YONO: Modeling Multiple Heterogeneous Neural Networks on Microcontrollers

Young D. Kwon
University of Cambridge
United Kingdom
ydk21@cam.ac.uk

Jagmohan Chauhan
University of Southampton
United Kingdom
J.Chauhan@soton.ac.uk

Cecilia Mascolo
University of Cambridge
United Kingdom
cm542@cam.ac.uk

ABSTRACT

Internet of Things (IoT) systems provide large amounts of data on all aspects of human behavior. Machine learning techniques, especially deep neural networks (DNN), have shown promise in making sense of this data at a large scale. Also, the research community has worked to reduce the computational and resource demands of DNN to compute on low-resourced microcontrollers (MCUs). However, most of the current work in embedded deep learning focuses on solving a single task efficiently, while the multi-tasking nature and applications of IoT devices demand systems that can handle a diverse range of tasks (such as activity, gesture, voice, and context recognition) with input from a variety of sensors, simultaneously.

In this paper, we propose YONO, a product quantization (PQ) based approach that compresses multiple heterogeneous models and enables in-memory model execution and model switching for dissimilar multi-task learning on MCUs. We first adopt PQ to learn codebooks that store weights of different models. Also, we propose a novel network optimization and heuristics to maximize the compression rate and minimize the accuracy loss. Then, we develop an online component of YONO for efficient model execution and switching between multiple tasks on an MCU at run time without relying on an external storage device.

YONO shows remarkable performance as it can compress multiple heterogeneous models with negligible or no loss of accuracy up to 12.37×. Furthermore, YONO's online component enables an efficient execution (latency of 16-159 ms and energy consumption of 3.8-37.9 mJ per operation) and reduces model loading/switching latency and energy consumption by 93.3-94.5% and 93.9-95.0%, respectively, compared to external storage access. Interestingly, YONO can compress various architectures trained with datasets that were not shown during YONO's offline codebook learning phase showing the generalizability of our method. To summarize, YONO shows great potential and opens further doors to enable multi-task learning systems on extremely resource-constrained devices.

CCS CONCEPTS

• **Computer systems organization → Embedded and cyber-physical systems**.

KEYWORDS

Multi Task Learning, Product Quantization, Microcontrollers.

1 INTRODUCTION

With the rise of mobile, wearable devices, and the Internet of Things (IoT), the proliferation of sensory type data has fostered the adoption of deep neural networks (DNN) in the modeling of a variety of mobile sensing applications [41]; researchers use DNN trained on sensory data in mobile sensing tasks such as human activity recognition [22, 77], gesture recognition [14], tracking and localization [29], mental health and wellbeing [53], and audio sensing applications [61]. While machine learning (ML) models are becoming more efficient on resource-constrained IoT devices [12], most existing on-device systems, designed for microcontroller units (MCUs), are targeted at one specific application [5, 16, 86]. Conversely, multi-application systems capable of directly supporting a wide range of applications on-device could be more versatile and useful in practice. Specifically, we envisage a system powered by MCUs that can recognize users' voice commands, activities and gestures, identify everyday objects and people, and understand the surrounding environments: this has the potential to boost the utilization of IoT devices in practice (e.g., help visually impaired individuals understand their environments [1]).

However, realizing such multi-tasking system faces three major challenges. **First**, multiple dissimilar tasks based on different modalities of incoming data (e.g., voice recognition (audio), activity recognition (accelerometer signals), object classification (image)) need to co-exist in the same framework. As discussed in [44], conventional multi-task learning (MTL) approaches cannot address *multiple heterogeneous networks* effectively. **Second**, IoT devices based on MCUs are extremely resource-constrained [15, 36]. For example, "high-end" MCUs (e.g., STMF767ZI) have only 512 KB Static Random-Access Memory (SRAM) for intermediate data and 2 MB on-chip embedded flash (eFlash) memory for program storage. **Finally**, in real-world deployment scenarios, context switching of different ML tasks at run-time could incur overheads on memory-constrained MTL systems as demonstrated in [44], where some models must reside in external storage devices due to the limited on-chip memory space. As on-chip memory operations are faster than external disk accesses, frequent model loading/swap between different tasks based on external storage increase the overall latency, exacerbating the usability and responsiveness of the system.

To solve these challenges, one of the common techniques employed is to compress individual models separately using pruning [25, 85] and quantization [28]. However, model compression techniques are limited since extensive and iterative finetuning is required to ensure high performance after compressing a model. Also, since models are trained independently, they cannot benefit from potential knowledge transfer between different tasks. In the literature, researchers proposed MTL-based approaches to achieve robustness and generalization of multiple tasks, while increasing the compression rate of the model by sharing network structures. However, *sharing/compressing multiple heterogeneous networks* has not been fully examined. Furthermore, prior work [44] attempts to solve the MTL of multiple heterogeneous networks by sharing weights of multiple models via virtualization. However, this method is complex, and the compression ratio is constrained to 8.08× (see §4.2 for detail), thereby limiting the type of IoT devices on which it can operate. Further, since only a simplified LeNet architecture is evaluated on an MCU, the system could not achieve high accuracy to be useful in practice (e.g., 59.26% on the CIFAR-10 dataset [35]).

This Work. To address the challenges and limitations of previous approaches, we propose **YONO** (**Y**ou **O**nly **N**eed **O**ne pair of codebooks), that adopts Product Quantization (PQ) [30] to maximize compression rate and on-chip memory operations to minimize external disk accesses for heterogeneous multi-task learning. PQ, originally proposed in the database community, aims to decompose the original high-dimensional space into the Cartesian product of a finite number of low-dimensional subspaces that are independently quantized. A model's weight matrix of any layer can be converted to codeword indexes corresponding to the subvectors of the weight matrix via a codebook.

Inspired by successful applications of PQ on approximate nearest neighbor search out of billions of vectors in the database community [17, 30, 31] and single layer compression in an individual model [20, 56, 73, 74, 80], we jointly apply PQ on multiple models instead of on a layer of a model. We find just one pair of codebooks that are generalizable and thus can be shared across many dissimilar tasks. We then propose a novel optimization process based on alternating PQ and finetuning steps to mirror the performance of the original models. Further, we introduce heuristics to consider the weight differences between the layers of the original model and the reconstructed layers from the codebooks to maximize the compression rate and accuracy. Finally, we develop an efficient model execution and switching framework to operate multiple heterogeneous models targeted for different tasks, reducing the overhead of context switching (i.e., model swap between tasks) at run-time.

YONO is comprised of two components. The first component is an offline phase in which a shared PQ codebook is learned and multiple models are incorporated. We implement the offline phase of our system on a server. The second component is an online phase in which multiple heterogeneous models are deployed on an extremely resource-constrained device (MCUs). To evaluate YONO, we first evaluated four image datasets and one audio dataset used in state-of-the-art prior work on heterogeneous MTL [44] for a fair comparison. We show that YONO achieves high accuracy of 93.7% on average across the five datasets, which is a 15.4% improvement over [44] due to our usage of the optimized network architecture (see §4.2 for detail) and is very close to the accuracy of the uncompressed models (0.4% loss in accuracy). Further, to evaluate the scalability of YONO to other modalities, we include data from modalities such as accelerometer signals from Inertial Movement Units (IMU) for human activity recognition (HAR) and surface electromyography (sEMG) signals for gesture recognition (GR). We then demonstrate that YONO effectively retains the accuracy of the uncompressed models across all the employed datasets of four different modalities (Image, Audio, IMU, sEMG). Next, to evaluate the generalizability of the learned codebooks of YONO, we apply YONO to compress new models trained on unseen datasets during the codebook learning in the offline phase. Surprisingly, YONO can maintain the accuracy of the uncompressed models and achieve a 12.37× compression ratio (53.1% higher than [44]). Finally, we evaluate the online component of YONO on the largest model and the smallest model to show the upper bound and lower bound results, respectively. We employ an MCU, STM32H747XI (see Section 3 for details), and demonstrate that YONO enables an efficient in-memory execution (latency of 16-159 ms and energy consumption of 3.8-37.9 mJ per operation) and model loading/swap framework for task switching (showing reductions of 93.3-94.5% in latency and 93.9-95.0% in energy consumption compared to the method using external storage access).

2 YONO

In this section, we first present the overview of our multitasking system, YONO (§2.1). Then, we introduce the background on PQ and its applications on single model compression (§2.2). We then explain how we utilize PQ to compress multiple heterogeneous networks into a pair of codebooks. The networks can be of any arbitrary architecture that consists of fully connected layers and convolutional layers. After that, we present our novel network optimization process to ensure the performance of the compressed networks remain close to original models (§2.4). On top of that, based on an observation (detailed in §2.5), we further propose optimization heuristics to maximize the performance gain with a minimal loss of the compression rate when using PQ-based compression. Finally, we describe our in-memory execution and model swapping framework on MCUs (§2.6).

2.1 Overview

In this subsection, we describe the overview of YONO that learns codebooks to represent the weights of multiple heterogeneous neural networks as well as enable on-chip memory operations on resource-constrained devices. In particular, YONO is composed of two components: (1) an offline phase where YONO learns a pair of codebooks on pretrained neural networks using PQ (will be explained in detail in §2.2) and (2) an online phase where YONO enables on-chip execution such as model execution and model loading/swapping. Note that we assume that the overall size of multiple neural networks is larger than the operational limit of the on-chip eFlash memory and SRAM of the targeted IoT devices. For example, in Section 4, we employ seven different models with a total size of 3.84 MB and evaluate our framework on MCU (STM32H747XI), which strictly has only 512 KB of SRAM and 1 MB of eFlash.

2.2 Product Quantization and Compressing Single Neural Network

We now provide an introduction to PQ and how it is used to compress a single model. PQ can be considered a special case of vector quantization (VQ) [21], in which it attempts to find the nearest codeword, $\mathbf{c}$, to encode a given vector, $\mathbf{w}$. Suppose we are given a codebook, $\mathbf{C}$, that contains a set of representative codewords, we can reconstruct/approximate the given vector $\mathbf{w}$ by using $\mathbf{c}$ and its associated index in the codebook. Thus, given a vector $\mathbf{w} \in \mathbb{R}^d$ to be encoded, the encoding problem of VQ can be formulated as follows.

$$\underset{b}{\operatorname{argmin}} \|\mathbf{w} - \mathbf{C}b\|^2 \tag{1}$$

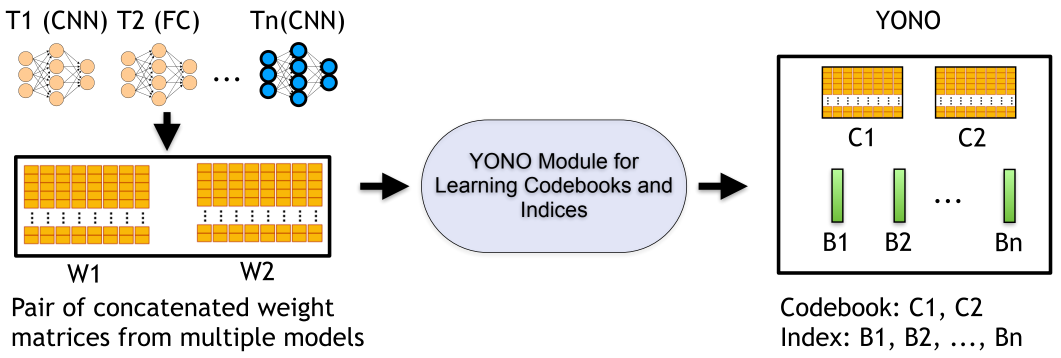

T1 (CNN) T2 (FC) Tn(CNN)
W1
W2
Pair of concatenated weight matrices from multiple models
YONO Module for Learning Codebooks and Indices
YONO
C1
C2
B1
B2
Bn
Codebook: C1, C2
Index: B1, B2, ..., Bn

Figure 1: Overview of the offline component of YONO. The offline module employs PQ to learn a pair of codebooks and identify indices to represent multiple heterogeneous neural networks. This module incorporates our novel optimization process and heuristics to minimize the accuracy loss compared to the original models.

where $\mathbf{C}$ is a d-by-K matrix containing K codewords of length d, and b is called a code (i.e., index of codebook pointing to a codeword, $\mathbf{c}$, nearest to the given vector, $\mathbf{w}$). $\|\cdot\|$ is a l_2 norm. Solving Equation 1 is equivalent to searching the nearest codeword. Besides, the codebook, $\mathbf{C}$, is learned by running the standard k-means clustering over all the given vectors [30].

The PQ is a particular case of VQ when the learned codebook is the Cartesian product of sub-codebooks. Given that there are two sub-codebooks, the encoding problem of PQ is as follows.

$$\begin{aligned} &\underset{b}{\text{argmin}}\, \|\mathbf{w} - \mathbf{C}b\|^2 , \\ &s.t. \quad \mathbf{C} = \mathbf{C}_1 \times \mathbf{C}_2 \end{aligned} \tag{2}$$

where $\mathbf{C}_1$ and $\mathbf{C}_2$ are two sub-codebooks of $\frac{d}{2}$-by-K matrices. Since any codeword of $\mathbf{C}$ is now the concatenation of a codeword of $\mathbf{C}_1$ and a codeword of $\mathbf{C}_2$, PQ can have K^2 different combinations of codewords. If a vector is divided into M partitions, then PQ can have K^M combinations of codewords. The number of sub-codebooks, M, can be any number between 1 and the length of the given vector, d (e.g., $1, 2, \ldots, d$). When M is set to 1, it is VQ. When M is set to d, it is equivalent to the scalar k-means algorithm.

We now describe how the encoding problem of PQ can be applied to compress a neural network. It is because instead of storing weight matrix $\mathbf{W}$ of any layer in neural networks explicitly, we can learn an encoding $\mathcal{B}(\mathbf{W})$ that needs much less storage space. Using the found encoding $\mathcal{B}$ and a learned codebook $\mathbf{C}$ based on PQ, we can reconstruct $\widehat{\mathbf{W}}$ which approximates the original weight matrix $\mathbf{W}$ of the layer. If we can find $\widehat{\mathbf{W}}$ close enough to $\mathbf{W}$, the reconstructed layer of a neural network will perform normally as demonstrated in prior works using PQ to compress a single neural network [20, 73].

2.3 Compressing Multiple Heterogeneous Networks

As described in §2.2, PQ is typically used to compress a single model in machine learning literature [56, 74]. In prior works, each layer is replaced by one small-sized codebook (e.g., K=256, D=8, M=1), and a high compression rate and little performance loss are achieved in large computer vision models with more than 10 M parameters (e.g., ResNet50 [26]). However, in small-sized models that are specially designed to be used on MCUs (i.e., the number of parameters is at most around 500K-1M), the same approach (having a codebook for each layer) no longer provides a high compression rate due to the overhead of storing many codebooks. Therefore, in our system, we propose to apply PQ to one or multiple neural networks while only sharing a pair of the learned codebooks to maximize the compression ratio. We will explain how we ensure high performance of the compressed models in the next subsections (§2.4 and §2.5).

As in Figure 1, we first concatenate weights of all the models of different tasks (i.e., $T_1, T_2, ..., T_n$). Then, we construct two weight matrices, W_1 and W_2, so that YONO takes into account spatial information of convolutional layer kernels as in other prior works [73]. For one weight matrix, W_1, we combine convolutional layers with a kernel size of 3×3. Then, in the other weight matrix, W_2, we concatenate convolutional layers with kernel size 1×1 and fully-connected layers. Then these concatenated weight matrices, W_1 and W_2, are given as an input to learn codebooks, C_1 and C_2, for different kernel sizes, respectively. Note that we also observed that neglecting such information in learning codebooks leads to worse performance. In our system design, we select kernel sizes of 3×3 and 1×1 as those are widely used kernel sizes in many of the optimized network architectures [27, 54, 67]. Also, since FC layers are essentially the same as point-wise convolution operation (i.e., kernel size of 1×1), we combine weights of FC layers together with those of 1×1 kernel convolution layers. Besides, we set M to 2 throughout our evaluation so that YONO can leverage the implicit codebook size of K^M. We observed that when M is 1, the codebook is not generalizable enough to compress multiple neural networks. When M is set to 3, the overhead of the codebooks decreases the compression rate without providing much accuracy benefit.

2.4 Network Optimization

After learning a pair of codebooks for multiple models as in §2.3, YONO performs finetuning on the reconstructed model in order to adjust the loss of information due to the compression (see Algorithm 1). As studied in [85], weights in the first and last layer of a model are the most important. Thus, in the finetuning stage, we select the first and last layer of a model and finetune them (Lines 2-4). The finetuning step largely recovers the accuracy of the original model by re-adjusting the first and last layer of the model according to the different weights induced by the codebooks. However, as we will show in our evaluation in Section 4 (this incurs 2-8% accuracy loss), a simple extension of PQ to multiple heterogeneous neural networks with a finetuning step cannot ensure high accuracy due to the increased weight differences between original models' weight matrices $\mathbf{W}_{T_1,...,T_n}$ and reconstructed models' weight matrices $\widehat{\mathbf{W}}_{T_1,...,T_n}$ although it shows a high compression rate.

Therefore, we introduce an optimization process to improve the performance of the decompressed models. As discussed in prior works [20, 74], in general, higher weight differences (i.e., errors) result in increased loss of accuracy. Thus, to minimize the impact of the weight differences, we adopt to use the iterative optimization procedure, inspired by the Expectation-Maximization (EM) algorithm [13] and prior work [74]. We iteratively adjust the weight drifts by reassigning indices on the updated weights from finetuning as the E-step (Lines 12-13) and by finetuning several selected layers (e.g., first and last layers) as the M-step (Lines 14-17). Note that our optimization procedure is novel in that (i) we perform network optimization across multiple heterogeneous networks and (ii) we do not update codewords in our learned codebooks since we want our codebooks to be generalizable to compress unseen models and datasets during the codebook learning procedure, different from single model compression methods [20, 56, 73, 74]. In Section 4, we demonstrate the generalizability of our learned codebooks and our system on new models that are trained on new datasets that YONO did not see in its codebook learning.

2.5 Optimization Heuristics

In addition, we further propose an optimization heuristic that can maximize performance improvement while ensuring a high compression rate. We observed that weight differences of each layer ($\mathbf{W}$ and $\widehat{\mathbf{W}}$) are not uniformly distributed. Besides, the number of parameters in each layer is considerably different. For example, MicroNet-KWS-M [5] (we adopt this network architecture in our evaluation. Refer to Section 4 for detail) contains 12 convolutional and FC layers. Among them, one convolutional layer has a 4-dimensional weight matrix ($\mathbf{W} \in \mathbb{R}^{C_{cout} \times C_{in} \times k \times k}$) with a size of {140, 1, 3, 3} which has 1,260 parameters, whereas another convolutional layer in the same model can have weight matrix with a size of {196, 112, 1, 1} which has 21,952 parameters. The latter has 17.4 times more parameters than the former. Thus, based on this observation, we propose our novel optimization heuristic to select layers for finetuning that have the largest weight difference and contain the least number of parameters (refer to Lines 22-24 in Algorithm 1). Hence, given a network $\mathbf{W}$ with L layers, we attempt to find a layer ℓ as follows.

$$\underset{\ell}{\operatorname{argmax}} \left\| \mathbf{W}^{\ell} - \widehat{\mathbf{W}}^{\ell} \right\|^2 / N^{\ell} \tag{3}$$

where $\mathbf{W}^{\ell} - \widehat{\mathbf{W}}^{\ell}$ is a weight difference of weight matrices of the layer ℓ, and N^{ℓ} is the number of the parameters of the layer ℓ.

Algorithm 1: YONO Network optimization and heuristics for a given task t

```
Input: Model weights $\mathbf{W}$, model indices $\mathbf{b}$, PQ codebooks $\mathbf{C}$,
       the number of layers $L$, error threshold $\epsilon$, heuristics
Output: Reconstructed model weights $\widehat{\mathbf{W}}$, model indices $\hat{\mathbf{b}}$
Data: Train data $\mathcal{D}^{TRAIN}$, Test data $\mathcal{D}^{TEST}$
/* Perform an initial finetuning step */
1  $\widehat{\mathbf{W}} \leftarrow \mathbf{C}(\mathbf{b})$  // reconstruct a model via codebooks and indices
2  for $\ell = 2, ..., L-1$ do
3      FreezeWeights($\widehat{\mathbf{W}}^{\ell}$)
   // run network training (e.g., BackProp) with loss function
4  Finetune($\widehat{\mathbf{W}}, \mathcal{D}^{TRAIN}$)
5  $acc\_orig \leftarrow$ Evaluate($\mathbf{W}, \mathcal{D}^{TEST}$))
6  $acc\_recon \leftarrow$ Evaluate($\widehat{\mathbf{W}}, \mathcal{D}^{TEST}$))
7  if $acc\_orig - \epsilon \leq acc\_recon$ then
8      return $\widehat{\mathbf{W}}, \mathbf{b}$
/* Perform a further network optimization step */
9  $S \leftarrow (1, L)$ // finetuning layer set
10 $\hat{\mathbf{b}} \leftarrow \mathbf{b}$
11 for $i = 1, ..., L-2$ do
       // E-step: code re-assignment
12     for $\ell \notin S$ do
13         $\hat{\mathbf{b}}^{\ell} \leftarrow \underset{b \in \hat{\mathbf{b}}^{\ell}}{\operatorname{argmin}} \left\| \hat{\mathbf{w}}^{\ell} - \mathbf{C}b \right\|^2$
       // M-step: model update
14     $\widehat{\mathbf{W}} \leftarrow \mathbf{C}(\hat{\mathbf{b}})$
15     for $\ell \notin S$ do
16         FreezeWeights($\widehat{\mathbf{W}}^{\ell}$)
17     Finetune($\widehat{\mathbf{W}}, \mathcal{D}^{TRAIN}$)
18     $acc\_orig \leftarrow$ Evaluate($\mathbf{W}, \mathcal{D}^{TEST}$))
19     $acc\_recon \leftarrow$ Evaluate($\widehat{\mathbf{W}}, \mathcal{D}^{TEST}$))
20     if $acc\_orig - \epsilon \leq acc\_recon$ then
21         return $\widehat{\mathbf{W}}, \hat{\mathbf{b}}$
22     if heuristics is $OURS$ then
           // choose a layer to finetune based on our heuristics
23         $\ell \leftarrow \underset{\ell}{\operatorname{argmax}} \left\| \mathbf{W}^{\ell} - \widehat{\mathbf{W}}^{\ell} \right\|^2 / N^{\ell}$
24         $S \leftarrow (S, \ell)$
```

In summary, through the optimization heuristics, YONO identifies a layer with the highest weight difference per parameter. After that, YONO finetunes the identified layer using our network optimization process introduced in §2.4. The process continues until the reconstructed model's accuracy is recovered to the target accuracy (Lines 20-21), i.e., accuracy loss is less than a given threshold ϵ (e.g., 2-3% in our evaluation). The number of layers to be finetuned is less than or equal to three in most cases. This process helps YONO maximize the compression ratio (small storage overhead) while retaining the accuracy of its compressed models close to their corresponding original (uncompressed) models. Note that the finetuned layers are then quantized into 8-bit integers in the online component of YONO as described in the next subsection.

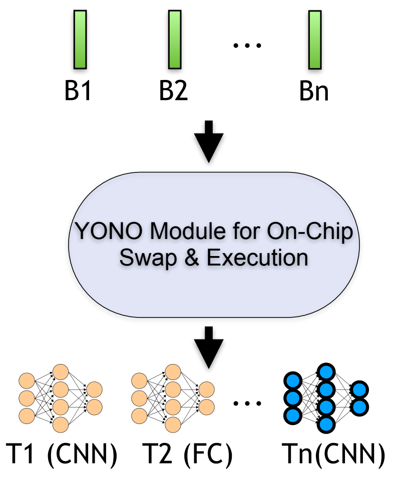


Figure 2: Overview of the online component of YONO. The online module enables fast and efficient model loading/swap and in-memory execution.

2.6 In-memory Execution and Model Swap Framework on MCUs

Having established the offline component of YONO, we now turn our attention to the online component of our system. At runtime, the online component of YONO enables the fast and efficient in-memory execution and model swap of multiple heterogeneous neural networks. Figure 2 illustrates the overview of the online component of YONO.

Data Structure for Deployment on MCUs: To begin with, we describe the data structures that are necessary for deploying ML models on MCUs. First, YONO requires one pair of learned PQ codebooks, model indices, and other relevant information to reconstruct a model. In addition, YONO needs a task executor to run the reconstructed model in-memory and a task switcher to swap an in-memory model to another reconstructed model.

Learned Codebooks: As described in subsections §2.2-2.5, YONO learns a pair of codebooks by applying PQ on multiple heterogeneous neural networks with our novel optimization procedure. Since SRAM is a scarce resource on MCUs, the codebooks are stored on eFlash. Also, because the codebooks are shared across different models compressed by YONO and static during runtime, they are stored on the read-only memory of eFlash.

Model Indices and Other Elements: Once a model is compressed through our system, YONO generates model indices that correspond to the weights of an original model via the learned codebooks and other relevant elements necessary to reconstruct the uncompressed model. For example, relevant elements include model architecture, operators, quantization information, and so on.

Task Executor: We now present the explanation of our task executor. As we adopt TensorFlow Lite for Microcontrollers (TFLM) [12] to run the deployed model on MCUs, YONO also follows its model representation and interpreter-based task execution. As model representation on MCUs, the stored schema of data and values represent the model. The schema is designed for storage efficiency and fast access on mobile and embedded platforms. Therefore, it has some features that help ease the development of MCUs. For example, operations are in a topologically sorted list instead of a directed-acyclic graph, making conducting calculations be a simple looping through the operation list in order. In addition, YONO adopts interpreter-based task execution by relying on TFLM. Thus, the interpreter refers to the schema of the model representation and loads a model. After that, the interpreter handles operations to execute. Since YONO adopts an interpreter-based task executor and loads a model in the main memory for execution, YONO allows model switching at run time, which is not allowed with the code-generator-based compiler method [48] because this method requires recompilation to switch a model.

Task Switcher: When a task needs to be switched (e.g., the target application is switched from image classification to voice command recognition), YONO replaces the loaded model in the memory with a new model to be executed. Using the same memory space between previous and new models, YONO can operate multiple models within a limited memory budget of SRAM. In addition, since YONO does on-chip memory operations to perform execution and model swap, YONO improves the response time and end-to-end execution time of different applications. It is because the access time to secondary storage devices is slower than that to internal memory and primary storage. Moreover, a system relying on external storage devices may have unpredictable overheads. For example, disk-writes on storage devices like flash and solid-state drives need to erase an entire block before a write operation.

Model Reconstruction: We now describe our model reconstruction scheme. To reconstruct a model, YONO utilizes the PQ codebooks, indices, and relevant elements, such as batch normalization layer's mean and variance, quantization information, stored in eFlash. The overall process is as follows. First, YONO retrieves model weights by matching indices of a model to be loaded on the main memory and its corresponding codewords of the PQ codebooks. Secondly, YONO loads relevant elements of the model and then writes this information and model weights to the preallocated memory address for the model on the main memory.

In addition, each value of the learned codewords in the PQ codebooks is stored in 16-bit float instead of 32-bit float type to further reduce the storage requirements on eFlash. In contrast, the weights of the model loaded on the main memory and executed need to be quantized to 8-bit integers. Thus, while loading each layer of the model, YONO converts 16-bit floats to 8-bit integers using the saved quantization information. Specifically, we use the quantization scheme used in [28] to minimize the information loss in quantization. We utilize an affine mapping of integer q to real number r for constant quantization parameters S and Z, i.e., $r = S(q-Z)$. S denotes the scale of an arbitrary positive real number. Z denotes zero-point of the same type as quantized value q, corresponding to the real value 0. As a result, the reconstructed model in the online component is based on 8-bit integers, and thus the use of codebooks does not affect computations of model execution.

3 SYSTEM IMPLEMENTATION

We introduce the hardware and software implementation of YONO.

Hardware. The offline component of our system is implemented and tested on a Linux server equipped with an Intel Xeon Gold 5218 CPU and NVIDIA Quadro RTX 8000 GPU. This component is used to learn PQ codebooks and find indices for each model to be compressed. Then, the online component of our system is implemented and evaluated on an MCU, STM32H747XI, having two cores (ARM Cortex M4 and M7) with 1 MB SRAM and 2 MB eFlash in total. However, our implementation of YONO uses only one core (ARM Cortex M7) since MCUs are typically equipped with one CPU core. We restrict the usage space of SRAM and eFlash to 512 KB and 1 MB, respectively, to enforce stricter resource constraints.

Software. We use PyTorch 1.6 (deep learning framework) and Faiss (PQ framework) to develop and evaluate the offline component of YONO on the Linux server. At the offline phase, we develop YONO using Python on the server and examine the accuracy of the models. In addition, we develop the online component of YONO using C++ on STM32H7 series MCUs. For running neural networks on MCUs, we rely on TFLM. Since eFlash memory of MCUs is read-only during runtime, YONO loads the model weights on SRAM (read-write during runtime) and swaps the models by replacing the models' weights using PQ codebooks and indices stored on eFlash. The binary size of our implementation on an MCU is only 0.41 MB, and the total size of PQ codebooks, indices, and other information to compress the eight heterogeneous networks evaluated in §4.4 is 0.35 MB. Note that the memory requirement of the seven models is 4.19 MB, which is 12.05× of what YONO requires and 4.19× of what typical MCUs with 1 MB storage can support.

4 EVALUATION

We now present the results of the evaluation on our system. §4.1 describes our experimental setup. We evaluate the effectiveness of our system in the offline phase regarding the performance (i.e., accuracy) and compression rate of the compressed models in an MTL scenario. To make a comparison with prior work [44] that tackles MTL of different neural networks, we begin with evaluating our system with the same datasets used in [44] consisting of five datasets for two modalities (i.e., image and audio) (§4.2). After that, we evaluate our system to what extent it can address multiple heterogeneous networks trained with different modalities. Thus, we employ four different modalities of data ((1) Image, (2) Audio, (3) IMU, (4) sEMG) by adding two more datasets in order to demonstrate the scalability of YONO on diverse modalities in §4.3. Further, to demonstrate the generalizabilty of YONO's learned codebooks, we select two additional datasets in each of the four modalities and evaluate our system to compress new models trained on these datasets that YONO did not learn during its codebook learning stage (§4.4). Finally, we present the results of our online in-memory model execution and swap operations in §4.5.

4.1 Experimental Setup

4.1.1 Task. Our target application scenarios are based on dealing with dissimilar multitask learning. For example, those applications are image classification, keyword spotting, human activity recognition, and gesture recognition.

4.1.2 Evaluation Protocol. Following prior works [62, 75], 10% of data is used as the test set and the remaining as the training set. In addition, to evaluate the effectiveness of the offline phase component of our system, we report the accuracy and compression rate of the compressed models using our system. We also use compressed model's error rate (i.e., accuracy loss) compared to the original model. Then, to evaluate the efficiency of the online phase component of our system, we report the execution time and load/swap time of the models on MCU.

4.1.3 Baseline Systems. To evaluate the effectiveness of our work, **YONO**, we include various baselines in our experiments as follows.

NWV: Neural Weight Virtualization (NWV) [44] is the state-of-the-art heterogeneous MTL system that treats weights of neural networks as consecutive memory locations which can be virtualized and shared by multiple models. Note that we use reported results of [44] on an MCU, which relies on simplified LeNet architecture.

Scalar Quantization (Int8): This baseline compresses a single model by quantizing 32-bit floats into low-precision fixed-point representation (e.g., 8-bit) [28, 34]. As in [34], we employ both post-training quantization and quantization-aware training schemes. We then report the results of the best-performing scheme in our evaluation. Besides, we only include 8-bit quantization as sub-byte datatypes (e.g., 4-bit or 2-bit) are not natively supported by MCUs [5]. We leave sub-byte quantization as future work.

PQ-S: This baseline uses PQ to compress a single model to a pair of the shared codebooks across layers in the model. As this baseline does not share the codebooks across multiple models, this can serve as a baseline for the single model compression and as the lower bound in compression ratio among the PQ variants.

PQ-M: This baseline uses PQ to compress multiple heterogeneous models to a pair of the shared codebooks but does not apply our optimization process and heuristics as described in Section 2. We include this to conduct an ablation study to evaluate the impact of the proposed optimization in our system.

PQ-MOpt: This baseline uses PQ to compress multiple heterogeneous models to a pair of the shared codebooks and also apply the optimization process without the heuristics described in Section 2. We include this to conduct an ablation study to evaluate the impact of the heuristics in our system.

Uncompressed (Original): An original model before compression. It is pretrained with available training data and serves as the upper bound in terms of the accuracy metric.

4.2 Performance

Following [44], we start by evaluating YONO in MTL scenarios on two modalities: images and audio signals which are widely used data modalities in mobile sensing applications.

Datasets. We employ the same datasets used in the prior work [44] to make a fair comparison. First, four image datasets are employed, namely MNIST [42], CIFAR-10 [35], SVHN [59], and GTSRB [70] associated with classifying objects of handwritten digits (grayscale), generic objects, numbers (RGB), and road signs, respectively. Then, one audio dataset of Google Speech Commands V2 (GSC) [78] for keyword spotting is used.

Model Architecture. We adopt optimized neural network architectures, designed to be used in the resource-constrained setting,

Table 1: Summary of datasets, model architectures, mobile applications used in §4.2 and §4.3.

Modality	Dataset	Architecture	Mobile Application
Image	MNIST	LeNet	Digit recognition
	CIFAR-10	MicroNet-AD	Object recognition
	SVHN	MicroNet-AD	Digit recognition
	GTSRB	MicroNet-AD	Road sign recognition
Audio	GSC	MicroNet-KWS	Keyword spotting
IMU	HHAR	MicroNet-AD	Activity recognition
sEMG	Ninapro DB2	Lightweight CNN	Gesture recognition

such as variants of MicroNet [5], simplified LeNet used in [44]. For MNIST, we use the simplified LeNet as it is used in [44] and the accuracy of such LeNet variant is very high at 98%. For other datasets (CIFAR-10, SVHN, GTSRB, GSC), we use variants of MicroNet architecture to construct pretrained models. To identify a high-performing and yet lightweight model to operate on embedded and mobile devices, we conduct a hyper-parameter search based on different variants of MicroNet (e.g., small, medium, large models), lightweight convolutional neural network (CNN) architectures [38], the number of convolutional filters. A basic convolutional layer consists of 3×3 convolution, batch normalization, and Rectified Linear Unit (ReLU). Then, as our final model architectures, we use MicroNet-KWS-M for GSC and MicroNet-AD-M (with the reduced number of convolutional filters {192}) for CIFAR-10, SVHN, GTSRB. Throughout model training for all of the datasets, ADAM optimizer [33] and learning rate of 0.001 are used. The datasets, architectures, and applications are summarized in Table 1.

Accuracy. We show the accuracy results here. Figure 3 shows the accuracy of each baseline so that we can analyze the impact of our proposed techniques in our system. To begin with, the uncompressed (original) model serves as a performance upper bound. 8-bit quantization and PQ-S achieve high accuracy close to that of the original model, showing a small average error rate of 0.9% and 2.8%, respectively, between each of the five models after compression and their corresponding original models. However, in the case of the CIFAR-10 dataset, PQ-S shows high error rates of 6.3% on average. This result indicates that the specialized codebooks which target only one model can help retain the performance of the original model in general but sometimes fail to retain it, as shown in the case of CIFAR-10. Besides, PQ-M shows an accuracy loss of 4.3% on average. For CIFAR-10, it shows a high error rate of 9.0%. In addition, although our proposed EM-based iterative network optimization procedure can help in improving the accuracy, PQ-MOpt still shows a substantial accuracy drop of 4.3% on average. This result indicates that compressing multiple neural networks based on only one pair of codebooks is very challenging. However, YONO shows that its accuracy drop is minimal (i.e., an average error rate of 0.4%). Interestingly, in the case of GSC, YONO outperforms the accuracy of the original model by 1.2% where YONO benefits from sharing weights via PQ codebooks.

This result indicates that YONO can effectively retain the accuracy of original models as observed in the prior work on multiple MTL systems [44] and other techniques focusing on a single model compression [25, 28, 74]. Further, it is an interesting result

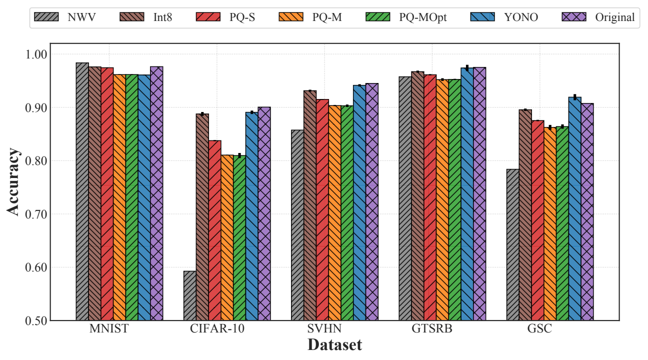


Figure 3: The inference accuracy of the heterogeneous MTL systems trained with five datasets of two modalities. Reported results are averaged over five trials, and standard-deviation intervals are depicted.

Table 2: The compression efficiency of the heterogeneous MTL systems trained with five datasets of two modalities.

	NWV [44]	**Int8**	**PQ-S**	**PQ-M**	**PQ-MOpt**	**YONO**	**Original**
Ratio	8.08×	3.04×	9.47×	12.07×	12.07×	11.57×	1×
Size	0.13 MB	0.96 MB	0.31 MB	0.24 MB	0.24 MB	0.25 MB	2.91 MB

because YONO can retain the accuracy of multiple heterogeneous models, which is more challenging given that simply performing MTL would lead to the accuracy drop as shown in the prior work, NWV [44]. Also, note that differently from [44] which used LeNet, we used optimized network architectures such as MicroNet and lightweight CNN that can execute on resource-constrained MCUs (refer to §4.5) and obtain very high accuracy. To name a few, the pretrained models in our work achieve 90.05%, 94.48%, 90.74% on CIFAR-10, SVHN, GSC compared to 59.26%, 85.74%, 78.38% reported in [44], respectively.

Compression Efficiency. Table 2 shows the overall efficiency in compressing heterogeneous networks trained with five datasets of two modalities. First, the combined storage overhead of the five uncompressed models is 2.91 MB which is three times the capacity of our target MCU's storage, which is 1 MB at maximum. However, considering that to perform an inference on MCUs, it is required to have a space for program codes of TFLM, input and output peripherals, input and output buffers, and other variables, etc., the space used to store models needs to be below the storage size of 1 MB. Thus, it is impossible to put those five models on an MCU and run multitask applications using the uncompressed models. 8-bit quantization shows the lowest compression rate of 3.04× among all the evaluated methods, and its storage size (0.96 MB) is just below the limit of our employed MCU. Then, PQ-S shows a moderate compression rate of 9.47× and decreases the required storage size down to 0.31 MB and thus can reside on an MCU. Other baseline systems, PQ-M and PQ-MOpt, show a high compression rate of 12.07× and reduce the storage requirement to 0.24 MB. This is because PQ-M and PQ-MOpt share the same codebooks across the different applications. However, the savings in storage come at the expense of loss of accuracy, as seen in the

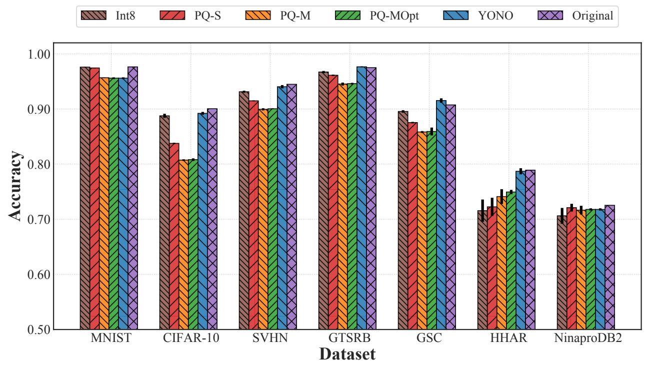


Figure 4: The inference accuracy of the heterogeneous MTL systems trained with seven datasets of four modalities. Reported results are averaged over five trials, and standard-deviation intervals are depicted.

accuracy results discussed before. In contrast, YONO achieves the best of both worlds, demonstrating a high compression rate close to PQ-M and PQ-MOpt and negligible accuracy loss compared to the uncompressed models. YONO obtains a 11.57× compression rate and decreases the storage overhead to 0.25 MB, showing a higher compression rate than NWV [44].

Overall, the results indicate that YONO can enable running multi-task applications on MCUs while retaining high accuracy and low storage footprints.

4.3 Scalability

In this subsection, we apply YONO on seven datasets consisting of four different data modalities ((1) Image, (2) Audio, (3) IMU, (4) sEMG) to investigate to what extent our system can effectively compress multiple networks trained on different data modalities without losing its accuracy and compressive power. We select IMU and sEMG as additional modalities because they are also widely used in mobile sensing applications [14, 72].

Datasets. On top of the five datasets used in the previous subsection, we add two datasets of two additional modalities: HHAR [72] and Ninapro DB2 [3], corresponding to activity recognition (based on IMU) and gesture recognition (based on sEMG), respectively. The HHAR and Ninapro DB2 datasets are some of the most widely used HAR and sEMG datasets, respectively.

Model Architecture. To identify the right model architecture for each dataset, we adopt to use the optimized neural architectures and also conduct a hyper-parameter search as described in §4.2. Then, we select the model which shows the best performance. As a result, we use MicroNet-AD for HHAR and lightweight CNN architecture for Ninapro DB2 (see Table 1 for detail).

Accuracy. Figure 4 presents the accuracy results of the seven datasets of four modalities so that we examine the scalability of YONO to various modalities of data. Overall, the accuracy of reconstructed models from baseline systems and YONO is slightly improved since the error rates on the new datasets are smaller than those of the other five datasets. Also, accuracy results of baseline

Table 3: The compression efficiency of the heterogeneous MTL systems trained with seven datasets of four modalities.

	Int8	PQ-S	PQ-M	PQ-MOpt	YONO	Original
Ratio	2.96×	9.27×	12.29×	12.29×	11.77×	1×
Size	1.27 MB	0.41 MB	0.31 MB	0.31 MB	0.32 MB	3.76 MB

systems and YONO reach similar observations as in §4.2. 8-bit quantization shows a small average error rate of 2.0% but a relatively high accuracy variance for HHAR. Also, PQ-S achieves high accuracy close to that of the uncompressed models with small error rates of 3.0% on average, whereas it shows high error in CIFAR-10. Then, the PQ-M and PQ-MOpt systems present an average error rate of 4.2% and 4.0% respectively, indicating that our proposed EM-based iterative network optimization procedure help improve the accuracy but still falls short of achieving the original model's accuracy. Also, in this setting, YONO performs the best and shows an negligible accuracy loss of 0.5% on average.

Compression Efficiency. Table 3 shows the overall efficiency in compressing heterogeneous models trained with seven datasets of four modalities. Similar to the compression results in §4.2, the total size of the seven uncompressed models (3.76 MB) is larger than the storage budget for our target MCU. In the case of 8-bit quantization, the required storage size of the seven compressed models is 1.27 MB, larger than our storage budget of 1 MB. This result indicates that 8-bit quantization is not suitable for operating many heterogeneous neural networks simultaneously on our target MCU. However, YONO requires at most 0.32 MB. Since our system can effectively compress multiple heterogeneous models (showing 11.77× compression ratio), the incurred storage requirement is minimal. For example, when two additional models (for HHAR and Ninapro DB2) are included in an MTL system, YONO incurs only 0.07 MB additional overhead, whereas the original models' storage size increases by 0.85 MB.

To summarize, our results show that YONO is scalable as it can accommodate many applications utilizing different input modalities while achieving high performance and small storage overhead.

4.4 Generalizability

We now investigate the generalizability of our multitasking system on new models/datasets and different network architectures unseen during the codebook learning phase of the offline component. Specifically, we evaluate whether YONO can achieve high accuracy on the unseen models from new datasets using the same codebooks that are learned previously (§4.3). This can be particularly useful since the learned codebooks of YONO can still be utilized to compress unseen models in different network architectures from new datasets without learning new codebooks again whenever a user wants to incorporate a new task/dataset into the system. Also, note that since the codebooks are not modified, the reported results in §4.3 are not affected, ensuring high accuracy on previous datasets. Then, in §4.4, we select two new datasets in each of the four modalities for a robust evaluation.

Datasets. In total, we add eight new datasets: two image datasets (1) FashionMNIST [81], (2) STL-10 [9], and two audio datasets (3) EmotionSense [62], (4) UrbanSound [66], and two HAR datasets

Table 4: Summary of datasets, model architectures, mobile applications used in §4.4.

Modality	Dataset	Architecture	Mobile Application
Image	FashionMNIST	DS-CNN	Object recognition
	STL-10	DS-CNN	Object recognition
Audio	EmotionSense	Lightweight CNN	Emotion recognition
	UrbanSound	DS-CNN	Sound classification
IMU	PAMAP2	MicroNet-AD	Activity recognition
	Skoda	MicroNet-AD	Activity recognition
sEMG	Ninapro DB3	Lightweight CNN	Gesture recognition
	Ninapro DB6	MicroNet-AD	Gesture recognition

(5) PAMAP2 [64], (6) Skoda [71], and lastly two sEMG datasets (7) Ninapro DB3 [3] and (8) Ninapro DB6 [60]. These are widely used real-world application datasets corresponding to classification problem as follows: (1) ten fashion items, (2) ten generic objects, (3) five emotions, (4) ten environmental sounds, (5) 12 activities, (6) ten activities, (7) ten gestures of amputees, (8) seven gestures of ordinary people, respectively.

Model Architecture. To demonstrate that YONO can effectively address new network architectures that were not shown during the offline codebook learning phase, we include another widely used architecture, DS-CNN [86], in our work. Then, we follow the same hyper-parameter search process as described in §4.2. Table 4 summarizes the identified network architectures for each dataset and its associated mobile application.

Accuracy. Note that we exclude PQ-S as it needs to learn PQ codebooks on a given dataset and then perform network finetuning on the given dataset. However, in this scenario, the system needs to adapt to new (unseen) datasets. This point makes the scenario particularly challenging since an MTL system needs to incorporate unseen datasets and network architectures. Nonetheless, an MTL system that can address this challenge could become very useful in practice since it is adaptable.

To begin with, Figure 5 shows the accuracy results on the eight unseen datasets with diverse network architectures. 8-bit quantization presents a moderate error rate of 2.5% similar to the results in §4.2 and §4.3 as the current evaluation setup does not make a difference for the single model compression approach. Conversely, PQ-M shows a substantial accuracy drop (9.4%) compared to the original model, which is worse than the previous two scenarios where it obtained error rates of 4.3% and 4.2%. In fact, on one dataset (Ninapro DB6), PQ-M shows a 33.0% error rate. Although PQ-MOpt improves upon PQ-M, the amount of improvement is small. PQ-MOpt shows a 8.4% accuracy drop on average compared to the original model. Also, for Ninapro DB6, the accuracy of PQ-MOpt shows a sharp decrease of 28.1% compared to the original model, demonstrating the difficulty of this scenario. Surprisingly, however, YONO does not experience a considerable accuracy loss. It shows only 0.6% accuracy loss on average. Besides, YONO shows a low variance of accuracy loss across the employed datasets. In fact, YONO even improves upon the accuracy of uncompressed models for some datasets such as EmotionSense, Skoda, and Ninapro DB6.

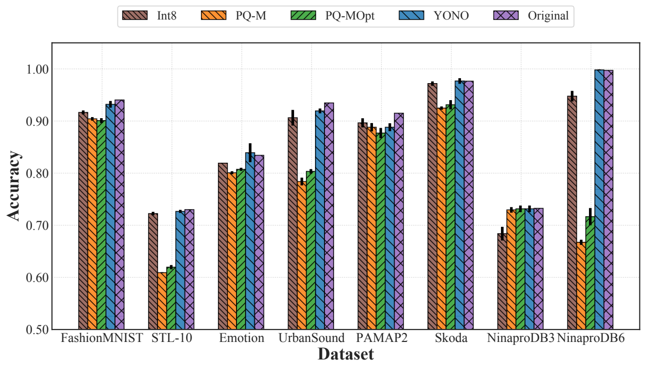


Figure 5: The inference accuracy of the heterogeneous MTL systems applied to unseen datasets of four modalities. Reported results are averaged over five trials, and standard-deviation intervals are depicted.

Table 5: The compression efficiency of the heterogeneous MTL systems applied to unseen datasets of four modalities.

	Int8	PQ-M	PQ-MOpt	YONO	Original
Ratio	2.80×	13.60×	13.60×	12.37×	1×
Size	1.47 MB	0.30 MB	0.30 MB	0.33 MB	4.11 MB

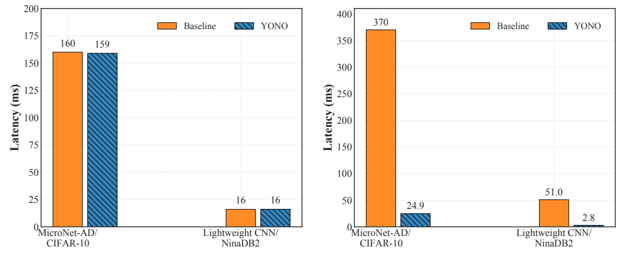
These results highlights that YONO is capable of retaining the accuracy of original models even in the most challenging scenario of incorporating unseen datasets and architectures.

Compression Efficiency. The compression results for heterogeneous models with eight unseen datasets are shown in Table 5. The size of the uncompressed models is the largest, 4.11 MB, in this setup compared to §4.2 and §4.3. YONO shows an impressive compression ratio of 12.37× and require storage size of 0.33 MB after compressing eight heterogeneous networks. It is worth noting that we included a new network architecture that YONO did not learn during its offline codebook learning phase. Yet, YONO successfully compress different architectures with an even higher compression rate (11.57× in §4.2 and 11.77× in §4.3) without loss of accuracy on all the unseen datasets.

In summary, the results here hint that YONO can effectively compress different heterogeneous models trained on unseen datasets without losing accuracy and demonstrate the generalizability of YONO's codebooks and the effectiveness of the proposed network optimization and optimization heuristics.

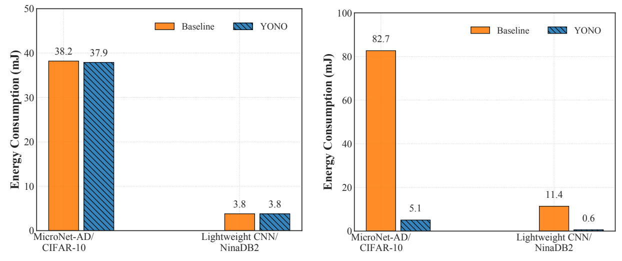
4.5 Evaluation on In-Memory Execution and Model Swapping Framework on MCUs

We finally examine the run-time performance of the online component of YONO, the in-memory execution and model swapping framework, introduced in §2.6. In specific, we evaluate the latency and energy consumption of model execution and model swapping of YONO on an MCU. Also, we include an alternative approach to YONO as a baseline that relies on an external SD card as a secondary storage device for storing heterogeneous networks



(a) Execution (b) Loading/Switching

Figure 6: The model execution and loading/switching time of YONO and the baseline.



(a) Execution (b) Loading/Switching

Figure 7: The energy consumption of model execution and loading/switching of YONO and the baseline.

and on in-memory execution similar to YONO. We employ the same datasets used in the previous subsections. In Figures 6 and 7, we report the results of upper bound (i.e., slowest or the most energy-consuming) and lower bound (i.e., fastest or the least energy-consuming) to show the range of latency and energy consumption of YONO and the baseline based on the identified network architectures trained on the datasets in §4.2-§4.4 (see Tables 1 and 4). We use a MicroNet-AD model based on CIFAR-10 as upper bound and a lightweight CNN model based on Ninapro DB2 as lower bound. Although results for other models and datasets are omitted, they reside within the reported latency and energy consumption as in Figures 6 and 7.

Latency. We measure the latency of the model execution and model loading/swap by using MBed Timer API, as shown in Figure 6. In terms of execution time, both YONO and the baseline show a swift execution time (16-160 ms per inference) that can be useful in practice, and there is no meaningful latency difference between them since both rely on in-memory execution. However, for model loading/swap time, YONO accelerates the model switching. YONO reduces model loading/swap time by 93.3% (370 ms vs. 24.9 ms) in a MicroNet-AD model based on CIFAR-10 and 94.5% (51.0 ms vs. 2.8 ms) in a lightweight CNN model based on Ninapro DB2 compared to the baseline. Note that we did not conduct a direct comparison on-device with the prior work [44] since its source code is not shared and the used MCUs for experiments are not the same.

Energy Consumption. We measure the energy consumption of model execution and loading/swap on the MCU using YONO and the baseline, as shown in Figure 7. We use the Tenma 72-7720 digital multimeter to measure the power consumption and then compute the energy consumption over time taken for each operation (i.e., inference and model loading). Similar to the latency result, the energy consumption for executing models does not show the difference as explained above. However, for the model loading/swap task, YONO decreases energy consumption by at minimum 93.9% (82.7 mJ vs. 5.1 mJ in a MicroNet-AD model on CIFAR-10) and at maximum 95.0% (11.4 mJ vs. 0.6 mJ in a lightweight CNN model on Ninapro DB2) compared to the baseline.

To summarize, the results demonstrate that YONO enables fast (low latency) and efficient (low energy footprints) model execution and loading/swap on an extremely resource-limited IoT device, MCU.

5 DISCUSSION

Impact on Heterogeneous MTL Systems. YONO represents the first framework that can compress multiple heterogeneous models and be applicable to unseen datasets. Also, YONO ensures negligible or no loss of accuracy in compressing many different models (architecture) on multiple datasets. This is achieved by only one pair of PQ-based codebooks, our novel optimization procedure, and heuristics. Thus, we envisage that YONO could become a practical system to deploy heterogeneous MTL systems on various embedded devices and platforms in many real-world applications in the future. We leave the wide deployment and performance evaluation of YONO on other embedded platforms under real-world application scenarios as future work.

Application Scenario. Let us consider an example of a real-world application. Given an intelligent authentication system for a smart home, the system would need to detect tenants' identification based on images and voice (image classification and voice recognition). Then, the system could take voice commands as inputs from the identified tenant (e.g., keyword spotting). This simple application scenario already needs three different models, which could satisfy the necessity of a heterogeneous MTL system, YONO.

Generalizability of YONO. In Section 4, we have demonstrated that YONO can incorporate heterogeneous models and datasets (four different modalities) consisting of 15 datasets (i.e., seven datasets for learning codebooks in §4.3 and the other eight unseen datasets in §4.4), which shows that YONO is a generalizable framework. Other datasets and network architectures (e.g., LSTMs [22] and CNNs with large-sized kernels like 5x5 or 7x7) that can be employed and tested on YONO are left as future work.

Limitation. To enable model switching during the runtime, we design YONO to load the model in the main memory instead of the storage of an MCU. However, since SRAM is a limited on-chip resource and typically smaller than eFlash, our design choice may limit the applicability of YONO, especially for low-end MCUs with smaller SRAM sizes such as 128 KB. Therefore, it would be worthwhile to further investigate memory-efficient ways to reduce the required main memory space for model execution while enabling the model switching at run time. Better usage of FlatBuffer serialization format to hold model weights can be interesting future work since the weights of a model takes the majority of the space.

6 RELATED WORK

Multitask Learning. Multi-task learning allows learning correlated tasks such that accuracy of both or one of the tasks is improved by exploiting the similarities and differences across tasks [7]. Common approaches include common feature learning [50, 58], low-rank parameter search [24, 57], task clustering [23, 32], and task relation learning [43, 51]. These works achieve limited compression by sharing the first few network layers. However, their main goal is to increase the robustness and generalization of multiple task learners. Thus, keeping multiple heterogeneous DNN models into the extremely limited memory of embedded devices, along with managing and executing these models (achieving different tasks) efficiently at run-time, are challenging to the aforementioned works. Comparing this, YONO allows to run multiple DNN models efficiently while remaining within the limited resource constraints on embedded devices.

Besides, NWV [44] was introduced to compress multiple heterogeneous models of different network architectures and tasks. NWV also minimizes the context switching overhead by retaining all shared weights on the memory. However, NWV's compression ratio is constrained to 8.08×, limiting the multi-tasking IoT system with a small memory footprint to operate many tasks in real-time. Also, the work only employs a simplified LeNet architecture in the experiments of IoT use cases, and thus the accuracy of the system is limited. Conversely, YONO not only increases compression rates but compresses even the highly optimized models (e.g., MicroNet, DS-CNN), while achieving high accuracy that is useful in practice.

Mobile and Embedded Sensing Applications. Deep learning is increasingly being applied in mobile and embedded systems as it achieves state-of-the-art performances on many sensing applications such as computer vision applications [15], audio sensing [40], activity recognition [72], gesture recognition [3]. First of all, there exist many vision applications, to name a few, tiny image classification [35, 59], traffic sign recognition [70]. Besides, audio sensing application is also one of the foundational mobile sensing applications [37, 65] that much research has focused on to deliver behavioral insights to users. The audio sensing tasks include Emotion Recognition (ER) [62], Speaker Identification [52], Environmental Sound Classification (ESC) [75], and Conversation Analysis [45], and Keyword Spotting (KWS) [86]. Next, one of the most widely studied mobile sensing application is HAR [22, 77], where the aim is to determine various human activities automatically using body-worn IMU (Inertial Movement Units) sensors. In application frequently used in mobile sensing is to recognize hand gestures (e.g., fist and open palm) using sEMG (surface Electromyography) signals generated during muscle contractions [6, 14]. sEMG signal is used for medical [84], rehabilitation [79], human-computer interactions [39, 69], upper-limb prostheses control [68], and authentication [8].

Model Compression. Many researchers focus on developing a method to improve efficiency without sacrificing the model's accuracy due to a large burden of training deep network architecture and its data [80]. First of all, many researchers have focused on designing and hand-drafting more efficient network architectures, namely, SqueezeNets [19], ShuffleNets [54], and MobileNets [27, 67], and MicroNet [5]. In particular, we employ MicroNet as one of our backbone network architectures since it shows impressive performance and efficiency on tiny IoT systems such as MCUs.

In addition, another thread of research is weight pruning methods that leverage the inherent redundancy in the weights of neural networks [25, 47, 49, 55, 82, 85]. Furthermore, quantization of model weights and activiations has been an active area of research. Many prior works quantize the weights and activations from 32-bit float to 8-bit integer [28], ternary values (2-bit) [46, 88], binary values (1-bit) [2, 10, 11, 63], and mixed precision [76, 83]. Also, weight clustering methods are proposed to group weights into several clusters to compress a model.

Moreover, researchers studied techniques that quantize an array of scalars of the weights to compress a model or a particular layer. Some works extended a sparse coding [87] to learn a compact representation that covers the feature space of weights of a model [4, 18]. Also, many researchers examined vector quantization-based methods [80]. For example, Gong et al. [20] conducted an empirical study to compare binarized networks, scalar quantization using k-means (i.e., weight clustering), Product Quantization (PQ) [30]. Several recent works apply PQ to compress a deep neural network with more than 11 million parameters [56, 73, 74]. Albeit its impressive results, all the prior works only focused on utilizing PQ to a single and bulky model at a scale of millions of parameters, lacking the understanding of how the method can be used to deal with heterogeneous MTL applications and compress tiny models that should fit into the extremely limited memory budget of MCUs (less than 512 KB). Thus, for the first time in this work, we develop YONO, a PQ-based model compression framework that operates heterogeneous models on tiny IoT devices. We propose a novel network optimization procedure and heuristics to achieve high accuracy close to the uncompressed models. Also, YONO enables fast and efficient model execution and swapping on an MCU.

7 CONCLUSIONS

We have presented an efficient MTL system, YONO, that compresses multiple heterogeneous models through PQ codebooks, our novel network optimization and heuristics. First, we implemented YONO's offline component on a server and its online component on a critically resource-constrained MCU. Then, we demonstrated its effectiveness and efficiency. YONO compresses multiple heterogeneous models up to 12.37× with minimal or near to no accuracy loss. Interestingly, YONO can successfully compress models trained with datasets unseen during its offline codebook learning phase. Finally, YONO's online component enables an efficient in-memory model execution and loading/swap with low latency and energy footprints on an MCU. We envision that methods developed for YONO and our research findings could pave the way to deploy practical heterogeneous multi-task deep learning systems on various embedded devices in the near future.

ACKNOWLEDGMENTS

This work is supported by a Google Faculty Award, ERC through Project 833296 (EAR), and Nokia Bell Labs through their donation for the Centre of Mobile, Wearable Systems and Augmented Intelligence to the University of Cambridge.

REFERENCES

[1] 2016. Wearable Device for Blind People Could be a Life Changer | NVIDIA Blog. https://blogs.nvidia.com/blog/2016/10/27/wearable-device-for-blind-visually-impaired/

[2] Milad Alizadeh, Javier Fernández-Marqués, Nicholas D. Lane, and Yarin Gal. 2018. An Empirical study of Binary Neural Networks' Optimisation. (Sept. 2018).

[3] Manfredo Atzori, Arjan Gijsberts, Claudio Castellini, Barbara Caputo, Anne-Gabrielle Mittaz Hager, Simone Elsig, Giorgio Giatsidis, Franco Bassetto, and Henning Müller. 2014. Electromyography data for non-invasive naturally-controlled robotic hand prostheses. *Scientific Data* 1 (Dec. 2014), 140053.

[4] Hessam Bagherinezhad, Mohammad Rastegari, and Ali Farhadi. 2017. LCNN: Lookup-Based Convolutional Neural Network. 7120–7129.

[5] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul Whatmough. 2021. MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers. *Proceedings of Machine Learning and Systems* 3 (March 2021).

[6] Vincent Becker, Pietro Oldrati, Liliana Barrios, and Gábor Sörös. 2018. Touchsense: Classifying Finger Touches and Measuring Their Force with an Electromyography Armband. In *Proceedings of the 2018 ACM International Symposium on Wearable Computers (ISWC '18)*. 1–8.

[7] Rich Caruana. 1997. Multitask Learning. *Machine Learning* 28, 1 (July 1997), 41–75.

[8] Jagmohan Chauhan, Young D. Kwon, Pan Hui, and Cecilia Mascolo. 2020. ContAuth: Continual Learning Framework for Behavioral-Based User Authentication. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 4, 4, Article 122 (Dec. 2020), 23 pages. https://doi.org/10.1145/3432203

[9] Adam Coates, Andrew Ng, and Honglak Lee. 2011. An Analysis of Single-Layer Networks in Unsupervised Feature Learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 15)*, Geoffrey Gordon, David Dunson, and Miroslav Dudík (Eds.). PMLR, Fort Lauderdale, FL, USA, 215–223.

[10] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. *Advances in Neural Information Processing Systems* 28 (2015).

[11] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv:1602.02830 [cs]* (March 2016).

[12] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, Pete Warden, and Rocky Rhodes. 2021. TensorFlow Lite Micro: Embedded Machine Learning for TinyML Systems. In *Proceedings of Machine Learning and Systems*, A. Smola, A. Dimakis, and I. Stoica (Eds.), Vol. 3. 800–811.

[13] A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)* 39, 1 (1977), 1–38.

[14] Junjun Fan, Xiangmin Fan, Feng Tian, Yang Li, Zitao Liu, Wei Sun, and Hongan Wang. 2018. What is That in Your Hand?: Recognizing Grasped Objects via Forearm Electromyography Sensing. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 4 (Dec. 2018), 161:1–161:24.

[15] Biyi Fang, Xiao Zeng, and Mi Zhang. 2018. NestDNN: Resource-Aware Multi-Tenant On-Device Deep Learning for Continuous Mobile Vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18)*. 115–127.

[16] Igor Fedorov, Ryan P. Adams, Matthew Mattina, and Paul Whatmough. 2019. SpArSe: Sparse Architecture Search for CNNs on Resource-Constrained Microcontrollers. 4977–4989.

[17] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2014. Optimized Product Quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 4 (April 2014), 744–755.

[18] Tiezheng Ge, Kaiming He, and Jian Sun. 2014. Product Sparse Coding. 939–946.

[19] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter Jin, Sicheng Zhao, and Kurt Keutzer. 2018. SqueezeNext: Hardware-Aware Neural Network Design. 1638–1647.

[20] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing Deep Convolutional Networks using Vector Quantization. *arXiv:1412.6115 [cs]* (Dec. 2014).

[21] R. Gray. 1984. Vector quantization. *IEEE ASSP Magazine* 1, 2 (April 1984), 4–29.

[22] Yu Guan and Thomas Plötz. 2017. Ensembles of Deep LSTM Learners for Activity Recognition Using Wearables. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 2 (June 2017), 11:1–11:28.

[23] Lei Han and Yu Zhang. 2015. Learning multi-level task groups in multi-task learning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

[24] Lei Han and Yu Zhang. 2016. Multi-stage multi-task learning with reduced rank. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30.

[25] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv:1510.00149 [cs]* (Feb. 2016).

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). arXiv:1512.03385

[27] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861 [cs]* (April 2017).

[28] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2017. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *CoRR* abs/1712.05877. arXiv:1712.05877

[29] Yifei Jiang, Xin Pan, Kun Li, Qin Lv, Robert P. Dick, Michael Hannigan, and Li Shang. 2012. ARIEL: Automatic Wi-fi Based Room Fingerprinting for Indoor Localization. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing (UbiComp '12)*. 441–450.

[30] H. Jégou, M. Douze, and C. Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (Jan. 2011), 117–128.

[31] Y. Kalantidis and Y. Avrithis. 2014. Locally Optimized Product Quantization for Approximate Nearest Neighbor Search. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2329–2336.

[32] Zhuoliang Kang, Kristen Grauman, and Fei Sha. 2011. Learning with whom to share in multi-task feature learning. In *ICML*.

[33] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]* (Jan. 2017).

[34] Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv:1806.08342 [cs, stat]* (June 2018).

[35] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[36] Aditya Kusupati, Manish Singh, Kush Bhatia, Ashish Kumar, Prateek Jain, and Manik Varma. 2018. FastGRNN: A Fast, Accurate, Stable and Tiny Kilobyte Sized Gated Recurrent Neural Network. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.). 9017–9028.

[37] Young D Kwon, Jagmohan Chauhan, Abhishek Kumar, Pan Hui, and Cecilia Mascolo. 2021. Exploring System Performance of Continual Learning for Mobile and Embedded Sensing Applications. In *ACM/IEEE Symposium on Edge Computing*.

[38] Young D. Kwon, Jagmohan Chauhan, and Cecilia Mascolo. 2021. FastICARL: Fast Incremental Classifier and Representation Learning with Efficient Budget Allocation in Audio Sensing Applications. In *Proc. Interspeech 2021*. 356–360. https://doi.org/10.21437/Interspeech.2021-1091

[39] Young D. Kwon, Kirill A. Shatilov, Lik-Hang Lee, Serkan Kumyol, Kit-Yung Lam, Yui-Pan Yau, and Pan Hui. 2020. MyoKey: Surface Electromyography and Inertial Motion Sensing-based Text Entry in AR. In *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. 1–4.

[40] Nicholas D. Lane, Petko Georgiev, and Lorena Qendro. 2015. DeepEar: Robust Smartphone Audio Sensing in Unconstrained Acoustic Environments Using Deep Learning. In *Proc. UbiComp*. 283–294.

[41] Nicholas D. Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T. Campbell. 2010. A survey of mobile phone sensing. *IEEE Communications Magazine* 48, 9 (Sept. 2010), 140–150.

[42] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (Nov. 1998), 2278–2324.

[43] Giwoong Lee, Eunho Yang, and Sung Hwang. 2016. Asymmetric multi-task learning based on task relatedness and loss. In *International conference on machine learning*. PMLR, 230–238.

[44] Seulki Lee and Shahriar Nirjon. 2020. Fast and scalable in-memory deep multitask learning via neural weight virtualization. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services (MobiSys '20)*. Association for Computing Machinery, New York, NY, USA, 175–190.

[45] Youngki Lee, Chulhong Min, Chanyou Hwang, Jaeung Lee, Inseok Hwang, Younghyun Ju, Chungkuk Yoo, Miri Moon, Uichin Lee, and Junehwa Song. 2013. SocioPhone: everyday face-to-face interaction monitoring platform using multi-phone sensor fusion. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services (MobiSys '13)*. Taipei, Taiwan, 375–388.

[46] Fengfu Li, Bo Zhang, and Bin Liu. 2016. Ternary Weight Networks. *arXiv:1605.04711 [cs]* (Nov. 2016).

[47] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning Filters for Efficient ConvNets. (Nov. 2016).

[48] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. 2020. MCUNet: Tiny Deep Learning on IoT Devices. *arXiv:2007.10319 [cs]* (July 2020).

[49] Ning Liu, Xiaolong Ma, Zhiyuan Xu, Yanzhi Wang, Jian Tang, and Jieping Ye. 2020. AutoCompress: An Automatic DNN Structured Pruning Framework for Ultra-High Compression Rates. *Proceedings of the AAAI Conference on Artificial Intelligence* 34, 04 (April 2020), 4876–4883.

[50] Wu Liu, Tao Mei, Yongdong Zhang, Cherry Che, and Jiebo Luo. 2015. Multi-task deep visual-semantic embedding for video thumbnail selection. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 3707–3715.

[51] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Philip S Yu. 2015. Learning multiple tasks with multilinear relationship networks. *arXiv preprint arXiv:1506.02117* (2015).

[52] Hong Lu, A. J. Bernheim Brush, Bodhi Priyantha, Amy K. Karlson, and Jie Liu. 2011. SpeakerSense: energy efficient unobtrusive speaker identification on mobile phones. In *Proceedings of the 9th international conference on Pervasive computing (Pervasive'11).* San Francisco, USA, 188–205.

[53] Hong Lu, Denise Frauendorfer, Mashfiqui Rabbi, Marianne Schmid Mast, Gokul T. Chittaranjan, Andrew T. Campbell, Daniel Gatica-Perez, and Tanzeem Choudhury. 2012. StressSense: Detecting Stress in Unconstrained Acoustic Environments Using Smartphones. In *Proc. UbiComp.* 351–360.

[54] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. 2018. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. 116–131.

[55] Arun Mallya and Svetlana Lazebnik. 2018. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 7765–7773.

[56] Julieta Martinez, Jashan Shewakramani, Ting Wei Liu, Ioan Andrei Barsan, Wenyuan Zeng, and Raquel Urtasun. 2021. Permute, Quantize, and Fine-Tune: Efficient Compression of Neural Networks. 15699–15708.

[57] Andrew M McDonald, Massimiliano Pontil, and Dimitris Stamos. 2014. Spectral k-Support Norm Regularization.. In *NIPS.* 3644–3652.

[58] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. 2016. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 3994–4003.

[59] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. 2011. Reading digits in natural images with unsupervised feature learning. (2011).

[60] Francesca Palermo, Matteo Cognolato, Arjan Gijsberts, Henning Müller, Barbara Caputo, and Manfredo Atzori. 2017. Repeatability of grasp recognition for robotic hand prosthesis control based on sEMG data. In *2017 International Conference on Rehabilitation Robotics (ICORR).* 1154–1159.

[61] Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuo-Yiin Chang, and Tara Sainath. 2019. Deep Learning for Audio Signal Processing. *IEEE Journal of Selected Topics in Signal Processing* 13, 2 (May 2019), 206–219.

[62] Kiran K. Rachuri, Mirco Musolesi, Cecilia Mascolo, Peter J. Rentfrow, Chris Longworth, and Andrius Aucinas. 2010. EmotionSense: a mobile phones based adaptive platform for experimental social psychology research. In *Proc. UbiComp.* 281–290.

[63] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *Computer Vision – ECCV 2016 (Lecture Notes in Computer Science)*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Cham, 525–542.

[64] A. Reiss and D. Stricker. 2012. Introducing a New Benchmarked Dataset for Activity Monitoring. In *2012 16th International Symposium on Wearable Computers.* 108–109.

[65] Sandra Servia Rodríguez, Cecilia Mascolo, and Young D. Kwon. 2021. Knowing when we do not know: Bayesian continual learning for sensing-based analysis tasks. *arXiv preprint arXiv:2106.05872* (2021).

[66] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. 2014. A Dataset and Taxonomy for Urban Sound Research. In *Proc. ACM MM.* 1041–1044.

[67] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 4510–4520.

[68] Erik Scheme and Kevin Englehart. 2011. Electromyogram pattern recognition for control of powered upper-limb prostheses: state of the art and challenges for clinical use. *Journal of Rehabilitation Research and Development* 48, 6 (2011), 643–659.

[69] Kirill A. Shatilov, Dimitris Chatzopoulos, Alex Wong Tat Hang, and Pan Hui. 2019. Using Deep Learning and Mobile Offloading to Control a 3D-printed Prosthetic Hand. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 3, 3 (Sept. 2019), 102:1–102:19.

[70] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. 2011. The German traffic sign recognition benchmark: a multi-class classification competition. In *The 2011 international joint conference on neural networks.* IEEE, 1453–1460.

[71] Thomas Stiefmeier, Daniel Roggen, Georg Ogris, Paul Lukowicz, and Gerhard Tröster. 2008. Wearable Activity Tracking in Car Manufacturing. *IEEE Pervasive Computing* 7, 2 (April 2008), 42–50.

[72] Allan Stisen, Henrik Blunck, Sourav Bhattacharya, Thor Siiger Prentow, Mikkel Baun Kj\a ergaard, Anind Dey, Tobias Sonne, and Mads Møller Jensen. 2015. Smart Devices Are Different: Assessing and MitigatingMobile Sensing Heterogeneities for Activity Recognition. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems (SenSys '15).* 127–140.

[73] Pierre Stock, Angela Fan, Benjamin Graham, Edouard Grave, Rémi Gribonval, Herve Jegou, and Armand Joulin. 2020. Training with Quantization Noise for Extreme Model Compression.

[74] Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. 2019. And the Bit Goes Down: Revisiting the Quantization of Neural Networks.

[75] Yu Su, Ke Zhang, Jingyu Wang, and Kurosh Madani. 2019. Environment Sound Classification Using a Two-Stream CNN Based on Decision-Level Fusion. *Sensors* 19, 7 (Jan. 2019), 1733.

[76] Diwen Wan, Fumin Shen, Li Liu, Fan Zhu, Jie Qin, Ling Shao, and Heng Tao Shen. 2018. TBN: Convolutional Neural Network with Ternary Inputs and Binary Weights. 315–332.

[77] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. 2019. Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letters* 119 (March 2019), 3–11.

[78] Pete Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *arXiv:1804.03209 [cs]* (April 2018).

[79] Brent D. Winslow, Mitchell Ruble, and Zachary Huber. 2018. Mobile, Game-Based Training for Myoelectric Prosthesis Control. *Frontiers in Bioengineering and Biotechnology* 6 (2018).

[80] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. 2016. Quantized Convolutional Neural Networks for Mobile Devices. 4820–4828.

[81] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.* arXiv:cs.LG/1708.07747 [cs.LG]

[82] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. 2017. Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning. 5687–5695.

[83] Shihui Yin, Zhewei Jiang, Jae-Sun Seo, and Mingoo Seok. 2020. XNOR-SRAM: In-Memory Computing SRAM Macro for Binary/Ternary Deep Neural Networks. *IEEE Journal of Solid-State Circuits* 55, 6 (June 2020), 1733–1743.

[84] Jamileh Yousefi and Andrew Hamilton-Wright. 2014. Characterizing EMG data using machine-learning tools. *Computers in Biology and Medicine* 51 (Aug. 2014), 1–13.

[85] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. 2018. A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers. 184–199.

[86] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. 2017. Hello Edge: Keyword Spotting on Microcontrollers. *arXiv:1711.07128 [cs, eess]* (Nov. 2017).

[87] Jingyuan Zhao, Zhang Sihao, and Zeng Jing. 2016. Review of the sparse coding and the applications on image retrieval. In *2016 International Conference on Communication and Electronics Systems (ICCES).* 1–5.

[88] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. 2016. Trained Ternary Quantization. (Nov. 2016).