FEDCENSUS: PARTITION-INVARIANT FEDERATED LEARNING FROM ADDITIVE SUFFICIENT STATISTICS

Anonymous authors
Paper under double-blind review

ABSTRACT

Federated learning trains a shared model across clients while keeping their data local. With weight averaging, the model drifts when client data is non-IID, and the accuracy is bounded by the lowest-computation client. Methods that share prototypes, logits, or synthetic data share only the knowledge from each client's own dataset. We present FedCensus, a federated model with a fine and a coarse codebook learned inside the federation, three additive feature statistics computed against them, and a closed-form classifier, so every parameter is computed from additive client statistics. Because additive statistics do not depend on how the data is split, the trained model is the same under every partition and every number of clients. Because the sums cover every class, each client's model achieves nearly the same accuracy on classes absent from its local dataset as on those present. A client with a lower training budget uploads its statistics in installments and chooses its deployment level after training, so the accuracy of the deployed model is not bound by the lowest-computation client. On four datasets, FedCensus exceeds the state-of-the-art computation-heterogeneous, prototype-based, and logit-based methods. On CIFAR-100, FedCensus is comparable to the best homogeneous method at matched deployment FLOPs. FedCensus reaches this accuracy in 18 communication rounds and reduces the communication overhead by 60% compared with FedAvg. The complete source code is available at `https://anonymous.4open.science/r/fedcensus-79E5/`.

1 INTRODUCTION

Overview Federated learning (FL) trains models across clients whose data cannot leave their devices (McMahan et al., 2017). Homogeneous FL trains one shared network by weight averaging, but every device must train the full network, and the updates conflict when client data is not independent and identically distributed (non-IID), a failure known as client drift (McMahan et al., 2017; Karimireddy et al., 2020; Acar et al., 2021). Model-heterogeneous FL lets each client pick its own architecture, so knowledge cannot be shared by averaging the weights and is instead shared by weaker channels such as prototypes, logits, or synthetic data (Li & Wang, 2019; Tan et al., 2022; Jeong et al., 2018; Zhang et al., 2022). Computation-heterogeneous FL keeps one shared architecture and trains subnetworks that fit client budgets (Diao et al., 2021; Alam et al., 2022). Yet subnetwork methods still aggregate conflicting updates, so drift remains. No direction so far shares knowledge faithfully, because every channel loses knowledge when the server aggregates it.

Motivation In homogeneous FL, the same model must fit every device, so the lowest-computation client bounds the accuracy of every client. Model-heterogeneous FL removes the bound, but clients can only share and receive knowledge related to their own local data distributions. Also, the accuracy reported for these methods is measured with per-client frequency-weighted metrics at each client's best round, and it degrades under unweighted metrics (Zhang et al., 2024; Lee et al., 2025) (Section 4.2). Computation-heterogeneous FL tries to solve both challenges by dividing a global model into subnetworks that fit the clients' budgets and averaging the subnetworks' weights. However, the small subnetworks are nested inside the large ones and receive more gradient updates, so the global model is trained unevenly, the client drift persists, and low-computation clients still bound the accuracy. Statistics-based FL avoids drift with closed-form aggregation, but it computes its features with a backbone pretrained outside the federation (Section 2). Therefore, resolving both

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.

challenges requires a framework that does not drift, shares knowledge beyond local data coverage, is not bounded by its low-computation clients, and learns its encoder inside the federation.

Proposed work We present FedCensus, a computation-heterogeneous FL framework that aims to solve the challenges above. FedCensus learns a codebook encoder inside the federation, encodes each image into additive statistics against the codebook, and solves the classifier in closed form, so every parameter is computed from additive client statistics without stochastic gradient descent (SGD) (Robbins & Monro, 1951).

Contributions Our contributions are fourfold.

- We design **two complementary codebooks**, learned from additive client statistics, which encode fine-grained activation responses and coarse feature interactions.
- We construct the representation from **three codebook-based feature statistics**, activation sums, pair products, and spatial moments, which capture activation, co-activation, and spatial information, and the classifier is solved in closed form.
- We enable **computation heterogeneity** by defining each deployment level as a subset of the fine codewords. All parameters are trained by all clients, so the accuracy of FedCensus is not bounded by the lowest-computation client.
- FedCensus uses additive statistics and closed-form solutions to **avoid client drift and dependence on the data partition**. Clients receive knowledge beyond their local data coverage, and the resulting model equals the centrally trained model.

2 Related Work

FedCensus builds on three bodies of work. Heterogeneous FL defines our challenges, statistics-based FL shares similar approaches, and gradient-free representation learning is our foundation.

Heterogeneous federated learning Model-heterogeneous FL shares knowledge through channels other than weights. FedProto and its successors upload class prototypes (Tan et al., 2022; Zhang et al., 2024; Lee et al., 2025), FedMD and later methods exchange logits on public data (Li & Wang, 2019; Jeong et al., 2018; Lin et al., 2020), and DENSE and other generator methods synthesize data to distill the client models (Zhang et al., 2022; Dai et al., 2024; Yao et al., 2026). Computation-heterogeneous FL trains subnetworks of a model, scaled in width (Diao et al., 2021; Alam et al., 2022) or depth (Kim et al., 2023) or selected by importance (Wu et al., 2024), but client drift persists under SGD and weight averaging even with control variates or regularization (Karimireddy et al., 2020; Acar et al., 2021; Li et al., 2021; Fan et al., 2024). FedCensus keeps the shared architecture but replaces SGD and weight averaging with closed-form solutions on additive statistics.

Statistics-based federated learning Statistics-based FL replaces gradient aggregation with closed-form solutions, which avoids the non-convexity that causes client drift (Yu et al., 2022; Pathak & Wainwright, 2020). Fed3R and its successors solve classifiers from aggregated feature statistics and prove the partition invariance of the solution (Fanì et al., 2024; Guan et al., 2025; He et al., 2025; Tang et al., 2025; 2026), and other methods aggregate curvature, posterior, or calibration statistics (Jhunjhunwala et al., 2024; Al-Shedivat et al., 2021; Luo et al., 2021). All of these methods compute their features with a backbone pretrained outside the federation, so the invariance covers only the layers above the pretrained features (Fanì et al., 2024; Tang et al., 2026), and the backbone forces a minimum inference cost on every client. FedCensus trains the whole model, encoder included, without pretraining, and the whole model is independent of the partition.

Gradient-training-free representation learning Whitened patches with a k-means codebook (Coates & Ng, 2012), whitened patch dictionaries (Thiry et al., 2021), and scattering features (Mallat, 2012; Oyallon et al., 2019) reach strong accuracy with a linear classifier, but only in the centralized setting. Statistical-query algorithms can be written as exact sums over data, which parallelizes them (Chu et al., 2006; Dhillon & Modha, 2000), and federated clustering trades exactness for one communication round (Dennis et al., 2021). FedCensus leverages the exact summation form for

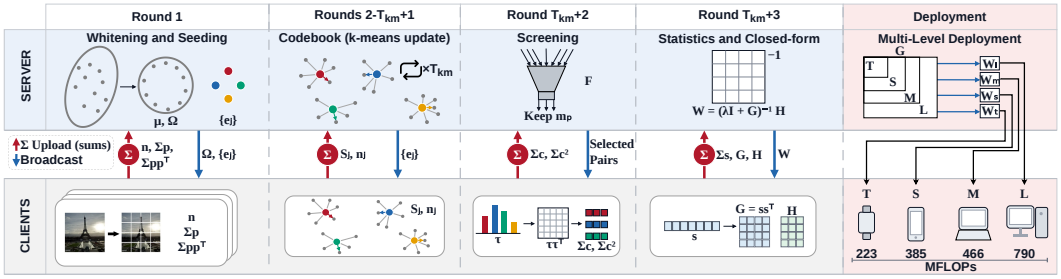

Figure 1: The FedCensus pipeline. Columns are the training rounds and the deployment stage. The upper lane is the server, and the lower lane is the clients. Red arrows indicate uploads and blue arrows broadcasts, labeled with what is sent. The last column shows the four deployment levels.

every encoder stage and the classifier, so the federated model equals the centrally trained one. To the best of our knowledge, FedCensus is the first federated model that learns its encoder from raw input without pretrained parameters and computes every parameter from additive client statistics.

3 FedCensus

3.1 Setting and Constraints

The K clients own datasets $\mathcal{D}_1, \ldots, \mathcal{D}_K$ that partition one training set with C classes. Each training sample belongs to exactly one client, and a client's dataset may be empty. We write n_k^c for the number of class-c samples on client k. A round consists of the clients' uploads and one broadcast, and the server aggregates and broadcasts once every requested upload has arrived. The uploads of a round arrive in one or more upload slots, one per installment. Each client has a training budget, which bounds the computation between two of its uploads, and a deployment budget, which bounds the cost of one forward evaluation. The budgets differ across clients (Diao et al., 2021; Alam et al., 2022). We require each client to deploy a model within its own deployment budget, while allowing the deployed model size and inference cost to differ among clients.

We assume full participation, so each client uploads every requested statistic, in one slot or in installments, and no client drops out. Clients do not synchronize at every step, because that makes the federation one centralized run with a large communication overhead (McMahan et al., 2017). We use no pretrained parameters, because a pretrained representation is learned outside the federation and a backbone forces a minimum inference cost on every device. We assume no public dataset, because many domains have none close to the clients' data (Zhang et al., 2022). We train no generative model on client data, because its samples can reveal that data (Hitaj et al., 2017; Yin et al., 2020).

3.2 Pipeline

The FedCensus model consists of a patch encoder and a closed-form classifier. The encoder splits each image into overlapping patches and compares them with learned codewords, and every comparison yields an activation. The activation sums, the pair products, and the spatial moments of the activations form the feature vector, and the server builds the classifier with a ridge regression over it. Training takes one whitening-and-seeding round, T_{km} codebook rounds, and one round each for screening and statistics, with no other communication. Every parameter is computed from additive client statistics, never by gradient descent. Section 3.3 states FedCensus's theoretical properties, and Appendix A.2 proves them. Figure 1 shows the pipeline, and Algorithm 1 lists the procedure.

Patches and whitening The encoder builds on whitened patches and the triangle encoding (Coates & Ng, 2012), and every parameter of the encoder is computed from client statistics without pretraining. Each image is split into overlapping $w \times w$ patches, because distances stay discriminative in the patch space and lose contrast at the whole-image dimension (Beyer et al., 1999). Every statistic is computed over each image's augmented copies, whose flips and crops are seeded per image. Contrast normalization subtracts each patch's mean and divides by its standard deviation, and whitening stops low-frequency correlations from dominating the patch distances. In the first round, clients up-

load the count, the sum, and the outer-product sum of their normalized patches $\tilde{p}$, and the aggregated sums give the mean μ, the covariance Σ, and a rank-r PCA whitening map Ω,

$$\Sigma \;=\; V\Lambda V^{\top}, \qquad \Omega \;=\; V_r\,(\Lambda_r + \varepsilon I)^{-1/2}, \qquad u \;=\; \Omega^{\top}(\tilde{p} - \mu), \tag{1}$$

where ε keeps the low-variance directions from amplifying noise.

Codebooks The server learns the two codebooks that the patches are compared against. The fine codebook contains J codewords and is used to compute the activation sums, while the coarse codebook contains J_P codewords and is used to compute the pair products and the spatial moments. The number of pair products grows quadratically with J_P. For each image, the client computes a few seeding centroids of its patches by a small k-means inside the image, seeded by the image's index, and uploads them. The server sorts the uploaded centroids lexicographically, selects the initial codewords with the global seed, and whitens them. Codeword indices follow this order, and two centroids tie only when they are the same vector. A centroid averages many patches, so no raw patch is leaked. The initialization does not depend on the partition. The server then learns both codebooks with batch k-means (Lloyd, 1982; Dhillon & Modha, 2000; Chu et al., 2006). In each codebook round, every client assigns its whitened patches to the nearest codeword and uploads per-codeword sums $S_j^{(k)}$ and counts $n_j^{(k)}$, and the server sets each codeword to the aggregated mean,

$$a(u) = \arg\min_j \|u - e_j\|_2, \qquad e_j \;\leftarrow\; \frac{\sum_{k=1}^{K} S_j^{(k)}}{\sum_{k=1}^{K} n_j^{(k)}}. \tag{2}$$

This update equals the centralized iteration (Dhillon & Modha, 2000; Chu et al., 2006), so the codebooks do not depend on the partition. A codeword with no assigned patches keeps its old value.

Activation sums The first statistic, the activation sums z_A, uses the fine codebook. For each patch p, the encoder computes the distance d_{pj} to every codeword and the activation $\tau_{pj} = \max(0, \bar{d}_p - d_{pj})$ (Coates & Ng, 2012), where the threshold $\bar{d}_p$, shared by both codebooks, is the patch's mean distance to the coarse codewords. The encoder sums each codeword's activations in each of the four quadrants, so $z_A \in \mathbb{R}^{4J}$ keeps position at quadrant granularity.

Pair products The second statistic, the pair products z_P, uses the coarse codebook and measures how strongly two codewords are activated together on the same patch, which the activation sums cannot express. For every pair of the J_P codewords, including self-pairs, the encoder averages the product of their activations over the image and takes the square root to stabilize the scale, $z_P[j,l] = \big(\frac{1}{|\mathcal{P}|}\sum_{p\in\mathcal{P}} \tau_{pj}^P\,\tau_{pl}^P\big)^{1/2}$, where $\mathcal{P}$ is the set of patch positions and $\tau_{pl}^P = \max(0, \bar{d}_p - \|u_p - e_l^P\|_2)$ is the activation of patch p at coarse codeword l. The $J_P(J_P{+}1)/2$ pairs are too many to keep, so the server screens them. Clients upload per-class sums, sums of squares, and counts for every pair, and the server keeps the m_P pairs with the largest analysis-of-variance F values,

$$F[j,l] \;=\; \frac{\mathrm{SS_b}[j,l]\,/\,(C-1)}{\mathrm{SS_w}[j,l]\,/\,(n-C)}, \tag{3}$$

where $\mathrm{SS_b}$ and $\mathrm{SS_w}$ are the between-class and within-class scatters and n is the total sample count. Every screening upload is a sum, so the selection is identical under any partition (Section 3.3).

Spatial moments The third statistic, the spatial moments z_M, uses the coarse codebook, because the activation sums keep position only at quadrant granularity and the pair products keep none. With normalized patch coordinates $g_p \in [-1,1]^2$, the encoder computes for each codeword j the mass m_j, the mean position $\bar{g}_j$, and the spread $\sqrt{v_j}$,

$$m_j = \textstyle\sum_p \tau_{pj}^P, \qquad \bar{g}_j = \frac{1}{m_j}\sum_p \tau_{pj}^P\, g_p, \qquad v_j = \frac{1}{m_j}\sum_p \tau_{pj}^P\, g_p^{\odot 2} - \bar{g}_j^{\odot 2}, \tag{4}$$

where the squares act on the two coordinates separately. The encoder scales the mean and the spread by $\sqrt{m_j}$, because both are unstable near zero mass, so a codeword with no activation contributes zeros. Each codeword yields $\sqrt{m_j}$ and the four scaled coordinates, so z_M has $5J_P$ entries.

Classifier The feature vector s concatenates the three statistics into $D = 4J + m_P + 5J_P$ entries, the features. In the statistics round, clients encode their samples and upload the Gram matrix $G_k = \sum_i s_i s_i^\top$, the class sums $H_k = \sum_i s_i Y_i^\top$, the feature sums $\sum_i s_i$, and the class counts $\sum_i Y_i$, where Y_i is sample i's one-hot label and y_i its class. The server standardizes each feature with the mean μ_s and standard deviation σ_s from the aggregated sums and solves the ridge regression,

$$W \;=\; \Big(\lambda I + \textstyle\sum_{k=1}^{K} G_k\Big)^{-1} \sum_{k=1}^{K} H_k \tag{5}$$

at a fixed ridge weight λ. The prediction is the class with the largest score $W^\top s$, and the deployed classifier contains W, μ_s, and σ_s.

Deployment After training, the model runs at several deployment levels without retraining or communication. A deployment level keeps the top J' fine codewords by their aggregated counts, so the encoder computes only $(J'+J_P)/(J+J_P)$ of the full distances. The kept activations equal the full model's, because $\bar{d}_p$ comes from the coarse codewords. The server solves the smaller classifier from the principal submatrix of the aggregated sums and the entries of μ_s and σ_s on the kept features. A client picks the level that fits its deployment budget, and uploads stay full, so the choice lowers only that client's accuracy. A client that cannot afford a full upload within its training budget splits every sum into without-replacement installments, and the round then has one upload slot per installment. The client uploads each installment as soon as it is computed, because storing it would occupy memory. The server aggregates after every installment has arrived, and the installments sum to the exact totals, so a low training budget adds upload slots and never changes the model. The server waits for every client, so the pipeline tolerates any delay but not a client that drops out.

3.3 PROPERTIES

We state four properties. The lemma and the proposition show that the model depends neither on how the uploads are split across upload slots nor on how the data is split across clients. The corollary applies the classical k-means descent to the federated codebooks, and the remark addresses privacy.

Write θ_t for what the server broadcasts at the end of round t, $z^{(i)}$ for sample i's z, $b(x)$ for image x's seeding centroids, and $\mathcal{L}$ for a level's feature index set. Client k's uploads and the broadcasts are

$$\begin{aligned}
&\text{(round 1)}\ \big(\textstyle\sum_{\tilde{p}} 1,\ \sum_{\tilde{p}} \tilde{p},\ \sum_{\tilde{p}} \tilde{p}\,\tilde{p}^\top,\ \biguplus b(x)\big) \xrightarrow{\text{Eq.(1)}} \theta_1 = (\mu, \Omega, \{e_j\}, \{e_l^P\}),\\
&\text{(rounds } 2, \ldots, T_{\mathrm{km}}{+}1)\ \big(S_j^{(k)},\ n_j^{(k)}\big) \xrightarrow{\text{Eq.(2)}} \theta_t = (\mu, \Omega, \{e_j\}, \{e_l^P\}),\\
&\text{(round } T_{\mathrm{km}}{+}2)\ \big(\textstyle\sum_{y_i=c} z_P^{(i)},\ \sum_{y_i=c} (z_P^{(i)})^{\odot 2},\ n_k^c\big) \xrightarrow{\text{Eq.(3)}} \theta_t = (\theta_{t-1},\ \mathrm{argtop}_{m_P} F),\\
&\text{(round } T_{\mathrm{km}}{+}3)\ \big(G_k,\ H_k,\ \textstyle\sum_i s_i,\ \sum_i Y_i\big) \xrightarrow{\text{Eq.(5)}} \theta_t = (\theta_{t-1},\ W,\ \mu_s,\ \sigma_s,\ \{W_{\mathcal{L}}\}),
\end{aligned} \tag{6}$$

where an arrow is the server's computation on the aggregated uploads, labeled by the equation it evaluates, and its right side is what the server broadcasts at the end of a round.

The lemma, the proposition, and the corollary are stated informally here. Appendix A.2 lists the assumptions, states the three results formally, and proves them.

Lemma 1 (Installment identity, informal)**.** *If a client submits an upload as without-replacement installments over several upload slots, and the server aggregates after every installment has arrived, the trained model is unchanged, so a low training budget only adds upload slots.*

Lemma 1 has no counterpart in gradient training, because partial SGD runs do not sum to a full run.

Proposition 1 (Partition invariance, informal)**.** *Under full participation and exact arithmetic, and with every random choice of the pipeline drawn per image from one global seed on the clients, the deployed model at every level does not depend on how the training set is split across the K clients, and it equals the centrally trained model.*

Under exact arithmetic, the trained models are equal bit for bit. In floating point, a partition changes only the order of addition, and Section 4 verifies that the deployed accuracy does not degrade. Prior work shows the same invariance for a classifier alone on a network pretrained outside the federation (Tang et al., 2026), while FedCensus is learned end to end inside the federation. Standard federated

training converges to the centralized solution at a rate that degrades as the data distribution is more non-IID, and under a fixed step size, it stops at a gap that grows the same way (Li et al., 2020b). FedCensus reaches the centrally trained model in $T_{\mathrm{km}}+3$ rounds under any partition.

Corollary 1 (Codebook descent, informal)**.** *Under exact arithmetic, the federated batch k-means iterations of both codebooks equal the centralized iterations on the sampled patches of the full training set (Proposition 1). Therefore, no iteration increases the quantization error (the sum of squared distances from every sampled patch to its nearest codeword), and the iterations converge to a fixed point of the update in finitely many rounds (Lloyd, 1982). We run T_{km} codebook rounds.*

Every other parameter is computed in closed form in one step.

Remark 1 (Privacy). Every upload except the seeding centroids is a sum, so FedCensus composes with secure aggregation (Bonawitz et al., 2017). Under installments, the server enforces a minimum number of clients per upload slot, because a single late installment would otherwise be aggregated alone and be visible to the server. Each seeding centroid averages many patches inside one image, so no raw patch is transmitted. The Gaussian differential-privacy mechanism (Dwork & Roth, 2014) applies to the sums after each image's contribution is clipped. A formal (ϵ, δ) analysis is left to future work, because for every useful (ϵ, δ), the noise added to the codebook uploads dominates the signal. We instead measure what a single-image column reveals without secure aggregation (Section 4.4).

4 Experiments

4.1 Protocol

Setting We evaluate on four image datasets: CIFAR-10, CIFAR-100 (Krizhevsky, 2009), Tiny-ImageNet (Le & Yang, 2015), and Flowers-102 (Nilsback & Zisserman, 2008). The main comparison uses CIFAR-100, and the channel measurements use all four. Each dataset is split $75/25$ into training and test sets, and the training set is partitioned across 20 clients with full participation. The partition is Dirichlet(α) label skew with $\alpha = 0.1$ by default, where a smaller α concentrates a class on fewer clients (Hsu et al., 2019). We also use an IID partition. The homogeneous baselines train CNN-S, a residual network (He et al., 2016) with 1.25M parameters (Table 5), and the references outside the setting train CNN-L, the same network with twice the filters in every layer and 4.94M parameters. FedCensus deploys at the levels defined in Section 3.2, and we name four by their per-image FLOPs. FedCensus-T, -S, -M, and -L keep $J' = 512$, 1536, 2048, and 4096 codewords and cost 223, 385, 466, and 790 MFLOPs. FedCensus-S has the FLOPs of CNN-S, and FedCensus-L is the full model. FedCensus runs $T_{\mathrm{km}} = 15$ codebook rounds, so training takes 18 rounds. Every run uses the configuration in Appendix A.3 and the settings in Appendix A.4 unless noted otherwise.

Metrics Every client is evaluated on one test set with the model it deploys and an argmax over all C classes (Zhang et al., 2025). The deployed model is the global model for homogeneous methods and the client's own model otherwise, and we call this the deployed protocol. From client k's per-class recalls, we derive three macro metrics. A_{all} averages all C recalls, A_{pres}, the present-class accuracy, averages those of the classes with $n_k^c > 0$, and A_{miss}, the missing-class accuracy, averages the rest, so A_{miss} measures what client k can only have received through the federation. Each metric is averaged over clients, and A_{miss} only over clients missing a class, so it is undefined under IID. A_{all} is balanced accuracy, which we call macro accuracy, and micro accuracy is frequency-weighted accuracy. Table 1 reports both. Standard deviations are calculated over three seeds runs.

Policy Methods that violate a constraint listed in Section 3.1 stay in the comparison with the violated constraint marked, methods that require a pretrained backbone form a separate reference table (Table 7), and Appendix A.6 discusses scope and limitations.

4.2 Main comparison

We ask two questions. The first is how accurate FedCensus is against methods under the same constraints at every level. The second is how much accuracy is lost to the constraints against unconstrained references at matched deployment FLOPs. The last two paragraphs report the training cost.

Table 1: Main comparison on CIFAR-100 ($\alpha = 0.1$). FLOPs is the per-image inference cost averaged over clients, and Comm is upload and download per client. Bold is best per block, underline second best. A value $0.040(3)$ means 0.040 ± 0.003.

			Micro (sample-weighted)			**Macro** (balanced)		
Method	FLOPs (M)↓	Comm↓	All↑	Pres↑	Miss↑	All↑	Pres↑	Miss↑
Methods that satisfy the constraints listed in Section 3.1								
computation-heterogeneous								
HeteroFL (Diao et al., 2021)	160	4.13 GB	0.240(11)	0.248(9)	0.233(11)	0.239(11)	0.247(9)	0.232(12)
FedRolex (Alam et al., 2022)	160	4.13 GB	0.280(9)	0.282(11)	0.279(8)	0.279(9)	0.281(11)	0.278(8)
DepthFL (Kim et al., 2023)	292	5.47 GB	0.462(10)	0.484(7)	0.443(13)	0.460(10)	0.481(7)	0.441(13)
FIARSE (Wu et al., 2024)	235	4.13 GB	0.218(8)	0.241(7)	0.200(10)	0.217(8)	0.240(7)	0.199(10)
prototype								
FedProto (Tan et al., 2022)	130	0.30 GB	0.022(1)	0.042(2)	0.005(2)	0.022(1)	0.042(2)	0.005(2)
FedTGP (Zhang et al., 2024)	130	0.30 GB	0.032(3)	0.070(6)	0.001(1)	0.031(3)	0.069(5)	0.001(1)
ProtoNorm (Lee et al., 2025)	**7**	0.30 GB	0.086(3)	0.188(8)	0.000(1)	0.086(3)	0.187(8)	0.000(1)
logit								
FD (Jeong et al., 2018)	130	59 MB	0.009(1)	0.009(1)	0.010(1)	0.010(1)	0.010(1)	0.011(1)
generator								
DENSE (Zhang et al., 2022)	390	**10 MB**	0.169(12)	0.206(10)	0.138(12)	0.167(11)	0.204(9)	0.136(11)
CoBoosting (Dai et al., 2024)	390	**10 MB**	0.174(6)	0.195(13)	0.153(6)	0.172(5)	0.193(11)	0.151(6)
FedMHO (Yao et al., 2026)	390	**10 MB**	0.040(4)	0.041(3)	0.040(5)	0.040(3)	0.040(3)	0.039(4)
statistics								
Scattering-ridge	20	0.49 GB	0.324(2)	0.328(2)	0.318(3)	0.323(2)	0.327(2)	0.317(2)
FedCensus-L (ours)	790	3.99 GB	**0.625(2)**	**0.632(2)**	**0.621(2)**	**0.625(2)**	**0.631(2)**	**0.620(2)**
FedCensus-S (ours)	385	3.99 GB	0.620(1)	0.627(1)	0.616(1)	0.619(1)	0.626(1)	0.615(1)
Methods that violate the deployment-budget requirement (weight averaging on CNN-S)								
FedAvg (McMahan et al., 2017)	**390**	**10.0 GB**	0.627(3)	0.645(3)	0.609(4)	0.626(3)	0.643(2)	0.608(4)
FedProx (Li et al., 2020a)	**390**	**10.0 GB**	**0.632(6)**	**0.648(7)**	**0.616(5)**	**0.630(6)**	**0.646(7)**	**0.615(5)**
SCAFFOLD (Karimireddy et al., 2020)	**390**	20.0 GB	0.608(14)	0.617(14)	0.599(15)	0.607(13)	0.616(13)	0.598(14)
MOON (Li et al., 2021)	**390**	**10.0 GB**	0.606(4)	0.624(5)	0.589(2)	0.605(4)	0.622(6)	0.587(3)
FedLESAM (Fan et al., 2024)	**390**	**10.0 GB**	0.625(10)	0.640(10)	0.612(9)	0.623(9)	0.638(9)	0.611(8)
FedHPro (Wang et al., 2026)	**390**	15.4 GB	0.298(22)	0.495(31)	0.132(12)	0.296(22)	0.494(31)	0.131(12)
References outside the setting								
Local-only	**390**	**0 MB**	0.135(4)	0.295(12)	0.000(1)	0.135(4)	0.294(12)	0.000(1)
Central, CNN-S	**390**	N/A	0.711(2)	N/A	N/A	0.709(2)	N/A	N/A
Central, CNN-L	1551	N/A	**0.734(4)**	N/A	N/A	**0.733(4)**	N/A	N/A
FedAvg, CNN-L (McMahan et al., 2017)	1551	39.5 GB	0.648(5)	**0.666(2)**	**0.631(8)**	0.647(5)	**0.665(2)**	**0.630(8)**

Constraint-matched comparison FedCensus leads every method that satisfies the constraints. Table 1 reports a macro accuracy of 0.625 ± 0.002 for FedCensus, against 0.460 for the best computation-heterogeneous method (DepthFL, Kim et al., 2023), 0.172 for the best generator method (CoBoosting, Dai et al., 2024), 0.086 for the best prototype method (ProtoNorm, Lee et al., 2025), and 0.323 for Scattering-ridge, the statistics method with fixed features. FedMHO trains a generative model on client data, which violates the constraint on generative models in Section 3.1, and it stays in the table for comparison. Among the methods that satisfy the constraints, the lead holds at every deployment level. FedCensus-L, -M, -S, and -T reach 0.625, 0.620, 0.619, and 0.614 macro accuracy, above every tier of DepthFL, HeteroFL, and FedRolex (Figure 2(d) and Table 8). The homogeneous weight-averaging rows, which violate the deployment-budget requirement, reach at most 0.631 after 1000 rounds, 0.006 above FedCensus-L. FedCensus classifies present and missing classes correctly at almost the same rate, 0.631 and 0.620. One thing to note is that the prototype and personalized methods report much higher numbers in their own papers, because they evaluate each client only on its own label distribution. Under that protocol, our replication of ProtoNorm reaches 0.289, and under the deployed protocol defined in Section 4.1, the same replication reaches 0.086. Every row is trained with the same augmentation.

FLOPs-matched comparison We quantify the performance loss caused by the constraints listed in Section 3.1 by comparing with methods that violate them at matched deployment FLOPs. The best homogeneous method on CNN-S, FedProx (Li et al., 2020a), reaches a macro accuracy of 0.631 after 1000 rounds with augmentation, and FedCensus-S, at the same FLOPs, reaches 0.619, so the accuracy lost to the constraints is 0.012. FedAvg on CNN-L, at twice the FLOPs of FedCensus-L, reaches 0.647 against 0.625. Central (SGD on the full training set) reaches 0.709 on CNN-S and 0.733 on CNN-L, while converged FedAvg on CNN-S reaches 0.626, so the constraints account for little of the gap to centralized training, and weight averaging accounts for most of it. Local-only training reaches 0.134 and cannot classify missing classes, as expected.

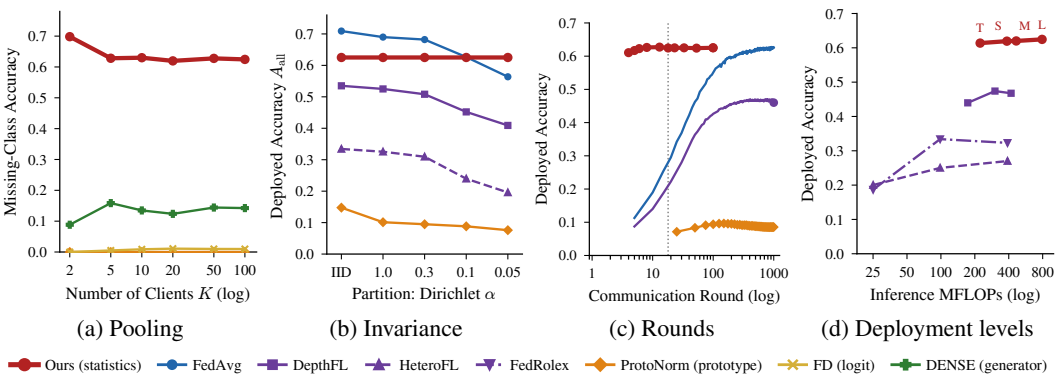

Figure 2: (a) Missing-class accuracy against the number of clients. (b) to (d) Deployed macro accuracy across partitions, against rounds, and against inference FLOPs, where T, S, M, and L mark the deployment levels of FedCensus.

Rounds FedCensus reaches its final accuracy in 18 rounds, and running the pipeline to 100 rounds does not change the accuracy (Figure 3). In Figure 2(c), the pipeline with one codebook round, which ends at round 4, reaches 0.611, the pipeline that ends at round 8 reaches 0.626, and the default pipeline ends at round 18 with 0.625. FedAvg on CNN-S reaches 0.60 after about 200 rounds and 0.625 after about 500.

Communication Table 10 lists the total communication per client, upload and download, over the training run. FedAvg on CNN-S sends 10.0 GB over its 1000 rounds, and FedCensus sends 3.99 GB, 40% of that. FedAvg on CNN-L sends 39.5 GB, ten times that of FedCensus. The statistics round accounts for 95% of the communication because G_k has $D(D{+}1)/2$ entries. The factor scheme in Table 10 divides that communication by three but is not compatible with secure aggregation.

4.3 Channel measurements

This section tests, one at a time, two possible causes of the failure of the other channels (missing classes and too little data per client). Under the IID partition, every client has every class, so the test is whether a channel can train a usable global model at all. Splitting the same data across 2 to 100 clients gives each client less data while the total stays fixed, so the test is whether the missing-class accuracy of a channel changes with the number of clients.

Channels Table 11 reports accuracy under the IID partition. With every class present at every client, the prototype channel trains a global model with a macro accuracy of at most 0.149, the logit channel 0.010, the generator channel 0.313, and the statistics channel 0.625, against 0.709 for weight averaging. Under Dirichlet $\alpha = 0.1$, the same trends hold for missing classes. The prototype channel classifies none of them correctly on any of the four datasets, the logit channel classifies at most 0.08 of them correctly, and FedCensus classifies 0.38 to 0.84 of them correctly, close to its present-class accuracy (Figure 4). The IID rows show that the channels are weak even with no class missing, and the Dirichlet results show that they transfer no knowledge of the missing classes.

Pooling Figure 2(a) plots the result of splitting the same data across 2 to 100 clients. The prototype channel classifies no missing class correctly at any K, and its present-class accuracy falls from 0.341 at $K = 2$ to 0.180 at $K = 100$. The logit channel classifies about 0.01 of the missing classes correctly at every K. The generator channel classifies 0.09 to 0.16 of them correctly at every K. FedCensus classifies 0.62 to 0.63 of the missing classes correctly at every $K \geq 5$, and 0.70 at $K = 2$, where each client misses about twelve classes. Giving a client more data does not help the prototype channel, and giving it less does not hurt FedCensus, so the difference is in the channel, not the data.

4.4 Properties on trained models

This section lists the ablations and verifies the properties stated in Section 3.3 on trained models.

Table 2: Configuration ablations (CIFAR-100, $\alpha = 0.1$). The default configuration keeps all three statistics, uses flip-and-crop augmentation, $\lambda = 100$, and $J = 4096$. Bold marks the default.

Ablation	Setting	A_{all}	Ablation	Setting	A_{all}
Statistics kept	z_A	0.5510	Ridge λ	10	0.6145
	z_A, z_M	0.5768		30	0.6201
	z_P, z_M	0.6101		100 (default)	**0.6251**
	z_A, z_P	0.6164		300	0.6253
	z_A, z_P, z_M (default)	**0.6251**		1000	0.6185
Augmentation	none	0.5967	Codebook size	$J = 4096$ (default)	**0.6251**
	flip-and-crop $\times 3$ (default)	**0.6251**		$J = 8192$	0.6263

Ablations Table 2 adds one kind of statistic at a time. With the activation sums alone, the deployed macro accuracy is 0.551. The accuracy reaches 0.577 with the spatial moments added and 0.616 with the pair products added, against 0.625 with all three, so the pair products contribute most, and each statistic adds accuracy. Training without augmentation loses 0.028 accuracy. The accuracy is not sensitive to λ, with 100 and 300 within 0.001 of each other, and both ends of the sweep 0.007 to 0.011 lower. Doubling the codebook to $J = 8192$ adds 0.001, so the codebook saturates at 4096.

Partition invariance Proposition 1 states that the deployed model does not depend on the partition. Table 15 confirms this on the trained models, and Table 16 lists the present-class and missing-class accuracies of every channel under every partition. Under the five partitions from IID to $\alpha = 0.05$, and with 100 clients instead of 20, FedCensus reaches the same deployed accuracy of 0.6242 and the same $15{,}000$ test predictions. Over the same range, FedAvg falls from 0.709 to 0.564, DepthFL from 0.535 to 0.409, and HeteroFL from 0.334 to 0.196 (Figure 2(b)). Summing the client statistics in a different order changes the accuracy by at most 6×10^{-4} in single precision and the classifier weights by 10^{-10} in double precision, so the partition's only effect is rounding.

Installments Lemma 1 states that installments reproduce the full upload. The experiment in Table 17 splits the statistics upload, which accounts for 95% of the communication, into 2, 4, and 8 installments, so that a client with half, a quarter, or an eighth of the training budget needs 2, 4, or 8 slots. The lemma covers the other rounds. The deployed accuracy stays at most 0.0008 below that of the full upload, and the weights change by at most 1.6×10^{-3}, within single-precision rounding.

Privacy Remark 1 states that every upload except the seeding centroids is a sum. Without secure aggregation, the server sees every H_k, whose column c sums the client's class-c encodings, so a class with one image gives that image's encoding. Appendix A.14 inverts such columns in closed form and tests whether a classifier trained on real images recognizes each reconstruction (Figure 5). On 200 targets, the classifier's top-1 accuracy is 0.417 on real images, 0.025 on closed-form reconstructions from single-image columns, and 0.027 on those from secure-aggregated columns, against a chance rate of 0.01. A refinement matching all three statistics raises the recognizability to 0.205, half the accuracy on originals, at a lower PSNR. Without secure aggregation, a single-image column leaks its class, and secure aggregation keeps the recognizability at the chance rate (Table 18).

5 Conclusion

We presented FedCensus, a federated model with a fine and a coarse codebook learned inside the federation, three additive feature statistics, and a closed-form classifier, so every parameter is computed from additive client statistics without gradients or pretraining. Because additive statistics do not depend on how the data is split, the trained model is the same under every partition and any number of clients. Because the sums cover every class, each client is nearly as accurate on missing classes as on present classes. Accuracy is not bounded by the lowest-computation client. On four datasets, FedCensus exceeds every computation-heterogeneous, prototype, and logit method. On CIFAR-100, FedCensus stays 0.012 below the best homogeneous method at matched FLOPs, in 18 rounds with 40% of FedAvg's communication. Two limitations remain. FedCensus does not exceed weight averaging at the same FLOPs, and every client must participate in each round.

Reproducibility statement

The code of FedCensus, our port of the baselines, the cached partitions, and the scripts that produce every table and figure will be publicly released upon publication. One global seed determines every random choice (Proposition 1), and the trained model is the same under every partition (Table 15), so a run can be reproduced from the seed alone. Appendix A.1 lists the training procedure as Algorithm 1. Appendix A.3 lists the datasets, the split, the partition, and the hyperparameters. Appendix A.4 lists the code and settings of every baseline, and Appendix A.9 lists the communication of every round. The main comparison, the deployment levels, the attack, and the rounds curve are means over three runs. The ablations, the partition and installment checks, the pretrained references, and the other-dataset baselines are single runs.

AI Use Statement

We did not use generative AI to generate synthetic datasets, develop theoretical models or conceptual frameworks, formulate mathematical claims, assist with proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, assist with translation, clean or reformat datasets, support qualitative or thematic data analysis, or interpret results. We used Claude Fable 5 solely to proofread and polish the original text for grammar, clarity, and readability. All AI-assisted edits were reviewed by the authors. The authors take responsibility for the final content of this work.

References

Durmus Alp Emre Acar, Yue Zhao, Ramon Matas, Matthew Mattina, Paul Whatmough, and Venkatesh Saligrama. Federated learning based on dynamic regularization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=B7v4QMR6Z9w.

Maruan Al-Shedivat, Jennifer Gillenwater, Eric Xing, and Afshin Rostamizadeh. Federated learning via posterior averaging: A new perspective and practical algorithms. In *International Conference on Learning Representations (ICLR)*, 2021.

Samiul Alam, Luyang Liu, Ming Yan, and Mi Zhang. Fedrolex: Model-heterogeneous federated learning with rolling sub-model extraction. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=OtxyysUdBE.

Mathieu Andreux, Tomás Angles, Georgios Exarchakis, Roberto Leonarduzzi, Gaspar Rochette, Louis Thiry, John Zarka, Stéphane Mallat, Joakim Andén, Eugene Belilovsky, et al. Kymatio: Scattering transforms in python. *Journal of Machine Learning Research*, 21(60):1–6, 2020.

Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *International conference on database theory*, pp. 217–235. Springer, 1999.

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, 2017.

Cheng-Tao Chu, Sang Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Kunle Olukotun, and Andrew Ng. Map-reduce for machine learning on multicore. *Advances in neural information processing systems*, 19, 2006.

Adam Coates and Andrew Y Ng. Learning feature representations with k-means. In *Neural Networks: Tricks of the Trade: Second Edition*, pp. 561–580. Springer, 2012.

Rong Dai, Yonggang Zhang, Ang Li, Tongliang Liu, Xun Yang, and Bo Han. Enhancing one-shot federated learning through data and ensemble co-boosting. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=tm8s3696Ox.

Don Kurian Dennis, Tian Li, and Virginia Smith. Heterogeneity for the win: One-shot federated clustering. In *International conference on machine learning*, pp. 2611–2620. PMLR, 2021.

Inderjit S Dhillon and Dharmendra S Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Large-scale parallel data mining*, volume 1759, pp. 245–260. Springer, 2000.

Enmao Diao, Jie Ding, and Vahid Tarokh. HeteroFL: Computation and communication efficient federated learning for heterogeneous clients. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=TNkPBBYFkXg.

Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–487, 2014.

Ziqing Fan, Shengchao Hu, Jiangchao Yao, Gang Niu, Ya Zhang, Masashi Sugiyama, and Yanfeng Wang. Locally estimated global perturbations are better than local perturbations for federated sharpness-aware minimization. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Eros Fanì, Raffaello Camoriano, Barbara Caputo, and Marco Ciccone. Accelerating heterogeneous federated learning with closed-form classifiers. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Zenghao Guan, Yucan Zhou, and Xiaoyan Gu. Capture global feature statistics for one-shot federated learning. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, volume 39 of *AAAI'25/IAAI'25/EAAI'25*, pp. 16942–16950. AAAI Press, 2025. ISBN 978-1-57735-897-8. doi: 10.1609/aaai.v39i16.33862. URL https://doi.org/10.1609/aaai.v39i16.33862.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Run He, Kai Tong, Di Fang, Han Sun, Ziqian Zeng, Haoran Li, Tianyi Chen, and Huiping Zhuang. AFL: A single-round analytic approach for federated learning with pre-trained models. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4988–4998. IEEE, 2025.

Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep models under the gan: Information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pp. 603–618, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349468. doi: 10.1145/3133956.3134012. URL https://doi.org/10.1145/3133956.3134012.

Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *CoRR*, abs/1909.06335, 2019. URL http://arxiv.org/abs/1909.06335.

Eunjeong Jeong, Seungeun Oh, Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data. In *Proceedings of the NeurIPS Workshop on Machine Learning on the Phone and other Consumer Devices*, 2018. URL https://arxiv.org/abs/1811.11479.

Divyansh Jhunjhunwala, Shiqiang Wang, and Gauri Joshi. FedFisher: Leveraging Fisher Information for One-Shot Federated Learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 1612–1620. PMLR, 2024.

Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. SCAFFOLD: Stochastic controlled averaging for federated learning. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5132–5143. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/karimireddy20a.html.

Minjae Kim, Sangyoon Yu, Suhyun Kim, and Soo-Mook Moon. DepthFL: Depthwise federated learning for heterogeneous clients. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=pf8RIZTMU58.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Ya Le and Xuan Yang. Tiny ImageNet visual recognition challenge. Cs231n course report, Stanford University, 2015.

Gyuejeong Lee, Jihwan Shin, and Daeyoung Choi. Heterogeneous federated learning with prototype alignment and upscaling, 2025. URL https://arxiv.org/abs/2507.04310.

Daliang Li and Junpu Wang. FedMD: Heterogenous Federated Learning via Model Distillation. In *Proceedings of the NeurIPS Workshop on Federated Learning for Data Privacy and Confidentiality*, 2019. URL https://arxiv.org/abs/1910.03581.

Qinbin Li, Bingsheng He, and Dawn Song. Model-contrastive federated learning. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10708–10717. IEEE, 2021.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, 2:429–450, 2020a.

Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. In *International Conference on Learning Representations*, 2020b. URL https://openreview.net/forum?id=HJxNAnVtDS.

Tao Lin, Lingjing Kong, Sebastian U. Stich, and Martin Jaggi. Ensemble distillation for robust model fusion in federated learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28 (2):129–137, 1982. doi: 10.1109/TIT.1982.1056489.

Mi Luo, Fei Chen, Dapeng Hu, Yifan Zhang, Jian Liang, and Jiashi Feng. No fear of heterogeneity: Classifier calibration for federated learning with Non-IID data. *Advances in Neural Information Processing Systems*, 34:5972–5984, 2021.

Stéphane Mallat. Group invariant scattering. *Communications on Pure and Applied Mathematics*, 65(10):1331–1398, 2012.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. Pmlr, 2017.

Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pp. 722–729, 2008. doi: 10.1109/ICVGIP.2008.47.

Edouard Oyallon, Sergey Zagoruyko, Gabriel Huang, Nikos Komodakis, Simon Lacoste-Julien, Matthew Blaschko, and Eugene Belilovsky. Scattering networks for hybrid representation learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(9):2208–2221, 2019. doi: 10.1109/TPAMI.2018.2855738.

Reese Pathak and Martin J. Wainwright. Fedsplit: an algorithmic framework for fast federated optimization. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, Dec 2015. ISSN 1573-1405. doi: 10.1007/s11263-015-0816-y. URL https://doi.org/10.1007/s11263-015-0816-y.

Fatema Siddika, Md Anwar Hossen, Wensheng Zhang, Anuj Sharma, J. Pablo Muñoz, and Ali Jannesari. Dual-distilled heterogeneous federated learning with adaptive margins for trainable global prototypes. In *2026 IEEE 26th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pp. 412–421, 2026. doi: 10.1109/CCGrid68966.2026.00050.

Yue Tan, Guodong Long, Lu Liu, Tianyi Zhou, Qinghua Lu, Jing Jiang, and Chengqi Zhang. FedProto: Federated prototype learning across heterogeneous clients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pp. 8432–8440, 2022.

Jianheng Tang, Zhirui Yang, Jingchao Wang, Kejia Fan, Jinfeng Xu, Huiping Zhuang, Anfeng Liu, Houbing Herbert Song, Leye Wang, and Yunhuai Liu. FedHiP: Heterogeneity-invariant personalized federated learning through closed-form solutions. *CoRR*, abs/2508.04470, 2025. doi: 10.48550/ARXIV.2508.04470. URL https://doi.org/10.48550/arXiv.2508.04470.

Jianheng Tang, Yajiang Huang, Kejia Fan, Feijiang Han, Jiaxu Li, Jinfeng Xu, Run He, Anfeng Liu, Houbing Herbert Song, Huiping Zhuang, and Yunhuai Liu. DeepAFL: Deep analytic federated learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ve3EzAvMGe.

Louis Thiry, Michael Arbel, Eugene Belilovsky, and Edouard Oyallon. The unreasonable effectiveness of patches in deep convolutional kernels methods. In *ICLR*, 2021. URL https://openreview.net/forum?id=aYuZO9DIdnn.

Huan Wang, Jun Shen, Haoran Li, Zhenyu Yang, Jun Yan, Ousman Manjang, Yanlong Zhai, Di Wu, and Guansong Pang. FedHPro: Federated hyper-prototype learning via gradient matching. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=35e16MmSL0.

Feijie Wu, Xingchen Wang, Yaqing Wang, Tianci Liu, Lu Su, and Jing Gao. Fiarse: model-heterogeneous federated learning via importance-aware submodel extraction. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Dezhong Yao, Tongtong Liu, Yuexin Shi, and Zhiqiang Xu. FedMHO: Heterogeneous one-shot federated learning towards resource-constrained clients. In *Proceedings of the ACM Web Conference 2026*, WWW '26, pp. 5686–5697, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400723070. doi: 10.1145/3774904.3792743. URL https://doi.org/10.1145/3774904.3792743.

Hongxu Yin, Pavlo Molchanov, Jose M Alvarez, Zhizhong Li, Arun Mallya, Derek Hoiem, Niraj K Jha, and Jan Kautz. Dreaming to Distill: Data-free Knowledge Transfer via DeepInversion. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8712–8721. IEEE, 2020.

Yaodong Yu, Alexander Wei, Sai Praneeth Karimireddy, Yi Ma, and Michael Jordan. TCT: Convexifying federated learning using bootstrapped neural tangent kernels. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=jzd2bE5MxW.

Jianqing Zhang, Yang Liu, Yang Hua, and Jian Cao. Fedtgp: Trainable global prototypes with adaptive-margin-enhanced contrastive learning for data and model heterogeneity in federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 16768–16776, 2024.

Jianqing Zhang, Xinghao Wu, Yanbing Zhou, Xiaoting Sun, Qiqi Cai, Yang Liu, Yang Hua, Zhenzhe Zheng, Jian Cao, and Qiang Yang. Htfllib: A comprehensive heterogeneous federated learning

library and benchmark. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25, pp. 5900–5911, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400714542. doi: 10.1145/3711896.3737379. URL https://doi.org/10.1145/3711896.3737379.

Jie Zhang, Chen Chen, Bo Li, Lingjuan Lyu, Shuang Wu, Shouhong Ding, Chunhua Shen, and Chao Wu. Dense: data-free one-shot federated learning. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Bo Zhao and Hakan Bilen. Dataset condensation with differentiable Siamese augmentation. In *International Conference on Machine Learning*, pp. 12674–12685. PMLR, 2021.

A Appendix

A.1 Training procedure

Algorithm 1 lists the pipeline as one server procedure and four client procedures. Every client procedure returns sums over the client's own data, marked *[+]*, and PatchStats also returns the per-image seeding centroids. Every round loops until all installments have arrived. This section describes, for each round, the statistics a client uploads and the closed form the server evaluates on the aggregated sums.

Round 1, whitening and seeding Client k uses a fixed subset of its images for whitening and the codebook, every third image by the global index, which is $12{,}000$ images over all clients, and we call this subset the whitening subset. From each of these images, client k draws 40 patches, with positions chosen per image by the global seed, and $\mathcal{B}_k$ is the set of all patches drawn by client k. The client uploads the patch count, the patch sum, and the outer-product sum, together with the seeding centroids $b(x)$ of each image. The seeding centroids of an image are 2 centroids computed on the image's own 40 patches by four iterations of nearest-centroid assignment and mean update, and a centroid is kept only when at least 8 patches are assigned to it. The server calculates the mean and covariance of all patches and the whitening map of Eq. 1,

$$N_p = \sum_k \sum_{\tilde{p}\in\mathcal{B}_k} 1, \qquad \mu = \frac{1}{N_p}\sum_k \sum_{\tilde{p}\in\mathcal{B}_k} \tilde{p}, \qquad \Sigma = \frac{1}{N_p}\sum_k \sum_{\tilde{p}\in\mathcal{B}_k} \tilde{p}\,\tilde{p}^\top - \mu\mu^\top,$$

sorts the union of the uploaded centroids lexicographically, whitens them as $\Omega^\top(b-\mu)$ with the map of Eq. 1, and draws J of them as the fine codebook and J_P as the coarse codebook with the global seed. The server broadcasts μ, Ω, and both codebooks.

Rounds 2 to $T_{\mathrm{km}}{+}1$, codebooks Client k whitens its patch sample $\mathcal{B}_k$ with Eq. 1, assigns every whitened patch u to its nearest fine codeword and, separately, to its nearest coarse codeword by Eq. 2, and uploads, for every codeword of both codebooks, the sum and the count of the patches assigned to it,

$$S_j^{(k)} = \sum_{\tilde{p}\in\mathcal{B}_k:\ a(u)=j} u, \qquad n_j^{(k)} = \sum_{\tilde{p}\in\mathcal{B}_k:\ a(u)=j} 1, \qquad u = \Omega^\top(\tilde{p}-\mu).$$

The server sets every codeword with a positive count to the aggregated mean, Eq. 2, keeps a codeword with no assigned patches unchanged, and broadcasts both codebooks. After the last codebook round, the server ranks the fine codewords by their aggregated counts, and a deployment level keeps the first J' codewords of this count ranking (Section 3.2).

Round $T_{\mathrm{km}}{+}2$, screening For every patch p of a sample, with whitened form u_p, the encoder computes the threshold and the activations against the fine and the coarse codewords, as in Section 3.2,

$$\bar{d}_p = \frac{1}{J_P}\sum_{l=1}^{J_P} \|u_p - e_l^P\|_2,$$
$$\tau_{pj} = \max\big(0, \bar{d}_p - \|u_p - e_j\|_2\big), \qquad \tau_{pl}^P = \max\big(0, \bar{d}_p - \|u_p - e_l^P\|_2\big).$$

Client k computes the pair products $z_P^{(i)}$ of every sample i over all $J_P(J_P{+}1)/2$ candidate pairs and uploads, for every class c, the sum, the sum of squares, and the count,

$$\sum_{i:\, y_i=c} z_P^{(i)}, \qquad \sum_{i:\, y_i=c} \big(z_P^{(i)}\big)^{\odot 2}, \qquad n_k^c.$$

The server aggregates these three uploads over clients into n_c, the class means $\bar{z}_c$, and the global mean $\bar{z}$, calculates the between-class and within-class sums of squares of every pair,

$$\mathrm{SS_b} = \sum_c n_c\,(\bar{z}_c - \bar{z})^{\odot 2}, \qquad \mathrm{SS_w} = \sum_c \Big(\sum_k \sum_{i:\, y_i=c} \big(z_P^{(i)}\big)^{\odot 2} - n_c\,\bar{z}_c^{\odot 2} \Big),$$

evaluates F by Eq. 3, and broadcasts the index set $\mathcal{I}$ of the m_P pairs with the largest F. The screening statistics exclude a fixed tenth of the samples, chosen by the global seed, and we call this tenth the screening hold-out.

Round $T_{\mathrm{km}}{+}3$, statistics Client k encodes every training image and its augmented copies into $s_i = [\, z_A^{(i)};\, z_P^{(i)}[\mathcal{I}];\, z_M^{(i)} \,] \in \mathbb{R}^D$, with z_A the activation sums and z_M the spatial moments of Eq. 4, and uploads

$$G_k = \sum_i s_i s_i^\top, \qquad H_k = \sum_i s_i Y_i^\top, \qquad \sum_i s_i, \qquad \sum_i Y_i.$$

The diagonal of G_k equals $\sum_i s_i^{\odot 2}$, so the sum of squares needs no separate upload. With $\mathbf{n} = \sum_k \sum_i Y_i$ the aggregated class counts and $N = \mathbf{1}^\top \mathbf{n}$ the total number of encoded samples, the server standardizes the aggregated sums and solves the ridge regression of Eq. 5 with the standardized aggregates $\tilde{G}$ and $\tilde{H}$ in place of $\sum_k G_k$ and $\sum_k H_k$,

$$\mu_s = \frac{1}{N} \sum_k \sum_i s_i, \qquad \sigma_s^{\odot 2} = \frac{1}{N} \sum_k \sum_i s_i^{\odot 2} - \mu_s^{\odot 2}, \qquad D_\sigma = \mathrm{diag}(\sigma_s)^{-1},$$

$$\tilde{G} = D_\sigma \Big(\sum_k G_k - N \mu_s \mu_s^\top \Big) D_\sigma, \qquad \tilde{H} = D_\sigma \Big(\sum_k H_k - \mu_s \mathbf{n}^\top \Big), \qquad W = \big(\lambda I + \tilde{G}\big)^{-1} \tilde{H}.$$

The standardized Gram matrix and class sums follow from the uploaded sums because

$$\sum_i D_\sigma (s_i - \mu_s)(s_i - \mu_s)^\top D_\sigma = D_\sigma \Big(\sum_i s_i s_i^\top - N \mu_s \mu_s^\top \Big) D_\sigma,$$

$$\sum_i D_\sigma (s_i - \mu_s)\, Y_i^\top = D_\sigma \Big(\sum_i s_i Y_i^\top - \mu_s \mathbf{n}^\top \Big),$$

where i runs over the samples of all clients and $\sum_i s_i = N \mu_s$. The server broadcasts W, μ_s, σ_s, the count ranking of the fine codewords, and $W_{\mathcal{L}}$ for every deployment level.

Levels and deployment A deployment level keeps the top J' fine codewords in the count ranking of the codebook rounds, so its feature index set $\mathcal{L}$ contains the $4J'$ matching activation sums and all pair and moment entries, and the server solves the classifier of the level from the principal submatrix of the same aggregated sums,

$$W_{\mathcal{L}} = \big(\lambda I + \tilde{G}_{\mathcal{L}\mathcal{L}}\big)^{-1} \tilde{H}_{\mathcal{L}}.$$

A client that deploys level $\mathcal{L}$ encodes an image into $s_{\mathcal{L}}$, standardizes it with the matching entries of μ_s and σ_s, and predicts $\arg\max_c \big(W_{\mathcal{L}}^\top D_{\sigma,\mathcal{L}} (s_{\mathcal{L}} - \mu_{s,\mathcal{L}})\big)_c$.

Algorithm 1 FedCensus training. Clients are indexed by $k \in [K]$, classes by $c \in [C]$, and a client's own samples by i, with label y_i and one-hot label Y_i; the superscript (i) marks sample i's value, and $(\cdot)^{\odot 2}$ denotes the elementwise square. Uploads marked *[+]* are additive statistics over the client's own data.

procedure SERVER
  whitening and seeding, one round
  repeat
    every client k uploads its next installment of PATCHSTATS(k)
  until every installment has arrived
  $\mu, \Sigma \leftarrow$ aggregated patch mean and covariance; Ω by Eq. 1
  sort the uploaded centroids lexicographically and initialize both codebooks with the global seed
  broadcast μ, Ω, and both codebooks
  codebook rounds
  for $t = 2, \ldots, T_{\mathrm{km}}+1$ **do**
    repeat
      every client k uploads its next installment of CODEWORDSUMS(k)
    until every installment has arrived
    $e_j \leftarrow \sum_k S_j^{(k)} / \sum_k n_j^{(k)}$ for every codeword with $\sum_k n_j^{(k)} > 0$ of both codebooks (Eq. 2)
    broadcast both codebooks
  end for
  screening round
  repeat
    every client k uploads its next installment of PAIRSTATS(k)
  until every installment has arrived
  keep the m_P candidate pairs with the largest F (Eq. 3); broadcast the kept pairs
  statistics round
  repeat
    every client k uploads its next installment of RIDGESTATS(k)
  until every installment has arrived
  standardize the features from the aggregated sums; W by Eq. 5
  broadcast W, μ_s, σ_s, and $W_{\mathcal{L}}$ for every level
end procedure

procedure PATCHSTATS(k) ▷ on client k, over its own data
  return patch count, patch sum, outer-product sum *[+]*, and the per-image seeding centroids
end procedure
procedure CODEWORDSUMS(k)
  assign every whitened patch u to its nearest codeword $a(u)$
  return $S_j^{(k)} = \sum_{u:a(u)=j} u$ and $n_j^{(k)}$ for every codeword of both codebooks *[+]*
end procedure
procedure PAIRSTATS(k)
  return $\sum_{y_i=c} z_P^{(i)}$, $\sum_{y_i=c} (z_P^{(i)})^{\odot 2}$, and n_k^c for every class c *[+]*
end procedure
procedure RIDGESTATS(k)
  encode every sample into s_i; **return** $G_k = \sum_i s_i s_i^\top$, $H_k = \sum_i s_i Y_i^\top$, $\sum_i s_i$, and $\sum_i Y_i$ *[+]*
end procedure

A.2 Formal statements and proofs

This section lists the assumptions, states the lemma, the proposition, and the corollary in Section 3.3 formally, and proves them.

Setting Let $\mathcal{D}$ be the training set, in which every image keeps its index under every partition. Let $\pi = (\mathcal{D}_1, \ldots, \mathcal{D}_K)$ be K pairwise disjoint subsets of $\mathcal{D}$, possibly empty, whose union is $\mathcal{D}$. We call π a partition of $\mathcal{D}$ across K clients. An empty client uploads the zero sum and an empty centroid list. Let ξ be the global seed. The one-client run is the run with $K = 1$ and $\pi = (\mathcal{D})$. The centrally trained model is the model of the one-client run, that is, the same pipeline on the full training set. A round ends with one broadcast, and the uploads of a round arrive in one or more upload slots. Without installments, a round has one upload slot. For round t write θ_{t-1} for the broadcast at the end of round $t-1$, with $\theta_0 = \xi$. Every upload of Eq. 6 is a sum of per-image terms $\rho_t(x; \theta_{t-1})$. The upload of round 1 also contains the multiset union of the per-image seeding centroids $b(x)$, which keeps equal vectors from different images as separate entries. In rounds 1 to $T_{\mathrm{km}}+1$, the term of an image is the sum of the terms of its sampled patches. The server adds the client uploads into $A_t = \sum_{k=1}^{K} \sum_{x \in \mathcal{D}_k} \rho_t(x; \theta_{t-1})$ and, in round 1, joins the client centroid lists into $U_1 = \biguplus_{k=1}^{K} \biguplus_{x \in \mathcal{D}_k} b(x)$. The server then computes $\theta_t = \Phi_t(A_t, U_1, \theta_{t-1}; \xi)$, where Φ_t is the server procedure of round t in Algorithm 1. The procedure evaluates the closed form of the round, Eq. 1, 2, 3, or 5. In round 1, the procedure also sorts the centroids and draws the initial codewords. In the codebook rounds, the procedure keeps a codeword with no assigned patches at its value in θ_{t-1}. In the screening and statistics rounds, the procedure copies θ_{t-1} into θ_t.

Assumptions

(A1) *Full participation.* Every client uploads every requested statistic, in one upload slot or as without-replacement installments over consecutive upload slots, and no client drops out. The server aggregates a round only after every client's upload is complete, and broadcasts only after aggregating.

(A2) *Exact arithmetic.* Every arithmetic operation of the pipeline is exact, on the clients and on the server.

(A3) *Per-image randomness.* The random choices made on a client are the membership of an image in the whitening subset, its patch positions, its augmented copies, its seeding centroids, and its membership in the screening hold-out. Each is a deterministic function of ξ and of the image's index in the training set, and not of the client that owns the image. The server's draw of the initial codewords is a deterministic function of ξ and the sorted list of U_1.

(A4) *Deterministic server.* The server procedure Φ_t of every round is one fixed procedure, including the eigendecomposition of Eq. 1 and the treatment of a pair with zero within-class scatter and of a feature with zero variance, so equal inputs give equal outputs. The ridge weight satisfies $\lambda > 0$. Ties in the count ranking, in the F ranking, and in the nearest-codeword assignment are broken in favor of the lowest codeword or pair index, never a client index.

Per-image form of the uploads The proofs use the fact that every upload of Algorithm 1 is a sum of per-image terms as defined in the setting, and we write out this form for the four round types. Write $h(x) \in \{0, 1\}$ for the membership of image x in the whitening subset, $\mathcal{B}(x)$ for the set of its 40 sampled patches, which is empty when $h(x) = 0$, $\mathcal{X}(x)$ for the image together with its augmented copies, Y_x for its one-hot label, and $\eta(x) \in \{0, 1\}$ for its membership in the screening hold-out. Round 1 uploads $\rho_1(x; \xi) = \big(|\mathcal{B}(x)|,\ \sum_{\tilde{p} \in \mathcal{B}(x)} \tilde{p},\ \sum_{\tilde{p} \in \mathcal{B}(x)} \tilde{p}\tilde{p}^\top\big)$ together with $b(x)$, which the client computes on $\mathcal{B}(x)$. Rounds 2 to $T_{\mathrm{km}}+1$ upload $\rho_t(x; \theta_{t-1}) = \big(S_j(x),\, n_j(x)\big)_j$ for both codebooks, with $S_j(x) = \sum_{\tilde{p} \in \mathcal{B}(x):\, a(u)=j} u$ and $n_j(x) = |\{\tilde{p} \in \mathcal{B}(x) : a(u) = j\}|$, where $u = \Omega^\top(\tilde{p} - \mu)$ and $a(u)$ depend on θ_{t-1} only. Round $T_{\mathrm{km}}+2$ uploads $\rho_t(x; \theta_{t-1}) = (1 - \eta(x)) \sum_{x' \in \mathcal{X}(x)} \big(z_P^{(x')} Y_x^\top,\ (z_P^{(x')})^{\odot 2} Y_x^\top,\ Y_x\big)$, whose class-$c$ column is the class-c part of the upload. Round $T_{\mathrm{km}}+3$ uploads $\rho_t(x; \theta_{t-1}) = \sum_{x' \in \mathcal{X}(x)} \big(s_{x'} s_{x'}^\top,\ s_{x'} Y_x^\top,\ s_{x'},\ Y_x\big)$. The choices h,

$\mathcal{B}$, $\mathcal{X}$, η, and the seed of b are the random choices of (A3), and every other quantity in the four terms is a deterministic function of x and θ_{t-1}.

Lemma (Lemma 1, formal). Assume (A1) to (A4). Fix a round t and write $\mathcal{Q}_k$ for the index set of client k's upload of round t, which is its set of sampled patches in rounds 1 to $T_{\mathrm{km}}+1$ and its set of images in rounds $T_{\mathrm{km}}+2$ and $T_{\mathrm{km}}+3$, because the terms of those two rounds need the whole feature vector of an image. For a patch index q, $\rho_t(q;\theta_{t-1})$ is the patch's term in the sum of its image. Let client k submit the upload over a disjoint split $\mathcal{Q}_k = \bigsqcup_{\ell=1}^{R_k} \mathcal{Q}_k^{\ell}$, that is, as without-replacement installments, one installment per upload slot, with the seeding centroids of an image placed in exactly one installment. Then A_t and U_1 equal the aggregates of the run in which every client uploads its round-t statistic in one slot. Every broadcast $\theta_{t'}$ with $t' \geq t$ and the deployed model of every level equal those of that run. Round t spans $\max_k R_k$ upload slots instead of one.

Proposition (Proposition 1, formal). Under (A1) to (A4), for every round t, the broadcast θ_t does not depend on $K \geq 1$ or on the partition π of $\mathcal{D}$ across K clients, and it equals the broadcast θ_t of the one-client run $\pi = (\mathcal{D})$. Consequently, the count ranking of the fine codewords, the screened pair set $\mathcal{I}$, and the classifier $W_{\mathcal{L}}$ of every deployment level $\mathcal{L}$ are the same for every partition, and the deployed model at every level equals the centrally trained model at that level.

Corollary (Corollary 1, formal). Let $\mathcal{U}$ be the whitened sampled patches of the training set, let E be either codebook, updated by its own nearest-codeword assignment over $\mathcal{U}$, and let $Q(E) = \sum_{u\in\mathcal{U}} \min_j \|u - e_j\|_2^2$ be its quantization error. Under (A1) to (A4), every codebook round maps E to E' with $Q(E') \leq Q(E)$, and the codebook reaches a fixed point of the update, $E' = E$, after finitely many rounds.

Proof of Lemma 1. Fix round t and client k. By (A1), the server broadcasts nothing before every installment of every client has arrived, so every installment is computed from the same broadcast θ_{t-1}. By (A3), the term of an index $q \in \mathcal{Q}_k$ does not depend on the upload slot in which it is computed. By (A2),

$$\sum_{\ell=1}^{R_k} \underbrace{\sum_{q\in\mathcal{Q}_k^{\ell}} a_t\big(q;\theta_{t-1}\big)}_{\text{installment } \ell} = \underbrace{\sum_{q\in\mathcal{Q}_k} a_t\big(q;\theta_{t-1}\big)}_{\text{the single upload}}, \tag{7}$$

and in round 1 every seeding centroid arrives in exactly one installment, so the multiset union over the installments equals $\biguplus_{x\in\mathcal{D}_k} b(x)$. The right sides are the single uploads, so A_t and U_1 are unchanged. By (A4), $\theta_t = \Phi_t(A_t, U_1, \theta_{t-1}; \xi)$ is unchanged. By (A3), every later term $\rho_{t'}(q;\theta_{t'-1})$ depends on the upload only through the broadcasts, so induction on t' shows that every later broadcast and the deployed model of every level are unchanged. The server waits for the last installment of the slowest client, so round t spans $\max_k R_k$ upload slots. □

Proof of Proposition 1. Fix K and a partition π. By (A3), every term $\rho_t(x;\theta_{t-1})$ and every $b(x)$ depends only on the image, the broadcast, and ξ. By (A1), every client uploads every statistic once, and because π is a partition, every image belongs to exactly one client, so every image contributes exactly once. A client that uploads in installments contributes the same sum as a single upload by Eq. 7. Summing over clients, the server obtains

$$\underbrace{\sum_{k=1}^{K} \underbrace{\sum_{x\in\mathcal{D}_k} a_t\big(x;\theta_{t-1}\big)}_{\text{client } k\text{'s upload}}}_{} = \underbrace{\sum_{x\in\mathcal{D}} a_t\big(x;\theta_{t-1}\big)}_{\text{the one-client total}}, \qquad \underbrace{\biguplus_{k=1}^{K} \underbrace{\biguplus_{x\in\mathcal{D}_k} b(x)}_{\text{client } k\text{'s centroids}}}_{} = \underbrace{\biguplus_{x\in\mathcal{D}} b(x)}_{\text{one-client union}}. \tag{8}$$

The right side of each equality depends on π only through θ_{t-1}. The server uses U_1 only through its lexicographically sorted list, and this list is the same for every order of arrival because equal vectors are interchangeable in the list. Hence Φ_1 is a function of the multiset U_1, and the codeword indices drawn from the list are the same for every partition. Under (A2) a partition changes which client evaluates each per-image term and the order of the additions, and neither changes an exact result. By (A4) Φ_t is a deterministic function of $(A_t, U_1, \theta_{t-1}, \xi)$, so θ_t depends on π only through θ_{t-1}. With $\theta_0 = \xi$, induction on t gives the same θ_t for every K and every π, including the one-client run. The same induction gives the same aggregates A_t for every π, because A_t depends on π only through θ_{t-1}. The count ranking of the fine codewords is a function of $A_{T_{\mathrm{km}}+1}$ and the tie rule of

(A4), and the pair set $\mathcal{I}$ is part of $\theta_{T_{\mathrm{km}}+2}$, so the index set $\mathcal{L}$ of every level is the same for every π. The standardized sums $\tilde{G}$ and $\tilde{H}$ are functions of $A_{T_{\mathrm{km}}+3}$, and $W_{\mathcal{L}}$ is solved from the principal submatrix $\tilde{G}_{\mathcal{L}\mathcal{L}}$ and the row block $\tilde{H}_{\mathcal{L}}$ (Appendix A.1), so $W_{\mathcal{L}}$ is the same for every π. The level encoder computes exactly the entries of the full feature vector indexed by $\mathcal{L}$, because the threshold $\bar{d}_p$ uses the coarse codebook only, so the deployed model at level $\mathcal{L}$ is the same for every π and equals the one-client run's model at that level. □

Proof of Corollary 1. By Proposition 1, the aggregates $\sum_k S_j^{(k)}$ and $\sum_k n_j^{(k)}$ equal the sums over $\mathcal{U}$, so a codebook round is one batch k-means iteration on $\mathcal{U}$. Write $Q(E, a) = \sum_{u \in \mathcal{U}} \|u - e_{a(u)}\|_2^2$ for the error of an assignment a, so that $Q(E) = \min_a Q(E, a)$. The assignment step chooses nearest codewords, so the chosen a has $Q(E, a) = Q(E)$. The update step sets every codeword with assigned patches to the mean of those patches, which is the unique minimizer of its term, and a codeword with no assigned patches contributes zero whatever its value, so $Q(E', a) \le Q(E, a)$ with equality only when $E' = E$ (Lloyd, 1982). Hence $Q(E') \le Q(E', a) \le Q(E)$. After an update, $Q(E', a)$ is a function of a alone, because every codeword with assigned patches equals the mean of those patches and every other codeword contributes zero. The assignment a ranges over a finite set. If an assignment repeats in a later round, the error was constant in between, so no update in between changed a codeword. By (A4), an unchanged codebook gives the same assignment in the next round, so the codebook is at a fixed point. Hence, the rounds reach a fixed point after finitely many rounds. The threshold $\bar{d}_p$ is used by the activations only and not by the assignment, so the argument does not depend on it. □

A.3 Protocol details

Table 3: The four datasets and the fixed $75/25$ split.

Dataset	Classes	Image size	Train	Test
CIFAR-10	10	32×32	45,000	15,000
CIFAR-100	100	32×32	45,000	15,000
Tiny-ImageNet	200	64×64	75,000	25,000
Flowers-102	102	64×64	6,141	2,048

Table 3 lists the four datasets. For each dataset, we merge the official training and test images and split the merged set $75/25$ into a training set and a test set once with seed 0. Every method uses this split. The training set is partitioned across 20 clients. Under the Dirichlet partition, each client receives a random fraction of every class, drawn from Dirichlet(α), and the whole assignment is redrawn when a client receives fewer than 32 samples. Under IID, the assignment shuffles the training set and splits it evenly.

Table 4 lists the model hyperparameters of FedCensus. The same values are used on every dataset and partition.

Table 5 lists the architecture of CNN-S and CNN-L.

The weight-averaging and computation-heterogeneous methods in Table 6 train with SGD at learning rates 0.05 and 0.01, respectively, momentum 0.9, weight decay 5×10^{-4}, and batch size 64, for 2 local epochs per round and 1000 rounds, with the same flip-and-crop augmentation as FedCensus. The one-shot, FedProtoKD, and Scattering-ridge rows follow the training settings described in Appendix A.4. Method-specific choices, such as the momentum-free updates of SCAFFOLD (Karimireddy et al., 2020) and FIARSE (Wu et al., 2024), follow the released code, and Table 6 lists the differences. The HtFLlib (Zhang et al., 2025) methods train with plain SGD at learning rate 0.01, without momentum or weight decay, for 1 local epoch per round, batch size 64, and 1000 rounds.

FedCensus runs on the CPU of one workstation with a 16-core AMD Ryzen 9 5950X and 128 GB of memory, using 24 threads, and the CIFAR-100 pipeline takes about 90 minutes. The baselines run on one Quadro RTX 8000 GPU.

Table 4: Model hyperparameters of FedCensus.

Symbol	Meaning	Value
w	patch height and width in pixels	6
r	dimension of a whitened patch	108
ε	value added to every eigenvalue in Eq. 1	0.1
none	number of images used for whitening and the codebook, over all clients	12,000
none	number of patches sampled from each of these images	40
none	number of centroids each image contributes to initialize the codebook	2
J	number of fine codewords	4,096
J_P	number of coarse codewords	512
T_{km}	number of codebook rounds	15
none	fraction of training samples excluded from pair screening	one tenth
m_P	number of pair products kept, out of 131,328	24,576
D	number of features, $4J + m_P + 5J_P$	43,520
λ	weight of the ridge penalty	100
none	number of augmented copies added to each image	2
none	padding of the random crop in pixels	4

Table 5: Architecture of CNN-S and CNN-L. The CNN-S and CNN-L columns list the number of filters of each stage, and the output size is for a 32×32 input. Parameters are counted for $C = 100$ classes.

Stage	Layers	CNN-S	CNN-L	Output
Stem	conv 3×3, batch normalization (BN), ReLU	64	128	32×32
Block 1	$2\times$(conv 3×3, BN), residual, stride 1	64	128	32×32
Block 2	$2\times$(conv 3×3, BN), 1×1 shortcut, stride 2	128	256	16×16
Block 3	$2\times$(conv 3×3, BN), 1×1 shortcut, stride 2	256	512	8×8
Head	global average pool, linear to C	256	512	C
Parameters		1.25M	4.94M	
FLOPs per image		390M	1551M	

A.4 IMPLEMENTATION DETAILS

Table 6 lists, for every method in Tables 1, 11, and 7, the number of training rounds and every difference from the released code. All methods use the same cached partitions and are evaluated with the same protocol (Section 4.1).

Table 6: Implementation details of every method. Every method is our port of the algorithm of its original paper under the shared training and evaluation protocol, except FedProto, FedTGP, and FD, which run in HtFLlib (Zhang et al., 2025) on our exported split.

Method	Rounds	Differences from the released code and notes
Weight averaging on CNN-S		
FedAvg (McMahan et al., 2017), FedProx (Li et al., 2020a), SCAFFOLD (Karimireddy et al., 2020), MOON (Li et al., 2021)	1000	none
FedLESAM (Fan et al., 2024)	1000	the release perturbs each tensor by one scalar (indexing bug); we implement the paper's per-weight perturbation
Computation-heterogeneous		
HeteroFL (Diao et al., 2021), FedRolex (Alam et al., 2022), DepthFL (Kim et al., 2023), FIARSE (Wu et al., 2024)	1000	clients are assigned to the three tiers round-robin (7/7/6); evaluated on the deployed subnetwork
FedHPro (Wang et al., 2026)	1000	official differentiable Siamese augmentation (DSA) kept; evaluated on each client's personalized model
Generator		
DENSE (Zhang et al., 2022), CoBoosting (Dai et al., 2024)	1	one-shot after 100 local epochs; official flip-and-crop augmentation; evaluated on the student model on the server
FedMHO (Yao et al., 2026)	1	one-shot after 200 local epochs; the released code selects the synthesized samples at random, and we implement the paper's selection rule
Prototype and logit		
FedProto (Tan et al., 2022), FedTGP (Zhang et al., 2024), FD (Jeong et al., 2018)	1000	HtFLlib with the HtFE-img-9 model zoo; flip-and-crop added to the training loader; evaluated post hoc with our protocol
ProtoNorm (Lee et al., 2025)	1000	arXiv preprint; the paper's own-distribution protocol reaches 0.289 in our replication
FedProtoKD (Siddika et al., 2026)	100	uses public data, outside the setting; kept as the most favorable logit case; peaks near round 90
Statistics reference		
Scattering-ridge	1	Kymatio scattering features with two scales and a ridge classifier; the fixed-feature reference
References outside the setting		
Local-only, Central	1000	CNN-S and CNN-L defined in Section 4.1; Central uses cosine decay
Pretrained references		
FedCGS (Guan et al., 2025), Fed3R (Fanì et al., 2024), DeepAFL (Tang et al., 2026)	1, 1, 41	ImageNet-pretrained frozen features (Table 7)
FedCensus	18	none

Weight averaging FedAvg, FedProx, SCAFFOLD, MOON (Li et al., 2021), and FedLESAM (Fan et al., 2024) train CNN-S with the settings listed in Appendix A.3. FedProx adds the proximal term with weight 0.01. SCAFFOLD updates its control variates with option II shown in (Karimireddy et al., 2020) and uses a server learning rate of 1. MOON uses the contrastive loss with temperature

0.5 and weight 1 on CIFAR-100. FedLESAM runs the FedLESAM-S variant of the released code with perturbation radius 0.5 and correction weight 0.1. FedHPro (Wang et al., 2026) averages the client models by FedAvg and keeps 5 hyper-prototypes and 50 meta prototypes per class on the server, in the 256-dimensional feature space, trained by gradient matching against the per-class feature and classifier gradients that the clients upload. The clients of FedHPro train for 10 local epochs at learning rate 0.1 with the DSA augmentation (Zhao & Bilen, 2021) of the released code, and each client is evaluated with its own model after local training.

Computation-heterogeneous methods The computation-heterogeneous methods train one global model from which each client extracts a subnetwork that fits its budget, so their evaluation needs clients with different budgets. The clients are assigned to the three budget tiers in order, client by client, so the tiers contain 7, 7, and 6 of the 20 clients. HeteroFL and FedRolex assign a tier 0.25, 0.5, or 1.0 of the filters of CNN-S in every layer, which costs 25, 98, and 390 MFLOPs per image. HeteroFL always keeps the first filters, and FedRolex shifts the window of filters by one position every round. DepthFL assigns a tier to the stem and the first one, two, or three blocks of CNN-S, with an early-exit classifier after each block, which costs 173, 301, and 419 MFLOPs. A client trains every exit it contains, and the exits distill into each other. FIARSE keeps 25%, 50%, or 100% of the weights with the largest magnitude, recomputed at every forward pass, which costs 125, 212, and 390 MFLOPs when the pruned weights are skipped. Each client is evaluated with the subnetwork it deploys, and the FLOPs column of Table 1 averages the tier costs over the clients.

Generator methods DENSE, CoBoosting (Dai et al., 2024), and FedMHO (Yao et al., 2026) are one-shot methods. In DENSE and CoBoosting, every client trains CNN-S on its own data for 100 epochs with the flip-and-crop augmentation and uploads the model once. The server then trains a generator against the ensemble of client models for 200 epochs with 30 generator steps per epoch, at learning rate 10^{-3} for the generator and 0.01 for the student, with distillation temperature 4. The server applies the flip-and-crop transform of the released code to the synthesized images. CoBoosting adds the sample hardness weighting and the ensemble reweighting of the released code. In FedMHO, half of the clients train CNN-S for 200 epochs with the same augmentation, and the other half train a conditional variational autoencoder with a 20-dimensional latent space for 30 epochs. The server averages the classifiers, samples 12,000 images from the autoencoders, keeps 80% of them by the paper's selection rule, and fine-tunes the averaged model on them for 30 epochs at learning rate 10^{-4}. All three are evaluated with the server model.

Prototype and logit methods FedProto, FedTGP (Zhang et al., 2024), and FD (Jeong et al., 2018) run in HtFLlib on its HtFE-img-9 model zoo, which contains four small ResNets of the library and ResNet-18, -34, -50, -101, and -152. Client k trains model $k \bmod 9$ with a 512-dimensional feature layer. FedProto adds to the loss, with weight 1, the distance from each sample to the global prototype of its class. FedTGP trains the global prototypes on the server for 100 epochs at learning rate 0.01 with a margin of at most 100, and FD adds a distillation term toward the per-class average logits with weight 1. HtFLlib loads our dataset split and adds the flip-and-crop augmentation to the training samples, and we evaluate the trained client models with the protocol defined in Section 4.1. ProtoNorm and FedProtoKD (Siddika et al., 2026) have no released code, so we implement them from the papers. ProtoNorm trains a zoo of four small convolutional networks with a linear projection to 512 dimensions, 5 clients per network, with prototype loss weight 1, one local epoch of batch size 32 per round, and a prototype scale of 200 on CIFAR-100. The server aligns the global prototypes on the unit sphere and scales them, and a client classifies a missing class with the scaled global prototype. FedProtoKD trains a zoo of four small convolutional networks with a learned projection to 512 dimensions on 2,500 public samples held out from every client, with 5 client epochs and 10 server epochs per round, batch size 32, margin 50, and every mixing weight at 0.5. Every client of these methods is evaluated with its own model.

Statistics references Scattering-ridge replaces the learned encoder of FedCensus with fixed wavelet features and keeps the rest of the pipeline. The features are the scattering transform of Kymatio (Andreux et al., 2020) with two scales and eight orientations, without spatial pooling of the scattering output, which gives $243 \times 8 \times 8 = 15{,}552$ features for a 32×32 image. The clients encode the same augmented copies and upload the same sums, and the server standardizes the features and solves the ridge regression with weight 100 in one round. Local-only trains CNN-S on each

client's own data for 1000 rounds of 2 epochs with the settings listed in Appendix A.3 and never communicates, and each client is evaluated with its own model. Central runs the FedAvg code with one client that owns the whole training set, on CNN-S and CNN-L, at learning rate 0.05 with cosine decay to zero over the 1000 rounds.

Pretrained references FedCGS, Fed3R, and DeepAFL use a frozen ResNet-18 pretrained on ImageNet-1k (Russakovsky et al., 2015). Every image is upsampled to 224×224 and normalized with the ImageNet mean and standard deviation, and the 512-dimensional output before the last layer is the feature. FedCGS uploads the per-class feature sums, the per-class counts, the total feature sum, and the outer-product sum in one round, and the server builds a linear classifier with a shared covariance from these sums. Fed3R uploads the Gram matrix and the class-sum matrix of the features in one round, and the server solves a ridge regression with weight 0.01 and normalizes every column of the classifier. DeepAFL stacks 20 analytic layers of random projections to 1024 dimensions with the Gaussian error linear unit (GELU) activation, solves the classifier of every layer in closed form with weight 1 and the layer-to-layer least squares with weight 0.05, and needs two rounds per layer, 41 rounds in total. All three are evaluated with the server classifier on every client.

A.5 Pretrained references

The frozen features of ImageNet-pretrained methods violate the constraints listed in Section 3.1. The representation of these methods is learned outside the federation from 1.2 million labeled images, and every client must run the frozen ResNet-18, which costs 3.63 GFLOPs per image at 224×224, 4.6 times the cost of FedCensus-L and 9 times the cost of CNN-S. FedCGS and Fed3R run one round on the frozen features, and DeepAFL runs 41 rounds. Table 7 reports the three methods as pretrained references, evaluated under the protocol defined in Section 4.1. The three methods solve their classifier the way FedCensus does. None of the three trains by gradient descent, and each classifier is solved in closed form from sums that the clients upload. FedCGS uploads per-class feature sums, counts, and the outer-product sum, Fed3R uploads the Gram matrix and the class-sum matrix, and DeepAFL uploads the same two matrices for every layer. As a consequence, the present-class and missing-class accuracies of the three methods differ by about 0.01, the same as for FedCensus, and their accuracy does not depend on the partition. A frozen ResNet-18 with a linear classifier reaches 0.631 with FedCGS and 0.642 with Fed3R, and FedCensus, trained from the 45,000 training images alone, reaches 0.625. The 20 analytic layers of DeepAFL add 0.07 and reach 0.710, which is a little above the accuracy of centralized training on CNN-S. The communication of the three methods is small because the feature has 512 dimensions. FedCGS and Fed3R upload about 1.3 MB per client, and DeepAFL exchanges 1024×1024 matrices for each of its 20 layers. The 3.99 GB of FedCensus is the communication cost of learning the representation inside the federation, with $D = 43{,}520$ features. In short, without pretraining and without gradients, FedCensus reaches the accuracy of a frozen ImageNet backbone with a linear classifier at less than a quarter of the inference cost, and the remaining gap to DeepAFL and to centralized training is about 0.085.

Table 7: ImageNet-pretrained references on CIFAR-100, evaluated under the protocol defined in Section 4.1. Rounds and Aug indicate the number of training rounds and whether the flip-and-crop augmentation was used. The best value of a column is bold and the second best is underlined.

			Micro (sample-weighted)			**Macro** (balanced)		
Method	Rounds	Aug	All↑	Pres↑	Miss↑	All↑	Pres↑	Miss↑
FedCGS	1	no	0.632	0.637	0.628	0.631	0.636	0.627
Fed3R	1	no	<u>0.643</u>	<u>0.647</u>	<u>0.639</u>	<u>0.642</u>	<u>0.646</u>	<u>0.638</u>
DeepAFL	41	no	**0.711**	**0.717**	**0.708**	**0.710**	**0.716**	**0.707**

A.6 Limitations and discussion

Two limitations follow from the design, and the others concern the scope of the evaluation. First, FedCensus assumes full participation. Every statistic is a sum over all clients, so a slow client adds upload slots through installments, but a client that drops out delays the round until it returns (Sec-

tion 3.2). The method, therefore, targets a fixed set of clients, and sampled participation is left to future work. Second, the accuracy is bounded by the choice of the three statistics and not by their size, because doubling the codebook adds 0.001 (Table 2). At the same time, FedCensus-L needs about twice the inference FLOPs of CNN-S, and FedCensus-S, with the same FLOPs as CNN-S, stays 0.012 below the best homogeneous method. At twice the FLOPs, weight averaging on CNN-L leads FedCensus-L by 0.022, and the pretrained references lead it by 0.017 to 0.085 (Table 7). As for scope, we evaluate methods only on 2 to 100 clients, and evaluation on more clients is left to future work. We also evaluate the models only on low-resolution images. DENSE and CoBoosting synthesize images by inverting the client models on the server. The synthesis stays within the constraints because no model is trained on client data, but the synthesized images resemble client data, and we do not evaluate what they reveal. The communication of 3.99 GB per client, upload and download, is 40% of that of converged FedAvg, and it grows with D^2, so a larger feature vector would raise it. Finally, the privacy claim is established by the attack measured in Section 4.4 and by secure aggregation, and a differential-privacy analysis is left to future work (Remark 1).

A.7 Deployment levels

A client with a smaller deployment budget keeps the first J' codewords of the count ranking, so its feature vector has $D' = 4J' + m_P + 5J_P$ entries, and its classifier is solved from the corresponding principal submatrix of the same Gram matrix. Table 8 lists the four deployment levels. The computation overhead of a deployment level is determined by the number of fine codewords used for the distance calculations. At the full level, FedCensus-L, these distance calculations account for 82% of the total FLOP count, and the whitening, the coarse-codeword distances, the pair products, the spatial moments, and the classifier cost 17, 81, 36, 3, and 9 MFLOPs at every level. The model has 5.00M parameters at the full level and 3.15M at FedCensus-T, and at both levels, most of the parameters are the weights of the classifier W. From FedCensus-L to FedCensus-T, the macro accuracy decreases by 0.011, and the present-class and missing-class accuracies decrease by the same amount, so the decrease in accuracy is not biased. The classifier of every deployment level is solved from the same aggregated statistics, so regardless of the deployment level of a client, the deployed model is well-trained from all clients' data. For the same reason, FedCensus-T stays above every other computation-heterogeneous method in Figure 2(d).

Table 8: Deployment levels of FedCensus on CIFAR-100 ($\alpha = 0.1$). The classifier of every level is solved from a principal submatrix of the same Gram matrix.

				Micro (sample-weighted)			**Macro** (balanced)		
Level	J'	D'	FLOPs (M)↓	All↑	Pres↑	Miss↑	All↑	Pres↑	Miss↑
FedCensus-L	4096	43520	790	**0.6255**	**0.6319**	**0.6207**	**0.6246**	**0.6310**	**0.6197**
FedCensus-M	2048	35328	466	0.6210	0.6274	0.6163	0.6200	0.6264	0.6152
FedCensus-S	1536	33280	385	0.6204	0.6267	0.6157	0.6194	0.6257	0.6146
FedCensus-T	512	29184	**223**	0.6146	0.6209	0.6100	0.6137	0.6200	0.6090

A.8 Rounds analysis

Figure 3 shows how the accuracy changes with the number of codebook rounds. For each point, we train the whole pipeline again with a different T_{km}, so the total number of rounds is $T_{\mathrm{km}}+3$, and we report the macro accuracy of the model at the end of that training. The accuracy rises quickly in the first codebook rounds, to 0.611 at round 4, 0.626 at round 8, and 0.627 at round 13. After that, the accuracy stays between 0.624 and 0.625 up to round 100. At round 18, the three seeds give 0.624, 0.623, and 0.626. The differences along the curve after round 8 are as small as the differences between seeds, so more codebook rounds do not change the accuracy. We fixed $T_{\mathrm{km}} = 15$ before this experiment and did not change it afterward. The default is 0.003 below the value at round 13, which is within the spread between seeds, and every run with at least 5 codebook rounds is within 0.003 of the default.

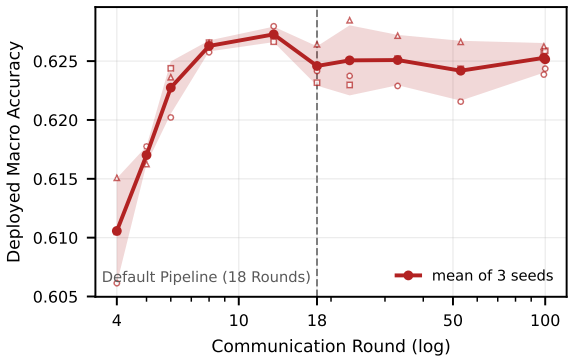

Figure 3: Deployed macro accuracy of FedCensus against the total number of communication rounds, with T_{km} codebook rounds and $T_{\mathrm{km}}+3$ rounds in total (CIFAR-100, $\alpha = 0.1$). The line is the mean over three seeds, the band is one standard deviation, and the default pipeline ends at round 18.

A.9 Communication and computation details

Table 9: Per-round per-client communication of FedCensus-L (CIFAR-100). Totals are per client over the full run. The patch statistics of round 1 stay in fp64 in both columns. The final broadcast excludes the classifiers of the three smaller deployment levels, which together are smaller than W.

Round	Upload	fp32 (MB)	fp16 (MB)
Round 1	patch statistics and seeding centroids	0.61	0.35
Rounds 2 to $T_{\mathrm{km}}+1$	codeword sums ($\times 15$)	30.14	15.07
Round $T_{\mathrm{km}}+2$	pair screening	105.08	52.54
Round $T_{\mathrm{km}}+3$	ridge statistics (G, H, sums, counts)	3805.82	1902.91
Total upload		3941.7	1970.9
Server broadcast, all rounds		49.8	24.9
Total, upload and download		3991.5	1995.8

Table 10: Per-client communication, upload and download, of the two upload schemes for the aggregated statistics of FedCensus-L (CIFAR-100, fp32). Δacc is the difference from the exact scheme. The last two rows list FedAvg on CNN-S and CNN-L for reference.

Scheme	Comm (MB)	Δacc	Secure aggregation
Exact	3991.5	0.000	✓
Factor	1360.8	0.000	×
FedAvg on CNN-S	10005.8	N/A	N/A
FedAvg on CNN-L	39540.5	N/A	N/A

Table 9 lists the per-client communication of the pipeline round by round. The server broadcast over the full run totals 49.8 MB in fp32. The Comm column of Table 1 counts, for the weight-averaging rows, the number of parameters, times 2 for upload and download, times the number of rounds, times 4 bytes. Under this count, CNN-S sends 1.0 GB in 100 rounds and 10.0 GB in 1000 rounds, CNN-L sends 39.5 GB in 1000 rounds, and SCAFFOLD sends twice as much as FedAvg because of its control variates. The computation-heterogeneous rows count the parameters of each client's deployed subnetwork. Scattering-ridge uploads the Gram, class-sum, and feature-sum statistics of its $D = 15{,}552$ features in fp32, and the entry of FedCensus is the fp32 upload-and-download total of Table 9.

A client can upload the statistics of the statistics round in two ways. In the exact scheme, a client uploads the Gram matrix $G_k = \sum_i s_i s_i^\top$, which has $D(D+1)/2$ entries and is a sum, so it is com-

patible with secure aggregation. In the factor scheme, a client uploads its standardized feature matrix Z_k, one row per augmented training image, and the server computes $\sum_k Z_k^\top Z_k$ itself. The upload grows with the number of rows times D instead of D^2, which divides the total communication by three, and the solved classifier is identical. The server then sees every client's per-image encodings, so the factor scheme is not compatible with secure aggregation, and the reconstruction attack described in Appendix A.14 would apply to each image. Table 10 lists both schemes.

A.10 CHANNEL TABLE

Table 11: Accuracy under the IID partition of CIFAR-100, grouped by channel. Rounds and Aug indicate the number of training rounds and whether the flip-and-crop augmentation was used. The best value of a column is bold and the second best is underlined.

Channel	Method	Rounds	Aug	Micro All↑	Macro All↑
weights	FedAvg	1000	✓	**0.710**	**0.709**
prototype	ProtoNorm	1000	✓	0.149	0.148
prototype	FedProto	1000	✓	0.131	0.130
prototype	FedTGP	1000	✓	0.150	0.149
logit	FD	1000	✓	0.009	0.010
logit with public data	FedProtoKD	100	✓	0.077	0.077
generator	DENSE	1	✓	0.315	0.313
generator	CoBoosting	1	✓	0.238	0.237
statistics	FedCensus-L (ours)	18	✓	<u>0.625</u>	<u>0.625</u>

Table 11 lists the accuracy of one or more methods per channel under the IID partition of CIFAR-100, with the rounds and augmentation of every run. Every client has every class, so a method is limited only by what its channel transfers and not by missing classes. FedAvg reaches 0.709, the accuracy of CNN-S with weight averaging. The prototype channel reaches 0.148 with ProtoNorm, 0.130 with FedProto, and 0.149 with FedTGP. The prototype loss only reduces the distance between the features of a class and a shared center, and each client still trains its classifier on its own $2{,}250$ images, so the classifier of a client is never trained on the images of the other clients. The logit channel reaches 0.010 with FD, the accuracy of a random guess among 100 classes. FedProtoKD uses a public dataset, which violates the constraints listed in Section 3.1. We include FedProtoKD because it gives the logit channel the most favorable condition, and even so it reaches only 0.077. The generator channel reaches 0.313 with DENSE and 0.237 with CoBoosting. The student of these two methods learns only from synthesized images, and the IID partition is the easiest case for the client ensemble that guides the synthesis, because every client model has seen every class. FedCensus reaches 0.625, the same value as under Dirichlet $\alpha = 0.1$, because its model does not depend on the partition (Proposition 1). For each of the prototype, logit, and generator channels, the gap to weight averaging under IID is at least 0.39 and close to the channel's gap under Dirichlet $\alpha = 0.1$ in Table 1, so the low accuracies in Table 1 are caused by the channels and not by the missing classes.

A.11 OTHER DATASETS

Tables 12, 13, and 14 repeat the comparison of Table 1 on CIFAR-10, Tiny-ImageNet, and Flowers-102. The methods that satisfy the constraints listed in Section 3.1 are trained without augmentation for the rounds listed. We list FedCensus both with and without augmentation for comparison. On CIFAR-10, FedCensus without augmentation reaches 0.831 macro accuracy, against 0.515 for the best computation-heterogeneous method (FedRolex), 0.217 for the prototype channel (ProtoNorm), and 0.100 for the logit channel (FD). On Tiny-ImageNet, FedCensus without augmentation reaches 0.375, against 0.338 for the best computation-heterogeneous method (DepthFL), 0.012 for the prototype channel, and 0.005 for the logit channel. On Flowers-102, FedCensus without augmentation reaches 0.801, against 0.631 for the best computation-heterogeneous method (DepthFL), 0.062 for the prototype channel, and 0.010 for the logit channel. Figure 4 shows the pattern on all four datasets. The prototype channel classifies no missing class correctly on any of them, the logit channel classifies at most 0.08 of the missing classes correctly, and the missing-class accuracy of FedCensus

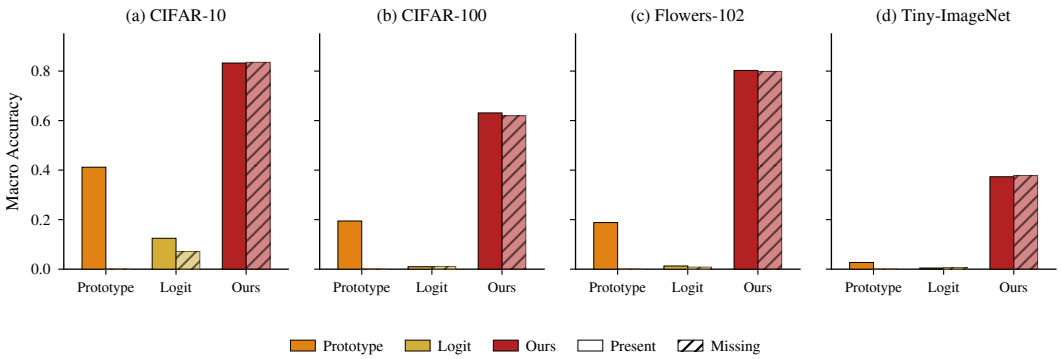

Figure 4: Present-class and missing-class accuracy per channel on four datasets under Dirichlet $\alpha = 0.1$. Colors mark the channel, solid bars are present classes, and hatched bars are missing classes. The bars are the rows of Table 1 for CIFAR-100 and of Tables 12, 13, and 14 for the other datasets, and the FedCensus bars are the runs without augmentation.

is within 0.012 of its present-class accuracy on every dataset. Among the methods that violate the deployment-budget requirement, SCAFFOLD on CNN-S reaches 0.838 on CIFAR-10 and 0.405 on Tiny-ImageNet, and FedAvg on CNN-S reaches 0.679 on Flowers-102, against 0.848, 0.396, and 0.842 for FedCensus with augmentation.

Table 12: Comparison on CIFAR-10 (Dirichlet $\alpha = 0.1$, deployed protocol defined in Section 4.1). Rounds and Aug indicate the number of training rounds and whether the flip-and-crop augmentation was used. Within each block, the best value of a column is bold and the second best is underlined.

			Micro (sample-weighted)			**Macro** (balanced)		
Method	Rounds	Aug	All↑	Pres↑	Miss↑	All↑	Pres↑	Miss↑
Methods that satisfy the constraints listed in Section 3.1								
computation-heterogeneous								
HeteroFL	1000	no	0.467	0.497	0.419	0.467	0.497	0.420
FedRolex	1000	no	0.514	0.537	0.476	0.515	0.538	0.477
DepthFL	1000	no	0.469	0.519	0.411	0.471	0.521	0.412
FIARSE	1000	no	0.451	0.455	0.471	0.452	0.455	0.472
prototype								
ProtoNorm	1000	no	0.216	0.412	0.000	0.217	0.412	0.000
logit								
FD	1000	no	0.099	0.124	0.070	0.100	0.125	0.071
statistics								
FedCensus-L (ours), no augmentation	18	no	<u>0.831</u>	<u>0.832</u>	<u>0.835</u>	<u>0.831</u>	<u>0.832</u>	<u>0.836</u>
FedCensus-L (ours)	18	✓	**0.847**	**0.849**	**0.851**	**0.848**	**0.848**	**0.852**
Methods that violate the deployment-budget requirement (weight averaging on CNN-S)								
FedAvg	1000	✓	<u>0.708</u>	<u>0.731</u>	<u>0.679</u>	<u>0.708</u>	<u>0.732</u>	<u>0.679</u>
FedLESAM	1000	✓	0.688	0.722	0.660	0.689	0.723	0.661
SCAFFOLD	1000	✓	**0.837**	**0.846**	**0.826**	**0.838**	**0.847**	**0.826**

Table 13: Comparison on Tiny-ImageNet (Dirichlet $\alpha = 0.1$, deployed protocol defined in Section 4.1). Rounds and Aug indicate the number of training rounds and whether the flip-and-crop augmentation was used. Within each block, the best value of a column is bold and the second best is underlined.

			Micro (sample-weighted)			**Macro** (balanced)		
Method	Rounds	Aug	All↑	Pres↑	Miss↑	All↑	Pres↑	Miss↑
Methods that satisfy the constraints listed in Section 3.1								
computation-heterogeneous								
HeteroFL	1000	no	0.160	0.159	0.160	0.161	0.160	0.161
FedRolex	1000	no	0.159	0.153	0.162	0.159	0.153	0.163
DepthFL	1000	no	0.336	0.358	0.318	0.338	0.359	0.319
FIARSE	1000	no	0.107	0.116	0.099	0.108	0.117	0.100
prototype								
ProtoNorm	1000	no	0.012	0.027	0.000	0.012	0.027	0.000
logit								
FD	1000	no	0.005	0.005	0.007	0.005	0.004	0.007
statistics								
FedCensus-L (ours), no augmentation	18	no	<u>0.373</u>	<u>0.371</u>	<u>0.376</u>	<u>0.375</u>	<u>0.373</u>	<u>0.378</u>
FedCensus-L (ours)	18	✓	**0.394**	**0.392**	**0.397**	**0.396**	**0.394**	**0.399**
Methods that violate the deployment-budget requirement (weight averaging on CNN-S)								
FedAvg	300	✓	<u>0.392</u>	<u>0.411</u>	<u>0.377</u>	<u>0.394</u>	<u>0.413</u>	<u>0.379</u>
FedLESAM	300	✓	0.386	0.398	0.376	0.388	0.401	0.378
SCAFFOLD	300	✓	**0.403**	**0.415**	**0.394**	**0.405**	**0.418**	**0.396**

Table 14: Comparison on Flowers-102 (Dirichlet $\alpha = 0.1$, deployed protocol defined in Section 4.1). Rounds and Aug indicate the number of training rounds and whether the flip-and-crop augmentation was used. Within each block, the best value of a column is bold and the second best is underlined.

			Micro (sample-weighted)			**Macro** (balanced)		
Method	Rounds	Aug	All↑	Pres↑	Miss↑	All↑	Pres↑	Miss↑
Methods that satisfy the constraints listed in Section 3.1								
computation-heterogeneous								
HeteroFL	1000	no	0.350	0.390	0.322	0.342	0.380	0.316
FedRolex	1000	no	0.330	0.355	0.311	0.316	0.343	0.299
DepthFL	1000	no	0.644	0.660	0.634	0.631	0.648	0.618
FIARSE	1000	no	0.279	0.342	0.244	0.269	0.330	0.240
prototype								
ProtoNorm	1000	no	0.069	0.200	0.000	0.062	0.188	0.000
logit								
FD	1000	no	0.011	0.014	0.009	0.010	0.013	0.009
statistics								
FedCensus-L (ours), no augmentation	18	no	<u>0.810</u>	<u>0.809</u>	<u>0.810</u>	<u>0.801</u>	<u>0.803</u>	<u>0.799</u>
FedCensus-L (ours)	18	✓	**0.854**	**0.856**	**0.851**	**0.842**	**0.847**	**0.838**
Methods that violate the deployment-budget requirement (weight averaging on CNN-S)								
FedAvg	1000	✓	**0.691**	**0.698**	**0.686**	**0.679**	**0.692**	**0.670**
FedLESAM	1000	✓	0.470	0.492	0.461	0.441	0.460	0.436
SCAFFOLD	1000	✓	<u>0.665</u>	<u>0.676</u>	<u>0.659</u>	<u>0.658</u>	<u>0.671</u>	<u>0.650</u>

A.12 ACCURACY ACROSS PARTITIONS

Table 15: Deployed macro accuracy on CIFAR-100 under the IID partition and four Dirichlet partitions. The client-grouped row adds the client statistics one client at a time and averages over the five partitions.

Partition	A_{all}	A_{pres}	A_{miss}
IID	0.6242	0.6242	N/A
$\alpha = 1.0$	0.6242	0.6237	0.6476
$\alpha = 0.3$	0.6242	0.6248	0.6222
$\alpha = 0.1$	0.6242	0.6306	0.6188
$\alpha = 0.05$	0.6242	0.6193	0.6267
Client-grouped order	0.6239	0.6242	0.6286
$K = 100$ clients, $\alpha = 0.1$	0.6242	0.6245	0.6239

Table 16: Deployed macro accuracy on CIFAR-100 under the IID partition and four Dirichlet partitions. All, Pres, and Miss are the metrics defined in Section 4.1, and Miss is undefined under IID. The weight-averaging, prototype, and logit runs use 1000 rounds, CoBoosting is one-shot, and FedCensus uses 18 rounds. Every run uses the flip-and-crop augmentation. Within each block, the best value of a column is bold and the second best is underlined.

			Macro (balanced)				
				Dirichlet α			
Channel	Method	Metric	IID	1.0	0.3	0.1	0.05
statistics	FedCensus-L (ours)	All↑	**0.624**	**0.624**	**0.624**	**0.624**	**0.624**
		Pres↑	**0.624**	**0.624**	**0.625**	**0.631**	**0.619**
		Miss↑	N/A	**0.648**	**0.622**	**0.619**	**0.627**
weights (depth)	DepthFL	All↑	0.535	0.525	0.508	0.452	0.409
		Pres↑	0.535	0.525	0.512	0.476	0.439
		Miss↑	N/A	0.532	0.494	0.430	0.397
weights (width)	HeteroFL	All↑	0.334	0.325	0.309	0.240	0.196
		Pres↑	0.334	0.325	0.314	0.247	0.196
		Miss↑	N/A	0.349	0.296	0.233	0.195
prototype	ProtoNorm	All↑	0.148	0.101	0.095	0.088	0.076
		Pres↑	0.148	0.104	0.120	0.195	0.249
		Miss↑	N/A	0.000	0.000	0.000	0.000
generator	CoBoosting	All↑	0.237	0.265	0.240	0.170	0.150
		Pres↑	0.237	0.265	0.248	0.195	0.165
		Miss↑	N/A	0.251	0.212	0.145	0.141
logit	FD	All↑	0.010	0.010	0.010	0.010	0.010
		Pres↑	0.010	0.010	0.011	0.010	0.011
		Miss↑	N/A	0.021	0.009	0.011	0.010
Reference: weight averaging on CNN-S, which violates the deployment-budget requirement							
weights	FedAvg	All↑	**0.709**	0.690	**0.682**	0.627	0.564
		Pres↑	**0.709**	0.690	**0.686**	0.645	0.585
		Miss↑	N/A	0.701	**0.670**	0.607	0.554
weights	FedProx	All↑	0.700	**0.699**	0.673	**0.637**	**0.574**
		Pres↑	0.700	**0.698**	0.677	**0.654**	**0.595**
		Miss↑	N/A	**0.706**	0.660	**0.620**	**0.564**

Table 16 lists the deployed macro accuracy of one method per channel under the IID partition and four Dirichlet partitions, with the present-class and missing-class accuracies of every run. FedCensus reaches 0.624 under every partition. The present-class and missing-class accuracies of FedCensus differ by at most 0.024 and change order between partitions, so the difference comes from which classes a partition assigns to each client and not from the model. The performance of the methods that average weights decreases as the partition becomes more skewed, and Section 4.4 lists the drops of DepthFL, HeteroFL, and FedAvg from IID to $\alpha = 0.05$. Under the skewed partitions, DepthFL

and FedAvg lose more on the missing classes than on the present classes, by 0.046 and 0.038 at $\alpha = 0.1$, while HeteroFL loses on both equally. The accuracy of the prototype, generator, and logit channels is low under every partition. ProtoNorm classifies no missing class correctly under any partition, CoBoosting classifies about a quarter of them correctly, and FD stays at the accuracy of a random guess. The partition, therefore, changes the accuracy of every other channel, but it does not change the accuracy of FedCensus.

A.13 Installments

Table 17 lists the result of splitting the statistics upload described in Section 3.2 into R installments. Each client divides its samples into R groups by the global seed. In each of the following R upload slots, the client encodes one group and uploads the contribution of that group to G, H, the feature sums, and the class counts, so its computation and upload per slot are $1/B$ of the full ones. The server adds the R partial sums before it solves the ridge regression. The statistics round, therefore, spans R upload slots, and the total upload does not change. The codebook rounds and the screening round are not split, because their uploads are only 2 MB and 105 MB per client. The column $\max|\Delta W|$ is the largest difference between a weight of the classifier solved from the installment sums and the same weight solved from the single upload. The three accuracy columns are those of the installment classifier. The differences of at most 1.6×10^{-3} come from the order of the floating-point additions. In double precision, summing the client statistics in a different order changes the weights by 10^{-10} (Section 4.4). The difference of 1.6×10^{-3} in single precision is therefore floating-point rounding, and the classifier solved from installments equals the classifier solved from the single upload up to rounding.

Table 17: Installments on CIFAR-100 ($\alpha = 0.1$). A client whose training budget is $1/R$ of the full upload uploads the statistics in R installments.

Training budget	Installments R	$\max\|\Delta W\|$	A_{all}	A_{pres}	A_{miss}
1	1	0	0.6242	0.6306	0.6188
$1/2$	2	1.6×10^{-3}	0.6234	0.6298	0.6182
$1/4$	4	1.6×10^{-3}	0.6240	0.6305	0.6187
$1/8$	8	1.5×10^{-3}	0.6242	0.6307	0.6189

A.14 Reconstruction attack

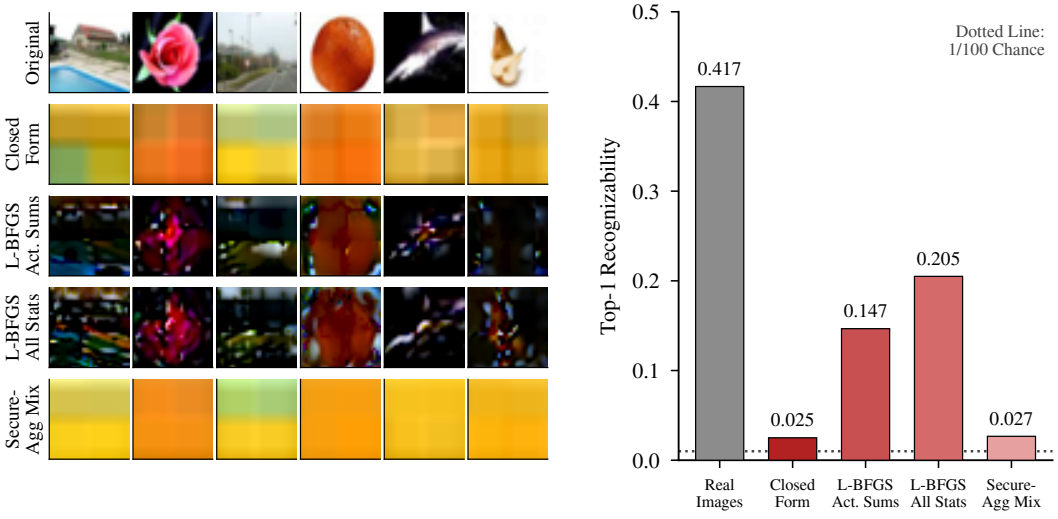

Figure 5: Reconstructions from uploaded statistics and their top-1 recognizability.

Table 18: Inversion of uploaded class columns on CIFAR-100, 200 targets. PSNR and SSIM compare a reconstruction with its original, and top-1 and top-5 are the recognizability under a linear classifier trained on all three statistics, whose accuracy on the originals is listed in the first row. Every value is the mean over 3 runs of the attack.

Input	PSNR (dB)↑	SSIM↑	Top-1↑	Top-5↑
Original images	N/A	N/A	0.417	0.675
Single-image column, closed form	8.81	0.117	0.025	0.088
Closed form and L-BFGS on the activation sums	7.95	0.071	0.147	0.278
Closed form and L-BFGS on all three statistics	8.24	0.080	0.205	0.407
Secure-aggregated column, closed form	7.55	0.081	0.027	0.068

The chance rates are 0.010 for top-1 and 0.050 for top-5.

Figure 5 shows six targets, their closed-form reconstructions from single-image columns, the same reconstructions after refinement on the activation sums and on all three statistics, and their reconstructions from secure-aggregated columns, with the top-1 recognizability of each. The closed-form inverse uses the activation sums only, and it is four flat patches, one per quadrant, because two steps of the encoder cannot be inverted from the activation sums alone. The contrast normalization removes the mean and the contrast of every patch, and the quadrant sums remove the position of a patch inside its quadrant. The pair products and the spatial moments keep more, and the refinement uses them.

Table 18 lists the results of the attack on 200 targets drawn from the single-image columns of the $\alpha = 0.1$ partition, with the encoder configuration of Appendix A.3, and every value is the mean over three runs of the attack. PSNR and SSIM are computed after both images are scaled to $[0, 1]$, because the contrast normalization removes the absolute brightness. The recognizability classifier is a ridge classifier on all three statistics, fitted on $5{,}000$ real training images with the same encoder, with the ridge weight chosen on a held-out fifth of them. Its accuracy on the originals is therefore the ceiling of the measurement, and the chance rates are 0.01 for top-1 and 0.05 for top-5 with 100 classes. The closed-form inverse recovers little, with a PSNR of 8.81 dB and a top-1 recognizability of 0.025. In the two refined rows, 60 iterations of L-BFGS refine the closed-form image by minimizing the standardized distance between the encoding of the image and the uploaded column, with a total-variation penalty, on the activation sums alone and on all three statistics. Matching the activation sums raises the top-1 recognizability to 0.147, and matching all three statistics raises it to 0.205, half of the ceiling. The PSNR stays below that of the closed-form inverse, so the refinement recovers the class information that the classifier uses and not the image. The refined images in Figure 5 keep the coarse color layout and the shape of a compact object such as an orange or a rose, and nothing of a scene. A secure-aggregated column sums the class-c encodings of up to 20 clients, and its closed-form inverse is a class average whose top-1 recognizability is 0.027.

Single-image columns are common. On the $\alpha = 0.1$ partition with 20 clients, 168 of the 928 class columns that clients upload contain one image, and every client uploads at least one such column, so without secure aggregation, the attack can target 18% of the columns of H_k. With secure aggregation, the server sees only the sum over clients, and the last row of Table 18 evaluates this case.