Bitwidth Heterogeneous Federated Learning with Progressive Weight Dequantization

Jaehong Yoon [1 *] **Geon Park** [1 *] **Wonyong Jeong** [1] **Sung Ju Hwang** [1 2]

Abstract

In practical federated learning scenarios, the participating devices may have different bitwidths for computation and memory storage by design. However, despite the progress made in device-heterogeneous federated learning scenarios, the heterogeneity in the bitwidth specifications in the hardware has been mostly overlooked. We introduce a pragmatic FL scenario with bitwidth heterogeneity across the participating devices, dubbed as Bitwidth Heterogeneous Federated Learning (BHFL). BHFL brings in a new challenge, that the aggregation of model parameters with different bitwidths could result in severe performance degeneration, especially for high-bitwidth models. To tackle this problem, we propose ProWD framework, which has a trainable weight dequantizer at the central server that progressively reconstructs the low-bitwidth weights into higher bitwidth weights, and finally into full-precision weights. ProWD further selectively aggregates the model parameters to maximize the compatibility across bit-heterogeneous weights. We validate ProWD against relevant FL baselines on the benchmark datasets, using clients with varying bitwidths. Our ProWD largely outperforms the baseline FL algorithms as well as naive approaches (e.g. grouped averaging) under the proposed BHFL scenario.

1. Introduction

In recent decades, the drastic evolution of hardware technologies for edge devices has changed our lives from the root. Smart edge devices, which have the ability to process sensory inputs and communicate, such as embedded sensors, drones, phones, smart watches, and augmented reality glasses, are now being used in our everyday lives. Such accessibility of edge devices has led to the emergence of *Federated Learning* (FL) (McMahan et al., 2017; Zhao et al., 2018; Chen et al., 2019), a learning framework in which multiple clients collaboratively train on private local data while periodically communicating the trained models across them, often through a server which aggregates and broadcasts the local models. Many previous works have investigated the potential and applicability of FL in various learning frameworks, such as semi-supervised learning (Jeong et al., 2021), bayesian learning (Wang et al., 2020), graph neural networks (Mei et al., 2019; Wu et al., 2021), meta-learning (Jiang et al., 2019; Fallah et al., 2020), and continual learning (Yoon et al., 2021).

A crucial challenge in FL is that there could be a large discrepancy among participants, in their data distribution, tasks, model architectures, and devices which often leads to incompatibility of the models that are being aggregated. This problem is often referred to as Heterogeneous Federated Learning (Jiang et al., 2020; Lin et al., 2020; He et al., 2020; Diao et al., 2021) problem. Many existing works have successfully alleviated the adverse effect of data-, model-, and device-heterogeneity. However, the most basic assumption even in such heterogeneous FL scenarios, is that all models have the same bitwidths.

Yet, in real-world FL scenarios, participating devices may have heterogeneous bitwidth specifications. Suppose that we are building a federated network of various health care providers, such as hospitals, community health centers, and clinics, as well as end-users, and even wearable devices, i.e. smart watches. Each client has a health disorder prediction model for the users themselves or their patients that continuously learns to perform a diagnosis given the heart rates or bioelectric signals. The scenario allows the participation of local clients with various hardware infrastructures, some of them using models built under lightweight devices based on low-bitwidth hardware operations using FPGA, ASIC, Raspberry Pi, or Edge GPUs. Here, BHFL enables local devices with different hardware specifications to participate

[*]Equal contribution [1]Korea Advanced Institute of Science and Technology (KAIST), South Korea [2]AITRICS, South Korea. Correspondence to: Jaehong Yoon <jaehong.yoon@kaist.ac.kr>, Geon Park <geon.park@kaist.ac.kr>, Sung Ju Hwang <sjhwang82@kaist.ac.kr>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

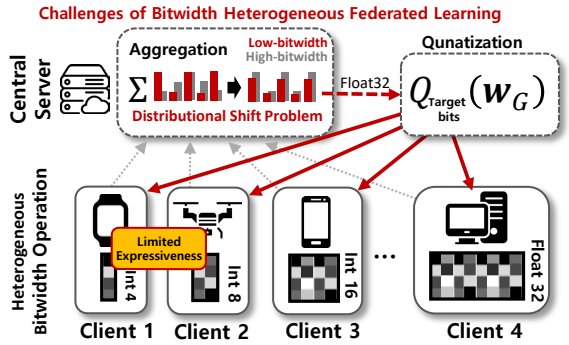


Figure 1: **Bitwidth Heterogeneous Federated Learning.** We consider a FL setting where the participating devices have heterogeneous bitwidths. FL with different bitwidth models may cause detrimental side effects due to (i) distributional shift of model weights, and (ii) the limited expressiveness in low-bit weights.

in a single federated learning framework without the need for uniformity of the infrastructure, enhancing the pool of devices that could participate in collaborative learning. We refer to this practical FL scenario as Bitwidth Heterogeneous Federated Learning (BHFL), which we illustrate in Figure 1.

Tackling BHFL is a nontrivial problem, as it poses new challenges such as (i) suboptimal loss convergence due to distributional shift after aggregating the weights of bitwidth-heterogeneous models, and (ii) inherent limitation of expressive power in low-bitwidth weights. As shown in Table 1, due to these challenges, existing methods cannot appropriately handle the new setting. While there exists a line of works that propose to quantize model weights to reduce the communication cost when transmitting them to the server, they assume that full-precision weights are being used at the local devices. However, processing full-precision weights may not be possible for resource-limited hardware devices.

In this paper, we propose a novel framework that can succesfully deal with the new challenges posed by the BHFL problem, which we name as **Pro**gressive **W**eight **D**equantization (**ProWD**), The ProWD framework enhances the compatibility across bit-heterogeneous weights with selective weight aggregation and weight dequantization. The selective weight aggregation discards outliers from the low-precision weights, and the trainable dequantizer at the server recovers high-precision weight information from the given low-bitwidth weights. These two methods collaboratively alleviate the information loss resulting from the aggregation of incompatible weights with heterogeneous bitwidths.

We evaluate the performance of our ProWD on various benchmark datasets and show that our method significantly outperforms previous FL methods that consider weight quantization, while also outperforming naive heuristics to tackle the bitwidth heterogeneity, such as grouped averaging. We also provide comprehensive analyses which show that existing FL methods suffer from poor convergence and adaptation under the bit-heterogeneous federated learning scenario, while ProWD consistently increases the performance of all local models regardless of their bitwidths.

In summary, our contributions are threefold:

- We define a practical federated learning scenario where the participating devices may have largely different bitwidths, which brings in new challenges such as degenerate distributional shift of federated weights and limited expressive power of low-bitwidth models.
- We propose *ProWD*, a novel framework that effectively tackles the bitwidth heterogeneous FL problem, by selectively aggregating the weights and hierarchically dequantizing the weights prior to aggregation.
- We demonstrate the efficacy of our ProWD framework by validating it on diverse compositions of bitwidth specifications in local clients, against recent FL methods as well as naive remedies.

2. Related Works

Quantization for Federated learning While no existing work considers the problem of federated learning across devices with different innate bitwidths, several works propose to quantize weights or gradients to reduce the communication cost, often referred to as Quantized Parameter Communication (QPC). FedPAQ (Reisizadeh et al., 2020) proposed to send the quantized changes of the weights at each client, instead of the full weights. FedCOMGATE (Haddadpour et al., 2021) extends this idea by adopting a global learning rate and guiding the update direction to stay close to each other with an accumulated correction vector, to further address data heterogeneity. While we empirically observed that QPC approaches alleviate the performance degeneration of the high-bitwidth models under the BHFL scenario since they do not drastically change the weights as simple averaging does, they are suboptimal since they do not explicitly tackle the challenges posed by the BHFL problem.

Low-bitwidth training Low-bitwidth training is an approach to train a model using only low-bitwidth operations and data types at training time, which enables on-device learning with lightweight edge devices. BNN (Hubara et al., 2016) and XNOR-Net (Rastegari et al., 2016) are approaches to accelerate the training of the convolutional neural networks using binary operations in the forward pass with binarized weights and activations, with floating-point operations to compute gradients in the backward pass. Except for that, diverse approaches have been proposed which focus on using 8-bit floating point numbers for

Table 1: Categorization of existing methods for bitwidth heterogeneous federated learning.

METHODS	FL Type	$\text{Bits}_{\text{Server}}$	$\text{Bits}_{\text{Clients}}$	$\text{Bits}_{\text{Uplink}}$	$\text{Bits}_{\text{Downlink}}$	Communication
FEDAVG (McMahan et al., 2017)	FL	Float32	Float32	Float32	Float32	Weights
FEDPROX (Li et al., 2018)	FL	Float32	Float32	Float32	Float32	Weights
FEDPAQ (Reisizadeh et al., 2020)	QPC[1]	Float32	Float32	Target bits	Float32	$Q_{\text{Target\_bits}}(\text{Diffs})$[2]
FEDCOM (Haddadpour et al., 2021)	QPC	Float32	Float32	Target bits	Float32	$Q_{\text{Target\_bits}}(\text{Diffs})$
FEDCOMGATE (Haddadpour et al., 2021)	QPC	Float32	Float32	Target bits	Float32×2	$Q_{\text{Target\_bits}}(\text{Diffs})$
PROWD (OURS)	BHFL	Float32	Client-specific	Client-specific	Client-specific	W_Q[3]

[1] Quantized Parameter Communication (QPC)
[2] $Q_{\text{Target\_bits}}(w)$: A function quantizes input w to the target bitwidth
[3] Weights obtained from clients' bit-dependent training

the weights, activations, and gradients (Zhou et al., 2018), devising ternary gradients to reduce communication cost (yet, it allows float32 operation for accumulating the gradients) (Wen et al., 2017), training without floating-point operations (Wu et al., 2018), adopting direction-sensitive gradient clipping (Zhu et al., 2020), training for mixed-bitwidth models (Zhang et al., 2020; Zhao et al., 2021; Sun et al., 2020). Most existing neural quantization approaches quantize only the weights or the gradients, while preserving parts of the network that has a large impact on the performance, such as activations, in full-precision, and thus are not applicable to training on devices with limited bitwidths.

Neural dequantization The dequantization of low-bitwidth signals into high-bitwidth signals has been studied for diverse applications. For image reconstruction, Xing et al. (2021) aims to recover the resolution of quantized sRGB images to full-dynamic-range RAW image data via an invertible dequantizer function. In generative flow models, Nielsen & Winther (2020) dequantizes the discrete-valued data by adding a uniform noise to guarantee that the data is able to have any value in the continuous domain. Dequantizatoin is also used for the process of converting the low-bit representation of the weights to high-bit without changing their values (Gholami et al., 2021), or to decode the encoded vectors using a codebook (Lee et al., 2020). In this paper, we refer to "dequantization" to describe the process of recovering the original high-bitwidth weights from the low-bitwidth quantized weights received from local clients , so that a server can utilize the high-performing recovered models for aggregation during FL. To our knowledge, our work is the first work that provides an appropriate method adopting the weight dequantization approach for solving practical FL scenarios.

3. Bitwidth Heterogeneous Federated Learning (BHFL)

We now introduce the problem setup for standard federated learning and extends it to the Bitwidth Heterogeneous Federated Learning (BHFL) scenario, where a subset of the participating devices train the local models with low-bitwidth operations, according to their hardware specifications (Section 3.1). We then describe the quantized computational flow for the training of low-bitwidth clients in Section 3.2.

3.1. Problem statements

Federated Learning In a standard Federated Learning (FL) scenario (McMahan et al., 2017; Chen et al., 2019), each client trains the local model on the private data and periodically transmits the model parameters to the central server, where the models are aggregated and broadcasted back to the clients. Let N different clients $\mathcal{C} = \{c_1, ..., c_N\}$ participate in an FL system. Given training samples $\boldsymbol{x}_n$ and its corresponding labels $\boldsymbol{y}_n$, we suppose that a client c_n solves a local optimization problem $\mathcal{L}_n = \text{CE}(f(\boldsymbol{x}_n; \boldsymbol{w}_n); \boldsymbol{y}_n)$, where CE is a cross-entropy loss and $f(\cdot; \boldsymbol{w}_n)$ is a neural network of client c_n parameterized by $\boldsymbol{w}_n$. At each communication round r, clients $\mathcal{C}^{(r)} \subseteq \mathcal{C}$ send the model parameters to the central server, and the server aggregates received weights, for exampling by averaging their weights.

Bitwidth heterogeneous federated learning The majority of existing FL methods assume full-precision operations for local clients, even when they consider device heterogeneity. However, the participating devices have largely heterogeneous bitwidths according to their hardware specifications. To this end, we introduce a practical FL scenario, named Bitwidth Heterogeneous Federated Learning (BHFL), in which we relax the strong assumption that all clients are capable of full-precision floating-point operations. We represent n-th local client as a tuple of the model weights $\boldsymbol{w}_n$ and the corresponding hardware bitwidth information $s_n \in \mathcal{S}$, where $\mathcal{S}$ is a set of available bitwidth specifications for local hardware devices. Then, the set of N clients, $\mathcal{C}$ can be represented as follows: $\mathcal{C} = \{(\boldsymbol{w}_1, s_1), \ldots, (\boldsymbol{w}_N, s_N)\}$. At each round of communication, a central server receives client tuples and aggregates model parameters which might be quantized in various levels depending on hardware specifications. We assume that the server allows the full-precision computation, which redistributes $\boldsymbol{w}_G^{(r)}$ to all clients after quantizing the aggregated weights according to the hardware specifications for each client; that is, $\boldsymbol{w}_n^{(r+1)} \leftarrow Q(\boldsymbol{w}_G^{(r)}, s_n), \forall n$.

In BHFL, what aggregation method we use could have a strong impact on the overall performance of the model being learned. Using a naive averaging technique, such as sim-

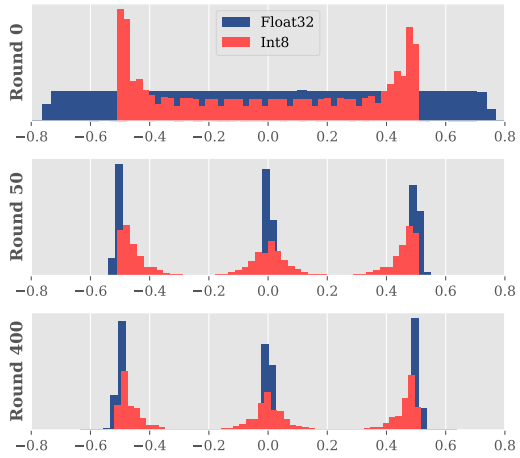


Figure 2: **Skewed weight distribution after the aggregation of mixed bitwidth weights**. The distribution of the last layer's weights of the full-precision and low-bitwidth models at the initial, after 50, and 400 aggregation rounds.

ple averaging, to aggregate local clients' model parameters hinders the convergence. This is because the model falls into a suboptimal local minimum due to the incompatibility across the model weights with heterogeneous bitwidths, due to large discrepancy in their distributions. The detrimental effect is more severe for larger bitwidth models, as aggregating the low-bitwidth model weights will result in the loss of expressiveness in the model. In Figure 2, we observe that the distribution of the full-precision weights quickly degenerates into three peaks that correspond to the ternary quantization values of the low-bitwidth models.

3.2. Local Computation for Limited Bitwidth Clients

We assume that the local clients are edge devices that perform limited bitwidth computations according to their hardware specifications. Float32 clients perform regular full-precision training, but quantized clients, such as Int6, Int8, and Int16, perform training using low-precision integer operations, following Wu et al. (2018).

Let $\boldsymbol{q}^l$ and $\boldsymbol{a}^l$ be s-bit quantized weights and activations at layer l for a client, respectively. We denote the convolution operator as $*$. Since convolution involves multiplication between the weights and the activations, naively computing $\boldsymbol{q}^l * \boldsymbol{a}^{l-1}$ requires the hardware to support fast multiplication of two s-bit integers. To relax this requirement, we ternarize the quantized weights before the convolution operation:

$$\boldsymbol{a}^l = \text{ReLU}\left(Q_s\left(Q_{\text{Int2}}\left(\boldsymbol{q}^l\right) * \boldsymbol{a}^{l-1}/\alpha^l\right)\right). \quad (1)$$

Thus, Equation 1 only needs s-bit addition and subtraction operations for training and inference, and the intermediate results can be stored in $(s+1)$-bits while achieving a good tradeoff between precision and computation (Li et al., 2016).

To prevent the activations from clipping out of range, we rescale the activations with a pre-defined layerwise scalar coefficient α^l so that they are within the range for s-bit integers. This rescaling can be implemented efficiently with a hardware bit-shift operation. We further provide details for training limited-bitwidth clients in Appendix B. After a few local training steps, limited-bitwidth clients broadcast original low-bitwidth weights $\boldsymbol{q}$ or the ternarized weights $Q_{\text{Int2}}\left(\boldsymbol{q}\right)$ to the server, like other QPC methods (Reisizadeh et al., 2020; Haddadpour et al., 2021).

4. Bitwidth Heterogeneous FL with ProWD

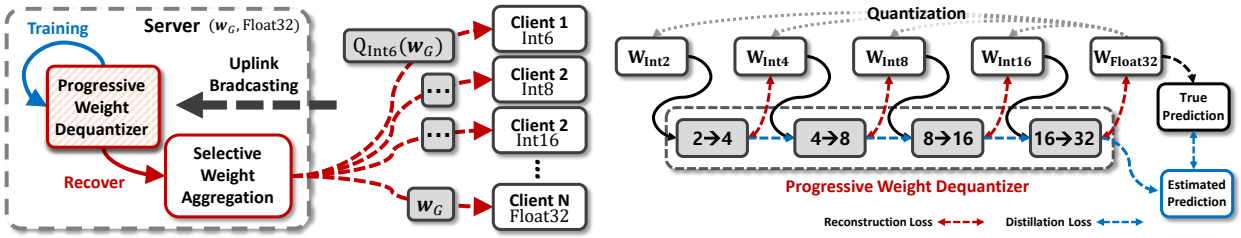
We first introduce our naive remedies for alleviating the challenges posed by the BHFL problem, and their limitations in Section 4.1. Then, we propose a novel FL framework to properly handle BHFL scenarios, named as *ProWD*, which consists of two core components: progressive weight dequantization and score-based selective weight aggregation, described in Section 4.2 and Section 4.3, respectively.

4.1. Naive Remedies for Bitwidth Heterogeneity in FL

BHFL deteriorates the performance of higher-bitwidth clients' models, while the local models from low-bitwidth clients often enjoy clear benefits due to knowledge transfer from the high-bitwidth models (See Figure 2). We first suggest a bitwidth-dependent averaging technique for BHFL, preventing interference across different bitwidth weights during aggregation, referred to as FedGroupedAvg. However, this method suffers from poor transferability, as the rich knowledge obtained by expressive high-bitwidth models is not transferred to low-bitwidth ones, while it preserves the performance of higher-bitwidth clients. Thus, we additionally suggest a modified version of FedGroupedAvg which allows the knowledge transfer from higher-bitwidth clients to the lower ones, but not vice versa, named as FedGroupedAvg-Asymmetric. These two simple baselines can improve the performance of local model, but are suboptimal in that it does not utilize the full knowledge of the participating models. We provide detailed descriptions for FedGroupedAvg variants in **??**. Thus, we propose a novel method to effectively tackle the challenges in BHFL, which overcomes the limitations of such naive remedies.

4.2. Progressive Weight Dequantization

Low-bitwidth models, while hardware-friendly, severely limits the expressiveness of the model. Aggregating the parameters with such limited information thus may degrade the quality of high-bitwidth models during federated learning. To this end, we propose a trainable dequantizer that reconstructs the full-precision weights for given low-precision weights. Yet, directly reconstructing the low-bit weights to the high-bit may not be as effective, especially when the



(a) Bitwidth heterogenrous federated learning with ProWD

(b) An example of progressive weight dequantizer

Figure 3: **(a) Illustration of our ProWD framework.** Clients send their local models and hardware bitwith specifications to the server. We reduce the distributional disparity among weights from different bitwidth devices during BHFL by introducing weight dequantization and selective aggregation. **(b) Progressive weight dequantizer** recovers low-bit weights into the high-bit via minimizing two loss terms.

bitwidth gap is large, since there exists a significant disparity between their distributions. To alleviate this issue with single-step reconstruction, our dequantizer breaks down the problem into block-wise weight dequantization problems by adopting a stack of network blocks.

Let a $\Pi = \{\pi_0, \cdots, \pi_k\} \supseteq \mathcal{S}$ be an ordered set of bitwidth precisions, where π_j is a lower precision bitwidth than π_{j+1} $\forall j$. The lowest and the higest bitwidths of Π correspond to those of $\mathcal{S}$, i.e., $\pi_0 = s_1$ and $\pi_k = s_m$. We denote a progressive weight dequantization function ϕ as a stack of k decomposable neural network blocks:

$$\phi := \phi^{\pi_0 \to \pi_k} = \phi^{\pi_0 \to \pi_1} \circ \phi^{\pi_1 \to \pi_2} \circ \cdots \circ \phi^{\pi_{k-1} \to \pi_k}, \quad (2)$$

where $\circ$ denotes the function composition and $\phi^{\pi_j \to \pi_{j+1}}$ indicates a block that recovers π_{j+1}-bit weights from π_j-bit weights. That is, we design the set of bitwidths for dequantizer blocks Π to include the client bitwidths so that the dequantizer can directly reconstruct the desired bitwidths from the received low-bit weights. We implement each block using a dimensionality-preserving function $h(\cdot; \boldsymbol{\theta})$ with a residual connection to its input. When the model $(\boldsymbol{w}, \pi_{j>0})$ arrives from a local client, the server quantizes $\boldsymbol{w}$ into j lower-bitwidth weights $\mathcal{Q}_{\boldsymbol{w}} = \{\boldsymbol{q}_{\pi_0}, ..., \boldsymbol{q}_{\pi_{j-1}}\}$. Given $\pi_{j<k}$-bit quantized weights $\boldsymbol{q}_{\pi_j}$, the block operation is formulated as follows:

$$\widehat{\boldsymbol{q}}_{\pi_{j+1}} = \phi^{\pi_j}(\boldsymbol{q}_{\pi_j}; \boldsymbol{\theta}_j) = \boldsymbol{q}_{\pi_j} + h(\boldsymbol{q}_{\pi_j}; \boldsymbol{\theta}_j). \quad (3)$$

The central server constructs a weight dataset out of received local model weights, by segmenting them into uniformly-sized blocks (e.g., $64 \times 24 \times 24$). Thus, our dequantizer can utilize differently-shaped weights from different layers or across heterogeneous neural architectures. We describe details of the weights dataset construction process and the design of network block ϕ in **??** and Appendix C. For the training of the weight dequantizer, we introduce two different loss terms. The first term is the reconstruction loss, which is defined as the average difference of the blockwise weight reconstruction of quantized input weights $\boldsymbol{q}_{\pi_j}$ and its higher bitwidth ground truth weights $\boldsymbol{q}_{\pi_{j+1}}$:

$$\mathcal{L}_{recon} = \sum_{j=0}^{k-1} \left\| \boldsymbol{q}_{\pi_{j+1}} - \phi^{\pi_j \to \pi_{j+1}}(\boldsymbol{q}_{\pi_j}; \boldsymbol{\theta}_j) \right\|_1. \quad (4)$$

Minimizing the blockwise reconstruction error allows the model to progressively extrapolate missing links between the low-bitwidth weights and high-bitwidth weights and not to stray far from the intermediate reconstructions. We note that there is no strict rule for the choices of the target bitwidths of dequantizer blocks, but we design the dequantizer in which the bitwidth difference between consecutive blocks is maintained to a similar degree until reaching the target high-bitwidth, avoiding drastic increase in bitwidth for each reconstruction step.

The second term is a distillation loss, for minimizing the descrepancy in the predictions from the model with high-bitwidth ground truth weights, and the model with dequantized low-bitwidth weights. To this end, the central server utilizes a tiny buffer $\mathcal{U}$ independent of the local data to compare the prediction between high-bit model and recovered model from the low-bit weights. Given the $\boldsymbol{w}$ and its quantized low-precision $\boldsymbol{q}_{\pi_0} = Q_{\pi_0}(\boldsymbol{w})$, we compute a distillation loss using a randomly sampled minibatch $\boldsymbol{u} \sim \mathcal{U}$ from the buffer as follows:

$$\mathcal{L}_{distill} = -\text{Sim}\Big(f(\boldsymbol{u}; \boldsymbol{w}), f\big(\boldsymbol{u}; \phi^{0 \to k}(\boldsymbol{q}_{\pi_0}; \Theta)\big)\Big), \quad (5)$$

where $\text{Sim}(\cdot)$ is a similarity metric for the two output distributions (we use cosine similarity) and $\Theta = \{\boldsymbol{\theta}_0, \cdots, \boldsymbol{\theta}_k\}$. Note that $f(\boldsymbol{u}; \boldsymbol{w})$ is a softmax class probability of a neural network parameterized by $\boldsymbol{w}$ given input $\boldsymbol{u}$. This distllation loss allows ProWD to directly tackle the prediction task with the reconstructed weights, which helps it recover full-precision weights while considering the downstream task performance. The final objective for training ProWD at the server is given as follows:

$$\mathcal{L} = \mathcal{L}_{recon} + \lambda \mathcal{L}_{distill}, \quad (6)$$

where λ is a hyperparameter to balance the two loss terms. Note that the model is not very sensitive to the choice of λ and thus we set $\lambda = 1$ in all our experiments. The training steps of our dequantizer is described in Algorithm 1.

Algorithm 1 Training of progressive weight dequantizer

input set of bitwidths $\Pi = \{\pi_0, \dots, \pi_k\}$, dequantizer $\phi_\Theta^{\pi_0 \to \pi_k}$, input weights and corresponding bitwidth $(\boldsymbol{w}, \pi_j)$, neural network f, unsupervised buffer $\mathcal{U}$, balancing coefficient λ.

$\boldsymbol{q}_{i<j} = \{Q_{\pi_i}(\boldsymbol{w})\}_{i=0}^{j-1}$ ▷ Quantize into each of the lower bitwidths
Construct data loader $\mathcal{D}_{\boldsymbol{w}}$ using $\{\boldsymbol{q}_{i<j}, \boldsymbol{w}\}$
for $(\boldsymbol{q}_0, \dots, \boldsymbol{q}_{j-1},\ \boldsymbol{w}) \sim \mathcal{D}_{\boldsymbol{w}}$ **do**
  Sample $\boldsymbol{u} \sim \mathcal{U}$ augmented with gaussian noise
  $\mathcal{L}_{recon} = \sum_{i=0}^{j-1} \left\| \boldsymbol{q}_{i+1} - \phi^{\pi_i \to \pi_{i+1}}(\boldsymbol{q}_i; \boldsymbol{\theta}_i) \right\|_1$
  $\mathcal{L}_{distill} = -\text{Sim}(f(\boldsymbol{u}; \boldsymbol{w}), f(\boldsymbol{u}; \phi_\Theta(\boldsymbol{q}_0)))$
  $\mathcal{L} = \mathcal{L}_{recon} + \lambda \mathcal{L}_{distill}$
  Update weight dequantizer ϕ_Θ to minimize $\mathcal{L}$

4.3. Score-based Selective Weight Aggregation

As described in Figure 2, using a naive aggregation technique such as simple averaging, may lead the model training to fall into a suboptimal local minimum. To prevent such distribution shifts of weights during BHFL, we assert that the low-bitwidth models should share similar gradient directions as the full-precision model, as inspired by the observation in Zhu et al. (2020) that there exists a strong correlation between the training stability and the deviation of the quantized model's gradient directions from the full-precision model, measured by the cosine similarity. Let us consider a simple two-client federated learning framework where a server communicates with a full-precision model parameterized with $\boldsymbol{w}_{\text{High}}$ and a low-bitwidth model parameterized with $\boldsymbol{w}_{\text{Low}}$. The goal of the low-bitwidth model then is to distill the knowledge of $\boldsymbol{w}_{\text{High}}$:

$$\left\langle \frac{\partial \ell(f(B; \boldsymbol{w}_{\text{High}}))}{\partial \boldsymbol{w}_{\text{High}}}, \frac{\partial_Q \ell(f_Q(B; \boldsymbol{w}_{\text{Low}}))}{\partial_Q \boldsymbol{w}_{\text{Low}}} \right\rangle \geq 0, \quad (7)$$

where ℓ is a task loss and $f(\cdot; \boldsymbol{w})$ is a neural network parameterized by $\boldsymbol{w}$. The subscript Q denotes the quantized operations on the weights, activations, and gradients.

However, unlike the setting of Zhu et al. (2020), under BHFL scenarios, it is impossible to preserve full-precision knowledge on low-bitwidth local devices as they have no means to represent them. Thus, we impose a selective weight aggregation technique based on the relevancy among the weights from the local clients. When a central server receives the local models from different bitwidth clients, we select sparse sub-weights from lower-bitwidth models that are compatible with the weights of the full-precision aggregated weights.

Let $\overline{\boldsymbol{w}}_{\text{High}}^{(r)}$ and $\overline{\boldsymbol{w}}_{\text{Low}}^{(r)}$ denote average high-bit weights and the low-bit weights that the server received at communication round r, respectively. When $\Delta\overline{\boldsymbol{w}}_{\text{High}} = \overline{\boldsymbol{w}}_{\text{High}}^{(r)} - \overline{\boldsymbol{w}}_{\text{High}}^{(r-1)}$ and $\Delta\overline{\boldsymbol{w}}_{\text{Low}} = \overline{\boldsymbol{w}}_{\text{Low}}^{(r)} - \overline{\boldsymbol{w}}_{Low}^{(r-1)}$, we encourage the high-bit and low-bit model weights to have similar update directions by disregarding a few outliers in the low-bit models. That is, Given a sparsity ratio τ, we encourage a server to obtain the binary mask c^* as follows:

$$\boldsymbol{c}^* = \underset{\boldsymbol{c}}{\text{argmax}} \frac{(\boldsymbol{c} \odot \Delta\overline{\boldsymbol{w}}_{\text{Low}})^\top \Delta\overline{\boldsymbol{w}}_{\text{High}}}{\|\boldsymbol{c} \odot \Delta\overline{\boldsymbol{w}}_{\text{Low}}\| \|\Delta\overline{\boldsymbol{w}}_{\text{High}}\|}, \text{ s.t. } |\boldsymbol{c}^*| \leq \tau. \quad (8)$$

For BHFL scenarios with multiple bitwidths, we split high-/low-bitwidths weights based on the mean bit-width of given a bitwidth set. We formulate the equation as a simple optimization problem to obtain a desired binary mask c^* to maximize the cosine similarity between sparsified averaged weight movement of low-bitwidth models and the averaged movement of high-bitwidth model. The process is rapidly optimized within a few steps (e.g., 10) and a marginal training time ($\sim 10ms$ per client). To this end, we perform selective weight aggregation as follows:

$$\boldsymbol{w}_G \leftarrow \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{c}_n \odot \boldsymbol{w}_n, \quad (9)$$

where $\boldsymbol{c}_n = \boldsymbol{c}^*$, if s_n is one of low-bitwidth specifications, otherwise, $\boldsymbol{c}_n$ is a all-one tensor with the same shape as $\boldsymbol{w}_n$. The overall procedure of our BHFL framework is described in Algorithm 2. Note that the dequantizer can be updated anytime during the FL process in a concurrent manner, because it does not require the latest client model weights for training. Thus, there is no training time bottleneck introduced by the training of our dequantizer.

Algorithm 2 ProWD framework for BHFL

input clients $\mathcal{C} \leftarrow \{\boldsymbol{w}_n, s_n\}_{n=1}^N$, $s_j \in \Pi = \{\pi_0, \dots, \pi_k\}$, $\forall j$, weight dequantizer $\phi_\Theta^{\pi_0 \to \pi_k}$, global weights $\boldsymbol{w}_G$.

for each round $r = 1, 2, ..., R$ **do**
  Sample $\mathcal{C}^{(r)} \subseteq \mathcal{C}$ where $|\mathcal{C}^{(r)}| = M$
  Distribute federated weights $\boldsymbol{w}_G$ to clients $\mathcal{C}^{(r)}$
  for each client $(\boldsymbol{w}_n, s_n) \in \mathcal{C}^{(r)}$ **in parallel do**
    $\boldsymbol{w}_n \leftarrow Q_{s_n}(\boldsymbol{w}_G)$ ▷ send s_n-bit quantized weights
    $\boldsymbol{w}_n \leftarrow \text{LocalUpdate}(\boldsymbol{w}_n, s_n)$
    Broadcast $(\boldsymbol{w}_n, s_n)$ to the central server
  $\boldsymbol{w}_n \leftarrow \phi_\Theta(\boldsymbol{w}_n)$, **if** $s_n \neq \pi_k$ ▷ Dequantize weights
  Obtain binary masks $\boldsymbol{c}^*$ using Equation 8
  $\boldsymbol{w}_G \leftarrow \frac{1}{M} \sum_{n=1}^{M} \boldsymbol{c}_n \odot \boldsymbol{w}_n$ ▷ Equation 9

5. Experiments

Datasets We validate our method against the relevant FL methods under several BHFL scenarios, with varying bitwidth configurations of participating clients. We use the widely used benchmark dataset for federated learning methods, CIFAR-10 to validate our method following the

Table 2: Average accuracy at each bitwidth and average accueacy across all clients on CIFAR-10 dataset. We set participating clients with 50% of Int8 and 50% of Float32 models (Left), and 50% of Int8 and 50% of Float32 models (Right). All of the results are measured by computing the 95% confidence interval over three independent runs.

METHOD	CIFAR-10, Int8 (50%) - Float32 (50%)				CIFAR-10, INT8 (80%) - FLOAT32 (20%)			
	INT8 ACC	FLOAT32 ACC	GAP	AVERAGE	INT8 ACC	FLOAT32 ACC	GAP	AVERAGE
LOCAL TRAINING	69.59 (± 0.34)	75.82 (± 0.41)	+6.23	72.71 (± 0.26)	69.39 (± 0.34)	76.41 (± 0.52)	+7.02	70.79 (± 0.34)
FEDAVG (McMahan et al., 2017)	76.88 (± 0.49)	76.23 (± 0.36)	-0.65	76.56 (± 0.36)	77.43 (± 0.83)	74.64 (± 1.21)	-2.79	76.87 (± 0.87)
FEDPROX (Li et al., 2018)	71.16 (± 0.35)	69.28 (± 0.90)	-1.88	70.22 (± 0.55)	69.60 (± 0.50)	66.28 (± 1.24)	-3.32	68.94 (± 0.57)
FEDPAQ (Reisizadeh et al., 2020)	78.01 (± 0.55)	84.93 (± 0.30)	+6.93	81.47 (± 0.29)	76.61 (± 0.60)	82.27 (± 0.42)	+5.66	77.74 (± 0.49)
FEDCOM (Haddadpour et al., 2021)	75.37 (± 0.52)	77.69 (± 0.45)	+2.32	76.53 (± 0.38)	73.60 (± 1.08)	80.73 (± 0.73)	+7.12	75.03 (± 0.94)
FEDCOMGATE (Haddadpour et al., 2021)	76.74 (± 0.59)	77.36 (± 0.55)	+0.62	77.05 (± 0.36)	74.48 (± 0.63)	81.18 (± 0.56)	+6.70	75.82 (± 0.53)
FEDGROUPEDAVG	61.85 (± 0.78)	85.08 (± 0.28)	+23.23	73.46 (± 0.29)	71.76 (± 0.68)	78.07 (± 0.58)	+6.31	73.02 (± 0.65)
FEDGROUPEDAVG-ASYMMETRIC	78.39 (± 0.54)	84.97 (± 0.27)	+6.57	81.68 (± 0.26)	70.14 (± 0.36)	78.70 (± 0.48)	+8.56	71.85 (± 0.36)
PROWD (OURS)	**82.87** (± 0.37)	**85.99** (± 0.20)	+3.12	**84.43** (± 0.22)	**79.23** (± 0.25)	81.26 (± 0.45)	+2.03	**79.63** (± 0.23)

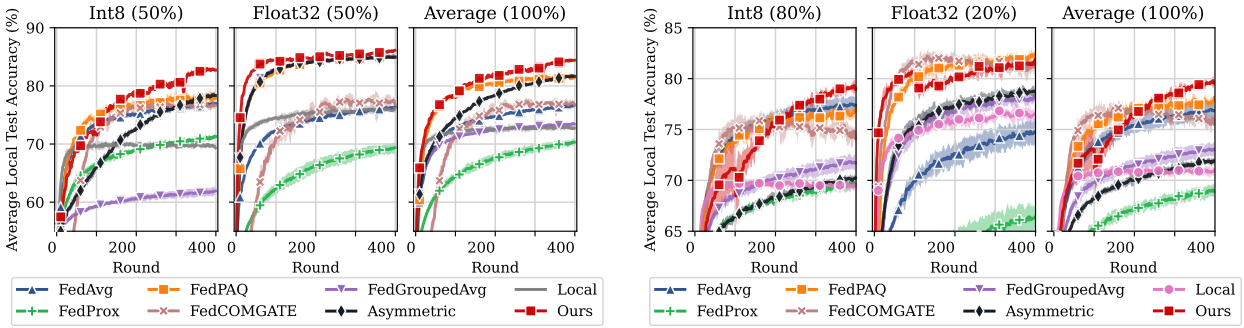


Figure 4: Visualization of average test accuracy on CIFAR-10 with 10 clients where the bitwidth of the clients is composed of **(a) 50% of Int8 and 50% of Float32** and **(b) 80% of Int8 and 20% of Float32** during BHFL. We average the results over three independent runs.

IID experimental settings of the existing works (Reisizadeh et al., 2020; Haddadpour et al., 2021). CIFAR-10 is a image classification dataset that consists of 10 object classes each of which has 5,000 training instances and 1,000 test instances. For FL purposes, we uniformly split the training instances per class by the number of clients participating in the federated learning system. We use a modified version of VGG-7 network (Simonyan & Zisserman, 2015) for all our experiments.

Data augmentation We perform the standard random crop with 4-pixel padding followed by a random horizontal flip and a random rotation of maximum 15° for all clients.

Baselines We compare our method *BiTAT* against following FL baselines: **FedAvg** (McMahan et al., 2017): A popular federated learning method that performs simple averaging of the local models at the server at each round. **FedProx** (Li et al., 2018): A federated learning method that aims to deal with device heterogeneity, with additional ℓ_2 distance regularization terms during the update of local models to prevent the model divergence. **FedPAQ** (Reisizadeh et al., 2020): A quantized parameter communication (QPC)-based FL method, which at each round quantizes the difference between the current local weights and the last aggregated weights at each client, then sends it to the server for aggregation. **FedCOM** (Haddadpour et al., 2021): An extension of FedPAQ with a learnable global learning rate, which achieves faster convergence in data-homogeneous settings. **FedCOMGATE** (Haddadpour et al., 2021): An extension of FedCOM to data-heterogeneous settings, which uses a correction vector that constrains the local models to evolve in a similar direction. **FedGroupedAvg**: A variant of FedAvg in which the server separately aggregates weights only from the same bitwidth clients, and redistributes them to corresponding local clients. **FedGroupedAvg-Asymmetric**: A modified version of FedGroupedAvg that sends clients only the aggregated weights of other clients that has the same bitwidth or higher.

Due to the page limit, we provide the details of hyperparameters for baselines and ours in Appendix A.

5.1. Quantitative Evaluation

We validate our methods under multiple BHFL scenarios with heterogeneous proportions of bitwidths among the clients. We first report the experimental results with 50% of Int8 and 50% Float32 clients (Left) and 80% of Int8 and 20% Float32 clients (Right) in Table 2. Int8 models in FedAvg obtain superior performance to local training, where each client trains independently on its local task due to positive knowledge transfer from the full-precision weights. This is also evident in the poor performance of Int8 clients in FedGroupedAvg which demonstrates that FL only with

low-bitwidth clients is highly limited due to the lack of information the low-bit weights provide. QPC-based FL methods, FedPAQ, FedCOM, and FedCOMGATE, communicate the quantized form of accumulated gradients at each round, and the server adds the accumulated gradients to the global model at each round, before broadcasting it to the clients. Such gradient communication does not degrade the local task information during FL, which is helpful for Float32 models to mitigate the interference from the low-bit weights. However, the low-bit clients (e.g., Int8) cannot directly amalgamate the expressive knowledge from the high-bitwidth weights, easily falling into suboptimal local minima. FedGroupedAvg variants, while preserving the performance of the high-bit models, they often obtain inferior performance as they do not exploit the full knowledge provided by other models. On the other hand, our ProWD consistently outperforms all baselines with varying compositions of bitwidths, yielding small performance gap between low- and high-bitwidths, which demonstrates the effectiveness of the proposed progressive weight dequantization scheme with selective weight aggregation. The convergence plot in Figure 4 shows that our method rapidly converges to good performance while baselines converge to suboptimal local minima.

We further demonstrate the versatility of the ProWD framework under a BHFL scenario with multiple bitwidths in Table 3, using devices with diverse bitwidths. Note that we allow local clients to send the parameters with local bitwidth for this experiment, rather than ternarizing them for uplink communication. Int6 and Int8 are considered as low-bitwidths and Int12 and Int16 are considered as high-bitwidths. Our ProWD consistently outperforms baselines with any bitwidths while achieving a small accuracy gap between Int6 and Int16 clients. We expect that the reduced performance gap between ours and FedAvg is due the smaller disparity between weight distributions compared to those in Table 2 (Int8↔Float32), since all clients perform low-bit operations with slightly different bitwidths, in which case FedAvg may suffer less from distributional shift at aggregation. To validate that, we further provide additional experiments with larger variance among the bitwidths of the participating devices in Figure 5, which shows a significant performance gain of our ProWD to FedAvg (Int6 clients acc: **5.9**%↑, average acc: **2.7**%↑, performance gap: **36**%↓).

Ablation study Now we explicate the effect of each ingredient of our ProWD on the CIFAR-10 with an ablation study. We experiment on both the uniform (50% of Float32 and 50% of Int8 clients) and low-bitwidth dominant scenario (20% of Float32 and 80% of Int8 clients). As shown in Table 4, thanks to its ability to reconstruct full-precision weights from low-bitwidth weights, our weight dequantizer (Deq) improves the performance by 5.5% and 2.8% over that of FedAvg, respectively for each scenario. Also, our selective weight aggregation (SWA) minimizes the discrepancy across the model updates from different bitwidth clients, obtaining the significant performance gain of 9.2% and 2.6% for each scenario. This experimental result confirms the efficacy of both components, progressive weight decomposition, and selective weight aggregation.

Table 3: Average accuracy at each bitwidth, and across all clients on CIFAR-10. We set clients with 30% of Int6, 30% of Int8, 20% of Int12, and 20% of Int16 models. All the results are measured by computing and standard deviation over three independent runs.

METHOD	INT6	INT8	INT12	INT16	GAP	AVGACC
LOCAL TRAINING	69.51	69.25	68.96	70.40	+0.89	69.50
FEDAVG	80.17	85.71	**86.49**	87.02	+6.85	84.47
FEDPAQ	80.78	85.30	85.98	86.93	+6.15	84.40
FEDCOM	39.47	55.14	53.07	53.26	+13.79	49.65
FEDCOMGATE	56.74	68.43	67.63	68.14	+11.69	64.70
FEDGROUPEDAVG	79.27	80.62	76.94	77.39	-1.88	78.83
FEDGROUP-ASYM	80.25	79.74	79.18	77.73	-2.52	79.38
PROWD (OURS)	**82.89**	**86.11**	**86.47**	**87.51**	+4.62	**85.50**

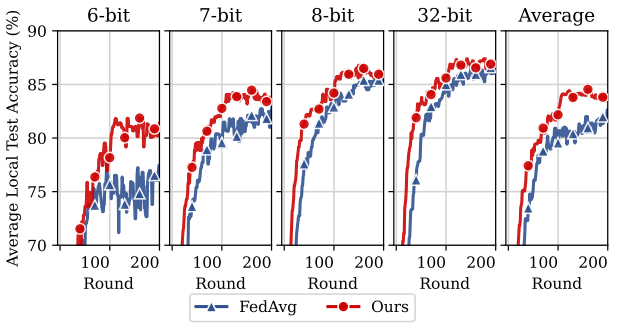


Figure 5: BHFL with Int6, Int7, Int8, and Float32 clients.

5.2. Qualitative Analysis

The effect of progressive weight dequantization To further analyze the role of the progressive weight dequantizer in our method, we visualize stepwise distributions of reconstructed weights using our dequantizer. For ease of interpretation, we use the initial input as the quantized weight from the last convolution layer in the Int8 client for training on CIFAR-10, where the result is illustrated in Figure 6. As low-bitwidth clients transmit ternarized model weights to the central server, the distribution of input weights to the dequantizer is visualized in three peaks. We also visualize the distribution from reconstructed weights after forwarding initial weights up to the first, second, and last block in our dequantizer, colored by *orange*, *pink*, and *navy*, respectively. As we expected, our dequantizer progressively recovers ternary weights towards the high-bitwidths by forwarding them through sequential dequantization blocks.

Weight distance between local clients Next, we visualize the distance of model weights between clients in Figure 7 to dissect the difference of learned representations

Table 4: Ablation study for ProWD. DEQ and SWA refer the progressive weight dequantizer and selevtive weight aggregation, respectively. We report the results over three independent runs.

METHOD			**CIFAR-10**, INT8 (50%) - FLOAT32 (50%)		
	DEQ	SWA	INT8 ACC	FLOAT32 ACC	AVERAGE
FEDAVG	–	–	76.55 (± 0.44)	75.71 (± 0.32)	75.83 (± 0.41)
+DEQ	✓	–	79.60 (± 0.39)	80.38 (± 0.44)	79.99 (± 0.26)
+SWA	–	✓	79.79 (± 0.98)	**85.94** (± 0.92)	82.84 (± 0.55)
OURS	✓	✓	**82.87** (± 0.37)	**85.99** (± 0.20)	**84.43** (± 0.22)
METHOD			**CIFAR-10**, INT8 (80%) - FLOAT32 (20%)		
	DEQ	SWA	INT8 ACC	FLOAT32 ACC	AVERAGE
FEDAVG	–	–	77.43 (± 0.83)	74.64 (± 1.21)	76.87 (± 0.87)
+DEQ	✓	–	78.52 (± 0.50)	76.08 (± 0.44)	79.03 (± 0.31)
+SWA	–	✓	78.42 (± 0.47)	80.82 (± 0.31)	78.90 (± 0.37)
OURS	✓	✓	**79.23** (± 0.25)	**81.26** (± 0.45)	**79.63** (± 0.23)

during BHFL. We use the cosine distance, computed by $1 - \cos\_\text{sim}(\boldsymbol{w}_n, \boldsymbol{w}_m)$ between n- and m-th local clients. Weights obtained using FedAvg have smaller distances between Int8 and Float32 models than the distances among the Int8 models, caused by the distributional shift of the high-bitwidth model weights towards the distribution of low-bitwidth model weights (Please see Figure 2).

Clients in FedPAQ communicate accumulated local gradients while keeping their local weights, alleviating the detrimental distributional shift of weights to some degree, resulting in a bigger similarity among the same bitwidth clients than in bit-heterogeneous cases. Int8 client weights in FedGroupedAvg stay close to each other while straying far from the Float32 client weights, which is expected as it only allows communication among clients with the same bitwidth. Interestingly, ProWD keeps the clients' weights sufficiently different between each bitwidthobtains sufficiently low proximity across bit-heterogeneous clients, successfully preventing the distributional shift in high-precision weights due to weight averaging. Our proposed method also allows high transferability of the learned knowledge across clients that shows better adaptation on the local tasks over QPC-based methods (Please see Table 2 and Figure 4). We provide more similarity analysis of the local models for other baselines and our models in Appendix D.

6. Conclusion

We proposed a novel yet practical heterogeneous federated learning scenario where the participating devices have different bitwidth specifications, in which case the model aggregation could have a highly detrimental effect. We further identify the two main causes of the performance degeneration, which are the suboptimal loss convergence due to distributional shift from aggregation, and the limited expressive power of the low-bitwidth weights. To tackle these challenges, we proposed a novel framework for bit-heterogeneous FL, based on progressive dequantization of the weights and selective weight aggregation. The progressive dequantizer at the server receives weights from low-bitwidth clients and recovers them into higher bitwidth weights by forwarding them through a sequence of dequantizer blocks. Further, selective weight aggregation determines which low-bit weight elements are compatible with higher-bit ones. Empirical evaluations of our framework across two BHFL scenarios with varying degrees of bitwidth-heterogeneity on the benchmark dataset demonstrate the effectiveness of our framework, which largely outperforms relevant FL baselines.

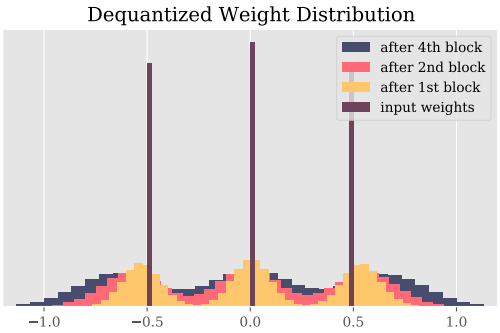


Figure 6: **The Weight Distribution after Progressive Dequantization.** Visualization of the distribution after the reconstruction of low-bitwidth model weights. We visualize the last Convolution layer weights in neural network, trained on CIFAR-10.

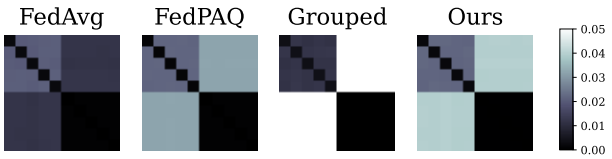


Figure 7: **Cosine distance matrix between the weights of each client.** Elements in a row and column describe the index of local clients. We set the first five clients to Int8 (Top left) and the other five to Float32 (Bottom right) for training on CIFAR-10. Darker colors indicate a bigger similarity.

7. Acknowledgement

This work is supported by Samsung Advanced Institute of Technology, Institute of Information communications Technology Planning Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2019-0-00075, Artificial Intelligence Graduate School Program(KAIST)), and the Engineering Research Center Program through the National Research Foundation of Korea (NRF) funded by the Korean Government MSIT (NRF-2018R1A5A1059921).

References

Chen, Y., Sun, X., and Jin, Y. Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation. *IEEE trans-*

actions on neural networks and learning systems, 2019.

Diao, E., Ding, J., and Tarokh, V. HeteroFL: Computation and communication efficient federated learning for heterogeneous clients. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Fallah, A., Mokhtari, A., and Ozdaglar, A. Personalized federated learning: A meta-learning approach. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., and Keutzer, K. A survey of quantization methods for efficient neural network inference. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Haddadpour, F., Kamani, M. M., Mokhtari, A., and Mahdavi, M. Federated learning with compression: Unified analysis and sharp guarantees. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2021.

He, C., Annavaram, M., and Avestimehr, S. Group knowledge transfer: Federated learning of large cnns at the edge. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

Iandola, F., Moskewicz, M., Karayev, S., Girshick, R., Darrell, T., and Keutzer, K. Densenet: Implementing efficient convnet descriptor pyramids. *arXiv preprint arXiv:1404.1869*, 2014.

Jeong, W., Yoon, J., Yang, E., and Hwang, S. J. Federated semi-supervised learning with inter-client consistency and disjoint learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Jiang, D., Shan, C., and Zhang, Z. Federated learning algorithm based on knowledge distillation. In *2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE)*. IEEE, 2020.

Jiang, Y., Konečnỳ, J., Rush, K., and Kannan, S. Improving federated learning personalization via model agnostic meta learning. *arXiv preprint arXiv:1909.12488*, 2019.

Lee, D., Kwon, S. J., Kim, B., Jeon, Y., Park, B., and Yun, J. FleXOR: Trainable fractional quantization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Li, F., Zhang, B., and Liu, B. Ternary weight networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

Lin, T., Kong, L., Stich, S. U., and Jaggi, M. Ensemble distillation for robust model fusion in federated learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

McMahan, H. B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

Mei, G., Guo, Z., Liu, S., and Pan, L. Sgnn: A graph neural network based federated learning approach by hiding structure. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019.

Nielsen, D. and Winther, O. Closing the dequantization gap: PixelCNN as a single-layer flow. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016.

Reisizadeh, A., Mokhtari, A., Hassani, H., Jadbabaie, A., and Pedarsani, R. FedPAQ: A Communication-Efficient Federated Learning Method with Periodic Averaging and Quantization. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

Salimans, T. and Kingma, D. P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

Sun, X., Wang, N., Chen, C.-y., and Ni, J.-m. Ultra-low precision 4-bit training of deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Wang, H., Yurochkin, M., Sun, Y., Papailiopoulos, D., and Khazaeni, Y. Federated learning with matched averaging. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

Wen, W., Xu, C., Yan, F., Wu, C., Wang, Y., Chen, Y., and Li, H. TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

Wu, C., Wu, F., Cao, Y., Huang, Y., and Xie, X. Fedgnn: Federated graph neural network for privacy-preserving recommendation. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2021.

Wu, S., Li, G., Chen, F., and Shi, L. Training and Inference with Integers in Deep Neural Networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

Xing, Y., Qian, Z., and Chen, Q. Invertible image signal processing. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Yoon, J., Jeong, W., Lee, G., Yang, E., and Hwang, S. J. Federated continual learning with weighted inter-client transfer. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2021.

Zhang, X., Liu, S., Zhang, R., Liu, C., Huang, D., Zhou, S., Guo, J., Guo, Q., Du, Z., Zhi, T., and Chen, Y. Fixed-Point Back-Propagation Training. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Zhao, K., Huang, S., Pan, P., Li, Y., Zhang, Y., Gu, Z., and Xu, Y. Distribution Adaptive INT8 Quantization for Training CNNs. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., and Chandra, V. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., and Zou, Y. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv:1606.06160 [cs]*, 2018.

Zhu, F., Gong, R., Yu, F., Liu, X., Wang, Y., Li, Z., Yang, X., and Yan, J. Towards Unified INT8 Training for Convolutional Neural Network. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

A. Hyperparameters

At each round, we train each client for 200 local steps. The Float32 network is trained with SGD with learning rate 0.1, and momentum value 0.9. Additionally, the gradient ℓ_2 norm is clipped to 2.0. The quantized fixed-point (Int2, Int4, Int8, Int12, and Int16) networks are trained with the low-bitwidth training method detailed in Appendix B, with $\eta = 8.0$. For FedProx, we explored several l2 coefficients (0.01, 0.1, and 1), and found that 0.01 had the best final test accuracy. For FedCOM and FedCOMGATE, we use the global step size $\gamma = 10$, since Haddadpour et al. (2021) only specifies that γ should be greater than or equal to the number of clients participating in the FL process. For FedGroupedAvg and FedGroupedAvg-Asymmetric, for fair comparison, we scale down the local learning rates by the number of clients with the same bitwidth divided by the total number of clients, because the number of clients participating in the aggregation affects the convergence speed of the FL process proportionately (Haddadpour et al., 2021). We train a weight dequantizer ϕ with a SGD optimizer with the learning rate of 0.01, batch size of 16, for 5 epochs for all experiments.

In the experiments in Table 2, we use $\Pi = \{\text{Int2}, \text{Int4}, \text{Int8}, \text{Int16}, \text{Float32}\}$ for the dequantizer blocks, whereas in the experiment in Table 3, we use $\Pi = \{\text{Int6}, \text{Int8}, \text{Int10}, \text{Int12}, \text{Int16}\}$.

B. Low-bitwidth Training for Bit-limited Hardware Devices

Weight initialization. In order to prevent weights from vanishing due to the intermediate ternarization, we adjust the scale of the initialization as follows:

$$\boldsymbol{w}_q \sim \mathcal{U}(-L, L), \tag{10}$$

where $L = \max\{0.75, \sqrt{3/\text{fan\_in}_l}\}$. Note that when $L = \sqrt{3/\text{fan\_in}_l}$, it is equivalent to the Kaiming uniform initialization. The layer-wise scaling factor α^l is defined as follows:

$$\alpha^l = \text{Shift}(0.75/\sqrt{3/\text{fan\_in}_l}). \tag{11}$$

This modified initialization strategy has consequences for the full-precision model, since the scale of the weights aggregated are not the same across bitwidths. We alleviate this problem by initializing the full-precision model weights with the same distribution as the quantized model weights, and using Weight Normalization (Salimans & Kingma, 2016) in the full-precision clients to compensate for the scale difference.

Backward pass. The errors are calculated by using the chain rule, except we normalize the errors at each layer to prevent saturation. Specifically, we apply the following before propagating the error value to every subsequent layer in the chain rule:

$$\boldsymbol{e}_q = Q_s(\boldsymbol{e}/\text{Shift}(\max\{|\boldsymbol{e}|\})), \tag{12}$$

where $\text{Shift}(x) = 2^{\lceil \log_2 x \rfloor}$ finds the nearest power-of-two to the input, and $\max\{|\boldsymbol{e}|\})$ represents the layer-wise maximum absolute value among the elements of the error $\boldsymbol{e}$, and s is the bitwidth of the client. The quantizer function is defined as

$$Q_s(x) := \text{clip}_s(\lceil (2^{s-1} \cdot x) \rfloor / 2^{s-1}), \tag{13}$$

$$\text{clip}_s(x) := \max(\min(x, (2^{s-1} - 1)/2^{s-1}), (-2^{s-1} + 1)/2^{s-1}). \tag{14}$$

For the weight update, we similarly apply the following rescaling operation to the gradient value before applying the weight update:

$$\boldsymbol{q} \leftarrow \text{clip}_s(\boldsymbol{q} - Q_s^{\text{stoch}}(\eta \cdot \boldsymbol{g}/\text{Shift}(max\{|\boldsymbol{g}|\}))), \tag{15}$$

where $Q_{s_n}^{\text{stoch}}(\cdot)$ is a stochastic quantization function defined elementwise as follows:

$$Q_s^{\text{stoch}}(x) = \begin{cases} 2^{1-s} \cdot \lceil |x| \rceil & \text{w.p. } |x| - \lfloor |x| \rfloor \\ 2^{1-s} \cdot \lfloor |x| \rfloor & \text{otherwise.} \end{cases} \tag{16}$$

Note that the scale of the learning rate η is different from regular full-precision network training, because of the rescaling of the gradient values.

C. Training of Progressive Weight Dequantizer

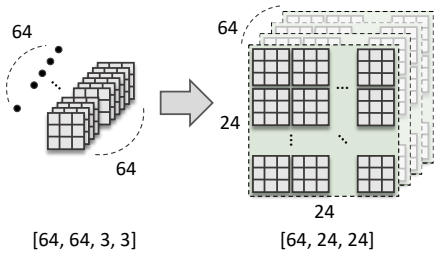


Figure 8: **Illustration of a reshaping process on the weight module.**

Construction of the weights dataset. Given the local model weights $\boldsymbol{w}$, we construct the weight datasets to learn the progressive weight dequantizer. Since layers in deep neural networks is often composed of the weights with different dimensionality each other, we split them into the uniformly-sized subweights. As following common structures for CNN models that mostly composed of a number of convolution layers, such as VGG (Simonyan & Zisserman, 2015) and ResNet (He et al., 2016), we basically utilize convolution weights with a filter size of 3×3 and input dimension is 64 or larger to construct the weight dataset. That is, we use all convolution weights except the weights from the first layer (input dimension is the channel of Image, 3). We split the weights at each convolution layer to partial modules with the shape of $64 \times 64 \times 3 \times 3$, (e.g., the weights with the shape of $256 \times 128 \times 3 \times 3$ is splitted to $4 \cdot 2 = 8$ different modules. Next, we reshape each module sized by 24×24 with 64 channels (i.e., $64 \times 24 \times 24$) to as illustrated in Figure 8.

Block design for progressive weight dequantizer. We implement our dimensionality-preserving network block ϕ for our dequantizer using two layered of affine coupling layers, and the design of each layer ρ is as follows:

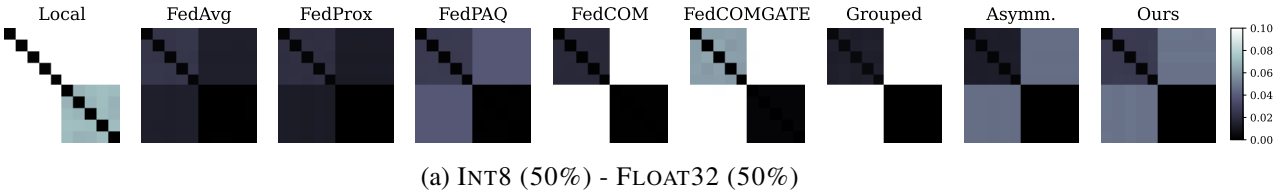
$$\begin{aligned} \widehat{\boldsymbol{w}}_{1:d} &= \boldsymbol{w}_{1:d} + \alpha(\boldsymbol{w}_{d+1:D}), \\ \widehat{\boldsymbol{w}}_{d+1:D} &= \boldsymbol{w}_{d+1:D} \odot \exp\left(\beta(\widehat{\boldsymbol{w}}_{1:d})\right) + \gamma(\widehat{\boldsymbol{w}}_{1:d}), \end{aligned} \tag{17}$$

where $\widehat{\boldsymbol{w}} = \rho(\boldsymbol{w})$, α, β, and γ are DenseNet (Iandola et al., 2014) blocks while we omit the notation of weights in each layer for readability. To this end, a j^{th} network block of progressive weight dequantizer is formulated as follows:

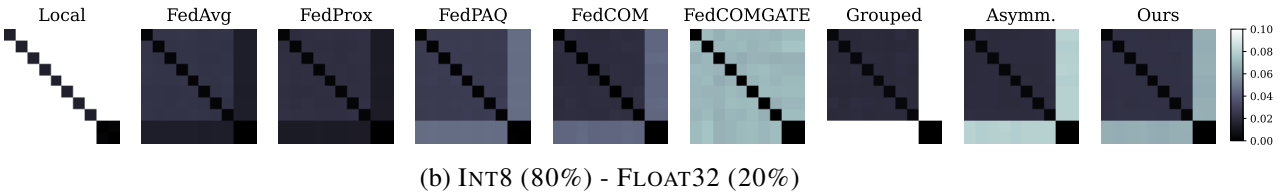
$$\widehat{\boldsymbol{w}} = \phi^{\pi_j \to \pi_{j+1}}(\boldsymbol{w}; \boldsymbol{\theta}_j) = \boldsymbol{w} + \tau\rho(\rho(\widehat{\boldsymbol{w}})), \tag{18}$$

where τ is a scaling coefficient and we set $\tau = 0.1$ for all experiments. We want to note that there is a huge potential to further develop the design of our dequantizer. We intend to suggest a better design for the dequantization function for recovering high-bitwidth weights from low-bitwidth weights in future work.

D. Additional Experiments



(a) INT8 (50%) - FLOAT32 (50%)



(b) INT8 (80%) - FLOAT32 (20%)

Figure 9: **Cosine distance matrix between the weights of each client.** Elements in a row and column describe the index of local clients. **(a)** the first five clients correspond to Int8 (Top left) and the other five to Float32 (Bottom right). **(b)** the first eight clients correspond to Int8, and the other two to Float32. Darker colors indicate higher similarities between clients.

Weight similarity anlaysis. We further provide the cosine distance matrix between the weights of each client for all baselines in Figure 9. Each client in the local training (Local) model does not share the knowledge with other clients, resulting in a large weight distance. FedProx (Li et al., 2018) shows similar tendency with FedAvg. Interestingly, the Int8 clients in FedCOMGATE show different behaviors with FedCOM, with the weight distance among them farther than any other BHFL methods. This phenomenon is due to the "correction vectors" utilized in the FedCOMGATE algorithm designed to adapt to data heterogeniety and enable the weights to behave semi-independently. This seems to improve the model accuracy compared to FedCOM with a marginal amount, as shown in Table 2. While FedGroupedAvg-Asymmetric (Asym.) shows a similar distance matrix with FedGroupedAvg (Grouped), it allows the knowledge transfer from-high-to-low-bitwidths clients, performing higher weight similarity between bitwidth heterogeneous clients.