Efficient Federated Random Subnetwork Training

Anonymous Author(s)
Affiliation
Address
email

Abstract

One main challenge in federated learning is the large communication cost of exchanging weight updates from clients to the server at each round. While prior work has made great progress in compressing the weight updates through gradient compression methods, we propose a radically different approach that does not update the weights. Instead, our method freezes the weights at their initial random values and learns how to sparsify the random network for the best performance. To this end, the clients collaborate in training a *stochastic* binary mask to find the optimal random sparse network within the original one. At the end of the training, the final model is a randomly weighted sparse network – or a subnetwork inside the random dense network. We show improvements in accuracy, communication bitrate (less than 1 bit per parameter (bpp)), convergence speed, and final model size (less than 1 bpp) over relevant baselines on MNIST, EMNIST, CIFAR-10, and CIFAR-100 datasets, in the low bitrate regime under various system configurations.

1 Introduction & Related Work

Federated learning (FL) is a distributed learning framework where clients collaboratively train a model by performing local training on their data and by sharing their local updates with a server every few iterations, which in turn aggregates the local updates to create a global model, that is then transmitted to all the clients to continue the training. While being an appealing approach for enabling model training without the need to collect client data at the server, communication of local updates (which typically have a size comparable to the model size) is a significant bottleneck in FL (Kairouz et al., 2021). This has motivated research in communication-efficient FL strategies (McMahan et al., 2017) and various gradient compression schemes via sparsification (Lin et al., 2018; Wang et al., 2018; Barnes et al., 2020; Isik et al., 2022), quantization (Alistarh et al., 2017; Bernstein et al., 2018), and low-rank approximation (Konečnỳ et al., 2016; Vargaftik et al., 2021, 2022; Basat et al., 2022).

In this work, we take a radically different approach for communication efficiency and propose a strategy that does not require an exchange of weight updates. Instead of training the weights, we

(1) initialize a random dense network at the server with d weights collected in the weights vector $\boldsymbol{w}^{\text{init}} = (w_1^{\text{init}}, w_2^{\text{init}}, \ldots, w_d^{\text{init}})$, using a random seed SEED, and broadcast SEED to the clients enabling them to reproduce the same weights vector $\boldsymbol{w}^{\text{init}}$,

(2) keep the weights frozen at their initial values at all times in both client and server nodes,

(3) train a Bernoulli mask with d parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_d) \in [0,1]^d$ in an FL setting,

(4) sample a binary mask from the trained Bernoulli mask, and generate a randomly weighted sparse network – or a subnetwork inside the initial random dense network as $\boldsymbol{w}^{\text{final}} = \text{Bern}(\boldsymbol{\theta}) \odot \boldsymbol{w}^{\text{init}}$, where $\text{Bern}(\cdot)$ is the Bernoulli sampling operation, and $\odot$ the element-wise multiplication.

We call this framework Federated Probabilistic Mask Training (FedPM). At first glance, it may be surprising that there exist subnetworks inside randomly initialized networks that could perform well

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

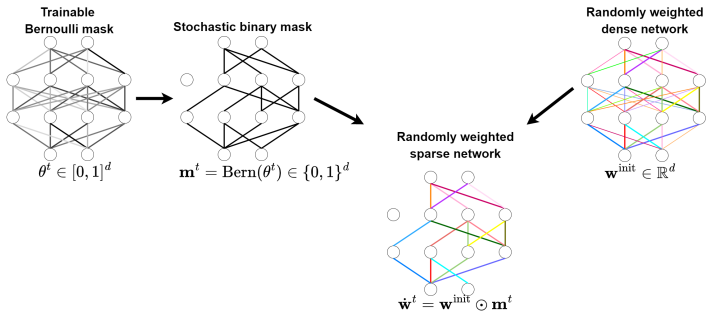

Figure 1: High-level description of extracting a randomly weighted sparse network from the trainable Bernoulli parameters $\boldsymbol{\theta}^t$ for forward-pass at round t (for clients and the server). In practice, clients collaboratively train continuous scores $\boldsymbol{s} \in \mathbb{R}^d$, and then at inference time, the clients (or the server) find $\boldsymbol{\theta}^t = \mathrm{Sigmoid}(\boldsymbol{s}^t) \in [0,1]^d$. We skip this step in the figure for the sake of simplicity.

without ever modifying the weight values. This phenomenon has been explored to some extent in prior work (Zhou et al., 2019; Ramanujan et al., 2020; Pensia et al., 2020; Diffenderfer & Kailkhura, 2020; Aladago & Torresani, 2021) with different strategies for finding the subnetworks. However, how to find these subnetworks in an FL setting has not attracted much attention. Some exceptions to this are works by Li et al. (2021); Vallapuram et al. (2022); Mozaffari et al. (2021), which provide improvements in other FL challenges such as personalization and poisoning attacks, while not being so competitive with existing (dense) compression methods such as QSGD (Alistarh et al., 2017), DRIVE (Vargaftik et al., 2021), and SignSGD (Bernstein et al., 2018) in terms of *accuracy* under the same communication budget. In this work, we propose a *stochastic* way of finding such subnetworks while reaching higher accuracy at a reduced communication cost – less than 1 bit per parameter (bpp).

In addition to the accuracy and communication gains, our framework also provides an efficient representation of the final model post-training by requiring less than 1 bpp to represent (1) the random seed that generates the initial weights $\boldsymbol{w}^{\mathrm{init}}$, and (2) a sampled binary vector $\mathrm{Ber}(\boldsymbol{\theta})$ (computed with the trained $\boldsymbol{\theta}$). Therefore, the final model enjoys a memory-efficient deployment – a crucial feature for machine learning at power-constrained edge devices. Another advantage our framework brings is the privacy amplification under some settings, thanks to the stochastic nature of our training strategy. We show that in the presence of a differential privacy mechanism, such as (Abadi et al., 2016; Andrew et al., 2021), over the Bernoulli mask $\boldsymbol{\theta}$, the Bernoulli sampling step may amplify the privacy further.

We show the efficacy of our approach on MNIST, EMNSIT, CIFAR-10, and CIFAR-100 datasets under both IID and non-IID data splits; and show improvements in accuracy, bitrate, convergence speed, and final model size over relevant baselines. Our contributions can be summarized as follows:

(1) We propose an FL framework where the clients do not train the model weights but instead a stochastic binary mask to be used in sparsifying the randomly weighted dense network. This differs from the standard training approaches in the literature.

(2) Our framework provides efficient communication from clients to the server, requiring (less than) 1 bpp per client, while yielding faster convergence and higher accuracy than the baselines.

(3) We propose a Bayesian aggregation strategy at the server side to better deal with partial participation and non-IID data splits.

(4) The final model (a randomly weighted sparse network) can be efficiently represented with a random seed and a binary mask which requires (less than) 1 bpp – at least $32\times$ more efficient storage and communication of the final model with respect to standard FL strategies.

2 Federated Probabilistic Mask Training (FedPM)

We first describe the simpler version of the FedPM framework in Section 2.1, which provides *unbiased* estimation of the mean of the learned Bernoulli vectors at the server with *bounded error*. Next, we propose a change in our aggregation strategy in Section 2.2. This helps boost the performance of FedPM in non-IID data splits, and in the case of partial participation of clients. We then discuss the details of the distribution of the initial weights in Section 2.3, and due to space limitations, we describe the privacy benefits of FedPM in Appendix F. Throughout the paper, we use capital letters

for random variables, small letters for deterministic quantities, and bold letters to denote vectors. Moreover, we indicate with $\boldsymbol{x}^{u,t}$ the state of the local vector $\boldsymbol{x}$ at client u during round t, and with $x_i^{u,t}$ its i-th component. Global values are denoted with $\boldsymbol{x}^{g,t}$ and $x_i^{g,t}$, and sets are indicated with calligraphic fonts.

2.1 Federated Probabilistic Mask Training (FedPM)

In this section, the general FedPM training pipeline is exposed. First, the server randomly initializes a neural network $f_{\boldsymbol{w}^{\text{init}}}$, parameterized by the weights vector $\boldsymbol{w}^{\text{init}} = (w_1^{\text{init}}, w_2^{\text{init}}, \dots, w_d^{\text{init}}) \in \mathbb{R}^d$, whose components are sampled IID according to a distribution P_w using a random seed SEED. The random seed is then communicated to all the clients, which can locally sample the same pseudo-random vector $\boldsymbol{w}^{\text{init}}$, which is kept fixed and never modified during training. The goal for the clients is to collaboratively train a global stochastic binary mask $\boldsymbol{m} \in \{0,1\}^d$, such that the function $f_{\dot{\boldsymbol{w}}}$ maximizes its performance on a given task, where $\dot{\boldsymbol{w}} = \boldsymbol{m} \odot \boldsymbol{w}^{\text{init}}$.

Specifically, FedPM learns the *probabilities* for the weights of being active, which are given by the Bernoulli mask $\boldsymbol{\theta} = (\theta_1, \theta_2, \dots, \theta_d) \in [0,1]^d$. To achieve this, at every round t, the server samples a set $\mathcal{K}_t$ of $|\mathcal{K}_t| = K$ participants (out of the total N clients), which individually train their local Bernoulli masks $\boldsymbol{\theta}^{k,t}$ $k \in \mathcal{K}_t$, with their local datasets $\mathcal{D}_k$, each composed by $D_k = |\mathcal{D}_k|$ samples. These local masks are then aggregated by the server in a communication-efficient way to estimate the optimal $\boldsymbol{\theta}$. At test time, the initial random network $f_{\boldsymbol{w}^{\text{init}}}$ is sparsified using the global Bernoulli mask $\boldsymbol{\theta}^{g,t}$ following the stochastic approach in Figure 1. In the following sections, we provide more details on the single steps occurring at each round. The pseudocode for FedPM is given in Appendix B.

2.1.1 Local Training of Bernoulli Masks

Upon receiving a global Bernoulli mask $\boldsymbol{\theta}^{g,t-1}$ from the server at round t, the client k performs local training and updates the mask via back-propagation. However, we have to guarantee that the updated Bernoulli mask satisfies $\boldsymbol{\theta}^{k,t} \in [0,1]^d$. While this can be achieved with a regularization term or clipping, these approaches may lead to a performance drop due to information loss. We introduce another mask, called *score mask*, $\boldsymbol{s} = (s_1, s_2, \dots, s_d) \in \mathbb{R}^d$ that has unbounded support and can be used to generate the Bernoulli masks through the one-to-one Sigmoid function as follows: $\boldsymbol{\theta} = \text{Sigmoid}(\boldsymbol{s})$. Then, the procedure for local training of the Bernoulli mask at round t is as follows:

(1) The K clients receive the global Bernoulli mask $\boldsymbol{\theta}^{g,t-1}$ from the server, and set their local score $\boldsymbol{s}^{k,t} = \text{Sigmoid}^{-1}(\boldsymbol{\theta}^{g,t-1})$.

(2) Then, the participants generate a binary mask by transforming back $\boldsymbol{\theta}^{k,t} = \text{Sigmoid}(\boldsymbol{s}^{k,t})$, and by sampling a binary mask $\boldsymbol{m}^{k,t}$ from $\boldsymbol{\theta}^{k,t}$ as shown in Figure 1: $\boldsymbol{m}^{k,t} \sim \text{Bern}(\boldsymbol{\theta}^{k,t})$.

(3) The binary mask then sparsifies the inital weights vector w_{init}: $\dot{\boldsymbol{w}}^{k,t} = \boldsymbol{m}^{k,t} \odot \boldsymbol{w}^{\text{init}}$.

(4) $\dot{\boldsymbol{w}}^{k,t}$ is used for forward pass and the loss $\mathcal{L}$ is backpropagated to update the score mask as follows: $\boldsymbol{s}^{k,t} = \boldsymbol{s}^{k,t} - \eta \nabla \mathcal{L}$, where η is the local learning rate.

Notice that all the local operations from step 2 to step 4 are differentiable, except for Bernoulli sampling. To backpropagate the gradients through the Bernoulli sampling, we use the first-order gradient of the Bernoulli function, which is simply equal to the Bernoulli mask $\boldsymbol{\theta}^{k,t}$. While the steps above (from step 2 to step 4) describe the first iteration in the local training step, the clients perform the same operations for a specific number τ of local iterations, as standard in the FL context.

2.1.2 Communication Strategy

Once the local training at round t is completed, the server needs to distill the global Bernoulli mask $\boldsymbol{\theta}^{g,t}$, for example by taking the empirical average of the local Bernoulli masks $\bar{\boldsymbol{\theta}}^{g,t} = \frac{1}{K}\sum_{k\in\mathcal{K}_t} \boldsymbol{\theta}^{k,t}$ from the clients. However, since we aim for communication efficiency, the clients do not send their local Bernoulli masks directly. Instead, they communicate a *stochastic* binary sample from their Bernoulli masks: $\boldsymbol{m}^{k,t} \sim \text{Bern}(\boldsymbol{\theta}^{k,t})$, and then the server estimates the global aggregate $\bar{\boldsymbol{\theta}}^{g,t}$ as $\hat{\bar{\boldsymbol{\theta}}}^{g,t} = \frac{1}{K}\sum_{k\in\mathcal{K}_t} \boldsymbol{m}^{k,t}$. This distributed mean estimation problem with communication constraints is summarized in Figure 3 in Appendix D. Our estimator $\hat{\bar{\boldsymbol{\theta}}}^{g,t} = \frac{1}{K}\sum_{k\in\mathcal{K}_t} \boldsymbol{m}^{k,t}$ is an unbiased estimate of the true mean, in that $\mathbb{E}_{\boldsymbol{m}\sim\text{Bern}(\boldsymbol{\theta})}[\hat{\bar{\boldsymbol{\theta}}}^{g,t}] = \mathbb{E}_{\boldsymbol{m}\sim\text{Bern}(\boldsymbol{\theta})}[\frac{1}{K}\sum_{k\in\mathcal{K}_t} \boldsymbol{m}^{k,t}] = \frac{1}{K}\sum_{k\in\mathcal{K}_t} \theta^{k,t} = \bar{\theta}^{g,t}$.

Moreover, the estimation error is upper bounded as (the proof is given in Appendix C)

$$\mathbb{E}_{\boldsymbol{m}\sim\text{Bern}(\boldsymbol{\theta})}\left[||\hat{\bar{\boldsymbol{\theta}}}^{g,t} - \bar{\boldsymbol{\theta}}^{g,t}||_2^2\right] \leq \frac{d}{4K}. \tag{1}$$

Since each client communicates a stochastic binary mask $\boldsymbol{m}^{k,t}$, 1 bit per parameter is the worst case bitrate for FedPM. We can reduce the bitrate to less than 1 bit by using arithmetic coding to encode $\boldsymbol{m}^{k,t}$. This gives us smaller bitrates whenever the frequency of 1's in $\boldsymbol{m}^{k,t}$ is far from 0.5 – which is usually the case for our method (see Figure 2 and Appendix H.2 for experimental results). In general, we know the bitrate is a function of the entropy of the mask's coefficients distribution, as $d \to \infty$.

We note that with a deterministic mask training approach as in FedMask (Li et al., 2021), arithmetic coding of $\boldsymbol{m}^{k,t}$'s does not provide any further gain in bitrate, as we have empirically observed that the frequency of 1's is always around 0.5 (see Figure 2 and Appendix H.2) – here we apply arithmetic coding for FedMask to improve our baseline although it was not proposed in the original paper. Moreover, FedMask (Li et al., 2021) and HideNSeek (Vallapuram et al., 2022) do not enjoy the convergence guarantees we have as their estimator (1) is not unbiased and (2) does not have an upper bound on the estimation error due to hard thresholding (Li et al., 2021) and sign operations (Vallapuram et al., 2022). This is another benefit of our stochastic sampling approach.

2.2 FedPM with Bayesian Aggregation

Another important aspect that differentiates our work from existing masking methods such as FedMask and HideNSeek is the Bayesian aggregation strategy, which exploits the underlying *stochastic mask* to synthetize a global model, boosting the performance in heterogeneous scenarios, e.g., local client data are not sampled from the same distribution. Given the probabilistic interpretation of the FedPM mask's values, at the server side we further model the Bernoulli mask parameters $\boldsymbol{\theta}^{g,t}$ with a Beta distribution Beta$(\boldsymbol{\alpha}^{g,t}, \boldsymbol{\beta}^{g,t})$, parameterized by the round-dependent parameters $\boldsymbol{\alpha}^{g,t}$ and $\boldsymbol{\beta}^{g,t}$, which are initialized to $\boldsymbol{\alpha}^{g,0} = \boldsymbol{\beta}^{g,0} = \boldsymbol{\lambda_0}$. At the beginning of the training process, there is no prior knowledge indicating which coefficient should be more important than the others, and so each Bernoulli parameter is uniformly distributed in $[0, 1]$. Consequently, the clients' updates are the *data* the server uses to update its belief on each coefficient score, and so the aggregation strategy corresponds now to a *posterior update*. Specifically, given the conjugate relation between the Beta-Bernoulli distributions, the new posteriors are still Beta distributions with parameters

$$\boldsymbol{\alpha}^{g,t} = \boldsymbol{\alpha}^{g,t-1} + \boldsymbol{m}^{\text{agg},t} \quad \text{and} \quad \boldsymbol{\beta}^{g,t} = \boldsymbol{\beta}^{g,t-1} + K \cdot \mathbf{1} - \boldsymbol{m}^{\text{agg},t} \quad \forall t \geq 1, \tag{2}$$

where $\boldsymbol{m}^{\text{agg},t} = \sum_{k\in\mathcal{K}_t} \boldsymbol{m}^{k,t}$, and $\mathbf{1}$ is the d-dimensional all-ones vector. Then, the server broadcasts to the clients the mode of the Bernoulli distributions, as suggested in Ferreira et al. (2021), i.e., $\boldsymbol{\theta}^{g,t} = \frac{\boldsymbol{\alpha}^{g,t}-1}{\boldsymbol{\alpha}^{g,t}+\boldsymbol{\beta}^{g,t}-2}$, where the division operation is element-wise applied. However, to obtain the best performance out of this method, the Beta parameters should be re-initialized to their original values $\boldsymbol{\lambda}_0$ with some regularity. We observe that, if $\boldsymbol{\lambda}_0 = \mathbf{1}$, and if the parameters are re-initialized at the beginning of each round, the method is equivalent to the aggregation strategy detailed in Section 2.1.2.

2.3 Weights Distribution

As mentioned in Section2.1, the fixed weights vector $\boldsymbol{w}^{\text{init}}$ is randomly initialized by sampling from the distribution P_w. We further observe that the choice of this distribution impacts two important aspects of FedPM: (1) the size of the sample space of P_w affects the number of bits needed to store the model during the inference process and (2) the values of $\boldsymbol{w}^{\text{init}}$ highly influence the final accuracy achieved by the model, as they represent the building blocks to extract a sub-network $f_{\boldsymbol{\dot{w}}}$ (see Figure 1), which should be rich enough to solve the learning task. We provide further details on the selection of P_w and how to find a good balance between (1) and (2) in Appendix E.

3 Experiments

In this section, we empirically show the performance of FedPM in terms of accuracy, bitrate, converge speed, and the final model size. We consider four datasets, CIFAR-10, CIFAR-100 (Krizhevsky et al., 2009), MNIST (Deng, 2012), and EMNIST (Cohen et al., 2017). For CIFAR-100, we use a 10-layer

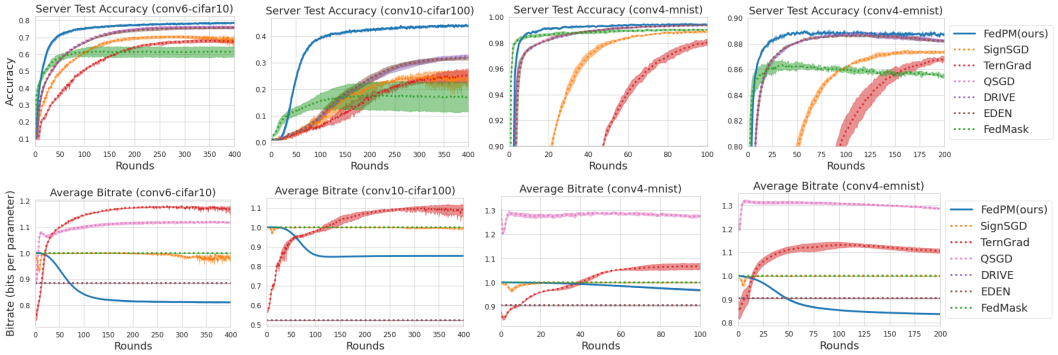

Figure 2: Accuracy (upper) and bitrate (lower) comparison of FedPM with baselines SignSGD (Bernstein et al., 2018), TernGrad (Wen et al., 2017), QSGD (Alistarh et al., 2017), DRIVE (Vargaftik et al., 2021), EDEN (Vargaftik et al., 2022), and FedMask (Li et al., 2021), all performing in the same bitrate regime, on CIFAR-10, CIFAR-100, MNIST, and EMNIST datasets.

convolutional network (CNN); for CIFAR-10 a 6-layer CNN; for MNIST and EMNIST a 4-layer CNN. Detailed descriptions of the architectures can be found in Ramanujan et al. (2020). We compare FedPM with SignSGD (Bernstein et al., 2018), TernGrad (Wen et al., 2017), QSGD (Alistarh et al., 2017), DRIVE (Vargaftik et al., 2021), EDEN (Vargaftik et al., 2022), and FedMask (Li et al., 2021) first on IID data split and full client participation in Section 3.1. We then extend our experiments to non-IID data splits and partial participation in Section 3.2. Clients perform 3 local epochs in all experiments. We present results averaged over 3 runs.

3.1 IID Data Split

In this section, we focus on IID data distribution and the case where all the clients participate in the training process at each round. We set the number of clients to $N = K = 10$. We adopt the aggregation strategy in Section 2.2. Regarding the bitrate calculation, for all the baselines, we compute the entropy of the empirical updates distributions as a proxy for the arithmetic coding. In Figure 2, we compare the accuracy, bitrate, and convergence speed of FedPM with baselines. As can be seen in the figure, FedPM converges to the highest accuracy on all 4 datasets. DRIVE, EDEN, and QSGD (they mostly overlap in the accuracy plots) seem to be the three baselines that perform the best after FedPM; however, their convergence speed is significantly lower than FedPM. In terms of convergence speed, FedMask is the fastest among our baselines – in fact, at the beginning of the training, FedMask is faster than FedPM as well. However, their final accuracy is not as high as others.

In terms of bitrate, SignSGD and FedMask consistently spend 1 bit per parameter, which is the default number when a binary mask or sign mask is communicated. This means binary values (1's and 0's) are almost equally distributed in their masks, which prevents them from enjoying additional bitrate gains. Unlike other baselines, including our work, QSGD can go down to very low bitrates by adjusting the number of quantization levels. In that extreme quantization regime, QSGD still has some reasonable accuracy (but lower than FedPM); however, the performance in that regime (lower bitrate, lower accuracy) is not a meaningful comparison point for FedPM. As can be seen from the plots, even with bitrates larger than 1, QSGD still underperforms FedPM. The only two baselines that challenge FedPM in terms of bitrate are DRIVE and EDEN. While FedPM has lower bitrates on CIFAR-10 and EMNIST, DRIVE and EDEN have better bitrates on CIFAR-100 and MNIST. However, the accuracy of DRIVE and EDEN on these datasets (specifically CIFAR-100) are significantly lower than that of FedPM, with slower convergence.

As to the final model size, FedPM needs only 0.8 bits per parameter (bpp) for the CONV-6 model trained on CIFAR-10, 0.85 bpp for the CONV-10 model trained on CIFAR-100, 0.96 bpp for the CONV-4 model trained on MNIST, and 0.83 bpp for the CONV-4 model trained on EMNIST. On the other hand, our baselines that train a *dense* model, namely SignSGD, TernGrad, QSGD, DRIVE, and EDEN, would need to represent each weight with their full precision value, i.e., 32 bpp. This implies that FedPM provides around $38.6\times$ improvement in the storage or the communication of the final model. Since FedMask also trains a sparse model, it enjoys a similar gain in the final model

size requiring 1 bpp across all the models. Due to the stochastic masking procedure and uneven distribution of 1's and 0's in the binary masks, FedPM has up to 0.17 bpp improvement over the deterministic procedure in FedMask, which adds up to a large gain due to the huge model size.

We also would like to highlight that while some of our baselines, such as FedMask and TernGrad, have a visibly high variance in accuracy, FedPM shows stable training behavior across all experiments.

3.2 Non-IID Data Split and Partial Participation

In this section, we consider the scenario when clients' datasets are generated from slightly different data distributions. We focus on CIFAR-10 and CONV-6 model, and compare FedPM against the most promising baselines, which, based on the results of Section 3.1, are DRIVE, EDEN, QSGD, and FedMask. Due to limited space, we provide the details on how to make the data distribution non-IID and unbalanced across clients in Appendix G. For the experiments in this section, the maximum number of classes per local dataset is set to $c_{\text{max}} = \{2, 4\}$, and the client participation ratio is set to $\rho = \{0.1, 0.5, 1\}$. When $\rho = 1$ and $\rho = 0.5$, the total number of clients is set to $N = 10$, and so K is equal to 10 and 5, respectively. When $\rho = 0.1$, $N = 50$, and so $K = 5$. When $\rho = 1$, for the FedPM algorithm, we keep the same aggregation strategy exposed in Section 2.1.2, and we switch to the Bayesian aggregation method with partial participation (see Section 2.2) when $\rho < 1$. The Bayesian aggregation method is revealed to be crucial for achieving good accuracy when $\rho < 1$ and data is non-IID, obtaining a large gain with respect to the simpler version. We adopt a simple heuristic schedule to reset the priors: Reset every 3 rounds when $\rho = 0.5$, and every 10 rounds when $\rho = 0.1$. As can be seen, the smaller the ratio ρ, the larger the number of rounds we should wait before resetting the priors, in order to collect information from a much more diverse pool of clients.

Avg. Accuracy - $c_{\text{max}} = 4$			
Algorithm	$\boldsymbol{\rho = 1}$	$\boldsymbol{\rho = 0.5}$	$\boldsymbol{\rho = 0.1}$
DRIVE (Vargaftik et al., 2021)	0.739 ± 0.005	0.632 ± 0.010	0.405 ± 0.018
EDEN (Vargaftik et al., 2022)	0.717 ± 0.006	0.665 ± 0.012	0.36 ± 0.016
QSGD (Alistarh et al., 2017)	0.709 ± 0.006	0.644 ± 0.014	0.399 ± 0.02
FedMask (Li et al., 2021)	0.531 ± 0.044	0.435 ± 0.057	0.362 ± 0.024
FedPM (Ours)	$\mathbf{0.748 \pm 0.003}$	$\mathbf{0.72 \pm 0.007}$	$\mathbf{0.496 \pm 0.007}$

Table 1: Average final accuracy $\pm\sigma$ in non-IID data split, when $c_{\text{max}} = 4$, and $\rho = \{0.1, 0.5, 1\}$, for FedPM, FedMask, and the strongest baselines in the IID experiments: EDEN, DRIVE, and QSGD. The training process duration was set to $t_{\text{max}} = 200$ rounds.

Table 1 reports the results when $c_{\text{max}} = 4$ (see Appendix H.1 for $c_{\text{max}} = 2$). As we can see, FedPM outperforms the baselines in every system configuration, as the Bayesian aggregation updates allow the central node to collect more data before resetting the priors, which is important when clients' data distributions are heterogeneous. This strategy can be seen as the FedPM counterpart of decreasing the learning rate (which we applied) in the other *dense* compression baselines, such as DRIVE, EDEN, and QSGD. Notice that FedMask is struggling in non-IID cases, as applying a hard threshold on the scores to binarize the mask does not provide a proper way to implement multiple-round aggregation, emphasizing the benefit of the stochastic process underlying FedPM. However, further features and implementations may close this gap even for threshold-based methods. It is interesting to notice that, especially when $c_{\text{max}} = 4$, the lower the value of ρ, the larger the gap between FedPM and the other baselines, corroborating the fact that the Bayesian strategy can better deal with partial participation. Additional results and further analysis in communication efficiency is provided in Appendix H.2.

4 Conclusion

In this work, we introduced Federated Probabilistic Mask Training (FedPM) – a communication-efficient FL strategy. FedPM relies on the idea of building a model on top of a randomly weighted network, which is then sparsified by a collaboratively trained *stochastic* binary mask. In addition to reducing the communication cost to ≤ 1 bit per parameter (bpp), FedPM also reaches higher accuracy with faster convergence than the relevant baselines, while additionally outputting a compressed final model with size less than 1 bpp. Due to limited space, we discuss the related work in Appendix A.

References

Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 308–318, 2016.

Naman Agarwal, Peter Kairouz, and Ziyu Liu. The skellam mechanism for differentially private federated learning. *Advances in Neural Information Processing Systems*, 34:5052–5064, 2021.

Alham Aji and Kenneth Heafield. Sparse communication for distributed gradient descent. In *EMNLP 2017: Conference on Empirical Methods in Natural Language Processing*, pp. 440–445. Association for Computational Linguistics (ACL), 2017.

Maxwell M Aladago and Lorenzo Torresani. Slot machines: Discovering winning combinations of random weights in neural networks. In *International Conference on Machine Learning*, pp. 163–174. PMLR, 2021.

Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in neural information processing systems*, 30, 2017.

Galen Andrew, Om Thakkar, Brendan McMahan, and Swaroop Ramaswamy. Differentially private learning with adaptive clipping. *Advances in Neural Information Processing Systems*, 34:17455–17466, 2021.

Sara Babakniya, Souvik Kundu, Saurav Prakash, Yue Niu, and Salman Avestimehr. Federated sparse training: Lottery aware model compression for resource constrained edge. *arXiv preprint arXiv:2208.13092*, 2022.

Leighton Pate Barnes, Huseyin A Inan, Berivan Isik, and Ayfer Özgür. rtop-k: A statistical estimation approach to distributed sgd. *IEEE Journal on Selected Areas in Information Theory*, 1(3):897–907, 2020.

Ran Ben Basat, Shay Vargaftik, Amit Portnoy, Gil Einziger, Yaniv Ben-Itzhak, and Michael Mitzenmacher. Quick-fl: Quick unbiased compression for federated learning. *arXiv preprint arXiv:2205.13341*, 2022.

Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signsgd: Compressed optimisation for non-convex problems. In *International Conference on Machine Learning*, pp. 560–569. PMLR, 2018.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pp. 2921–2926. IEEE, 2017.

Rong Dai, Li Shen, Fengxiang He, Xinmei Tian, and Dacheng Tao. Dispfl: Towards communication-efficient personalized federated learning via decentralized sparse training. *arXiv preprint arXiv:2206.00187*, 2022.

Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

James Diffenderfer and Bhavya Kailkhura. Multi-prize lottery ticket hypothesis: Finding accurate binary neural networks by pruning a randomly weighted network. In *International Conference on Learning Representations*, 2020.

Paulo Abelha Ferreira, Pablo Nascimento da Silva, Vinicius Gottin, Roberto Stelling, and Tiago Calmon. Bayesian signsgd optimizer for federated learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2018.

Farzin Haddadpour, Belhal Karimi, Ping Li, and Xiaoyun Li. Fedsketch: Communication-efficient and private federated learning via sketching. *arXiv preprint arXiv:2008.04975*, 2020.

Farzin Haddadpour, Mohammad Mahdi Kamani, Aryan Mokhtari, and Mehrdad Mahdavi. Federated learning with compression: Unified analysis and sharp guarantees. In *International Conference on Artificial Intelligence and Statistics*, pp. 2350–2358. PMLR, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, 2015. doi: 10.1109/ICCV.2015.123.

Jacob Imola and Kamalika Chaudhuri. Privacy amplification via bernoulli sampling. *arXiv preprint arXiv:2105.10594*, 2021.

Berivan Isik, Tsachy Weissman, and Albert No. An information-theoretic justification for model pruning. In *International Conference on Artificial Intelligence and Statistics*, pp. 3821–3846. PMLR, 2022.

Shaoxiong Ji, Wenqi Jiang, Anwar Walid, and Xue Li. Dynamic sampling and selective masking for communication-efficient federated learning. *arXiv preprint arXiv:2003.09603*, 2020.

Yuang Jiang, Shiqiang Wang, Victor Valls, Bong Jun Ko, Wei-Han Lee, Kin K Leung, and Leandros Tassiulas. Model pruning enables efficient federated learning on edge devices. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

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

Ang Li, Jingwei Sun, Binghui Wang, Lin Duan, Sicheng Li, Yiran Chen, and Hai Li. Lotteryfl: Personalized and communication-efficient federated learning with lottery ticket hypothesis on non-iid datasets. *arXiv preprint arXiv:2008.03371*, 2020.

Ang Li, Jingwei Sun, Xiao Zeng, Mi Zhang, Hai Li, and Yiran Chen. Fedmask: Joint computation and communication-efficient personalized federated learning via heterogeneous masking. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, pp. 42–55, 2021.

Yujun Lin, Song Han, Huizi Mao, Yu Wang, and Bill Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *International Conference on Learning Representations*, 2018.

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.

Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th computer security foundations symposium (CSF)*, pp. 263–275. IEEE, 2017.

Nicole Mitchell, Johannes Ballé, Zachary Charles, and Jakub Konečnỳ. Optimizing the communication-accuracy trade-off in federated learning with rate-distortion theory. *arXiv preprint arXiv:2201.02664*, 2022.

Hamid Mozaffari, Virat Shejwalkar, and Amir Houmansadr. Frl: Federated rank learning. *arXiv preprint arXiv:2110.04350*, 2021.

Ankit Pensia, Shashank Rajput, Alliot Nagle, Harit Vishwakarma, and Dimitris Papailiopoulos. Optimal lottery tickets via subset sum: Logarithmic over-parameterization is sufficient. *Advances in Neural Information Processing Systems*, 33:2599–2610, 2020.

Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11893–11902, 2020.

Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *International Conference on Artificial Intelligence and Statistics*, pp. 2021–2031. PMLR, 2020.

Daniel Rothchild, Ashwinee Panda, Enayat Ullah, Nikita Ivkin, Ion Stoica, Vladimir Braverman, Joseph Gonzalez, and Raman Arora. Fetchsgd: Communication-efficient federated learning with sketching. In *International Conference on Machine Learning*, pp. 8253–8265. PMLR, 2020.

Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Robust and communication-efficient federated learning from non-iid data. *IEEE transactions on neural networks and learning systems*, 31(9):3400–3413, 2019.

Sejin Seo, Seung-Woo Ko, Jihong Park, Seong-Lyun Kim, and Mehdi Bennis. Communication-efficient and personalized federated lottery ticket learning. In *2021 IEEE 22nd International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pp. 581–585. IEEE, 2021.

Ananda Theertha Suresh, X Yu Felix, Sanjiv Kumar, and H Brendan McMahan. Distributed mean estimation with limited communication. In *International conference on machine learning*, pp. 3329–3337. PMLR, 2017.

Anish K Vallapuram, Pengyuan Zhou, Young D Kwon, Lik Hang Lee, Hengwei Xu, and Pan Hui. Hidenseek: Federated lottery ticket via server-side pruning and sign supermask. *arXiv preprint arXiv:2206.04385*, 2022.

Shay Vargaftik, Ran Ben-Basat, Amit Portnoy, Gal Mendelson, Yaniv Ben-Itzhak, and Michael Mitzenmacher. Drive: one-bit distributed mean estimation. *Advances in Neural Information Processing Systems*, 34:362–377, 2021.

Shay Vargaftik, Ran Ben Basat, Amit Portnoy, Gal Mendelson, Yaniv Ben Itzhak, and Michael Mitzenmacher. Eden: Communication-efficient and robust distributed mean estimation for federated learning. In *International Conference on Machine Learning*, pp. 21984–22014. PMLR, 2022.

Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. Powersgd: Practical low-rank gradient compression for distributed optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

Hongyi Wang, Scott Sievert, Shengchao Liu, Zachary Charles, Dimitris Papailiopoulos, and Stephen Wright. Atomo: Communication-efficient learning via atomic sparsification. *Advances in Neural Information Processing Systems*, 31, 2018.

Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. *Advances in neural information processing systems*, 30, 2017.

Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. *Advances in neural information processing systems*, 32, 2019.

A Related Work

In this section, we briefly discuss the related work in (1) communication-efficient federated learning, (2) pruning for federated learning, and (3) finding subnetworks in a random network.

Communication-Efficient Federated Learning. One way of improving communication efficiency in FL is to compress the model updates using gradient compression methods like sparsification (Aji & Heafield, 2017; Lin et al., 2018; Wang et al., 2018; Barnes et al., 2020; Isik et al., 2022), quantization (Alistarh et al., 2017; Wen et al., 2017; Suresh et al., 2017; Bernstein et al., 2018; Mitchell et al., 2022), and low-rank approximation (Wang et al., 2018; Vogels et al., 2019; Vargaftik et al., 2021, 2022; Basat et al., 2022); or more FL-oriented compression schemes such as (Konečnỳ et al., 2016; McMahan et al., 2017; Sattler et al., 2019; Rothchild et al., 2020; Reisizadeh et al., 2020; Haddadpour et al., 2020, 2021), while training a *dense* network. Our framework, FedPM, differs from these *dense* compression methods substantially due to the unconventional stochastic mask training strategy; however, we take SignSGD (Bernstein et al., 2018), TernGrad (Wen et al., 2017), QSGD (Alistarh et al., 2017), DRIVE (Vargaftik et al., 2021), and EDEN (Vargaftik et al., 2022) as our baselines since they work in the same bitrate regime as ours – around 1 bit per parameter.

Pruning for Federated Learning. Since the explosion of the Lottery Ticket Hypothesis (LTH) (Frankle & Carbin, 2018), there has been growing interest in finding sparse and *trainable* networks at initialization. The main hypothesis in this line of work is that there exist sparse networks (lottery tickets) inside randomly initialized dense networks such that those sparse networks can be *trained* to a surprisingly well performance – sometimes comparable if not higher to the performance of the trained dense network. In the original paper, the strategy for finding these lottery tickets is to first iteratively train the dense network, i.e., finding the lottery tickets is expensive. We distinguish our approach from the FL papers that utilize the LTH (Li et al., 2020; Ji et al., 2020; Seo et al., 2021) and pruning (Jiang et al., 2022; Babakniya et al., 2022; Dai et al., 2022) for three mains reasons: (1) These methods require training the weight values, and thus cannot provide an efficient representation of the final model as FedPM does. Recall that we achieve at least $32\times$ more efficient storage and communication of the final model by representing it with just a random seed and a binary mask. (2) Some of these works require finding the lottery tickets prior to FL training (Li et al., 2020). While this could improve the communication cost during the FL training, since they just communicate sparse networks, it increases the computation cost significantly, due to the burden of finding lottery tickets via training. (3) As opposed to the LTH- or pruning-based FL works, our framework learns with what probability a particular weight should stay in the final model, i.e., the final sparsity level is also a learned parameter optimized for the best performance. Overall, since FedPM is not in the same bitrate regime as these works (they require higher bitrates to communicate continuous weight values), we do not compare against them.

Finding Subnetworks Inside a Random Network. Our work is closest to recent works of Zhou et al. (2019); Ramanujan et al. (2020); Pensia et al. (2020); Aladago & Torresani (2021) as they find subnetworks (or supermasks) inside a *randomly weighted* dense network that perform surprisingly well without ever training the weights, but in a centralized scenario. In this work, we take advantage of the existence of such subnetworks to **reduce the communication budget** in FL to **less than 1 bit per parameter** with **faster convergence** and **higher accuracy** than our relevant baselines in the same bitrate regime, while **further compressing the final model**, all simultaneously. Prior work (Li et al., 2021; Vallapuram et al., 2022; Mozaffari et al., 2021) in finding subnetworks inside a randomly weighted network in an FL setting differs from our approach on several levels. First of all, they focus on different challenges in FL, such as personalization and poison attacking, which limits their ability to improve over existing compression methods in accuracy-communication bitrate tradeoff. One fundamental reason for this is their deterministic mask training strategy, which involves hard thresholding or sign operations. On the other hand, the stochasticity in FedPM allows us to (1) enjoy a better accuracy-communication cost tradeoff, (2) have an unbiased estimate of the true aggregate of the local mask with a provable upper bound on the error, (3) design an improved aggregation strategy with a Bayesian approach so that the previous masks at the server are not hard replaced – a useful strategy specifically in unbalanced non-IID splits, and (4) gain privacy benefits via amplification in the Bernoulli sampling step. To demonstrate these benefits over deterministic schemes, we compare our method with FedMask (Li et al., 2021) (a *sparse* baseline) by adapting it slightly to make it a strategy that purely aims communication efficiency, and not personalization, to

improve its accuracy-communication efficiency performance. More specifically, we discard the initial pruning stage that was deployed for personalization. This change was necessary because (a) this paper does not study personalization, so this pruning step would put FedMask in a disadvantageous position in our experiment setup, (b) the initial pruning step requires extra training at client devices, which is computationally more expensive than FedPM and the *dense* baselines.

B FedPM Algorithm

We provide the pseudocode for FedPM in Algorithm 1. In Algorithm 2, the priors resetting scheduling policy is controlled by the procedure *ResPrior(t)*, which may depend on other quantities other than the round number t, e.g., loss.

Algorithm 1 FedPM.

Hyperparameters: learning rate η, minibatch size $\mathcal{B}$, number of local iterations τ.
Inputs: local datasets $\mathcal{D}_i$, $i = 1, \ldots, N$
Output: random seed SEED and stochastic binary mask parameters $\boldsymbol{m}^{g,T}$

Initialize a random dense network with weight vector $\boldsymbol{w}^{\text{init}} \in \mathbb{R}$ using SEED.
Initialize a Beta priors $\boldsymbol{\alpha}^{g,0} = \boldsymbol{\beta}^{g,0} = \boldsymbol{\lambda}_0$ at the server.
Broadcast randomly initialized weights $\boldsymbol{w}^{\text{init}}$ and initialized Bernoulli mask $\boldsymbol{\theta}^{g,0}$ to all nodes.
for $t = 0, \ldots, T-1$ **do**
  Sample a subset $\mathcal{K} \subset [N]$ of $|\mathcal{K}| = K$ clients without replacement.
  On Client Nodes:
  for $k \in \mathcal{K}$ **do**
    Receive $\boldsymbol{\theta}^{g,t-1}$ from the server and set $\boldsymbol{s}^{k,t} = \text{Sigmoid}^{-1}(\boldsymbol{\theta}^{g,t-1})$.
    for $l = 1, \ldots, \tau$ **do**
      $\boldsymbol{\theta}^{k,t} \leftarrow \text{Sigmoid}(\boldsymbol{s}^{k,t})$
      Sample binary mask $\boldsymbol{m}^{k,t} \sim \text{Bern}(\boldsymbol{\theta}^{k,t})$.
      $\dot{\boldsymbol{w}}^{k,t} \leftarrow \boldsymbol{w}^{\text{init}} \odot \boldsymbol{m}^{k,t}$
      $\text{grad}_{\boldsymbol{s}^{k,t}} \leftarrow \frac{1}{\mathcal{B}} \sum_{b=1}^{\mathcal{B}} \nabla \ell(\dot{\boldsymbol{w}}^{k,t}; B_j^k)$; $\{B_j^k\}_{j=1}^{\mathcal{B}}$ is uniformly chosen from $\mathcal{D}_k$
      $\boldsymbol{s}^{k,t} \leftarrow \boldsymbol{s}^{k,t} - \eta \cdot \text{grad}_{\boldsymbol{s}^{k,t}}$
    end for
    $\boldsymbol{\theta}^{k,t} \leftarrow \text{Sigmoid}(\boldsymbol{s}^{k,t})$
    Sample a binary mask $\boldsymbol{m}^{k,t} \sim \text{Bern}(\boldsymbol{\theta}^{k,t})$.
    Send the arithmetic coded binary mask $\boldsymbol{m}^{k,t}$ to the server.
  end for

  On the Server Node:
  Receive $\boldsymbol{m}^{k,t}$'s from K client nodes and decode it.
  $\boldsymbol{\theta}^{g,t} = \text{BayesAgg}\big(\{\boldsymbol{m}^{k,t}\}_{k \in \mathcal{K}_t}, t\big)$
  Broadcast $\boldsymbol{\theta}^{g,t}$ to all client nodes.
end for
Sample the final binary mask $\boldsymbol{m}^{\text{final}} \sim \text{Bern}(\boldsymbol{\theta}^{g,T})$.
Generate the final model: $\dot{\boldsymbol{w}}^{\text{final}} \leftarrow \boldsymbol{w}^{\text{init}} \odot \boldsymbol{m}^{\text{final}}$.

C Proof of the Upper Bound on the Estimation Error

We now provide proof of the upper bound on the estimation error in Eq. 1. Recall that our true mean is $\bar{\boldsymbol{\theta}}^{g,t} = \frac{1}{K} \sum_{k \in \mathcal{K}_t} \boldsymbol{\theta}^{k,t}$ whereas our estimate is $\hat{\bar{\boldsymbol{\theta}}}^{g,t} = \frac{1}{K} \sum_{k \in \mathcal{K}_t} \boldsymbol{m}^{k,t}$, where $\boldsymbol{m}^{k,t} \sim \text{Bern}(\boldsymbol{\theta}^{k,t})$. Then we can compute the error as

Algorithm 2 BayesAgg.

Inputs: clients' updates $\{\boldsymbol{m}^{k,t}\}_{k\in\mathcal{K}_t}$, and round number t
Output: global Bernoulli mask $\boldsymbol{\theta}^{g,t}$

if ResPriors(t) **then**
  $\boldsymbol{\alpha}^{g,t-1} = \boldsymbol{\beta}^{g,t-1} = \boldsymbol{\lambda}_0$
end if
$\boldsymbol{\alpha}^{g,t} = \boldsymbol{\alpha}^{g,t-1} + \boldsymbol{m}^{\text{agg},t}$
$\boldsymbol{\beta}^{g,t} = \boldsymbol{\beta}^{g,t-1} + K \cdot \mathbf{1} - \boldsymbol{m}^{\text{agg},t}$
$\boldsymbol{\theta}^{g,t} = \frac{\boldsymbol{\alpha}^{g,t}-1}{\boldsymbol{\alpha}^{g,t}+\boldsymbol{\beta}^{g,t}-2}$
return $\boldsymbol{\theta}^{g,t}$

$$
\begin{aligned}
\mathbb{E}_{\boldsymbol{m}\sim\text{Bern}(\boldsymbol{\theta})}\left[||\hat{\bar{\boldsymbol{\theta}}}^{g,t} - \bar{\boldsymbol{\theta}}^{g,t}||_2^2\right] &= \sum_{i=1}^{d} \mathbb{E}_{\boldsymbol{m}_i\sim\text{Bern}(\boldsymbol{\theta}_i)}\left[\left(\hat{\bar{\boldsymbol{\theta}}}_i^{g,t} - \bar{\boldsymbol{\theta}}_i^{g,t}\right)^2\right] && (3)\\
&= \sum_{i=1}^{d} \mathbb{E}_{\boldsymbol{m}_i\sim\text{Bern}(\boldsymbol{\theta}_i)}\left[\left(\frac{1}{K}\sum_{k\in\mathcal{K}_t}(\boldsymbol{m}_i^{k,t} - \boldsymbol{\theta}_i^{k,t})\right)^2\right] && (4)\\
&= \frac{1}{K^2}\sum_{i=1}^{d} \mathbb{E}_{\boldsymbol{m}_i\sim\text{Bern}(\boldsymbol{\theta}_i)}\left[\left(\sum_{k\in\mathcal{K}_t}(\boldsymbol{m}_i^{k,t} - \boldsymbol{\theta}_i^{k,t})\right)^2\right] && (5)\\
&= \frac{1}{K^2}\sum_{i=1}^{d} \mathbb{E}_{\boldsymbol{m}_i\sim\text{Bern}(\boldsymbol{\theta}_i)}\left[\sum_{k\in\mathcal{K}_t}\left(\boldsymbol{m}_i^{k,t} - \boldsymbol{\theta}_i^{k,t}\right)^2\right] && (6)\\
&= \frac{1}{K^2}\sum_{i=1}^{d}\sum_{k\in\mathcal{K}_t} \mathbb{E}_{\boldsymbol{m}_i\sim\text{Bern}(\boldsymbol{\theta}_i)}\left[(\boldsymbol{m}_i^{k,t} - \boldsymbol{\theta}_i^{k,t})^2\right] && (7)\\
&= \frac{1}{K^2}\sum_{i=1}^{d}\sum_{k\in\mathcal{K}_t}\left(\mathbb{E}_{\boldsymbol{m}_i\sim\text{Bern}(\boldsymbol{\theta}_i)}[(\boldsymbol{m}_i^{k,t})^2] - (\boldsymbol{\theta}_i^{k,t})^2\right) && (8)\\
&= \frac{1}{K^2}\sum_{i=1}^{d}\sum_{k\in\mathcal{K}_t}\left(\boldsymbol{\theta}_i^{k,t} - (\boldsymbol{\theta}_i^{k,t})^2\right) && (9)\\
&\leq \frac{d}{4K}. && (10)
\end{aligned}
$$

From (3) to (4), we use the definition of $\hat{\bar{\boldsymbol{\theta}}}^{g,t} = \frac{1}{K}\sum_{k=1}^{K}\boldsymbol{m}_i^{k,t}$ and $\bar{\boldsymbol{\theta}}_i^{g,t} = \frac{1}{K}\sum_{k=1}^{K}\boldsymbol{\theta}_i^{k,t}$. From (5) to (6), we use the fact that $\mathbb{E}_{\boldsymbol{m}_i\sim\text{Bern}(\boldsymbol{\theta}_i)}[\boldsymbol{m}_i^{k,t} - \boldsymbol{\theta}_i^{k,t}] = 0$; and $\boldsymbol{m}_i^{k,t} - \boldsymbol{\theta}_i^{k,t}$ and $\boldsymbol{m}_i^{l,t} - \boldsymbol{\theta}_i^{l,t}$ are independent for $l \neq k \in [K]$. Finally, the inequality in (6) follows from $\boldsymbol{\theta}^{k,t} \in [0,1]$ for all $k \in [K]$.

D Communication Strategy in FedPM

Figure 3 shows the diagram of the distributed mean estimation problem with communication constraints, considered in FedPM.

E Weights Distribution

As mentioned in Section2.1, the fixed weights vector $\boldsymbol{w}^{\text{init}}$ is randomly initialized by sampling from the distribution P_w. We further observe that the choice of this distribution impacts two important aspects of FedPM: (1) the size of the sample space of P_w affects the number of bits needed to store the model during the inference process and (2) the values of $\boldsymbol{w}^{\text{init}}$ highly influence the final

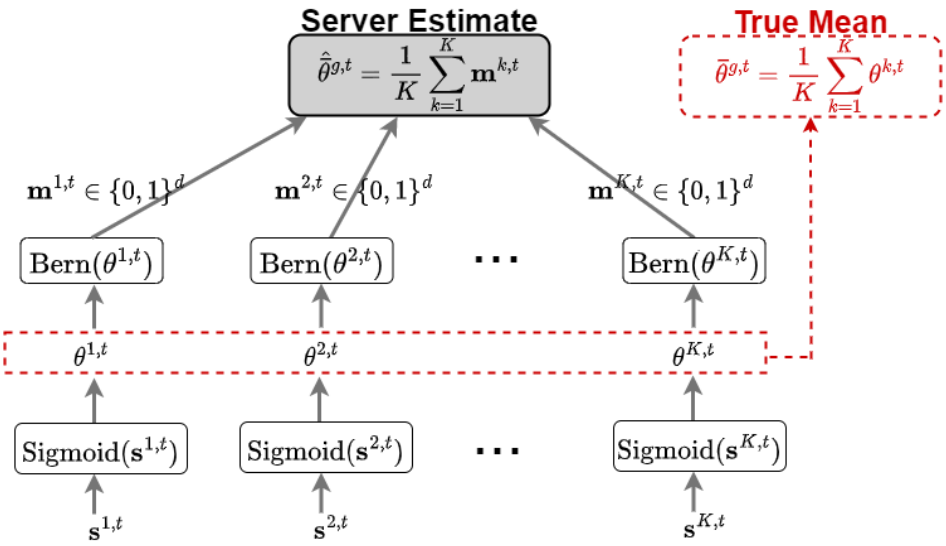

Figure 3: Communication-efficient estimation of the mean of the Bernoulli masks $\bar{\boldsymbol{\theta}}^{g,t}$. Each client communicates a stochastic binary mask $\boldsymbol{m}^{k,t}$ sampled from the local Bernoulli mask $\boldsymbol{\theta}^{k,t}$. We reduce the bitrate to less than 1 bit per parameter by using arithmetic coding to encode $\boldsymbol{m}^{k,t}$. When the frequency of 1's is far from 0.5 (which is usually the case with FedPM), the number of bits per parameter to communicate $\boldsymbol{m}^{k,t}$ is less than 1. See Figure 2 for more details.

accuracy achieved by the model, as they represent the building blocks to extract a sub-network $f_{\dot{\boldsymbol{w}}}$ (see Figure 1), which should be rich enough to solve the learning task. Regarding (1), we observe that, even if knowing the value of SEED is enough to perfectly reconstruct the vector $\boldsymbol{w}^{\text{init}}$, one would have to re-sample the entire vector at every inference step. Consequently, to achieve fast inference, the actual values of the weights may need to be stored in the memory of the devices for the duration of applications. Regarding (2), as also proposed in Ramanujan et al. (2020), we adopt to sample weights from a uniform distribution, whose domain is $\{-\sigma, +\sigma\}$, where σ is the standard deviation of the Kaiming Normal distribution (He et al., 2015). In this way, we can both control the variance of the neurons' output to be ~ 1 (which avoids the vanishing or the explosion of neurons' activation values), and store the values using just 1 bit per weight, being a binary distribution. Previous experiments in Zhou et al. (2019); Ramanujan et al. (2020) also demonstrated the superior performance achieved by binary weights distributions when compared to standard continuous counterparts, e.g., Gaussian.

F Privacy

Differential privacy (DP) is another challenge in FL as the model updates (or, in our case, the stochastic binary masks $\boldsymbol{m}^{k,t}$) may leak information about the client data. While there have been many recent DP strategies developed for FL (Agarwal et al., 2021; Andrew et al., 2021), or deep learning in general (Abadi et al., 2016), these strategies typically suffer from severe performance degradation due to perturbed model updates. To make DP practical, researchers have explored certain randomization mechanisms that amplify privacy. When these mechanisms are part of the FL framework, such as sampling and shuffling, the amplification comes for free, i.e., without an additional performance drop. FedPM promises one such amplification due to the stochastic Bernoulli step. In particular, Imola & Chaudhuri (2021) have shown that when a sample $\boldsymbol{m} \in \{0,1\}^d$ from an already privatized vector $\boldsymbol{\theta} \in [c, 1-c]^d$, where $0 < c < 0.5$, is released to a third party (instead of $\boldsymbol{\theta}$ itself), the privacy is amplified under some conditions. More precisely, when there is an (α, ϵ)-Rényi Differential Privacy (RDP) mechanism (Mironov, 2017) that privatizes $\boldsymbol{\theta} \in [c, 1-c]$ with $0 < c < 0.5$, releasing a sample from $\text{Ber}(\boldsymbol{\theta})$ yields an improved privacy budget (the smaller ϵ, the better privacy): $\epsilon_{amp} \leq \min\{\epsilon, d \cdot r_\alpha(c)\}$. Here, $r_\alpha(p)$ is the binary symmetric Rényi divergence function defined as $r_\alpha(p) = \frac{1}{\alpha-1}\log\left(p^\alpha(1-p)^{1-\alpha} + (1-p)^\alpha p^{1-\alpha}\right)$. Notice that FedPM already

involves this step in the communication protocol on client side and in generating the final mask $\boldsymbol{m}^{g,\text{final}} = \text{Ber}(\boldsymbol{\theta}^{g,\text{final}})$ at the end of the training on server's side. However, the d term in the upper bound limits the amplification for large model sizes. We believe it is worth working on deriving a tighter upper bound on ϵ_{amp} to enjoy the privacy amplification in FedPM with practical models. Nonetheless, in Appendix F.1, we demonstrate the impact of this amplification on a distributed mean estimation problem, described in Figure 3, where the goal is to estimate the true mean of the Bernoulli masks $\bar{\boldsymbol{\theta}} = \frac{1}{K}\sum_{k=1}^{K} \boldsymbol{\theta^k}$ under communication and privacy constraints. We also provide a bias correction mechanism, specific to our scheme in Figure 1, that mitigates the bias due to the DP mechanism we adopt, and that reduces the estimation error.

F.1 Privacy Amplification and Bias Correction

To demonstrate the privacy gains due to the inherent sampling mechanism in FedPM, we consider a simple differential privacy (DP) strategy as shown in Figure 4, where the Bernoulli masks $\boldsymbol{\theta} \in [c, 1-c]^d$ are first corrupted by Gaussian noise, and then clipped to a pre-specified range $[c, 1-c]$. Our goal is, as before, to estimate the true mean $\bar{\boldsymbol{\theta}} = \frac{1}{K}\sum_{k\in\mathcal{K}_t} \boldsymbol{\theta}^k$ by averaging the sampled binary masks, i.e., $\hat{\bar{\boldsymbol{\theta}}} = \frac{1}{K}\sum_{k\in\mathcal{K}_t} \boldsymbol{m}^k$. Differently from our previous experiments, we have privacy constraints now, meaning that we want to guarantee (ϵ, δ)-DP by injecting a Gaussian noise with variance $\sigma^2 = \frac{2\ln(1.25/\delta)\Delta_2^2}{\epsilon^2}$ with a small ϵ, where $\delta \approx \frac{1}{N^2}$ and Δ_2^2 is the ℓ_2-sensitivity of the Bernoulli masks (in our case $\Delta = (1-2c)\sqrt{d}$).

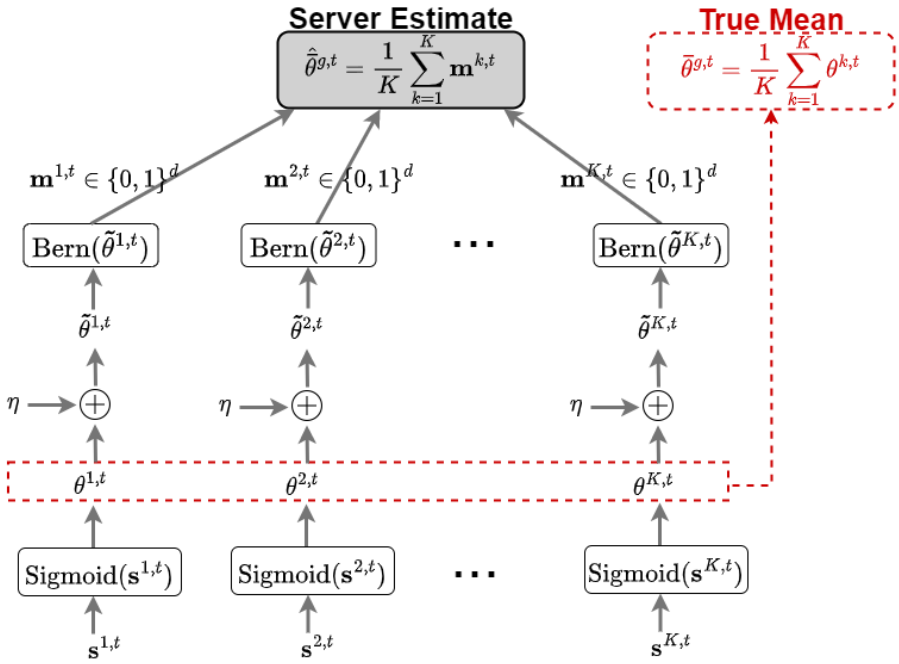

Figure 4: Distributed mean estimation scheme in FedPM, modified for differential privacy.

Since clipping after the noise addition step would lead to bias in the estimated mean, we work out a bias correction mechanism. We denote with θ one general parameter at client k for one parameter, with $\tilde{\theta}$ the noisy version and with $\hat{\theta} = \text{clip}(\tilde{\theta})$ its clipped version. Specifically, if $\tilde{\theta} = \theta + \eta$ is the noisy version of the parameter, where $\eta \sim \mathcal{N}(0, \sigma^2)$, then

$$\text{clip}(\tilde{\theta}) = \begin{cases} \tilde{\theta}, & \text{if} \quad c \leq \theta + \eta \leq 1-c \\ 1-c, & \text{if} \quad \theta + \eta > 1-c \\ c, & \text{if} \quad \theta + \eta < c. \end{cases} \tag{11}$$

We now compute $\mathbb{E}[\hat{m}]$, where $\hat{m} \sim \text{Bern}(\hat{\theta})$, to analyze the bias $\mathbb{E}[\hat{m}] - \mathbb{E}[m] = \mathbb{E}[\hat{m}] - \theta$, where $m \sim \text{Bern}(\theta)$. First of all, notice that

$$\mathbb{E}[\hat{m}] = \int_0^1 \mathbb{E}\left[\hat{m}|\hat{\theta} = \rho\right] f(\rho)d\rho = \int_0^1 \rho f(\rho)d\rho = \mathbb{E}[\hat{\theta}].$$

And we now compute the mean of the clipped parameter

$$\begin{aligned}
\mathbb{E}\left[\hat{\theta}\right] &= \int_0^1 \rho f(\rho)d\rho = \int_{-\infty}^{+\infty} \text{clip}(\theta+\eta)f(\eta)d\eta \\
&= \int_{-\infty}^{c-\theta} c \cdot f(\eta)d\eta + \int_{c-\theta}^{1-c-\theta} (\theta+\eta)\cdot f(\eta)d\eta + \int_{1-c-\theta}^{+\infty} (1-c)\cdot f(\eta)d\eta \\
&= c\Phi_\sigma(c-\theta) + \theta\int_{c-\theta}^{1-c-\theta} f(\eta)d\eta + \int_{c-\theta}^{1-c-\theta} \eta f(\eta)d\eta + 1 - \Phi_\sigma(1-c-\theta) \\
&= c\Phi_\sigma(c-\theta) + \theta\left[\Phi_\sigma(1-c-\theta) - \Phi_\sigma(c-\theta)\right] - \frac{-\sigma}{2\sqrt{2\pi}}\left[e^{\frac{-(1-c-\theta)^2}{2\sigma^2}} - e^{\frac{-(c-\theta)^2}{2\sigma^2}}\right] + \\
&\quad + 1 - \Phi_\sigma(1-c-\theta) \\
&= 1 + [\theta-1]\Phi_\sigma(1-c-\theta) + [c-\theta]\Phi_\sigma(c-\theta) - \frac{-\sigma e^{\frac{-(c-\theta)^2}{2\sigma^2}}}{2\sqrt{2\pi}}\left[e^{2(c-\theta)-1} - 1\right],
\end{aligned}$$

where $\Phi_\sigma(\cdot)$ is the cumulative distribution function of a Gaussian random variable with standard deviation σ. We use this relation to correct the bias in $\hat{\bar{\boldsymbol{\theta}}}$.

F.2 Experiments

We conduct our experiments with $N = 100$ clients, each having independent Bernoulli masks with dimension $d = 5$ and range $[0.2, 0.8]$, i.e., $\boldsymbol{\theta} \in [0.2, 0.8]^5$. Figure 5 shows the estimation error $||\hat{\bar{\boldsymbol{\theta}}}^{g,t} - \bar{\boldsymbol{\theta}}^{g,t}||_2^2$ under no noise injection case (i.e. no DP) with the black line. The red curve corresponds to the ϵ-estimation error behavior if Bernoulli sampling did not amplify the privacy. (Recall that smaller ϵ indicates stronger DP guarantees.) The blue curve corresponds to the ϵ_{amp}-estimation error behavior when the amplification due to Bernoulli sampling reduces ϵ. The blue curve overlaps with the red curve for ϵ values smaller than $d \cdot r_\alpha(c) = 8.96$, where there is no privacy amplification. However, notice that the blue line never reaches ϵ's higher than this value due to amplification, while enjoying smaller estimation errors that the red curve can only achieve with very large ϵ. This shows the promise of FedPM in having a better privacy-accuracy performance than most baselines that do not have amplification. Finally, the green curve shows that bias correction improves this performance further even with $\epsilon < d \cdot r_\alpha(c) = 8.96$.

G Additional Experimental Details

G.1 Non-IID Data Split and Partial Participation

In the experiments in Section 3.2, to model the size of each dataset $|\mathcal{D}_n| = D_n$, for each client $n \in \{1, \dots, N\}$, an integer j_n is sampled uniformly in $\{10, \dots, 100\}$. Then, a coefficient $p_n = \frac{j_n}{\sum_n j_j}$ is computed, which represents the size of the local dataset D_n as a fraction of the size of the full dataset, i.e., the training set of CIFAR-10. In this way, highly unbalanced datasets can be generated from the central one. We then impose a maximum number of different labels, or classes, c_{max}, one client can see. Consequently, clients need cooperation to learn the statistics of other classes' distributions, as the test dataset contains samples from all classes. In addition to this, partial participation is also considered, meaning that at each round the server uniformly samples a fraction $\rho = \frac{K}{N}$ of clients to participate to the training round.

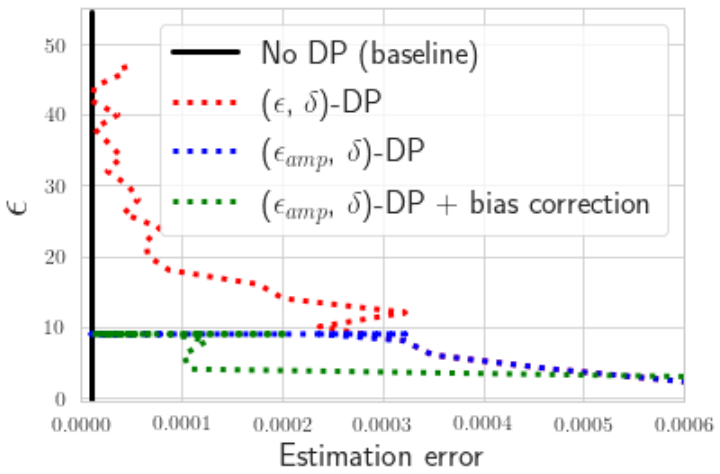

Figure 5: The effect of privacy amplification and bias correction in the privacy budget (ϵ) vs. estimation error behavior. Comparing red and blue curves, we see that we can reach small estimation errors without increasing ϵ thanks to the amplification (see the vertical blue line at low estimation error.). While red curve and blue curve overlap for $\epsilon < d \cdot r_\alpha(c) = 8.96$, in that regime we benefit from the our bias correction strategy to reach lower error.

H Additional Experimental Results

H.1 Non-IID accuracy - $c_{\max} = 2$

We first highlight that in this case, to distill the final model, we may apply both stochastic sampling, as during training, or a hard-threshold method, similar to the one adopted in FedMask (Li et al., 2021). In the latter, a mask coefficient m_i is set to 1 if $\theta_i > \alpha_{\text{ths}}$, and 0 otherwise. For all experiments but one, when $\alpha_{\text{ths}} \in [0.4, 0.6]$, the threshold test accuracy is always higher than the sampling method, and so we use the threshold method. However, in the extreme case $c_{\max} = 2$ and $\rho = 0.1$, the optimal values for $\alpha_{\max}$ were in $[0.2, 0.4]$ and $[0.6, 0.8]$ in all experiments, probably due to the high randomness given by the highly heterogeneous scenario. Consequently, for the last experiment we just adopt the stochastic sampling strategy to evaluate the model, as further optimizing the α_{ths} means adapting to the test dataset, which may corrupt the ability of the model to generalize. We now report the missing numbers for the case $c_{\max} = 2$.

Avg. Accuracy - $c_{\max} = 2$			
Algorithm	$\boldsymbol{\rho = 1}$	$\boldsymbol{\rho = 0.5}$	$\boldsymbol{\rho = 0.1}$
DRIVE (Vargaftik et al., 2021)	0.434 ± 0.025	0.376 ± 0.014	0.221 ± 0.003
EDEN (Vargaftik et al., 2022)	0.535 ± 0.050	0.461 ± 0.016	0.219 ± 0.005
QSGD (Alistarh et al., 2017)	0.476 ± 0.033	0.464 ± 0.002	0.243 ± 0.014
FedMask (Li et al., 2021)	0.420 ± 0.028	0.387 ± 0.062	0.197 ± 0.030
FedPM (Ours)	$\mathbf{0.6425 \pm 0.016}$	$\mathbf{0.556 \pm 0.031}$	$\mathbf{0.277 \pm 0.003}$

Table 2: Average final accuracy $\pm\sigma$ in non-IID data split, when $c_{\max} = 2$, and $\rho = \{0.1, 0.5, 1\}$, for FedPM, FedMask, and the strongest baselines in the IID experiments: EDEN, DRIVE, and QSGD. The training process duration was set to $t_{\max} = 200$ rounds.

H.2 Bitrate Considerations on non-IID Data

In this section, we report the bitrate and bpp considerations on the non-IID data split experiments described in Section 3.2. Tab.s 3 and 4 report the average bitrate needed by the different algorithms over the whole training process when $c_{\max} = 4$ and $c_{\max} = 2$, respectively. By simply multiplying the obtained average bitrate by the total number of rounds $t_{\max} = 200$, we obtain the lower bound on the total number of bits one model coefficient needs to converge to its final value, on average, indicating the total amount of information communicated during the training process.

We first observe that both DRIVE and EDEN consume almost the same amount of bits no matter which system configuration or round number is used (very small variance), and it is instead model dependent (see Fig. 2). On the other hand, FedPM and QSGD report higher bitrate variability, as it depends on both the training phase and system setting. As already observed in Section 3.1, FedMask balances almost uniformly the binary updates, leading to a bitrate that is basically fixed to 1. In both tables, FedPM presents the smallest bitrate in the case $\rho = 1$, whereas for the other scenarios EDEN and DRIVE are slightly more efficient. We argue that this is motivated by the fact that, as the learning task becomes harder due to the high system heterogeneity, all the models struggle converging to good and stable solutions, which means that FedPM is still uncertain on the *weights importance probabilities* $\boldsymbol{\theta}$, setting many of them close to 0.5. However, we think that this may be a useful feature of FedPM to quantify its internal uncertainty which, we will further analyze.

To conclude the analysis, we also report the FedPM bpp for the final model, which is an indication on the average number of bits needed to store one coefficient of the trained model. In the case $c_{\text{max}} = 2$, the bpps are 0.79, 0.834, and 0.99, when $\rho = \{0.1, 0.5, 1\}$, respecitvely. When $c_{\text{max}} = 2$, the bpps are 0.8, 0.817, and 0.992. Consequently, at the end of the training process, FedPM remains the most efficient option, as already observed in Section 3.1.

Avg. Bitrate - $c_{\text{max}} = 4$			
Algorithm	$\boldsymbol{\rho = 1}$	$\boldsymbol{\rho = 0.5}$	$\boldsymbol{\rho = 0.1}$
DRIVE (Vargaftik et al., 2021)	$0.885 \pm 9 \cdot 10^{-5}$	$\mathbf{0.885 \pm 1 \cdot 10^{-4}}$	$\mathbf{0.885 \pm 1 \cdot 10^{-4}}$
EDEN (Vargaftik et al., 2022)	$0.885 \pm 1 \cdot 10^{-4}$	$\mathbf{0.885 \pm 1 \cdot 10^{-4}}$	$\mathbf{0.885 \pm 1 \cdot 10^{-4}}$
QSGD (Alistarh et al., 2017)	0.982 ± 0.027	0.923 ± 0.029	0.91 ± 0.05
FedMask (Li et al., 2021)	$1 \pm 3 \cdot 10^{-6}$	$1 \pm 8 \cdot 10^{-8}$	$1 \pm 6 \cdot 10^{-7}$
FedPM (Ours)	$\mathbf{0.863 \pm 0.077}$	0.912 ± 0.056	0.996 ± 0.003

Table 3: Average bitrate $\pm\sigma$ over the whole training process in non-IID data split, when $c_{\text{max}} = 4$, and $\rho = \{0.1, 0.5, 1\}$, for FedPM, FedMask, and the strongest baselines in the IID experiments: EDEN, DRIVE, and QSGD. The training process duration was set to $t_{\text{max}} = 200$ rounds.

Avg. Bitrate - $c_{\text{max}} = 2$			
Algorithm	$\boldsymbol{\rho = 1}$	$\boldsymbol{\rho = 0.5}$	$\boldsymbol{\rho = 0.1}$
DRIVE (Vargaftik et al., 2021)	$0.885 \pm 7 \cdot 10^{-5}$	$\mathbf{0.885 \pm 2 \cdot 10^{-4}}$	$\mathbf{0.885 \pm 2 \cdot 10^{-4}}$
EDEN (Vargaftik et al., 2022)	$0.885 \pm 1 \cdot 10^{-4}$	$\mathbf{0.885 \pm 7 \cdot 10^{-5}}$	$\mathbf{0.885 \pm 7 \cdot 10^{-5}}$
QSGD (Alistarh et al., 2017)	1.230 ± 0.043	1.234 ± 0.038	1.082 ± 0.01
FedMask (Li et al., 2021)	$1 \pm 2 \cdot 10^{-6}$	$1 \pm 2 \cdot 10^{-6}$	$1 \pm 2 \cdot 10^{-7}$
FedPM (Ours)	$\mathbf{0.868 \pm 0.076}$	0.904 ± 0.063	0.997 ± 0.01

Table 4: Average bitrate $\pm\sigma$ over the whole training process in non-IID data split, when $c_{\text{max}} = 2$, and $\rho = \{0.1, 0.5, 1\}$, for FedPM, FedMask, and the strongest baselines in the IID experiments: EDEN, DRIVE, and QSGD. The training process duration was set to $t_{\text{max}} = 200$ rounds.