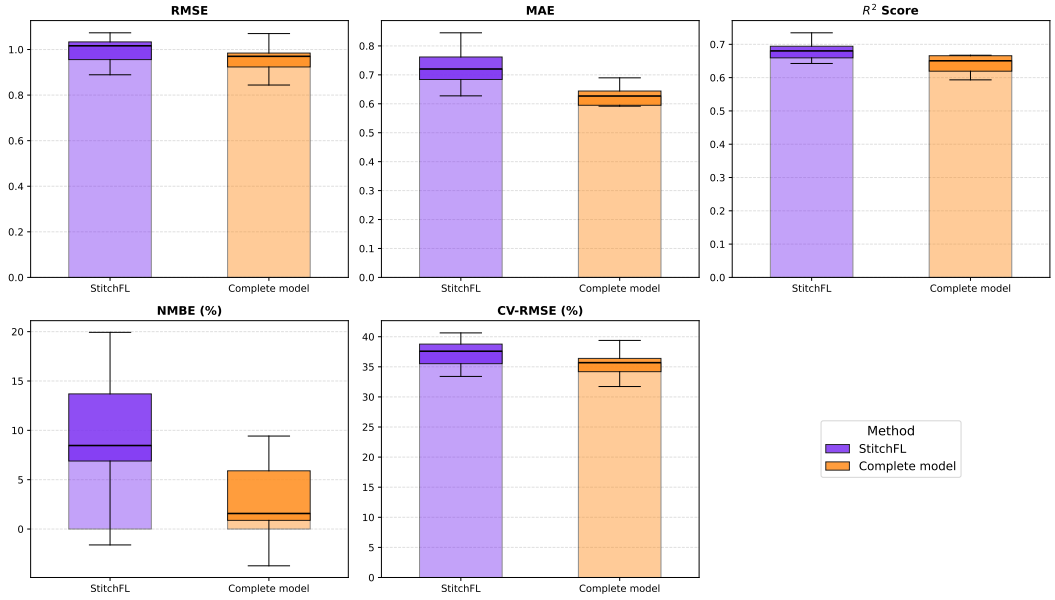


Figure 10: One-shot fusion of the STITCHFL LSTM against the complete model on the PLEIAData forecasting task.

35.69%). As in the federated setting, the only metric with a visible gap is the normalised mean bias error (8.46% against 1.58%), the expected footprint of template completion and anchor-based head recalibration. That a single fusion of overlapping recurrent submodels can match a model trained end-to-end confirms that the shared random template does give units a common identity across clients, which is the assumption on which cell-by-cell averaging rests.

D.3.4 ANALYSIS

Two conclusions follow. First, when a method performs poorly in the federated setting, this experiment identifies whether the cause is the fusion operator or its interaction with the round structure—a

distinction the multi-round figures alone cannot make. Second, the ranking of the methods is not preserved across regimes: FEDGSE is the second-best method after ten rounds on the MLP but the worst by a large margin in one shot on the CNN, so any deployment with a constrained communication budget would select a very different method than the multi-round results suggest. STITCHFL is the only one of the four whose behaviour is stable across both regimes, which is precisely the property required if the number of rounds is to be reduced for privacy reasons.

D.4 ABLATION STUDY

The experiments of Section 4.2.2 evaluate a single, fixed partitioning of the global model. Because STITCHFL imposes no structural constraint on how the architecture is divided, however, the space of admissible client assignments is vast, and the quality of the reconstruction is expected to depend strongly on which point of that space is used. In this section we therefore probe the behaviour of the fusion mechanisms along four orthogonal degrees of freedom of the federated scenario, each isolated in a *family* of configurations and varied across three cases ordered from the mildest (case 1) to the most extreme (case 3). To keep the analysis tractable and interpretable we restrict it to the multilayer perceptron and fix the aggregator model to a network with three hidden layers of 60 neurons each. Every configuration is run five times with independent seeds, and for each of the five classification metrics we report the median across repetitions, joined by a line within each family, so that the trend along the axis is visible, together with the boxplot of the five runs, which exposes the run-to-run variability that a single aggregate would hide. The four families are the following.

- **Family A — data heterogeneity.** Controls how the training data are distributed across clients, from the original, approximately IID split to increasingly skewed partitions obtained with a Dirichlet allocation of concentration $\alpha = 0.5$ and $\alpha = 0.1$. The architectural partitioning is held fixed, so the family isolates the effect of statistical heterogeneity alone.
- **Family B — concentration.** Controls how evenly the neurons are distributed among clients, from a balanced split in which all clients receive submodels of equal size to a highly concentrated one in which a single client holds the majority of the network and the remaining clients are left with small slices.
- **Family C — disjointness / overlap.** Controls how much the client submodels overlap, from fully disjoint slices in which no neuron is shared by two clients to the highest degree of balanced overlap short of full replication.
- **Family D — stitching.** Controls the number of layers that clients skip entirely and that must therefore be folded back through the least-squares procedure of Section 3.2.1, from one stitching operation to three.

Families A–C are evaluated for the four methods under comparison (STITCHFL, HALOFL, FEDGSE and SUBFLOT), each of which receives exactly the same scenario so that only the extraction and fusion rules differ. Family D is reported for STITCHFL alone: allowing a client to hold no neuron at all in a given layer is precisely the constraint that the competing methods do not relax—all of them require every client to train a fraction of every layer—so no baseline is even definable in these configurations. That the family exists only for our method is in itself part of the result.

D.4.1 FAMILY A: FROM IID TO NON-IID.

Figure 11 reports the effect of statistical heterogeneity, with the architectural partitioning held fixed and the data redistributed from the original, approximately IID split (A1) to Dirichlet allocations of concentration $\alpha = 0.5$ (A2) and $\alpha = 0.1$ (A3). Every method degrades monotonically along the axis, but they do not degrade at the same rate, and the ranking changes along the way. In the IID case STITCHFL is clearly ahead, with an accuracy of 0.782 against 0.719 for FEDGSE, 0.688 for SUBFLOT and 0.308 for HALOFL. At $\alpha = 0.5$ it has lost 16.7 points (0.615) and is overtaken by FEDGSE (0.652), and at $\alpha = 0.1$ the three competitive methods converge to a narrow band between 0.322 and 0.327, with HALOFL at 0.159.

The steeper slope of STITCHFL is a direct consequence of what makes it data-free. Its fusion averages overlapping slices position by position, weighting every client equally, and repairs the readout from anchors on which the clients act as an ensemble. Both primitives assume that the

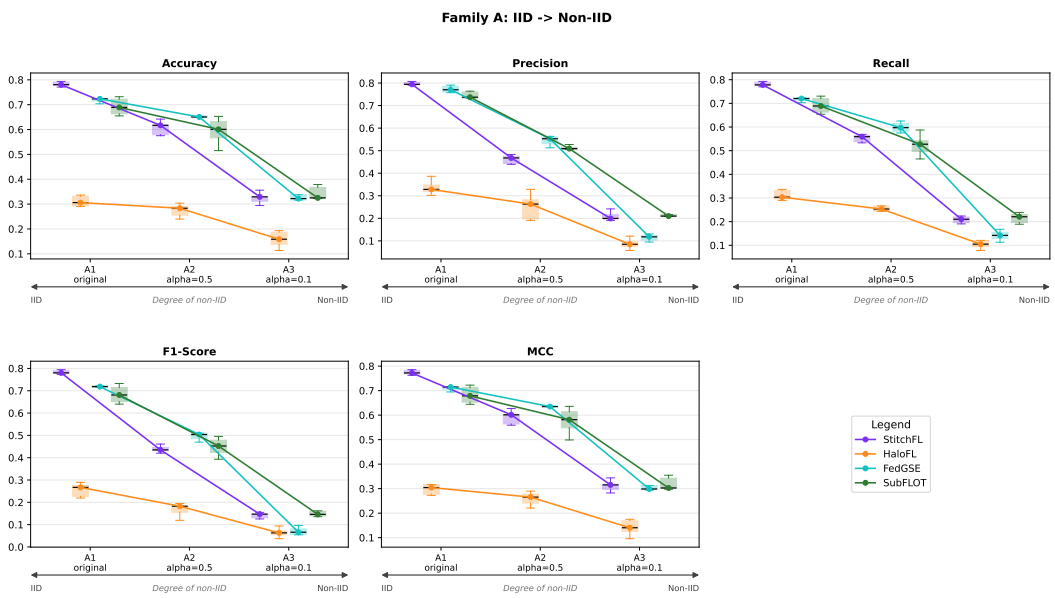


Figure 11: Family A: effect of data heterogeneity, from the original IID split to Dirichlet allocations with $\alpha = 0.5$ and $\alpha = 0.1$. Lines join the medians over five runs; boxplots show the run-to-run variability.

clients are estimating comparable functions. Under a strong label skew this assumption breaks: each client fits a different class-conditional problem, so the slices of a shared neuron encode conflicting features and their unweighted average is not a good estimate of either. The baselines select which parameters a client trains from data-dependent statistics, which gives them a partial adaptation to the skew, and this is exactly what shows up as the crossing at A2. It is also the reason why the comparison of Section 4.2.2 deliberately keeps the data as close to IID as possible: mixing both effects would confound the cost of partitioning the model with the cost of heterogeneous data.

The metrics that account for class balance qualify this reading. At $\alpha = 0.1$ the F1-score collapses far below the accuracy for every method, which indicates that the fused models predict only a subset of the classes. In that regime StitchFL retains an F1-score of 0.143 and an MCC of 0.313, against 0.065 and 0.300 for FedGSE, so the apparent parity in accuracy hides the fact that the model fused by StitchFL is still the one that spreads its predictions over more of the label space. In other words, extreme heterogeneity erodes the advantage of our fusion but does not invert it once the imbalance of the errors is taken into account. We will overcome the effect of non-IID in the future work in order to make StitchFL more robust against these scenarios.

D.4.2 Family B: from balanced to one-client focus.

Figure 12 varies how evenly the neurons are distributed among clients, from submodels of equal size (B1) to a configuration in which a single client holds the majority of the network and the rest receive small slices (B3). The behaviour of StitchFL is essentially flat, moving from 0.780 to 0.799 and 0.803 in accuracy, with equally flat curves on the remaining metrics. The baselines behave in the opposite way and improve markedly as the partition becomes concentrated: HaloFL rises from 0.372 to 0.660, SubFLOT from 0.688 to 0.817 and FedGSE from 0.724 to 0.743, the latter after a dip at B2 (0.688) whose box spans roughly 0.62 to 0.72 and is therefore the least stable point of the family.

The explanation is that concentration dissolves the problem these methods are solving. When one client trains almost the entire architecture, the fused model is close to that client's model, most parameters are covered by a single well-trained submodel, and the allocation rule has little left to decide. SubFLOT exploits this best and ends 1.4 points above StitchFL at B3. This is not an advantage that transfers to our setting, for two reasons. First, the configuration that favours the baselines is the one in which submodel training is least meaningful, since the federation degenerates

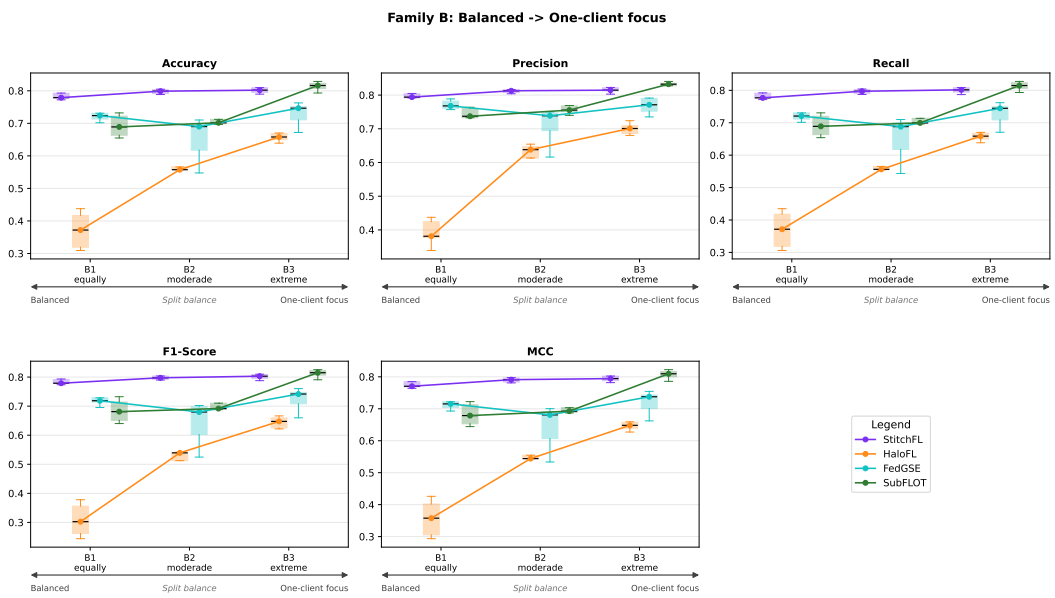


Figure 12: Family B: effect of the concentration of the partition, from equally sized submodels to a configuration in which one client holds the majority of the network. Lines join the medians over five runs; boxplots show the run-to-run variability.

into one client doing nearly all the work. Second, and more important, that client is exposed to a model of nearly full capacity, which Appendix D.1.2 shows to be the most vulnerable to membership inference, so the utility gained at B3 is paid for with the privacy that motivates the whole scheme. The relevant property of StitchFL in this family is therefore its insensitivity: it attains its best results without requiring any client to hold a large share of the network.

D.4.3 Family C: from disjoint to overlapping submodels.

Figure 13 varies the degree of overlap between client submodels, from fully disjoint slices (C1) to the highest degree of balanced overlap short of full replication (C3). All the methods improve monotonically, which is expected: overlap creates redundancy, and every position covered by several clients is averaged over several independent estimates instead of being taken from a single one, or left to the stitching and repair primitives. The gain is largest for the weakest method, HaloFL, which climbs from 0.357 to 0.800. StitchFL leads the two configurations in which overlap is scarce or moderate, with 0.784 against 0.724 at C1 and 0.878 against 0.874 at C2, which is the regime its primitives are designed for: normalised averaging of partially overlapping slices, pseudo-inverse stitching of the layers a client skips, and a readout repaired from anchors are precisely the mechanisms needed when there is little consensus to rely on.

At the highest overlap, the ordering changes marginally, with FedGSE reaching 0.928 against 0.918 for StitchFL, a difference of 1.0 point that is also reflected in F1-score (0.928 against 0.918) and MCC (0.926 against 0.918). However, this regime is less representative of the main challenge addressed by model-heterogeneous FL. As submodels grow, a larger number of neurons is necessarily shared across clients, making the training process progressively closer to standard FedAvg; consequently, differences between fusion strategies become less informative. The more demanding and relevant setting is instead the one in which clients can only accommodate small fractions of the global model, since this is precisely where submodel-training methods are expected to provide value. Moreover, overlap is not free: enlarging the slice assigned to each client increases its exposed model capacity, and the leakage of a submodel grows with its capacity, as established in Section 4.2.1 and confirmed in Appendix D.1.2. Configuration C3 therefore achieves a modest accuracy gain at the cost of exposing each client to a substantially larger fraction of the network and moving the setting closer to full-model federated training. Seen jointly with Section 4.2.3, overlap should therefore be understood as a tuning parameter of the privacy–utility trade-off rather than as a free source of

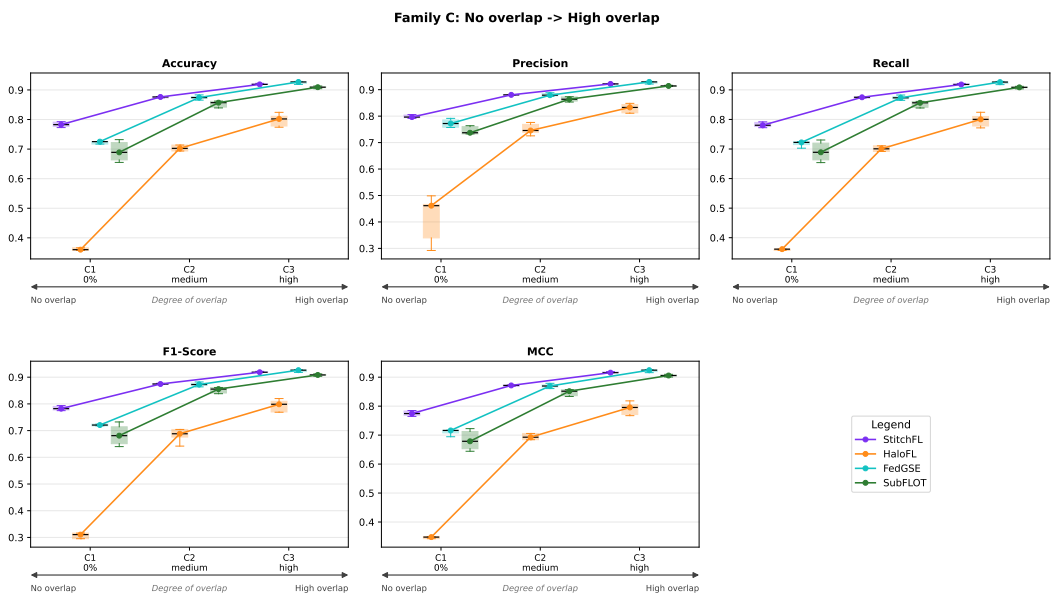


Figure 13: Family C: effect of the overlap between client submodels, from fully disjoint slices to a high degree of balanced overlap. Lines join the medians over five runs; boxplots show the run-to-run variability.

improvement, with the most informative operating points being those where client capacity remains genuinely constrained.

D.4.4 Family D: amount of stitching.

Figure 14 isolates the pseudo-inverse stitching procedure by increasing the number of clients that skip a hidden layer entirely, from one to three. The degradation is monotone but mild: the median accuracy falls from 0.7707 to 0.7659 and 0.7488, a total loss of 2.2 points, with F1-score and MCC following the same shape (0.7732 to 0.7508 and 0.7622 to 0.7393). This is what the operator predicts, since each skipped layer chains an additional least-squares solve through the Moore–Penrose pseudo-inverse, so one local-linearity approximation accumulates per push and the folded weight inherits the conditioning of every intermediate block it traverses. The errors compose rather than cancel, but the composition stays bounded: no repetition shows the amplification that an ill-conditioned factor could in principle cause, and the intermediate case is the most variable one only because its spread reflects which layers happen to be skipped. Skipping layers is therefore a usable degree of freedom rather than merely an admissible one, which matters because every skip further reduces the capacity a client holds, and with it the surface exposed to an attacker.

Taken together, the four families confirm the intuitions built into the design of StitchFL and yield concrete deployment guidance. Our method performs best in most of the close-to-IID configurations, with its advantage largest precisely where fusion is hardest, namely disjoint or barely overlapping submodels, and it is the only one insensitive to how capacity is distributed, since it does not need a client to concentrate most of the network in order to fuse well. Its limits are equally clear: strong label skew erodes its advantage, because its fusion never queries the data distribution and an unweighted average of slices fitted to conflicting class-conditional problems is no longer a good estimate, while heavily overlapping or concentrated partitions let the baselines catch up at the cost of exposing a larger fraction of the model to each client. Both are configurations one would not choose in a privacy-motivated deployment, where the point is precisely that no participant holds much of the network. Within the regimes such a deployment defines, StitchFL is accurate, stable across runs, and degrades gracefully when clients skip layers altogether.

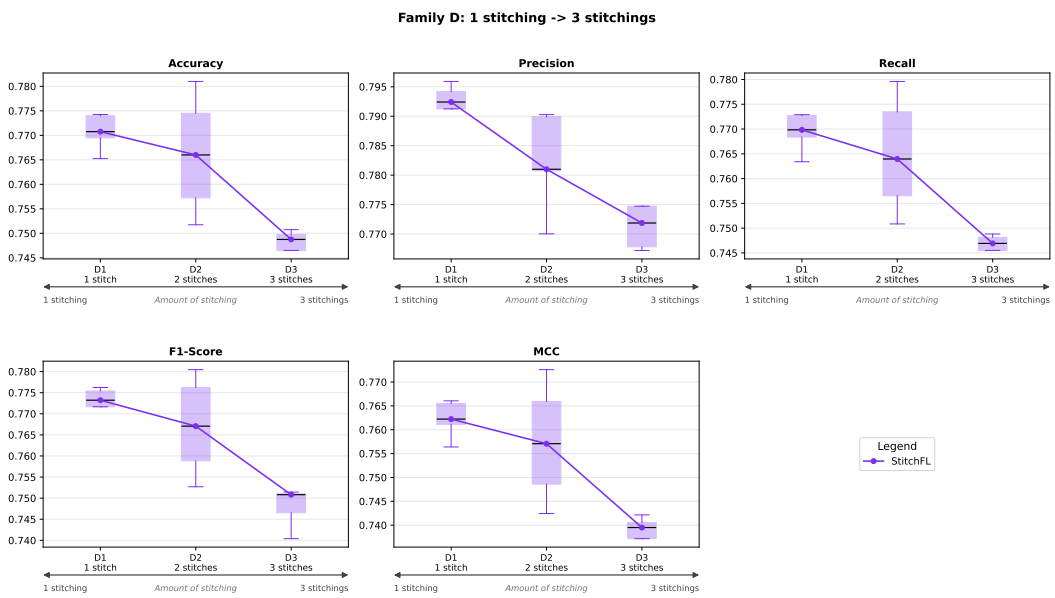


Figure 14: Family D: effect of the number of stitching operations.

D.4.5 Ablation on the components of StitchFL

The four families above compare the methods under a common scenario. This section instead isolates the two data-free primitives that are specific to StitchFL, the anchor-based readout recalibration and the pseudo-inverse stitching of skipped layers, by switching each of them off and by varying the number of anchors. The starting points are two configurations of the previous families: B1, the balanced partition in which every client receives a submodel of the same size and no client skips a layer, and D1, the mildest stitching configuration, in which one client skips a hidden layer entirely and the resulting skip-edge must be folded back through the least-squares procedure of Section 3.2.1. Table 2 reports the accuracy of the fused model in each case.

The readout recalibration is the component with the largest individual effect. Removing it altogether, so that the fused network keeps the averaged output layer, costs 4.6 points of accuracy (0.7380 against 0.7844). This is the expected failure: after averaging, the last hidden layer of the merged model is not the one any client was trained to read from, so the inherited readout is applied to features it was not fitted on. Recalibration repairs precisely this mismatch, and it does so without touching client data, since the server generates synthetic anchors and the clients only return their predictions on them.

The number of anchors controls the quality of that repair and shows clear diminishing returns. With only 200 anchors the recalibration already recovers most of the gap (0.7629, that is 2.5 points above no recalibration), and accuracy then grows to 0.7759 with 1000, 0.7819 with 5000 and 0.7844 with 20000. The behaviour follows from the nature of the operation: the anchors act as the design matrix of a ridge regression whose unknowns are the weights of the output layer, so their number governs how well conditioned that system is, and the estimation error decreases with the number of rows until the solution is essentially determined. Beyond a few thousand anchors the remaining gain is a fraction of a point. Since anchors are synthetic and cost only one forward pass per client in a single round at the end of training, a large value can be afforded, but the ablation shows that the method is not sensitive to this choice: any value above a few hundred anchors retains most of the benefit.

The stitching experiment measures a different trade-off. In D1 one client holds no neuron of a hidden layer, so its submodel is strictly smaller than in B1 and it never sees that layer at all. This configuration is not available to any of the baselines, which require every client to hold at least one unit of every layer, and it is the most favourable one from the privacy standpoint, since the capacity exposed to that client is reduced further and Appendix D.1.2 shows leakage to grow with capacity. The price is a loss of utility: with stitching, the fused model reaches 0.7651, 1.9 points below

Configuration	Accuracy
B1 – balanced partition, no skipped layers	
StitchFL (default)	**0.7844**
Without readout recalibration	0.7380
200 anchors	0.7629
1000 anchors	0.7759
5000 anchors	0.7819
20000 anchors	0.7844
D1 – one client skips a hidden layer	
Without stitching (zero connections)	0.7459
With stitching	**0.7651**

Table 2: Ablation on the components of StitchFL.

the 0.7844 of B1. The relevant comparison, however, is against the alternative way of handling a skipped layer, namely leaving the corresponding connections at zero. Doing so yields 0.7459, so the least-squares stitching recovers 1.9 of the points that a zero default would lose. In other words, allowing a client to skip a layer costs accuracy but buys privacy, and stitching halves the cost of that decision, which is what makes the free partitioning of StitchFL practical rather than merely admissible.