This document is published by IEEE Transactions on Multimedia:

H. Afrabandpey and H. R. Tavakoli, ”P 2 U: Progressive Precision Update For Efficient Model Distribution,” in IEEE Transactions on Multimedia, doi: 10.1109/TMM.2025.3642916.

# P$^2$U: Progressive Precision Update For Efficient Model Distribution

Homayun Afrabandpey, Hamed Rezazadegan Tavakoli

***Abstract*—Efficient model distribution is becoming increasingly critical in bandwidth-constrained environments. In this paper, we propose a simple yet effective approach called Progressive Precision Update (P$^2$U) to address this problem. Instead of directly transmitting the original high-precision model, P$^2$U transmits a lower-bit precision model, coupled with a model update representing the difference between the original high-precision model and the transmitted low precision version. With extensive experiments on various model architectures, ranging from small models ($1-6$ million parameters) to a large model (more than $100$ million parameters) and using three different data sets, e.g., chest X-Ray, PASCAL-VOC, and CIFAR-100, we demonstrate that P$^2$U consistently achieves a better tradeoff between accuracy, bandwidth usage, and startup latency, i.e., the time it takes for the receiver to start inference. Moreover, we show that when bandwidth or startup time is the priority, aggressive quantization (e.g., 4-bit) can be used without severely compromising performance. These results establish P$^2$U as a practical solution for scalable and efficient model distribution across distributed environments such as federated learning, edge computing, and IoT deployments. Given that P$^2$U complements existing compression techniques and can be implemented alongside many compression methods, e.g., sparsification, quantization, pruning, etc., the potential for improvement is even greater.**

***Index Terms*—Efficient model distribution, Low-latency deployment, Progressive precision update**

## I. Introduction

WITH the increasing complexity and explosive growth in the size of Machine Learning (ML) models, efficient model transmission plays a key role in applications with distributed and resource-constrained nature such as Internet of Things (IoT) and Federated Learning (FL). Transmitting large models in their original high-bit precision, e.g., 32-bit float, across networks with limited bandwidth can result in substantial communication overhead while maintaining the highest possible accuracy. Another issue is the increased startup latency, i.e., the time it takes for the receiver to start inference. Increasing startup latency can have detrimental effects on a user's Quality of Experience (QoE). As an example, assume a user starts an app on their edge device, e.g., mobile phone, to perform real-time object detection in the video stream captured by the device camera, for example in an Augmented Reality (AR) use case. The app may use a pre-trained model for the task. An example of such a model may be VGG16 [1]. A pre-trained VGG16 can be up to $528$ Mega Bytes (MB) in size, which can take substantial time to be downloaded from a service provider model repository, even if the repository is hosted in the 5G network. For example, on a link with an average capacity of $100$Mbps with a low access latency of $10$ms, it may take up to a minute for the model to be downloaded. One minute of startup latency can be quite detrimental in a latency-sensitive scenario such as AR.

In response to this issue, a large body of literature exists on different types of compression techniques including sparsification [2]–[4], pruning [5], [6], quantization [7]–[9] and encoding [10], [11]. These techniques generally apply transformations to the model weights or architecture to reduce the model size before transmission, aiming to achieve a smaller model that can be transmitted in one go. This way they sacrifice accuracy for bandwidth usage while also decreasing startup latency. In this paper, looking at the problem from a different angle, we propose a novel approach called Progressive Precision Update (P$^2$U) that focuses on model compression *during* the transmission process (not prior to that). P$^2$U centers on the concept of precision-driven model transmission. Instead of compressing and sending the full precision model directly, we advocate for the initial transmission of a compressed lower precision version of the model, for instance, an 8-bit integer model. This lower-precision model serves as a compressed representation that captures the essential characteristics of the original model while drastically reducing the amount of transmitted data. The receiver can start inference upon receiving this low-precision model. To bridge the accuracy degradation due to the precision gap, we then transmit a compressed model update(s), representing the discrepancy between the original high-precision model and its lower-precision counterpart. The motivation behind this approach is three-fold. First, it addresses the bandwidth limitations inherent in distributed learning scenarios, providing a more lightweight alternative for model transmission. Second, the reduction in the model size and effective information transfer leads to decreased startup latency. This makes our approach particularly suitable for real-time applications. Third, P$^2$U provides fine-grained control over the transmission process, allowing for tailored updates based on available bandwidth or specific accuracy requirements. It is important to note that P$^2$U is complementary to model compression techniques and could be used in conjunction with them to further improve bandwidth saving.

Our method is particularly effective at navigating the tradeoff between accuracy, bandwidth usage, and startup latency. For instance, in the case of MobileNet-v2 and VGG16 on the PASCAL-VOC dataset, P$^2$U with an 8-bit low-precision model outperforms even the 16-bit quantized baseline in terms of accuracy, while maintaining significantly lower communication cost and startup time. This confirms that the transmitted lower-precision model acts as an efficient approximation, capturing essential features of the original model. We present a comprehensive analysis of our method using diverse data sets and model architecture settings, showcasing its efficacy across various scenarios. Our findings emphasize the potential

H. Afrabandpey was with the Nokia Technologies, Finland. Email: homayun.afrabandpey@nokia.com
Hamed R. Tavakoli is with the Nokia Technologies, Finland. Email: hamed.rezazadegan_tavakoli@nokia.com

of precision-driven model transmission as a viable solution for bandwidth optimization in distributed learning, paving the way for more scalable and responsive systems in resource-constrained environments.

## II. Related Works

Several lines of research have addressed the challenge of efficient model distribution and inference in bandwidth-constrained or latency-sensitive environments. Broadly, existing approaches can be categorized into model compression techniques, progressive transmission schemes, and federated learning compression methods.

Model compression itself encompasses a range of strategies, including quantization, pruning, sparsification, tensor decomposition, knowledge distillation, and hybrid combinations of these techniques.

Quantization [12] is one of the most extensively studied model compression techniques, aiming to reduce the number of bits required to represent weights and activations. Approaches range from classic signal processing methods, such as uniform quantization, to more advanced schemes like parameter and data scaling [13], as well as a wide variety of recent innovations [14]–[16]. These techniques have enabled compression to extreme levels, including ternary, binary, and even 1-bit representations [17]–[19]. However, such aggressive quantization often incurs a substantial performance gap compared to the original high-precision model. A large body of work has sought to bridge this gap, particularly for binary networks, achieving varying degrees of success [20]–[22]. While these methods achieve good compression ratios, they offer limited flexibility: once the model is quantized to a certain precision level, it must be distributed and used as-is. They do not support progressive quality upgrades or adaptive tradeoffs between accuracy and transmission cost at runtime. A P$^2$U related line of research in this area is quantization with adaptive bit-widths [23]. This approach aims at the adaptive execution of models at different bit-widths and requires training quantized neural networks with different precisions adaptive to different requirements, while our method aims for efficient model distribution and inference.

Pruning and sparsification are two other extensively researched techniques in recent years [24]–[29]. Matrix decomposition-based approaches are yet another favored approach for reducing the size of a neural network for transfer. Familiar with matrix decomposition methods from linear algebra, one may decompose a weight tensor into smaller matrices. Various works have explored this aspect [30]–[32].

Another related family of methods includes model distillation and proxy-based distribution approaches [33], [34], which use smaller student models or surrogate networks with different structure to reduce deployment cost. These approaches typically prioritize inference efficiency but often sacrifice accuracy or require expensive teacher-student training pipelines. P$^2$U, by contrast, uses a single model architecture and progressively improves it without retraining.

Progressive transmission methods, such as BitSplit [35] and progressive pruning [36], aim to deliver model components in multiple stages to gradually refine performance. While they support staged deployment, they often require specialized architectures or substantial retraining. Moreover, they rarely account for the importance of minimizing startup latency, as they focus on full model reconstruction eventually. In contrast, P$^2$U is model-agnostic and designed to enable useful inference even from a low-precision base model, allowing quick deployment followed by progressive refinement.

The demand for neural model compression is significant that there has been standardization activities related to the topic of neural compression [37]. The neural model compression standard ensures an interoperable mechanism for delivering compressed neural networks. The pipeline consists of means for sparsification, pruning, quantization, and entropy coding of a model. It furthermore ensures mechanisms for encoding incremental updates, similar to federated learning, are supported. P$^2$U builds on top of the capabilities of the neural model compression standard to demonstrate the effectiveness of incremental delivery of a compressed model.

A relevant, yet distinct, domain is compression of federated communications [38]. These methods explore how to transmit model updates efficiently under communication constraints. For example, [39], [40] consider communication channel characteristics during federated learning to reduce the amount of bits required for transmitting a model. Other approaches to compressing federated communications may follow the principles of compression of neural networks for deployment, i.e., sparsification/pruning, quantization, and entropy coding, e.g., [41]–[43]. Our approach may share conceptual commonalities with these methods where weight updates are communicated between entities. While they optimize bandwidth during training across distributed clients, they do not directly address startup latency or allow inference from intermediate representations. P$^2$U brings these ideas to the inference stage by allowing the user to begin working with a low-precision model almost immediately, followed by incremental improvements as higher-precision updates are received.

Another related concept is incremental model transfer, which is defined as determining a split point that allows partial transfer of the model and partial execution of the computational graph akin to the split learning and inference [44] until the complete model is transferred and executed. P$^2$U is orthogonal to the incremental model transfer. In our approach, the model is deployed completely at different bit increments that allow enabling bit-incremental model deployment.

In summary, while prior works make important contributions toward reducing model size, preserving model performance, or reducing inference latency, they typically optimize along a single dimension. In contrast, P$^2$U is designed to operate in a tri-objective space: achieving a balance between accuracy, bandwidth efficiency, and startup latency. As demonstrated in our experiments, P$^2$U enables flexible deployment strategies, supporting rapid startup with coarse models and seamless refinement toward high-accuracy solutions – making it particularly suitable for resource-constrained or latency-sensitive applications.

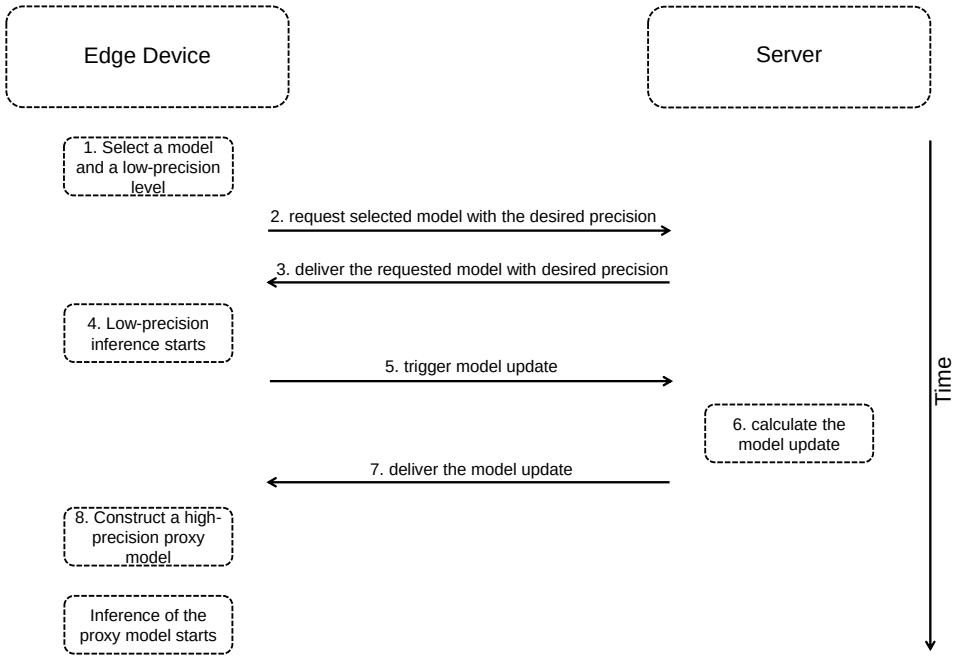


Fig. 1. Workflow of P$^2$U. After receiving the request for a ML model with a specific low-precision level from the edge device, the server first delivers the model with the requested precision level and then sends an update as the difference between the original (high-precision) model and the delivered low-precision model.

## III. Method

Imagine a smart city equipped with an advanced traffic management system that leverages ML to optimize traffic flow, reduce congestion, and minimize environmental impact. As a new car enters the city, equipped with IoT sensors and communication capabilities, it initiates communication with the central traffic management server. The car's onboard systems seek an ML model that can dynamically adapt to real-time traffic conditions, predict congestion patterns, and optimize the vehicle's navigation route to reduce travel time and energy consumption. In this scenario, a big challenge lies in the efficient transmission of the ML model to the new vehicle, considering potential bandwidth limitations and the need for rapid inference. Models must be delivered quickly and with minimal communication overhead, while still preserving inference accuracy. To address this, we propose P$^2$U, which aims to reduce bandwidth consumption during model distribution and decrease startup latency in latency-critical applications.

Since compressed model transmission is usually done in the quantized domain, we assume that the model requested by the edge device is either available in the server in quantized form with different precisions or the server is able to quantize the model to the required precisions. Figure 1 graphically demonstrates the workflow of P$^2$U. P$^2$U works in steps as follows:

1. a model and a desired low-precision level is selected by the edge device based on, e.g., device resources, network conditions, etc.,
2. the edge device sends a request for the selected model with the desired low-precision level to the server,
3. the server identifies the selected model in its model repository and prepares the requested precision if it is not already available in the repository and sends the model with the requested precision to the edge device,
4. the edge device starts inferencing upon receiving the low-precision model,
5. the edge device triggers a model update (parallel to the inferencing process of step 4). The update is a precision update of the model currently at the edge device rather than a new model,
6. the server computes the model update as the difference between the original high-precision and transferred low-precision model,
7. the model update is sent to the edge device,
8. the edge device constructs a proxy to the original high-precision model in the server by adding the update to the low-precision model and adopts this high-precision proxy for the inference.

In step 5, the edge device may trigger a model update immediately after receiving the model or it may trigger the update after a period of time. The update may be triggered based on a variety of factors including, but not limited to, model update delivery time, accuracy achieved at the edge device using the low-precision model, prospective accuracy improvement achievable with a model update, change in accuracy requirements, etc. The following subsection provides a theoretical proof that, under certain assumptions, the accuracy loss between the original high-precision model and its proxy

constructed in the edge device by adding the update to the low-precision model, is minimal.

### A. Theoretical Analysis

Consider a neural network with high precision, e.g., 32-bit int, and low precision, e.g., 8-bit int, quantized weights, denoted as $\boldsymbol{W}_Q^h$ and $\boldsymbol{W}_Q^l$, respectively. Let $\boldsymbol{W}^h$ and $\boldsymbol{W}^l$ be the corresponding dequantized 32-bit float versions of these weights. The update is defined as:

$$\Delta \boldsymbol{W} = \boldsymbol{W}^h - \boldsymbol{W}^l, \tag{1}$$

with its quantized version being shown as $\Delta \boldsymbol{W}_Q$. At the receiver, we construct a high-precision proxy for $\boldsymbol{W}^h$ as $\boldsymbol{W}' = \boldsymbol{W}^l + \Delta \boldsymbol{W}$. Given an input $\boldsymbol{x}$, the outputs of the high-precision model and low-precision model are denoted as $f\left(\boldsymbol{W}^h, \boldsymbol{x}\right)$ and $f\left(\boldsymbol{W}^l, \boldsymbol{x}\right)$, respectively. The error induced by using the low-precision model for inference is:

$$\epsilon\left(x\right) = f\left(\boldsymbol{W}^h, \boldsymbol{x}\right) - f\left(\boldsymbol{W}^l, \boldsymbol{x}\right). \tag{2}$$

Our goal is to show that the high-precision proxy model $\boldsymbol{W}'$ compensates for the error $\epsilon\left(\boldsymbol{x}\right)$. In other words, we aim to show that $f\left(\boldsymbol{W}', \boldsymbol{x}\right) \approx f\left(\boldsymbol{W}^h, \boldsymbol{x}\right)$.

We analyze the behavior of the function $f$ using a first-order Taylor expansion. Assuming $f$ is smooth and differentiable, we expand $f\left(\boldsymbol{W}^h, \boldsymbol{x}\right)$ around $\boldsymbol{W}^l$:

$$f\left(\boldsymbol{W}^h, \boldsymbol{x}\right) \approx f\left(\boldsymbol{W}^l, \boldsymbol{x}\right) + \nabla f\left(\boldsymbol{W}^l, \boldsymbol{x}\right).\left(\boldsymbol{W}^h - \boldsymbol{W}^l\right) + R_h, \tag{3}$$

where $R_h$ represents high-order residual terms. Similarly, expanding $f\left(\boldsymbol{W}', \boldsymbol{x}\right)$ around $\boldsymbol{W}^l$, we get:

$$\begin{aligned} f\left(\boldsymbol{W}', \boldsymbol{x}\right) = f\left(\boldsymbol{W}^l + \Delta \boldsymbol{W}, \boldsymbol{x}\right) \approx f\left(\boldsymbol{W}^l, \boldsymbol{x}\right) + \\ \nabla f\left(\boldsymbol{W}^l, \boldsymbol{x}\right) \cdot \left(\boldsymbol{W}^h - \boldsymbol{W}^l\right) + R'. \end{aligned} \tag{4}$$

Given 3 and 4, we obtain:

$$f\left(\boldsymbol{W}', \boldsymbol{x}\right) \approx f\left(\boldsymbol{W}^h, \boldsymbol{x}\right) + \left(R' - R_h\right).$$

Considering these, the difference between the output of the proxy model and the high-precision model obtains as:

$$|f\left(\boldsymbol{W}', \boldsymbol{x}\right) - f\left(\boldsymbol{W}^h, \boldsymbol{x}\right)| \leq |R' - R_h|. \tag{5}$$

Assuming[1]:

1. $\| \boldsymbol{W}^h - \boldsymbol{W}^l \| \leq \delta$ where $\delta$ is relatively small compared to the scale of the weighs themselves ($\| W^h \|$ and $\| W^l \|$), and
2. $\| R_h \|, \| R' \| \leq \frac{M}{2} \| \boldsymbol{W}^h - \boldsymbol{W}^l \|^2$,

where $\| . \|$ refers to the Euclidean norm. Since the absolute difference of the residual terms, i.e., $|R' - R_h|$, is on the order of $O\left(\delta^2\right)$, the proxy model approximates the high-precision model with high fidelity, leading to negligible inference error in practice.

## IV. Experiments

### A. Experimental Setup

In this subsection, we provide detailed settings of our experiments necessary for reproducing the results.

[1]the first assumption ensures the validity of first-order Taylor approximation, and the second assumption ensures that higher-order terms diminish quadratically, making their effect negligible for small $\delta$

TABLE I
Number of parameters (in Million) and Size (MB) of each model architecture used in the experiment.

| | MobileNet-v2 | ResNet18 | EfficientNet-b4 | VGG16 |
|---|---|---|---|---|
| **Parameters** | 3.5 | 11.7 | 19.4 | 138.4 |
| **Size** | 14 | 41 | 74.5 | 489 |

*1) Data sets:* We perform a set of experiments on three standard image classification data sets:

- Chest X-Ray [45] is a binary image classification data set with in total $5856$ images from which $5232$ images are used for training and the remaining $624$ images are used as test data. From all images in this data set, $1583$ images are "Normal" and the rest are "Pneumonia" cases.
- PASCAL-VOC [46] version 2012, which contains in total $11540$ images from $20$ different classes. We adopted a pre-defined split of the data set with $6925$ images for training, $2308$ images for testing, and the remaining $2307$ images for validation.[2]
- CIFAR-100 [47] consists of $60000$ color images of size $32 \times 32$ from $100$ different classes. There are $50000$ training images, i.e., $500$ training images per class, and $10000$ test images, i.e., $100$ test images per class.

*2) Network Architectures:* We used MobileNet-v2, ResNet18, EfficientNet-b4, and VGG16, all pre-trained on ImageNet and loaded from PyTorch model zoo. Table I demonstrates the number of parameters (in millions) and the model sizes (in MB) when loaded from the PyTorch model zoo. Since these models were originally trained on ImageNet with $224 \times 224$ images and $1000$ classes, directly applying them to data sets with differing number of classes leads to reduced accuracy. To address this, we modified each model by adjusting the output dimension of the final fully connected layer to match the number of classes in the target dataset and when necessary (in the case of CIFAR-100 where image sizes are much smaller than ImageNet) also modified initial feature layer. Finally, to enhance performance, we retrained each model on the training set of the selected dataset before initiating model transfer. During this process, all parameters except those modified in the transformation step were frozen.

*3) Implementation:* We used PyTorch framework for the implementation. As a preprocessing step for the Chest X-Ray data set, we resized the training images into $150 \times 150$ and applied a random affine transformation[3]. Test images are also resized to $150 \times 150$ without any other transformation. In the preprocessing of the PASCAL-VOC data set, when using MobileNet-v2, ResNet18, and VGG16, both train and test images are resized into $256 \times 256$, center cropped by $224 \times 224$, and normalized to have zero mean and a standard deviation of $1$. In the special case where EfficientNet-b4 is used with PASCAL-VOC, train and test images are resized into $412 \times 412$, center cropped by $380 \times 380$, and normalized to have zero mean and a standard deviation of $1$. This is because naturally, EfficientNet expects larger images and might not work well with datasets with small images without resizing to larger

[2]although, we have not used validation data in our implementation.

[3]without rotation ("degree" = 0) with scale in the range $(0.8, 1.2)$

sizes or making other adjustments to the architecture. Finally, the preprocessing step of CIFAR-100 includes normalization of both training and test data to zero mean and a standard deviation of 1. For quantization and entropy coding, we used NNCodec software [48] that implements the standard-compliant implementation of the Neural Network Coding (NNC) standard (ISO/IEC 15938-17) [49].

For model training, we used the Adam optimizer with learning rate $1e-3$ for 50 epochs and with batch size 64[4]. It is important to clarify that achieving state-of-the-art accuracy for each model is not our primary objective. Consequently, we did not fine-tune the learning rates or number of epochs extensively for each model. Instead, we selected hyperparameter values that yield satisfactory performance for each model on every dataset. Experiments related to MobileNet-v2, ResNet18, and VGG16 were conducted on a system equipped with two NVIDIA GeForce RTX 2080 Ti GPUs (12 GB RAM), with CUDA version 11.4. For experiments with EfficientNet-b4, we used two NVIDIA Tesla V100 GPUs (16 GB RAM) from a DGX server with CUDA version 11.4.

*4) Baseline:* As discussed in Section I, our method is complementary to common compression techniques such as quantization, sparsification, and entropy coding. Accordingly, the baseline in our comparisons corresponds to a standard compression pipeline without P$^2$U. For this purpose, we used the NNC software, configured as follows:

- Quantization: We apply scalar uniform quantization to parameter tensors using:

$$w_q = \lfloor \frac{w}{2^{qp}} \rfloor$$

  where $qp$ controls the quantization step size. The reconstructed values in the decoded tensor are integer multiples of the quantization step.
- Entropy coding: DeepCABAC (Deep Context Adaptive Binary Arithmetic Coding) [11] is utilized to produce bitstreams. DeepCABAC consists of the following three stages [42]:
  1. binarization: quantized weights are decomposed into series of binary bins,
  2. context modeling: a probability model is assigned to each bin to provide probability estimates,
  3. binary arithmetic coding: an arithmetic coding engine encodes each bin according to its estimated probability.

NNC supports a variety of techniques to further improve compression, such as structured and unstructured sparsification, Local Scaling Adaptation (LSA), BatchNorm Folding (BNF), and quantization parameter optimization ($opt\_qp$). To ensure fair comparison and reduce inference overhead, we deactivate these options in our baseline and only use scalar uniform quantization with optimized $qp$ values selected from the set $\{-25, -30, -35, -40, -45\}$, $qp\_density = 2$, and the base DeepCABAC entropy coding.

For each experiment, the model is first retrained for 50 epochs on the sender side to achieve acceptable performance. On the receiver side, the compressed bitstream is decoded and dequantized into the floating-point domain.

When applying P$^2$U, we follow the same compression pipeline, but separately encode both the low-precision model and the model update. At the receiver, these components are decoded, dequantized, and combined to reconstruct the final proxy model.

### *B. Results and Analysis*

In this section, we evaluate P$^2$U and investigate several important questions related to this approach. All reported results in the following subsections are averaged over 10 runs with different seed values. Also, the high-precision level of P$^2$U is set to 32-bit INT in all experiments.

*1) How does P$^2$U compare with direct compression with different bit-widths?:* Table II compares the performance of P$^2$U against direct quantization baselines at 16-, 8-, and 4-bit precision levels across three datasets. For comparison purposes, we employed three distinct metrics:

- size of the bitstream (in MB) generated by the encoder as a proxy for required bandwidth size,
- the time (in seconds) it takes for the receiver device to begin inference with the received model (either the low-precision version or the high-precision proxy model). This includes the necessary preparation steps - such as encoding, decoding, and dequantizing the recevied model,
- Top-1 classification accuracy of the dequantized model in the receiver over test data.

For better comparison, we report measurements for the baseline, the low-bit precision model (referred to as "Low-Prec."), the model update (referred to as "Update"), and the high-precision proxy model (referred to as "Proxy"). Note that since inference is not possible using the model update, Top-1 accuracy is not reported for it.

For the baselines, results are reported at all precision levels, while for P$^2$U, we show results only for its best-performing low-precision configuration (4-bit in all cases)[5]. Remarkably, P$^2$U with 4-bit low-precision model, consistently surpasses direct quantization even at the highest precision (16-bit) in terms of final top-1 accuracy, achieving gains of $+7.76\%$, $+0.54\%$, and $+1.33\%$ on Chest X-Ray, PASCAL-VOC, and CIFAR-100, respectively. It also incurs significantly lower total transmission sizes and startup times.

While direct quantization with lower bitwidths (e.g., 8-bit and 4-bit) reduces bandwidth usage and startup time, these benefits come at the cost of significantly degraded accuracy. In contrast, P$^2$U decouples low-precision startup from high-precision performance by progressively refining the model, thus enabling better trade-offs in bandwidth-constrained and latency-sensitive scenarios. This highlights the advantage of our update-based reconstruction strategy over direct quantization, especially when balancing accuracy with resource constraints.

*2) How does P$^2$U perform across various model architectures?:* In this section, we investigate the effect of model architectures of different sizes and/or complexities on the

[4]batch size 64 has been selected according to our GPU capacity to run larger models.

[5]For other precision levels, check Tables II-IV.

TABLE II

COMPARISON OF TOP-1 ACCURACY, TRANSMISSION SIZE (MB), AND TRANSMISSION TIME (S) BETWEEN DIRECT QUANTIZATION BASELINES (AT 16-, 8-, AND 4-BIT) AND P$^2$U WITH 4-BIT LOW-PRECISION LEVEL ACROSS THREE DATASETS. P$^2$U CONSISTENTLY ACHIEVES HIGHER ACCURACY THAN ALL DIRECT QUANTIZATION LEVELS, INCLUDING 16-BIT, WHILE MAINTAINING LOWER OR COMPARABLE TRANSMISSION SIZES AND STARTUP TIMES.

| | | Direct Compression | | | | P$^2$U | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Size | | | Time | | | Top-1 Acc | |
| Dataset | Model | Bitwidth | Size | Time | Top-1 Acc | Low-Prec. | Update | Total | Low-Prec. | Update | Total | Low-Prec. | Proxy |
| Chest X-Ray | MobileNet-v2 | 16 | 4.11 | 1.67 | 75.97 | 0.76 | 1.18 | 1.94 | 0.61 | 0.64 | 1.25 | 56.04 | 83.73 |
| | | 8 | 1.92 | 1.09 | 67.86 | | | | | | | | |
| | | 4 | 0.76 | 0.61 | 56.04 | | | | | | | | |
| PASCAL-VOC | ResNet18 | 16 | 19.51 | 6.87 | 72.05 | 2.95 | 5.41 | 8.36 | 1.69 | 2.37 | 4.07 | 41.62 | 72.59 |
| | | 8 | 8.69 | 3.94 | 72.02 | | | | | | | | |
| | | 4 | 2.95 | 1.69 | 41.62 | | | | | | | | |
| CIFAR-100 | VGG16 | 16 | 26.45 | 8.23 | 52.29 | 4.31 | 8.99 | 13.3 | 1.58 | 3.5 | 5.08 | 43.57 | 53.62 |
| | | 8 | 12.06 | 4.66 | 52.19 | | | | | | | | |
| | | 4 | 4.31 | 1.58 | 43.57 | | | | | | | | |

performance of P$^2$U. In the experiment, we considered all four network architectures introduced in Section IV-A2 for the image classification task using PASCAL-VOC data set.

For P$^2$U, we set the low-bit precision to 8-bit INT for all model architectures. Similar to the previous section, baselines map to sending the quantized models at different bitwidths, 16-, 8-, and 4-bit. Table III demonstrates the results. The table provides several key observations as follows.

First, Across all models, direct quantization shows a clear trade-off between model size, startup time, and accuracy. While reducing bitwidth from 16 to 4 significantly improves transmission size and startup time, it also results in considerable accuracy degradation–most dramatically seen in MobileNet-v2 and EfficientNet-b4, where 4-bit quantization drops accuracy to 11.68% and 11.76%, respectively. This suggests that for lightweight models like MobileNet-v2, P$^2$U provides substantial accuracy gains with minimal added cost.

Second, in ResNet18, the low-precision model already performs reasonably well (72.02%), and the update adds a modest 0.02 MB and 0.81s to reach a final accuracy of 72.51%. Although the final improvement in accuracy (+0.49%) is less pronounced, the total size (8.71 MB) is still smaller than the 16-bit model (19.51 MB), making P$^2$U a more bandwidth-efficient choice with comparable or better accuracy.

Thirds, For larger and more complex models like EfficientNet-b4, P$^2$U's benefits become even more evident. The 8-bit model starts at 79.42% accuracy with 14.65 MB and 14.77s startup time, and the update (3.04 MB, 5.83s) lifts the final accuracy to 79.7%. This is very close to the 16-bit baseline (79.83%) but with nearly 44% smaller transmission size (17.69 MB vs. 31.76 MB) and reduced startup time (20.6s vs. 20.01s). This shows P$^2$U maintains competitive accuracy while reducing overhead in heavier models.

Fourth, in VGG16, known for its large parameter count and size, the results are similarly compelling. The final P$^2$U proxy accuracy is 75.6%, surpassing both the 8-bit and 16-bit baselines (75.12% and 75.09%) despite a transmission size of only 113 MB compared to 242.8 MB for 16-bit. The startup time (54.25s) is also considerably shorter than that of the 16-bit model (81.63s). This demonstrates that even for bandwidth-heavy architectures, P$^2$U enables substantial resource savings without compromising performance.

Another observation is that for VGG16, the top-1 accuracy of the lower precision model, e.g., 8-bit, is better than that of the higher precision model, e.g., 16-bit. This can be due to the regularizing effects of compression and has been reported in other works [48].

It is important to note that the reported values for bitstream sizes (Size) and startup latencies (Time) of the high-precision "Proxy" model represent worst-case estimates. These values are calculated as the sum of those for the "Low-Prec." and "Update" components. However, in bandwidth-constrained scenarios where accuracy must be preserved, the actual bandwidth requirement for P$^2$U is capped by the larger of the bitstream sizes of the "Low-Prec." and "Update" not by the sum of these two values. This is because the sender initiates the transmission of the model update only after the receiver has fully received the bitstream of the low-precision model. If the transmission of the low-precision model and update is done in parallel, startup latency can be reduced, although this would result in a cumulative bandwidth requirement equal to the sum of both components.

Furthermore, in our experiments, we set the high-precision model to 32-bit integers, which consequently means that the model updates - computed as the difference between the high-precision and low-precision models - are also stored and transmitted using 32-bit integers. This choice ensures that the update values can be safely represented without overflow, preserving the exact difference between the two models. However, this approach may overestimate the required bitwidth for storing the update values, especially in cases where the actual differences are small and can be sufficiently represented using lower-precision formats such as 16-bit integers. A more efficient alternative would be to dynamically assess the value range of the update tensor and determine whether a smaller integer precision suffices. If so, P$^2$U can adaptively quantize the update to a lower bitwidth, such as 16 bit INT, thereby reducing the bitstream size of the model update and improving overall communication efficiency. This adaptive update encoding can yield additional bandwidth savings without affecting the final model accuracy, further strengthening the benefits of P$^2$U in bandwidth-constrained environments.

Finally, it should also be noted that in our calculations for startup latency, we did not consider the delay in the communication channel. Assuming the communication channel's delay is $C$, for P$^2$U we have $2C$ delay to get the reconstructed model while for the baseline, this only equals $C$. Though, the

TABLE III
COMPARISON OF P$^2$U AND DIRECT QUANTIZATION BASELINES ACROSS FOUR MODEL ARCHITECTURES IN THE IMAGE CLASSIFICATION TASK USING PASCAL-VOC. DIRECT QUANTIZATION IS EVALUATED AT 16-, 8-, AND 4-BIT PRECISION LEVELS. FOR P$^2$U, RESULTS ARE SHOWN FOR THE 8-BIT LOW-PRECISION MODEL ("LOW-PREC."), THE TRANSMITTED UPDATE ("UPDATE"), AND THEIR COMBINATION ("PROXY")

| | | MobileNet-v2 | | | ResNet18 | | | EfficientNet-b4 | | | VGG16 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Size | Time | Top-1 Acc. | Size | Time | Top-1 Acc. | Size | Time | Top-1 Acc. | Size | Time | Top-1 Acc. |
| Direct Quantization | 16-bit | 4.15 | 1.68 | 73.64 | 19.51 | 6.87 | 72.05 | 31.76 | 20.01 | 79.83 | 242.8 | 81.63 | 75.09 |
| | 8-bit | 1.95 | 1.1 | 70.45 | 8.69 | 3.94 | 72.02 | 14.65 | 14.77 | 79.42 | 112.83 | 47.35 | 75.12 |
| | 4-bit | 0.76 | 0.6 | 11.68 | 2.95 | 1.7 | 41.62 | 5.74 | 9.52 | 11.76 | 42.8 | 17.33 | 67.38 |
| P2U | Low-Prec. | 1.95 | 1.1 | 70.45 | 8.69 | 3.94 | 72.02 | 14.65 | 14.77 | 79.42 | 112.83 | 47.35 | 75.12 |
| | Update | 0.48 | 0.41 | – | 0.02 | 0.81 | – | 3.04 | 5.83 | – | 0.17 | 6.9 | – |
| | Proxy | 2.43 | 1.51 | 74.06 | 8.71 | 4.75 | 72.51 | 17.69 | 20.6 | 79.7 | 113 | 54.25 | 75.6 |

channel's delay is usually negligible compared to the startup times reported in the table.

*3) How is P$^2$U affected by different precision levels?:* The benefits of transferring a model from a sender to a receiver using P$^2$U comes with an essential burden – selecting the proper precision levels for low-precision models. To evaluate the sensitivity of P$^2$U to the precision level of the base (low-precision) model, we conduct experiments using three levels of integer quantization, e.g., 16-bit, 8-bit, and 4-bit, for both VGG16 and ResNet18 models on the PASCAL-VOC dataset. Tables IV and V present the average results from 10 runs for VGG16 and ResNet18, respectively. In these tables, each row index demonstrates one experiment with different precision level for the low-precision model.

Starting with VGG16, we observe that moving from 16-bit to 8-bit precision causes negligible performance degradation for the low-bit precision model; the proxy model accuracy, however, slightly increases from 75.49% to 75.6%. Furthermore, the bandwidth usage (Size) of the proxy model is significantly reduced from 243.16 MB to 113 MB (a 53.5% reduction), and inference time drops from 87.37s to 54.25s (a 38% reduction). This suggests that P$^2$U with 8-bit quantization achieves a favorable trade-off between efficiency and performance compared to 16-bit quantization.

However, when precision is further reduced to 4-bit, a marked decline in standalone low-precision performance is observed – accuracy of the 4-bit model alone falls sharply to 67.38%. Interestingly, when combined with the 32-bit update via P$^2$U, the proxy model restores its performance to 75.18%, nearly matching the higher-precision setups. Despite the low baseline accuracy of the 4-bit model, the 32-bit update effectively compensates for lost precision. On the other hand, this setup achieves the smallest bandwidth usage (43.43 MB) and lowest inference time (24.85 s), demonstrating the compression power of aggressive quantization when paired with P$^2$U.

In ResNet18, similar trends are evident. The 16-bit base model combined with the 32-bit update yields 72.58% accuracy with a bandwidth usage of 19.53 MB and an inference time of 7.35 s. Reducing to 8-bit slightly lowers standalone accuracy (to 72.02%) but maintains high proxy performance (72.51%) and further reduces bandwidth usage to 8.71 MB (a 55.4% reduction). This again indicates the robustness of P$^2$U in leveraging 8-bit models for efficient transmission. At 4-bit precision, the standalone ResNet18 performance drops dramatically to 41.62%, highlighting the information loss due to aggressive quantization. Yet, the P$^2$U proxy model still reaches 72.59%, nearly identical to the 16- and 8-bit configurations. This demonstrates that while 4-bit models alone are insufficient for accurate inference, their combination with a compact 32-bit update recovers most of the accuracy while keeping communication cost low (8.35 MB). In fact, the 4-bit proxy performance is achieved with the smallest total communication cost and startup time.

Overall, the results confirm that P$^2$U effectively decouples model transmission efficiency from model precision, allowing for aggressive quantization (as low as 4-bit) without significant performance degradation, provided that a lightweight 32-bit update is transmitted alongside. The analysis highlights that the optimal balance between communication efficiency and model accuracy is achieved using 8-bit quantization, which offers strong performance and reduced overhead. Nevertheless, 4-bit quantization, though riskier in standalone use, provides the best overall compression when paired with the 32-bit update in P$^2$U, enabling effective model delivery under tight bandwidth constraints.

An important practical takeaway from these findings is that the choice of quantization level in P$^2$U should be guided by the specific priorities of the application scenario. In cases where model accuracy is not critical – such as in some edge computing or IoT deployments, where bandwidth and latency constraints dominate – the most effective setup is to use P$^2$U with aggressive quantization, going as low as 4-bit. This configuration significantly reduces startup time and bandwidth usage, enabling fast model deployment and updates, while still yielding reasonable performance for low-stakes tasks.

On the other hand, in applications where accuracy must be preserved but some bandwidth savings are still desired, using P$^2$U with 8-bit or 16-bit quantization offers an optimal balance. These configurations provide a substantial reduction in communication cost compared to full-precision models, while retaining high model accuracy that is close to that of uncompressed baselines. Notably, even with 8-bit quantization, P$^2$U achieves competitive or superior accuracy relative to standard quantized models and low-bit training from scratch, thanks to the guidance provided by the high-precision model update.

This flexibility makes P$^2$U particularly well-suited for scenarios where communication is constrained but accuracy requirements vary – allowing the practitioner to tailor the precision level to meet the application's needs without retraining the model from scratch.

TABLE IV

PERFORMANCE OF VGG16 ON PASCAL-VOC USING P$^2$U AT DIFFERENT QUANTIZATION LEVELS. P$^2$U CONSISTENTLY IMPROVES TOP-1 ACCURACY WITH SLIGHT INCREASE OF STARTUP TIME AND BANDWIDTH USAGE ACROSS ALL BIT-WIDTHS, PARTICULARLY UNDER AGGRESSIVE QUANTIZATION (4-BIT), DEMONSTRATING ITS EFFECTIVENESS IN COMMUNICATION-CONSTRAINED SETTINGS.

| | Sender | | | | Communicate | Performance in Receiver | | |
|---|---|---|---|---|---|---|---|---|
| | Low Precision | | | 32-bit Update | | Top-1 Acc. | Size | Time |
| | 16-bit | 8-bit | 4-bit | | | | | |
| 1 | * | | | | $\mathbf{W}^{16\text{-bit}}$ | 75.09 | 242.8 | 81.63 |
| | | | | * | $\Delta\mathbf{W}^{32\text{-bit}}$ | – | 0.18 | 5.74 |
| | | | | | Proxy | 75.49 | 243.16 | 87.37 |
| 2 | | * | | | $\mathbf{W}^{8\text{-bit}}$ | 75.12 | 112.83 | 47.35 |
| | | | | * | $\Delta\mathbf{W}^{32\text{-bit}}$ | – | 0.17 | 6.9 |
| | | | | | Proxy | 75.6 | 113 | 54.25 |
| 3 | | | * | | $\mathbf{W}^{4\text{-bit}}$ | 67.38 | 42.8 | 17.33 |
| | | | | * | $\Delta\mathbf{W}^{32\text{-bit}}$ | – | 0.63 | 7.52 |
| | | | | | Proxy | 75.18 | 43.43 | 24.85 |

TABLE V

PERFORMANCE OF RESNET18 ON PASCAL-VOC USING P$^2$U AT DIFFERENT QUANTIZATION LEVELS. SIMILAR TO VGG16 (TABLE IV), P$^2$U YIELDS SIGNIFICANT ACCURACY GAINS IN LOW-BIT REGIMES, OFFERING A FAVORABLE TRADE-OFF BETWEEN MODEL SIZE AND PREDICTIVE PERFORMANCE.

| | Sender | | | | Communicate | Performance in Receiver | | |
|---|---|---|---|---|---|---|---|---|
| | Low Precision | | | 32-bit Update | | Top-1 Acc. | Size | Time |
| | 16-bit | 8-bit | 4-bit | | | | | |
| 1 | * | | | | $\mathbf{W}^{16\text{-bit}}$ | 72.05 | 19.51 | 6.87 |
| | | | | * | $\Delta\mathbf{W}^{32\text{-bit}}$ | – | 0.02 | 0.48 |
| | | | | | Proxy | 72.58 | 19.53 | 7.35 |
| 2 | | * | | | $\mathbf{W}^{8\text{-bit}}$ | 72.02 | 8.69 | 3.94 |
| | | | | * | $\Delta\mathbf{W}^{32\text{-bit}}$ | – | 0.02 | 0.81 |
| | | | | | Proxy | 72.51 | 8.71 | 4.75 |
| 3 | | | * | | $\mathbf{W}^{4\text{-bit}}$ | 41.62 | 2.95 | 1.69 |
| | | | | * | $\Delta\mathbf{W}^{32\text{-bit}}$ | – | 5.4 | 2.37 |
| | | | | | Proxy | 72.59 | 8.35 | 4.07 |

*4) How does P$^2$U work for tasks with different levels of difficulties?:* Table VI presents a comparative evaluation of P$^2$U and standard low-precision baselines across three datasets with varying levels of classification difficulty: Chest X-Ray (binary classification), PASCAL-VOC (20 classes), and CIFAR-100 (100 classes). All experiments utilize VGG16 as the base model with the low-precision level set to 8-bit. The results clearly highlight the scalability and effectiveness of P$^2$U, particularly under increasing task complexity.

On the simplest task, Chest X-Ray, both methods perform comparably in terms of accuracy (91.9% for P$^2$U vs. 91.26% for the baseline). However, P$^2$U introduces only a minor increase in total model size (from 17.05 MB to 17.09 MB) and startup time (from 6.63s to 7.68s), offering a slight but consistent gain in accuracy. This demonstrates that even in low-difficulty settings, P$^2$U's adaptive update mechanism provides a tangible improvement without introducing significant computational or communication overhead.

For the moderately complex PASCAL-VOC dataset, P$^2$U yields a more noticeable improvement in accuracy, raising it from 75.12% to 75.6%. This 0.5% gain becomes more significant given the task complexity and highlights P$^2$U's robustness. While the startup time increases from 47.35s to 54.25s, and the model size from 112.83 MB to 113 MB, these increases are marginal relative to the accuracy gain and can be justified in applications where performance is critical but resource use must remain efficient.

The most striking results appear in the CIFAR-100 experiment, which involves fine-grained classification across 100 classes. Here, the baseline low-precision model achieves only 52.19% accuracy, whereas P$^2$U reaches 53.60%, showing a substantial 1.41% absolute improvement. Given the challenging nature of the dataset, this boost is non-trivial and underscores P$^2$U's ability to retain important representational fidelity even at low bit-widths. Meanwhile, the model size grows modestly from 12.06 MB to 12.1 MB, and the startup time increases from 4.65s to 5.39s, demonstrating the minimal cost of the added update component.

These results align with the broader insight that if the target application prioritizes startup time or bandwidth over marginal accuracy gains, then P$^2$U with aggressive quantization (e.g., 4-bit) offers an ideal compromise. However, when moderate accuracy and stability are desired, 8-bit or 16-bit variants of P$^2$U provide consistent performance advantages over conventional quantized models.

### C. Ablation Study

Our theoretical analysis assumes that the update size, measured as the Euclidean norm of the difference between the high- and low-precision models, $\| W^h - W^l \|$, is relatively small compared to the scale of the original weights. In this section, we empirically verify this assumption using the same

TABLE VI
PERFORMANCE COMPARISON OF P$^2$U IN TASKS WITH DIFFERENT LEVELS OF DIFFICULTIES, E.G., BINARY, 20-CLASS, AND 100-CLASS CLASSIFICATION TASKS. VGG16 IS USED AS THE MODEL.

| | Chest X-Ray | | | PASCAL-VOC | | | Cifar-100 | | |
|---|---|---|---|---|---|---|---|---|---|
| | Top-1 Acc. | Size | Time | Top-1 Acc. | Size | Time | Top-1 Acc. | Size | Time |
| Low Prec. | 91.26 | 17.05 | 6.63 | 75.12 | 112.83 | 47.35 | 52.19 | 12.06 | 4.65 |
| Update | – | 0.04 | 1.04 | – | 0.17 | 6.9 | – | 0.04 | 0.74 |
| Proxy | 91.9 | 17.09 | 7.68 | 75.6 | 113 | 54.25 | 53.60 | 12.1 | 5.39 |

TABLE VII
AVERAGE RATIOS OF PERTURBATION NORM TO HIGH- AND LOW-PRECISION MODEL NORMS IN 10 RUNS.

| | MobileNet_V2 | ResNet18 | VGG16 |
|---|---|---|---|
| $r_h$ | $1.496e-3$ | $2.603e-2$ | $3.022e-2$ |
| $r_l$ | $1.498e-3$ | $2.608e-2$ | $3.03e-2$ |

experimental setup as in Section B.1, i.e., MobileNet_v2 trained on Chest X-Ray, ResNet18 trained on PASCAL_VOC, and VGG16 trained on CIFAR-100. To quantify the relative size of the update, we compute two ratios:

$$r_h = \frac{\| W^h - W^l \|}{\| W^h \|},$$
$$r_l = \frac{\| W^h - W^l \|}{\| W^l \|},$$

where $r_h$ and $r_l$ measure the perturbation relative to the high- and low-precision weight norms, respectively. Table VII reports the averages over 10 independent runs. It is observed that the perturbation is sufficiently small to make the first-order Taylor approximation a reasonable heuristic in our settings. Across all models, both ratios remain below $3.1 \times 10^-2$, confirming that the perturbation size is indeed small compared to the overall weight scale. This supports the validity of the first-order Taylor approximation as a heuristic in our analysis.

Figure 2 provides a finer-grained view of the perturbation by showing tensor-wise differences between the high- and low-precision weights for each model, using a single seed. For clarity, only layers with $\| W^h - W^l \| > 0.01$ are shown on the x-axis. The overall norms of the weight differences are 3.55, 2.51, and 2.48 for MobileNet_v2 (Chest X-Ray), ResNet18 (PASCAL-VOC), and VGG16 (CIFAR-100), respectively. These values further illustrate that update sizes are consistently small across architectures and datasets.

## V. SUMMARY AND DISCUSSION

In this work, we proposed a simple yet effective precision-driven transmission approach named Progressive Precision Update (P$^2$U), for transmitting and deploying compressed neural models in resource-constrained environments. By decoupling the model into a low-precision proxy and a small, high-precision update as the difference between the original high-precision model and the transmitted low-precision version, P$^2$U enables fast startup, reduced bandwidth usage, and strong predictive performance. P$^2$U is complementary to existing compression techniques such as quantization and sparsification. Extensive experiments across diverse architectures (e.g., VGG16, ResNet18) and datasets (Chest X-ray, PASCAL-VOC, CIFAR-100) show that P$^2$U consistently achieves accuracy improvements over standard quantization, with negligible overhead in communication and latency. Particularly in challenging settings such as 100-class classification, P$^2$U recovers up to 1.4% of accuracy lost due to quantization. Moreover, we show that when bandwidth or startup time is the priority, aggressive quantization (e.g., 4-bit) can be used without severely compromising performance. These results establish P$^2$U as an effective and practical solution for scalable and efficient model distribution in low-resource settings, including federated learning, edge computing, and IoT deployments. Designing an automated approach for low-precision level selection that best suits the scenario is an interesting venue for future work. We did not evaluate the performance of P$^2$U for transmitting Large Language Models (LLMs) and for tasks other than image classification. These aspects will be addressed in future research.

## REFERENCES


[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations (ICLR)*, 2015.

[2] I. Fedorov, R. P. Adams, M. Mattina, and P. Whatmough, "SpArSe: Sparse architecture search for cnns on resource-constrained microcontrollers," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[3] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Sparse binary compression: Towards distributed deep learning with minimal communication," *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2019.

[4] J. Wangni, J. Wang, J. Liu, and T. Zhang, "Gradient sparsification for communication-efficient distributed optimization," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 31, 2018.

[5] X. Liu, L. Wu, C. Dai, and H.-C. Chao, "Compressing cnns using multilevel filter pruning for the edge nodes of multimedia internet of things," *IEEE Internet of Things Journal*, vol. 8, no. 14, pp. 11 041–11 051, 2021.

[6] C. Qi, S. Shen, R. Li, Z. Zhao, Q. Liu, J. Liang, and H. Zhang, "An efficient pruning scheme of deep neural networks for internet of things applications," *EURASIP Journal on Advances in Signal Processing*, vol. 31, no. 1, 2021.

[7] R. Goutham, H. Afrabandpey, F. Cricri, H. Zhang, E. Aksu, M. Hannuksela, and H. R. Tavakoli, "Stochastic binary-ternary quantization for communication efficient federated computation," *IEEE International Conference on Image Processing (ICIP)*, pp. 2097–2101, 2022.

[8] Y. Li, Y. Bao, and W. Chen, "Fixed-sign binary neural network: An efficient design of neural network for internet-of-things devices," *IEEE Access*, vol. 8, pp. 164 858–164 863, 2020.

[9] Y. Yang, Z. Zhang, and Q. Yang, "Communication-efficient federated learning with binary neural networks," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3836–3850, 2021.

[10] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, 2016.

[11] S. Wiedemann, H. Kirchhoffer, S. Matlage, P. Haase, A. Marban, T. Marinč, D. Neumann, T. Nguyen, H. Schwarz, T. Wiegand *et al.*, "DeepCABAC: A universal compression algorithm for deep neural

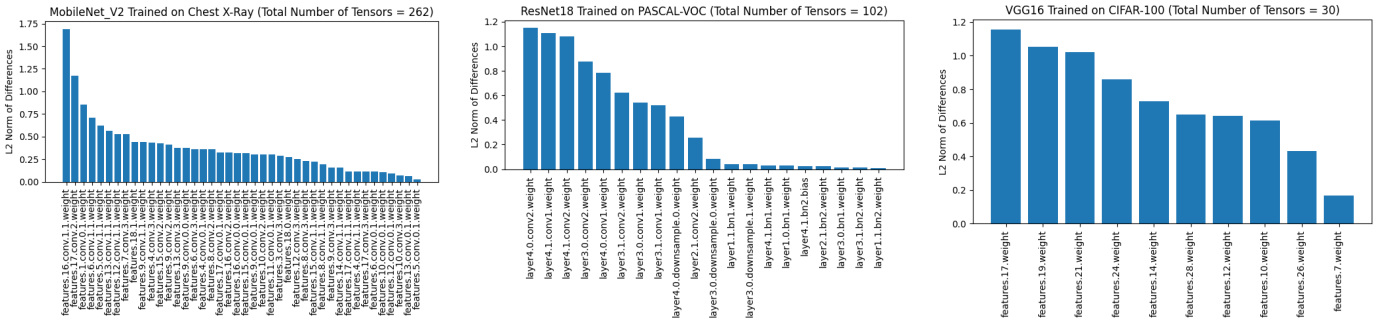


Fig. 2. Tensor-wise Euclidean norm of the differences of the high- and low-precision models. In each case, the total number of tensors are shown in the title. Only layers with $\| W^h - W^l \| > 0.01$ are shown on the x-axis.

networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 4, pp. 700–714, 2020.

[12] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, "A white paper on neural network quantization," *arXiv preprint arXiv:2106.08295*, 2021.

[13] E. Michaud, Z. Liu, U. Girit, and M. Tegmark, "The quantization model of neural scaling," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, pp. 28 699–28 722, 2023.

[14] M. Van Baalen, C. Louizos, M. Nagel, R. A. Amjad, Y. Wang, T. Blankevoort, and M. Welling, "Bayesian bits: Unifying quantization and pruning," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 5741–5752, 2020.

[15] Z. Yao, Z. Dong, Z. Zheng, A. Gholami, J. Yu, E. Tan, L. Wang, Q. Huang, Y. Wang, M. Mahoney *et al.*, "Hawq-v3: Dyadic neural network quantization," *International Conference on Machine Learning (ICML)*, pp. 11 875–11 886, 2021.

[16] S. Kim, C. R. C. Hooper, A. Gholami, Z. Dong, X. Li, S. Shen, M. W. Mahoney, and K. Keutzer, "Squeezellm: Dense-and-sparse quantization," *International Conference on Machine Learning (ICML)*, pp. 23 901–23 923, 2024.

[17] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Pétrot, "Ternary neural networks for resource-efficient ai applications," *International Joint Conference on Neural Networks (IJCNN)*, pp. 2547–2554, 2017.

[18] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary neural networks: A survey," *Pattern Recognition*, vol. 105, p. 107281, 2020.

[19] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," *European Conference on Computer Vision (ECCV)*, pp. 525–542, 2016.

[20] H. Qin, X. Zhang, R. Gong, Y. Ding, Y. Xu, and X. Liu, "Distribution-sensitive information retention for accurate binary neural network," *International Journal of Computer Vision*, vol. 131, no. 1, pp. 26–47, 2023.

[21] H. Qin, Y. Ding, X. Zhang, J. Wang, X. Liu, and J. Lu, "Diverse sample generation: Pushing the limit of generative data-free quantization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 10, pp. 11 689–11 706, 2023.

[22] H. Qin, X. Liu, X. Ma, L. Ke, Y. Zhang, J. Luo, and M. Magno, "Bivm: Accurate binarized neural network for efficient video matting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2025.

[23] Q. Jin, L. Yang, and Z. Liao, "Adabits: Neural network quantization with adaptive bit-widths," *IEEE/CVF conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2146–2156, 2020.

[24] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 29, 2016.

[25] S. P. Singh and D. Alistarh, "Woodfisher: Efficient second-order approximation for neural network compression," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 18 098–18 109, 2020.

[26] H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 6377–6389, 2020.

[27] H. R. Tavakoli, J. Wabnig, F. Cricri, H. Zhang, E. Aksu, and I. Saniee, "Hybrid pruning and sparsification," *IEEE International Conference on Image Processing (ICIP)*, pp. 3542–3546, 2021.

[28] F. Tung and G. Mori, "Deep neural network compression by in-parallel pruning-quantization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 3, pp. 568–579, 2018.

[29] S.-K. Yeom, P. Seegerer, S. Lapuschkin, A. Binder, S. Wiedemann, K.-R. Müller, and W. Samek, "Pruning by explaining: A novel criterion for deep neural network pruning," *Pattern Recognition*, vol. 115, p. 107899, 2021.

[30] S. Chen, J. Zhou, W. Sun, and L. Huang, "Joint matrix decomposition for deep convolutional neural networks compression," *Neurocomputing*, vol. 516, pp. 11–26, 2023.

[31] W. Liu, M. Zhang, C. Shi, N. Zhang, and J. Liu, "Deep convolutional neural network compression method: Tensor ring decomposition with variational bayesian approach," *Neural Processing Letters*, vol. 56, no. 2, pp. 1–17, 2024.

[32] S. Swaminathan, D. Garg, R. Kannan, and F. Andres, "Sparse low rank factorization for deep neural network compression," *Neurocomputing*, vol. 398, pp. 185–196, 2020.

[33] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," in *International Conference on Learning Representations (ICLR)*, 2018.

[34] V. Sanh, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter." in *Proceedings of Thirty-third Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[35] P. Wang, Q. Chen, X. He, and J. Cheng, "Towards accurate post-training network quantization via bit-split and stitching," *International Conference on Machine Learning (ICML)*, pp. 9847–9856, 2020.

[36] S. Ye, T. Zhang, K. Zhang, J. Li, K. Xu, Y. Yang, F. Yu, J. Tang, M. Fardad, S. Liu *et al.*, "Progressive weight pruning of deep neural networks using ADMM," *arXiv preprint arXiv:1810.07378*, 2018.

[37] H. Kirchhoffer, P. Haase, W. Samek, K. Müller, H. Rezazadegan-Tavakoli, F. Cricri, E. B. Aksu, M. M. Hannuksela, W. Jiang, W. Wang, S. Liu, S. Jain, S. Hamidi-Rad, F. Racapé, and W. Bailer, "Overview of the neural network compression and representation (nnr) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 5, pp. 3203–3216, 2022.

[38] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," *Artificial Intelligence and Statistics (AISTATS)*, pp. 1273–1282, 2017.

[39] J.-H. Ahn, O. Simeone, and J. Kang, "Wireless federated distillation for distributed edge learning with heterogeneous data," *IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pp. 1–6, 2019.

[40] S. Oh, J. Park, E. Jeong, H. Kim, M. Bennis, and S.-L. Kim, "Mix2fld: Downlink federated learning after uplink federated distillation with two-way mixup," *IEEE Communications Letters*, vol. 24, no. 10, pp. 2211–2215, 2020.

[41] H. Afrabandpey, G. Rangu, H. Zhang, F. Criri, E. Aksu, and H. R. Tavakoli, "On the importance of temporal dependencies of weight updates in communication efficient federated learning," *IEEE International Conference on Visual Communications and Image Processing (VCIP)*, pp. 1–5, 2022.

[42] D. Becking, K. Müller, P. Haase, H. Kirchhoffer, G. Tech, W. Samek, H. Schwarz, D. Marpe, and T. Wiegand, "Neural network coding of difference updates for efficient distributed learning communication," *IEEE Transactions on Multimedia*, pp. 1–16, 2024.

[43] Y. Oh, N. Lee, and Y.-S. Jeon, "Quantized compressed sensing for

communication-efficient federated learning," *2021 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–6, 2021.
[44] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, vol. 116, pp. 1–8, 2018.
[45] D. Kermany, K. Zhang, and M. Goldbaum, "Large dataset of labeled optical coherence tomography (OCT) and chest x-ray images," *Mendeley Data*, vol. 3, pp. 10–17 632, 2018.
[46] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results," http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.
[47] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.
[48] D. Becking, P. Haase, H. Kirchhoffer, K. Müller, W. Samek, and D. Marpe, "Nncodec: An open source software implementation of the neural network coding iso/iec standard," in *ICML 2023 Workshop Neural Compression: From Information Theory to Applications*, 2023.
[49] H. Kirchhoffer, P. Haase, W. Samek, K. Müller, H. Rezazadegan-Tavakoli, F. Cricri, E. B. Aksu, M. M. Hannuksela, W. Jiang, W. Wang *et al.*, "Overview of the neural network compression and representation (nnr) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 5, pp. 3203–3216, 2021.

**Homayun Afrabandpey** received his B.Sc. and M.Sc. degrees in computer science from the Isfahan University of Technology, Iran, in 2011 and 2014, respectively, and the P.hD. degree in computer science from Aalto University, Espoo, Finland, in 2019. He spent half of 2020 as Postdoctoral researcher with the department of computer science, University of Helsinki. From late 2020 to the end of 2024, he was a senior researcher with Nokia Technology, Finland. He is currently an Assistant Professor with the Faculty of Technology and Engineering, University of Mazandaran, Iran. His research interests include neural networks compression, heterogeneous federated learning, and machine learning interpretability.

**Hamed Rezazadegan-Tavakoli** (Senior Member, IEEE & ELLIS member) received the B.Sc. and M.Sc. in Computer Engineering and Artificial Intelligence from Azad University, Mashhad, Iran, in 2004 and 2008, respectively. He earned his Doctor of Science and Technology in Computer Science from the University of Oulu, Finland, in 2014. From 2015 to 2019, he served as postdoctoral researcher in the Department of Computer Science at Aalto University, with a joint appointment at the Department of Signal Processing, Tampere University of Technology for the duration of 2017-2018.

In 2019, he joined Nokia Technologies, Finland, where he contributed to the standardization of neural network compression and visual AI projects. Since 2022, he has led the Visual AI Systems Research team, focusing on AI applications in video, neural network, and tensor data compression, as well as related standardization activities.

Dr. Tavakoli has played an active role in ISO/IEC and MPEG, serving as editor for ISO/IEC 15938-17:2024, ISO/IEC 15938-18:2025, and ISO/IEC TR 23888-1 on neural network compression and "MPEG-AI Vision and Scenarios." He has made numerous technical contributions to ISO/IEC and authored over 100 papers and inventions. His research interests span computer vision and machine learning, including low-level vision, visual attention, image formation and perception, representation learning, multimodal systems (video understanding, captioning and retrieval), neural network compression and efficiency, federated learning, generative AI techniques.