LifeLearner: Hardware-Aware Meta Continual Learning System for Embedded Computing Platforms

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

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

Hong Jia
University of Cambridge
United Kingdom
hj359@cam.ac.uk

Stylianos I. Venieris
Samsung AI Center, Cambridge
United Kingdom
s.venieris@samsung.com

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

ABSTRACT

Continual Learning (CL) allows applications such as user personalization and household robots to learn on the fly and adapt to context. This is an important feature when context, actions, and users change. However, enabling CL on resource-constrained embedded systems is challenging due to the limited labeled data, memory, and computing capacity.

In this paper, we propose LifeLearner, a hardware-aware meta continual learning system that drastically optimizes system resources (lower memory, latency, energy consumption) while ensuring high accuracy. Specifically, we (1) exploit meta-learning and rehearsal strategies to explicitly cope with data scarcity issues and ensure high accuracy, (2) effectively combine lossless and lossy compression to significantly reduce the resource requirements of CL and rehearsal samples, and (3) developed hardware-aware system on embedded and IoT platforms considering the hardware characteristics.

As a result, LifeLearner achieves near-optimal CL performance, falling short by only 2.8% on accuracy compared to an Oracle baseline. With respect to the state-of-the-art (SOTA) Meta CL method, LifeLearner drastically reduces the memory footprint (by 178.7×), end-to-end latency by 80.8-94.2%, and energy consumption by 80.9-94.2%. In addition, we successfully deployed LifeLearner on two edge devices and a microcontroller unit, thereby enabling efficient CL on resource-constrained platforms where it would be impractical to run SOTA methods and the far-reaching deployment of adaptable CL in a ubiquitous manner. Code is available at https://github.com/theyoungkwon/LifeLearner.

CCS CONCEPTS

• **Computer systems organization** → *Embedded and cyber-physical systems*; • **Human-centered computing** → *Ubiquitous and mobile computing*.

SenSys '23, November 12–17, 2023, Istanbul, Turkiye

ACM ISBN 979-8-4007-0414-7/23/11.
https://doi.org/10.1145/3625687.3625804

KEYWORDS

Continual Learning, Meta Learning, On-device Training, Latent Replay, Product Quantization, Edge Computing, Microcontrollers.

ACM Reference Format:
Young D. Kwon, Jagmohan Chauhan, Hong Jia, Stylianos I. Venieris, and Cecilia Mascolo. 2023. LifeLearner: Hardware-Aware Meta Continual Learning System for Embedded Computing Platforms. In *The 21st ACM Conference on Embedded Networked Sensor Systems (SenSys '23), November 12–17, 2023, Istanbul, Turkiye.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3625687.3625804

1 Introduction

With the rise of embedded and Internet of Things (IoT) devices, the adoption of deep neural networks (DNN) has revolutionized various applications ranging from computer vision [27], audio [77] and sensing applications [54]. However, in real-world setups, where a deployed model may need to dynamically learn new tasks (i.e., new classes or inputs) from users [8] and adapt to changing input distributions [69], existing learning approaches often fail, due to the constrained nature of available resources on edge devices and *catastrophic forgetting (CF)* [66]. CF describes the situation when a deployed model is able to perform new tasks but forgets previously learned knowledge. Efficient Continual Learning (CL) systems that can learn new tasks from growing data streams [8, 71, 76] are now being recognized as an important step forward as they also enable many practical applications. For example, household robotic devices need to continually learn to recognize new objects, while smart appliances need to learn different voice commands.

Many CL approaches have been proposed in the literature, including *regularization-based* [44, 103], *dynamic architecture-based* [34, 82, 101], and *rehearsal-based methods* [8, 74, 81]. Among these, rehearsal-based methods largely alleviate the forgetting issue of a learned model. Nonetheless, they are excessively data-hungry as they require a large number of labeled samples to learn new information and to be stored as rehearsal samples [71], incurring high computational and memory overheads.

Another stream of work has recently attempted to utilize meta-learning [29] in CL to address the problem of the scarce labeled data. A number of Meta CL methods [4, 37, 55] relying on a few samples of new classes to adapt and learn have been proposed. However, Meta CL's performance degrades when many classes are added during deployment, leading to low scalability (refer to Figure 1a).

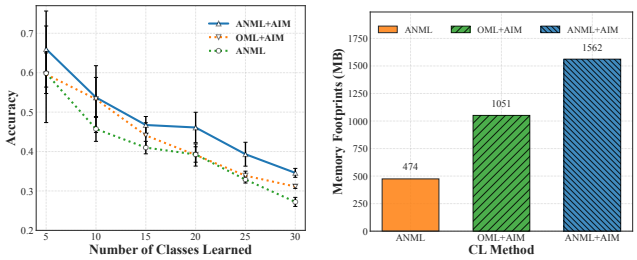
Additionally, state-of-the-art (SOTA) Meta CL methods, OML+AIM and ANML+AIM [55], exhibit large memory footprint, easily exceeding the RAM size on many embedded devices (e.g., 1 GB) (refer to Figure 1b). Further, we observed that the end-to-end latency of SOTA Meta CL methods to continually learn multiple classes is computationally expensive. These aspects render prior Meta CL methods not deployable on resource-constrained devices. As such, there is an emerging need for novel system design approaches that facilitate the broader deployment of CL systems on various IoT devices by bringing down resource requirements of CL methods without jeopardizing their accuracy.

To address the aforementioned limitations, we develop *LifeLearner*, the first hardware-aware system that fully enables data- and memory-efficient CL on the constrained edge and IoT devices. **First,** contrary to the existing Meta CL methods that primarily rely on regularization and suffer from accuracy loss, we introduce *rehearsal-based Meta CL*; we co-design meta-learning with an efficient rehearsal strategy, enabling LifeLearner to rapidly learn new classes using only a few samples while alleviating catastrophic forgetting of the already learned classes upon deployment (Section 3.1). **Second,** we propose a *CL-tailored algorithm/software co-design approach* that minimizes the on-device resource overheads of CL. At the algorithmic level, we design a latent replay scheme, where rehearsal samples are extracted from an intermediate layer of the target DNN instead of holding copies of raw inputs. By strategically selecting the rehearsal layer for high compressibility, we facilitate the subsequent compression of rehearsal samples, enabling their efficient storage on-device. Besides, based on an observation that latent replays are sparse, we further design a novel *Compression Module* via an intelligent combination of lossless compression to utilize sparsity and lossy compression to yield a high compression rate, fast encoding and decoding, and minimal resource usage (Section 3.2). **Finally,** we develop our *hardware-aware system* by employing hardware-friendly optimization techniques and considering the unique characteristics of hardware (e.g., write operation on Flash of IoT devices is costly during runtime) to optimize the runtime efficiency of CL operations on-device (Section 4).

We make the following key contributions:

(1) A novel Meta CL method comprises a rehearsal strategy that alleviates catastrophic forgetting and a deployment-time inner-and outer-loop training structure that achieves both fast adaptation to new classes and refreshing of already learned classes. LifeLearner achieves previously unattainable levels of on-device accuracy, outperforming all existing Meta CL methods by 4.1-16.1% on image and audio datasets, while being within 2.8% of an oracle.

(2) A new algorithm/software co-design method that co-optimizes the rehearsal strategy and the compression pipeline to significantly reduce the resource requirements of CL and rehearsal samples. As a result, LifeLearner requires only 3.40–15.45 MB of memory and obtains a compression rate of 11.4–178.7× compared to the SOTA Meta CL method, ANML+AIM. This allows LifeLearner to run on edge devices, something impossible for current SOTA methods due to their large memory requirements (>1.05 GB).



(a) Performance (b) Memory Overhead

Figure 1: Preliminary analysis of the prior Meta CL methods (i.e., ANML, OML+AIM, ANML+AIM). (a) shows the CL accuracy degradation of the Meta CL methods after learning c number of classes on CIFAR-100 [47]. (b) shows the memory footprint needed to run the Meta CL methods on MiniImageNet [95] with a batch size of 8.

(3) Our hardware-aware system implementation successfully deployed LifeLearner on two embedded devices (Jetson Nano and Raspberry Pi 3B+) and a microcontroller (STM32H747). Through extensive experiments, we demonstrate that LifeLearner outperforms existing CL and Meta CL baselines in terms of latency and energy consumption. Specifically, compared to ANML+AIM, LifeLearner obtains 80.8-94.2% lower end-to-end latency and 80.9-94.2% lower energy consumption on Jetson Nano. Also, we developed LifeLearner on an extremely resource-constrained IoT device, STM32H747 with 512 KB of SRAM (2,000× smaller memory than Pi 3B+ with 1 GB RAM). To our knowledge, this is the first implementation of a CL framework onto this constrained and challenging platform, opening the door for the ubiquitous deployment of learning systems adaptive to users and environments over time continually.

2 Background and Motivation

2.1 Continual Learning

CL allows DNN models to learn over time from non-stationary data streams by acquiring new knowledge while avoiding the forgetting problem of the learned experiences [39, 48, 71]. In the literature, various approaches attempt to solve the forgetting problem [13, 65, 66]. The first group of approaches includes regularization-based methods [2, 44, 84, 85, 103]: these add a regularization term to the loss function to minimize changes to important weights of a model for previously learned classes to prevent forgetting. This approach can be very efficient regarding computation and memory costs. However, it is shown to be less effective than other methods that utilize additional resources such as expanding architectures and storing additional samples [13], as introduced in the following. The second group of approaches includes the dynamic architecture-based methods [34, 82, 101] that dynamically expand and freeze DNN architectures to incorporate new classes and prevent forgetting. Despite the promising performance, dynamic architectures pose the costly requirement of modifying the model architecture. This leads to higher computational costs as the model expands and prohibits the utilization of compile-time optimizations on a fixed computation graph

of the model. The last group of approaches among conventional CL includes rehearsal-based methods [7, 8, 26, 49, 63, 67, 74, 81, 98]. These prevent forgetting by replaying the saved rehearsal samples from earlier classes, typically leading to superior CL performance over the other methods at the cost of increased memory footprint.

In this work, we opt to use a rehearsal-based method due to its primarily superior performance in CL settings and the avoidance of dynamic expansion of the model architecture during deployment, allowing us to apply system optimizations on the static computation graph of the model (see last paragraph of Section 4 for details).

Given a single trajectory of samples from a stream of classes $\mathcal{T}$, minimizing the CL loss of a DNN that is trained end-to-end is more challenging than conventional DNN training [37]. This is because various complex challenges need to be solved together: (1) the forgetting problem incurred when learning a stream of different classes, (2) the issue with the lack of labeled samples, and (3) training DNNs is extremely sample-inefficient: the minimization problem requires multiple training epochs to converge to a reasonable solution. Specifically, many CL methods [48, 71] are proposed to alleviate the forgetting problem. However, they require a large amount of labeled data (a few thousand) and many training epochs. Another learning approach, called meta-learning, is proposed to make DNN more sample-efficient [15, 29, 53, 99], requiring only a few samples to adapt/learn new data distributions from a correlated data stream [1, 68]. However, existing meta-learning methods often neglect the forgetting problem of the already learned classes as it primarily aims at fast adaptation towards new tasks only [9, 19, 22, 24, 30, 79, 86, 93].

2.2 Meta Continual Learning

To overcome the challenges mentioned thus far, researchers proposed a novel approach, Meta CL, that utilizes meta-learning in CL to enable data-efficient and fast adaptation to new classes and also attempts to alleviate forgetting of already learned classes through novel ways of regularization and/or modification of the model architecture [4, 37, 55]. First, to enable fast adaptation with only a few samples, Meta CL methods are based on the training procedure of meta-learning. The meta-learning uses an outer loop and an inner loop where the outer loop takes steps to improve the learning ability of the inner loop that optimizes the DNN model with a few samples. This phase is called *meta-training*, which is typically performed on an offline server. The meta-training phase aims to find a better weight initialization of DNNs for fast adaptation with a few samples. After the meta-training is finished, the learned DNNs are tested given a few examples of new classes, referred to as the *meta-testing* phase, that could run on embedded systems. Secondly, to prevent the forgetting problem, Meta CL methods separate the network architecture into the feature extractor and the classifier. During the meta-training phase, Meta CL adopts the concept of fast and slow learning on an architecture level. The feature extractor is updated in the outer loop (slow weights) using random samples from learned classes to prevent forgetting. The classifier is updated in the inner loop (fast weights) to learn new classes swiftly. This approach has proven useful in alleviating CF [4, 37, 55].

Although prior works in Meta CL enable CL with limited data samples, they have certain limitations. For example, Online-aware Meta-Learning (OML) [37] and A Neuromodulated Meta-Learning (ANML) [4] can retain high CL performance on the Omniglot dataset [52] over many classes. Also, Attentive Independent Mechanisms (AIM) module [55] captures independent concepts to learn new knowledge. In fact, AIM and its combinations, ANML+AIM and OML+AIM, have achieved SOTA results. However, as prior Meta CL only relies on inner-loop optimization in the meta-testing phase, it does not utilize the concept of learning fast and slow weights during deployment. Further, these methods fail to generalize (see Figure 1a; low accuracy on CIFAR-100 [47]) and have extremely high memory requirements (see Figure 1b), which limits their applicability to low-end devices. Hence, we aim to design an efficient Meta CL system that obtains high accuracy and less forgetting while making the practical deployment on embedded devices a reality.

2.3 Efficient Deep Learning Systems

Scarce memory and compute resources are major bottlenecks in deploying DNNs on constrained embedded and IoT devices. In this context, researchers have largely focused on optimizing *the inference stage* (i.e., forward pass) by proposing lightweight DNN architectures [20, 56, 57, 64, 83], pruning [25, 61], quantization [35, 46, 80], leveraging heterogeneous processors [38, 59, 60], and offloading computation [100].

In addition, many works focus on reducing the overall system resources required for *DNN training* [6, 11, 18, 21, 31–33, 36, 43, 51, 70, 73, 78, 87, 92, 96, 102]. For example, researchers control the layerwise growth of the model structure to enable efficient DNN training on mobile phones [104]. Other methods optimize sparse activations and redundant weights to avoid unnecessary storage of activations and weight updates during DNN training [5, 28, 58]. In particular, for memory-efficient training, researchers proposed efficient meta-learning approaches by tackling memory issues during meta-training [92] and meta-testing [78]. However, dynamically changing the updated parameters as in [78] is not suitable to be used for MCUs because Flash memory space where the model weights are stored is read-only during runtime, and SRAM is even more limited than Flash in terms of memory capacity. Thus, it is difficult to incorporate the dynamic parameter update on MCUs. Also, prior work [45] examines various lossless compression techniques (e.g., Huffman coding), which show at most a 3.3× compression ratio on activations. Lossy compression [10, 62] based on scalar quantization shows up to 12× memory savings without accuracy degradation. A promising method that can achieve even higher compression ratios (e.g., 128×) is Vector/Product Quantization (PQ) [41, 88, 89]. However, as it requires storing a separate codebook containing representative vectors, a brute-force utilization of PQ may not achieve actual memory savings. In this work, we demonstrate that PQ can be a key component towards efficient continuous learning and show how the on-device CL pipeline should be designed to accommodate it (see Section 3.2.2 and Figure 3 for details).

In contrast to previous works, LifeLearner realizes efficient continual learning that was previously considered impractical for many embedded devices. By developing rehearsal-based Meta CL, effective algorithm/software co-design, and hardware-aware system implementation considering the unique characteristics of a wide range of embedded and IoT platforms (e.g., Jetson Nano, Pi 3B+,

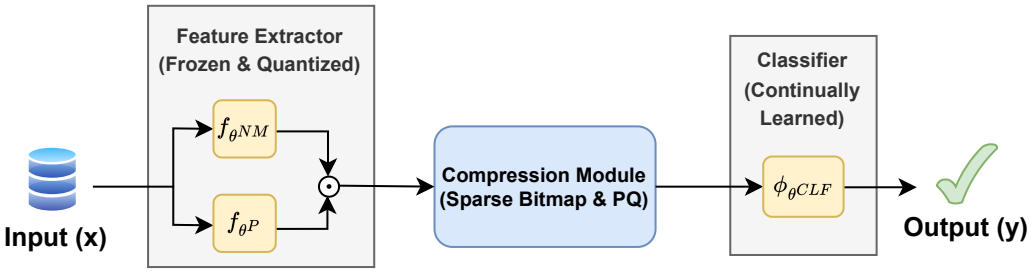


Figure 2: The system overview. LifeLearner consists of the frozen/quantized feature extractor, the continually learned classifier, and the compression module based on sparse bitmap and PQ. The compression module takes the feature extractor's outputs (activations) as inputs and compresses them to be saved as latent replay samples.

and STM32H747), LifeLearner yields both high accuracy and low resource overheads.

3 LifeLearner

LifeLearner leverages the idea of Meta CL and rehearsal-based learning and minimizes the system overheads on embedded devices. LifeLearner consists of two phases. The first phase, i.e., meta-training, is performed on a server to obtain a good weight initialization by utilizing meta-learning in the CL setup with a few samples. The second phase is meta-testing: a meta-trained model is deployed on embedded devices and learns new classes continually without forgetting previously learned classes. Additionally, as shown in Figure 2, LifeLearner has two components to ensure superior performance and efficiency when it is deployed on resource-constrained devices: (1) co-utilization of Meta CL and rehearsal strategy together with a deployment-time inner- and outer-loop optimization to resolve the accuracy degradation issue, (2) a design scheme that co-optimizes LifeLearner's rehearsal strategy and compression pipeline (*Compression Module* in Figure 2) to minimize the memory footprint, compute cost, and energy consumption when running CL.

3.1 Co-utilization of Meta-Learning and Rehearsal Strategy

Current Meta CL methods rely on regularization in order to minimize radical changes to the already trained weights when learning new classes. As such, given a small set of training data from a stream of classes, all samples are discarded once they have been used. However, recent results from the CL literature [13] indicate that the alternative approach of rehearsal-based methods often outperforms regularization-based CL. Driven by this observation, we design our Meta CL method, called *rehearsal-based Meta CL*, which introduces a rehearsal strategy into the Meta CL to improve CL performance. Concretely, we introduce a Replay Buffer that stores informative samples from already learned classes; these serve as additional training samples when learning new classes, form a mechanism for refreshing the weights of the model, and avoid catastrophic forgetting.

In addition, existing Meta CL systems are limited by their sole use of inner-loop optimization during meta-testing. Instead, we construct a variant of the learning fast and slow weights approach: we utilize the samples of new classes during inner-loop updates to enable rapid adaptation to new classes, followed by outer-loop iterations with the rehearsal samples of the previously learned classes to alleviate catastrophic forgetting.

System Overhead. Despite the learning benefits of our rehearsal-based Meta CL method (see Section 5.2 for details), it comes at a system cost. With respect to memory, the Replay Buffer has to store a number of representative samples for each of the already encountered classes, so that they can be fetched during meta-testing. With respect to computation, the samples have to be processed by the DNN with both forward and backward passes to perform CL. Unless alleviated, these overheads can lead to a sharp increase in storage and computational requirements, hindering its deployment on mobile and embedded devices, where continual learning is most needed. In the next section, we present LifeLearner's co-design approach for alleviating these system costs.

3.2 CL-tailored Algorithm/Software Co-Design

To alleviate the system costs of rehearsal-based Meta CL and enable its deployment on resource-constrained devices, we present an algorithm-software co-design method, optimized for Continual Learning. At the algorithmic level, we design a *rehearsal strategy* that minimizes the computational overhead while maximizing the compressibility of the rehearsal samples. At the software level, we design a two-stage *Compression Module* that enables the efficient compression, storage and decompression of rehearsal samples, while inducing minimal on-device resource usage.

3.2.1 Rehearsal Strategy. Key design decision in rehearsal-based methods constitutes the form of the rehearsal samples. A standard approach followed by many CL methods [8, 63, 81] is *native rehearsal (i.e., raw data replay)*, which stores and replays the input data in their raw format, e.g., images are stored for computer vision tasks and MFCC features for audio tasks. Under this scheme, a random subset of the given classes is stored as rehearsal samples, which are later replayed to mitigate the forgetting issue. The drawbacks of this approach are the significant computational overhead, as the samples have to be processed from the full model, and the compression variability as compressibility varies substantially in a per-sample manner.

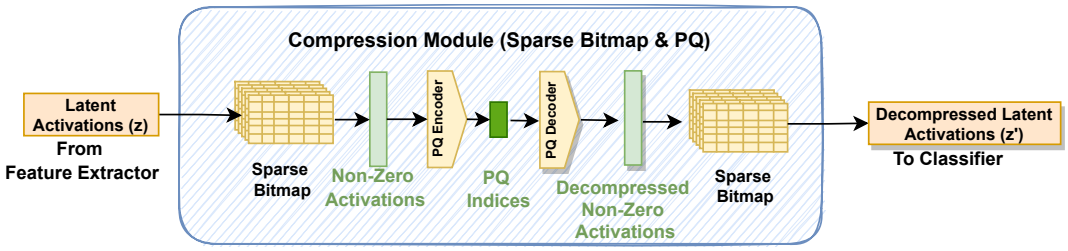


Figure 3: The overview of our compression module. It consists of (1) a sparse bitmap to filter out zero from activations or to reconstruct decompressed activations from non-zero activations, (2) a PQ encoder that further compresses non-zero activations into PQ indices, and (3) a PQ decoder that decompresses PQ indices back into decompressed non-zero activations.

To counteract these drawbacks, we introduce *latent replay* into our rehearsal strategy. Under this scheme, instead of holding copies of raw inputs, we store their latent representations, i.e., intermediate activations at the output of a selected layer of the target DNN. In LifeLearner, we employ two techniques in order to enable the utilization of latent replay: *i)* select the last layer of the model's feature extractor as the rehearsal point; and *ii)* we freeze the feature extractor upon deployment and perform CL only on the classifier. With the feature extractor frozen, we render latent replay functionally equivalent to raw data replay. On the computational front, the forward pass of the feature extractor can be omitted when replaying latent representations and the backward propagation is performed until the last layer, inducing significant computational gains.

On the memory front, we make the following observation. In DNN training, the activations for each layer are saved during the forward propagation so that those activations are utilized for computing the gradients during the backward propagation. As in [87], storing activations requires a large memory footprint depending on the batch size used for training. However, commonly used ReLU non-linearity in many DNN models results in sparse activations in the successive layers. Also, we observe that more than 90% of the activation values of the latent layer are zero due to the usage of ReLU from our analysis of the network architecture on all three datasets. By strategically selecting the rehearsal layer in the DNN and treating ReLU activations as the rehearsal samples, LifeLearner's rehearsal strategy facilitates their compression and subsequent efficient storage on-device.

3.2.2 Compression Module for Latent Replays We now introduce the *Compression Module* that is responsible for *i)* compressing rehearsal samples (i.e., latent activations in our work) when new classes are encountered and storing them in the Replay Buffer, and *ii)* fetching and decompressing them to perform CL at runtime. This component comprises two stages: sparse bitmap compression and product quantization (PQ).

Sparse Bitmap Compression. To leverage the sparsity of our latent replays for efficient storage, we employ sparse bitmap compression [28]. This scheme enables the Compression Module in LifeLearner to filter out the majority of zero values (typically 90% or more) in latent activations and save the remaining non-zero values to increase the compression rate for saving latent activations.

Figure 3 depicts the compression and decompression processes. For compression, when latent activations are given to our system, a bitmap with the same dimensions as the latent activations sets a bit to 1 for non-zero values' indices and 0 for the remainders. Then, non-zero values and the sparse bitmap are stored in 32-bit floats and the bitmap format, respectively. For decompression, we traverse all elements of the bitmap and a vector containing the stored non-zero values, reconstructing in this process the latent activations by using either the saved non-zero value or zero if a bitmap element is 1 or 0, respectively. The compression and decompression processes are linear in runtime: $O(n)$, where n is the total number of elements of latent activations. With respect to memory, the footprint is reduced from $(4n)$ when a dense format is used for storing latent activations to $(4 \times \text{number of non-zero values} + \frac{1}{8}n)$ with the bitmap.

Product Quantization. To further minimize the resource overhead of rehearsal samples, we introduce a second stage to our compressor (Figure 3) utilizing PQ [41]. The output of the sparse bitmap compressor contains a vector of non-zero values. With PQ being a vector compression method that can compress a given vector $\mathbf{v} \in \mathbb{R}^d$ into s number of PQ indices using a PQ codebook with s columns, it is suitable to further reduce the size of the encoded rehearsal samples. Each column of the PQ codebook contains a set of representative vectors that well approximate s sub-vectors of $\mathbf{v}$ when $\mathbf{v}$ is partitioned into s sub-vectors.

For compression, the PQ encoder applies PQ to the non-zero activations $\mathbf{v} \in \mathbb{R}^d$ that are already filtered out by the first-stage sparse bitmap compression. We use 1 byte to store each PQ index and set $d/s = \{128, 32, 8\}$ (length of each sub-vector). Then, each sub-vector of length d/s containing 32-bit floats is encoded to a 1-byte PQ index via our PQ encoder for more analysis regarding hyper-parameters). LifeLearner learns the PQ codebook offline using the latent activations during the meta-training phase, which is then stored on-device. For decompression, the PQ decoder reconstructs the non-zero activations $\mathbf{v}'$ using the stored PQ indices and the PQ codebook.

Finally, as in Algorithm 2 (see Lines 7, 9, and 10), our compression module is seamlessly incorporated in the inner- and outer-loop optimization of LifeLearner, enabling on-the-fly compression of the latent activations during deployment.

Algorithm 1: Meta-Training Procedure of LifeLearner

Require: N sequential classes $\mathcal{T}$; learning rates (LR) α, β; inner-loop iterations k; modules f_θ, ϕ_θ; given samples S

// Outer-loop starts here
1 **for** $t = 1, ..., N$ **do**
2 $S_{traj} \sim \mathcal{T}_t$, $S_{rand} \sim \mathcal{T}$
  // Inner-loop starts here
3 **for** $i = 1, ..., k$ **do**
4 Update fast weights using S_{traj} ▷ LR: α
  /* OML(+AIM): $\phi_{\theta^{PLN}}(f_{\theta^W})$,
  ANML(+AIM): $f_{\theta^P}, \phi_{\theta^{CLF}}(f_{\theta^W})$, LifeLearner: $\phi_{\theta^{CLF}}$ */
5 Update slow weights using $\{S_{traj}, S_{rand}\}$ ▷ LR: β
  /* OML(+AIM): $f_{\theta^{RLN}}$, ANML(+AIM): $f_{\theta^{NM}}, f_{\theta^P}, \phi_{\theta^{CLF}}$,
  LifeLearner: $f_{\theta^{NM}}, f_{\theta^P}, \phi_{\theta^{CLF}}$ */

Algorithm 2: Meta-Testing Procedure of LifeLearner

Require: N sequential unseen classes $\mathcal{T}$; learning rates (LR) α, β; inner-loop iterations k; modules f_θ, ϕ_θ, $BitPQ_{compress,decompress}$; samples S

1 $S_{train} = \{\}$, $S_{rehearsal} = \{\}$
// Outer-loop starts here
2 **for** $t = 1, ..., N$ **do**
3 $S_{traj} \sim \mathcal{T}_t$
4 $S_{train} = \{S_{train}, S_{traj}\}$
  // Inner-loop starts here
5 **for** $i = 1, ..., k$ **do**
6 Update fast weights using S_{traj} ▷ LR: α
  /* OML(+AIM): $\phi_{\theta^{PLN}}(f_{\theta^W})$,
  ANML(+AIM): $f_{\theta^P}, \phi_{\theta^{CLF}}(f_{\theta^W})$, LifeLearner: $\phi_{\theta^{CLF}}$ */
  // Get latent activations from compressed rehearsal samples
7 $S_{latent} = BitPQ_{decompress}(S_{rehearsal})$
8 Update slow weights using $\{S_{traj}, S_{latent}\}$ ▷ LR: β
  /* OML(+AIM): $f_{\theta^{RLN}}$, ANML(+AIM): $f_{\theta^{NM}}, f_{\theta^P}, \phi_{\theta^{CLF}}$,
  LifeLearner: $\phi_{\theta^{CLF}}$ */
  // Get latent activations
9 $S_{latent} = f_{\theta^{NM}}(S_{traj}) \odot f_{\theta^P}(S_{traj})$
  // Store compressed activations for rehearsal
10 $S_{rehearsal} = \{S_{rehearsal}, BitPQ_{compress}(S_{latent})\}$
11 $S_{test} = \mathcal{T} - S_{train}$ // Held-out test set
12 Evaluate on S_{train}, S_{test} // Eval on training/test set

3.3 Putting It All Together

Having described the main components of LifeLearner we now present the complete meta-training and meta-testing procedures that take place offline and online, respectively.

Meta-Training Procedure. Algorithm 1 shows the procedure of meta-training of Rehearsal-based Meta CL, LifeLearner. Firstly, the meta-training process of rehearsal-based Meta CL is the same as that of Meta CL [4]. In detail, it is comprised of an inner loop inside an outer loop of optimization. In the inner loop, the classifier part is updated (fast weights, e.g., θ^{PLN} for OML and $\theta^{P,CLF}$ for ANML, $\theta^{PLN,W}$ for OML+AIM, and $\theta^{P,CLF,W}$ for ANML+AIM) (Lines 4-5). The number of weight update iterations is determined by the number of samples k (e.g., 10-30) of a given sample set, S_{traj}, of a new class, $\mathcal{T}_t$. After the k sequential updates, the meta-loss in the outer loop (Line 6) is computed using all the given samples on the new class (S_{traj}) and randomly sampled samples from all the meta-training classes (S_{rand}). All the weights of DNN are updated through outer-loop gradient updates using an Adam optimizer [42]. The learning rates, α for the inner loop and β for the outer loop, are used as hyper-parameters.

Meta-Testing Procedure. After executing the meta-training phase on a server, our system is deployed on resource-constrained devices and evaluated on its ability to learn unseen classes in the meta-testing phase. Algorithm 2 shows the meta-testing phase of the rehearsal-based Meta CL. In prior Meta CL, the meta-testing procedure contains only inner-loop optimization without outer-loop optimization, i.e., only fast weights except for slow weights are fine-tuned. In contrast, LifeLearner leverages the full potential of meta-learning by using both inner-and outer-loop optimization in the meta-testing phase. Specifically, our proposed meta-testing procedure starts with the inner-loop weight updates to learn new classes swiftly using a few samples (Lines 5-6), followed by the outer-loop weight updates to retain the knowledge on the previously learned classes using the replayed samples plus the new samples (Line 8). Note that although the outer-loop iteration could run multiple epochs, the performance converges after one or two epochs (refer to Section 5.4 for more analysis). Also, LifeLearner integrates the compression module that compresses (Lines 9-10) and decompresses (Line 7) the latent activations during outer-loop optimization, as described in Section 3.2.

Our Contribution. Our method conceptually leverages existing concepts. We solve the challenge of incorporating these concepts in a coordinated, efficient end-to-end system (as discussed in Section 2.3). We achieve higher accuracy than baselines while reducing the memory footprint drastically. Our key contributions are (1) co-designing the algorithmic innovation (rehearsal strategy) with an intelligent combination of lossless (bitmap) and lossy (PQ) compression to significantly reduce the resource requirements of CL and latent replay samples (Section 3), (2) successfully deploying LifeLearner end-to-end on two embedded devices and MCU on which many prior works fail to run (Section 4).

4 Hardware-Aware System Implementation

We develop the first phase, meta-training, of Meta CL methods on a Linux server to initialize the neural weights that can enable fast adaptation during deployment scenarios. After that, for the second phase, meta-testing, (i.e., actual deployment scenarios), we implemented our hardware-aware system by considering the hardware capacity and unique runtime characteristics of our target devices: (1) embedded and mobile systems such as Jetson Nano and Raspberry Pi 3B+, and (2) a microcontroller unit such as STM32H747. To further optimize the system efficiency, we adopt hardware-friendly optimization techniques in our implementation[1]

[1] https://github.com/theyoungkwon/LifeLearner

Embedded Device. Jetson Nano has a quad-core ARM Cortex-A57 processor, and 4 GB of RAM, while Pi 3B+ contains a quad-core ARM Cortex-A53 processor with 1 GB of RAM. Note that the free memory space of Jetson Nano and Pi 3B+ during idle time is roughly 1.7 GB and 600 MB, respectively, due to the memory footprints pre-occupied by background, concurrent applications, and an operating system. As software platforms, we employ Faiss (PQ Framework) [40] and PyTorch 1.8 (Deep Learning Framework) [72] to develop and evaluate the meta-training and meta-testing phases on embedded systems.

Microcontroller Unit (MCU). To demonstrate the feasibility of the broader deployment of CL systems at the extreme edge, we further optimized and developed LifeLearner on MCUs. We implemented the online component of LifeLearner using C++ on an STM32H747 device equipped with ARM Cortex M4 and M7 cores with 1MB SRAM and 2 MB eFlash in total. However, we only utilize one core (ARM Cortex M7), as most MCUs have one CPU core. Also, we restrict the usage space of SRAM and eFlash to 512 KB and 1 MB, respectively, to enforce stricter resource constraints (an order of magnitude smaller memory space than other embedded devices with larger than 1 GB RAM).

To deploy LifeLearner on MCUs effectively and efficiently, we addressed many technical challenges and considered hardware characteristics. First of all, the memory requirements of the MetaCL methods developed on embedded devices, including LifeLearner, far exceed the hardware capacity of a "high-end" MCU such as STM32H747 (refer to Section 5.2). Hence, we first searched for a smaller yet accurate architecture for MCUs by experimenting with various width modifiers [56, 57, 83] (see Section 5.5 for details).

We then implemented our Compression Module (sparse bitmap compression and PQ) to reduce memory usage of latent replay samples on SRAM. In particular, we consider hardware characteristics and constraints: (1) the write operation on the storage (Flash) of MCUs is costly [90], and (2) Flash is read-only during runtime [3, 50]. Hence, in our MCU implementation of LifeLearner, to minimize the memory footprint and energy consumption required for latent replay, we first compress latent replay samples using our Compression Module and then store them on SRAM, which has more limited memory but is faster and cheaper to perform read/write operations on than Flash. Note that our learned PQ codebook, used to encode and decode the latent replay samples after sparse bitmap compression, is stored on Flash to leave more space for scarce resources of SRAM. Also, PQ codebooks are static once deployed; they can be stored on the read-only memory of Flash.

In addition, we rely on the TFLM framework [12] to perform inference of the feature extractor on MCUs. However, TFLM does not support training (i.e., backpropagation). We developed our Backpropagation Engine based on C/C++ using Eigen [23] as a data structure and matrix multiplication library. Based on our Backpropagation Engine, we construct the classifier part on the fly whose weights are allocated on SRAM and can be continually learned during deployment whenever more data for new classes become available. Our lightweight Backpropagation Engine enables the implementation of the first CL system on MCUs.

Lastly, the binary size of our Compression Module and Backpropagation Engine, excluding C++ Standard Library (STL) on an MCU, is only 80 KB, introducing minimal overhead on storage.

Hardware-friendly Optimization. We further optimize LifeLearner's CL operations on-device. By freezing the model's feature extractor during deployment, LifeLearner significantly reduces the computational cost for the already learned classes during replay by omitting the forward and backward passes. In addition, we utilize the hardware-friendly 8-bit integer arithmetic [91] by reducing the precision of weights/activations of the feature extractor from 32-bit floats to 8-bit integers, increasing the computation throughput and minimizing latency and energy. The scalar quantization scheme [35, 46] is used to minimize the information loss in quantization. Then, we utilize the QNNPACK [17] backend engine and TFLM to execute the quantized model on two embedded devices and MCUs, respectively.

5 Evaluation

5.1 Experimental Setup

We briefly describe our experimental setup in this subsection.

5.1.1 Metrics As in [4], we use testing accuracy on unseen samples of all the new classes learned continually as a key performance metric, representing the generalization ability of CL systems. In addition, we measure the memory footprint (model parameters, optimizers, activations, and rehearsal samples), end-to-end training latency and energy consumption to continually learn all the given classes for a deployed DNN on embedded devices.

5.1.2 Datasets We employ three datasets of two different data modalities in our evaluation.

CIFAR-100 [47]: Following [55], we employ CIFAR-100 in our evaluation as it is widely used dataset. CIFAR-100 consists of 60,000 images of 100 classes. Each class has 500 train images and 100 test images. 70 classes are used for meta-training and the remaining 30 for meta-testing. During both meta-training and meta-testing, up to only 30 training images are sampled for training in each class, which holds for both MiniImageNet and GSCv2 datasets. Then, during meta-testing, a total of 900 samples are given to perform CL.

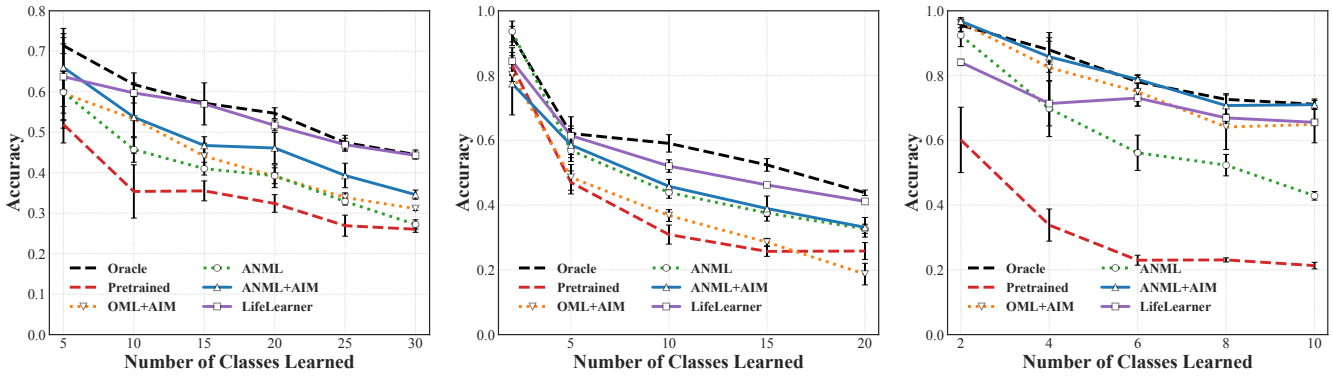
MiniImageNet [95]: Following [55], we employ MiniImageNet containing 64 classes for meta-training and 20 classes for meta-testing. Each class has 540 images for training and 60 images for testing. During meta-testing, a total of 600 samples are given.

GSCv2 [97]: To generalize our results to another data modality, we include Google Speech Command V2 (GSCv2) as it is a widely used audio dataset. GSCv2 consists of a total of 35 classes of different keywords. We use 25 classes for meta-training and 10 classes for meta-testing. Each class has 2,424 and 314 input data for training and testing, respectively. During meta-testing, 300 samples in total are given for CL.

5.1.3 Baselines We compare our system, *LifeLearner*, with five baseline systems as follows.

Oracle: The CL performance of Oracle represents the upper bound performance of the experiments. It is because Oracle has access to all the classes at once in an i.i.d. fashion and performs DNN training for many epochs until the performance converges.

Pretrained: This baseline initializes the model weights based on conventional DNN training without the meta-learning procedure.



(a) CIFAR-100 (b) MiniImageNet (c) GSCv2

Figure 4: The accuracy of the CL systems on the three datasets of two different modalities. Reported results are averaged over three trials, and standard-deviation intervals are depicted.

Then, it finetunes the weights using given samples in the meta-test phase, similar to prior Meta CL methods.

OML+AIM [55]: This is a Meta CL method based on OML with an Attentive Independent Mechanisms (AIM) module, capturing independent concepts to learn new knowledge.

ANML [4]: It is the representative Meta CL method. As this method is often reported to outperform OML [37], we only employ ANML in our evaluation. Also, note that the proposed components of LifeLearner build on top of ANML.

ANML+AIM [55]: ANML+AIM is a Meta CL method based on ANML with an AIM module. This baseline serves as the SOTA Meta CL method as it often outperforms other Meta CL methods including OML+AIM.

5.1.4 Model Architecture LifeLearner employs the network architecture used in the prior CL works for a fair comparison [4, 55]. As in Figure 2, it consists of the feature extractor and the final classifier. For ANML-based model architectures, the feature extractor consists of a neuromodulatory network, $f_{\theta^{NM}}$, and a prediction network, $f_{\theta^{P}}$, followed by the classifier part, $f_{\theta^{CLF}}$. The neuromodulatory and prediction networks are 3-layer convolutional networks with 112 and 256 channels, respectively. The classifier has a single fully-connected layer. In this case, LifeLearner utilizes the last layer of the feature extractor as the latent replay layer, following the natural structure of the ANML architecture.[2] The SOTA method, ANML+AIM, adds AIM layers $f_{\theta^{W}}$ between the feature extractor and the classifier, which alleviates forgetting and helps learn new classes. In addition, for OML and OML+AIM, the feature extractor has a 6-layer convolutional network with 112 channels, followed by the classifier of two fully-connected layers with an AIM module between the feature extractor and the classifier. Note that the model architectures deployed on embedded devices (i.e., Jetson Nano and Pi 3B+) and an MCU (i.e., STM32H747) are different due to the strict resource constraint on the MCU. Thus, a smaller version of the model architecture described above is adopted for the MCU deployment (see Section 5.5 for details).

5.1.5 Training Details We followed the meta-training procedure used in prior Meta CL works [4, 37, 55]. For instance, we used a batch size of 1 and 64 for the inner- and outer-loop updates over 20,000 steps, respectively. We experimented with different learning rates for the inner loop and outer loop to obtain the meta-trained DNN that provides the best accuracy on a validation set. As a result, for CIFAR-100 and GSCv2 datasets, the inner-loop learning rate (α) is set to 0.001, and the outer-loop learning rate (β) is also set to 0.001. For the MiniImageNet dataset, the optimal settings are $\alpha = 0.001$ and $\beta = 0.0005$. During the meta-testing phase, ten different learning rates are tried for all the methods, and the best-performing results are reported. Besides, to obtain the accuracy results of systems that perform replays, we experimented with batch sizes of 8 and 16 and observed little difference in CL performance. Thus, we employ a batch size of 8, as a smaller batch size reduces the memory footprint.

5.2 Experimental Results

Accuracy. We start by evaluating the CL performance (testing accuracy) of LifeLearner compared to the baselines on the employed datasets. Figure 4 presents the accuracy results of the meta-testing phase. Pretrained serves as the lower bound. The low accuracy (24.4% on average for three datasets) of Pretrained demonstrates that the conventional transfer learning approach cannot address the challenging scenarios of learning new classes with only a few samples. ANML improves upon Pretrained, however, the improvement is marginal (i.e., average 9.9% accuracy gain compared to Pretrained but 18.9% accuracy drop on average compared to Oracle which shows the upper bound accuracy). Note that it is very challenging to achieve high testing accuracy even for Oracle as the number of available samples is very limited during meta-testing: all evaluated systems are given only 30 samples per class, accounting for only 2.57%, 1.74%, and 0.5% of all training samples during meta-training of CIFAR-100, MiniImageNet, and GSCv2, respectively.

[2]When targeting a different model architecture, the latent replay layer selection is a configurable design decision. We leave this investigation as future work.

Table 1: The required memory footprint and the compression ratio for the baselines and our system to perform CL during the meta-testing phase on the three datasets.

Dataset	Metrics	Pretrained	ANML	OML+AIM	ANML+AIM	Oracle	LifeLearner
CIFAR-100	Memory	39.69MB	39.69MB	834.1MB	1,093MB	39.93MB	**15.45MB**
	Ratio	27.5×	27.5×	1.3×	1.0×	27.4×	**70.8×**
Mini-ImageNet	Memory	474.5MB	474.5MB	1,051MB	1,562MB	475.0MB	**136.7MB**
	Ratio	3.3×	3.3×	1.5×	1.0×	3.3×	**11.4×**
GSCv2	Memory	10.16MB	10.16MB	135.2MB	608.2MB	10.20MB	**3.40MB**
	Ratio	59.9×	59.9×	4.5×	1.0×	59.6×	**178.7×**

LifeLearner achieves near-optimal CL performance, falling short by only 2.8% accuracy compared to Oracle. Also, LifeLearner outperforms all the Meta CL methods with substantial accuracy gains of 4.1-16.1% on average for the three datasets. Specifically, LifeLearner shows almost no loss of accuracy, i.e., 0.2% for CIFAR-100 and 2.7% for MiniImageNet compared to Oracle. In contrast, ANML+AIM (i.e., the previous SOTA Meta CL method) shows notable accuracy drops (9.9% for CIFAR-100 and 10.7% for MiniImageNet). In the case of GSCv2, LifeLearner reveals a slight accuracy decline of 5.6% compared to Oracle, while ANML+AIM shows a minor 0.2% drop in accuracy relative to Oracle.

Although LifeLearner shows a slightly lower accuracy for GSCv2 than ANML+AIM, it still outperforms ANML+AIM by 4.1% on average over all datasets. In addition, LifeLearner is essentially designed for edge devices to require drastically lower system resources (memory, latency, and energy) than the previous SOTA. As explained in the following, the excessive resource overhead of ANML+AIM makes it unsuitable to operate on resource-constrained devices.

Peak Memory Footprint. We investigate the peak memory footprint required to perform CL. Precisely, we measure the memory space required to perform backpropagation and to store rehearsal samples. The memory requirement to perform backpropagation consists of three components: (1) model memory that stores model parameters, (2) optimizer memory that stores gradients and momentum vectors, and (3) activation memory that is comprised of the intermediate activations (stored for reuse during backpropagation). Then, the memory requirement for rehearsal samples is included.

Table 1 shows the peak memory footprint for various baselines and our system. First, the AIM variants (OML+AIM and ANML+AIM) require an enormous memory footprint of 135.2-1,051 MB and 608.2-1,562 MB, respectively, as their AIM module has many parameters. This required memory easily exceeds the RAM size of embedded devices such as Pi 3B+ (i.e., 1 GB) and barely fits on Jetson Nano. Conversely, baseline systems such as Pretrained, ANML, and Oracle show modest memory requirements, which are around 10.16-10.20 MB for GSCv2, 39.7-39.9 MB for CIFAR-100, and 474.5-475.0 MB for MiniImageNet. However, as shown earlier, Pretrained and ANML methods are not highly accurate, and Oracle does not support CL. In contrast, LifeLearner shows the impressive results that it only requires 15.45 MB for CIFAR-100, 136.7 MB for MiniImageNet, and 3.40 MB for GSCv2, demonstrating a very high compression rate of 70.8×, 11.4×, and 178.7× compared to ANML+AIM, respectively. Compared to Oracle, LifeLearner shows a tight range of the compression (2.5-3.5×), indicating that we can estimate the compression gain within this range agnostic to the dataset.

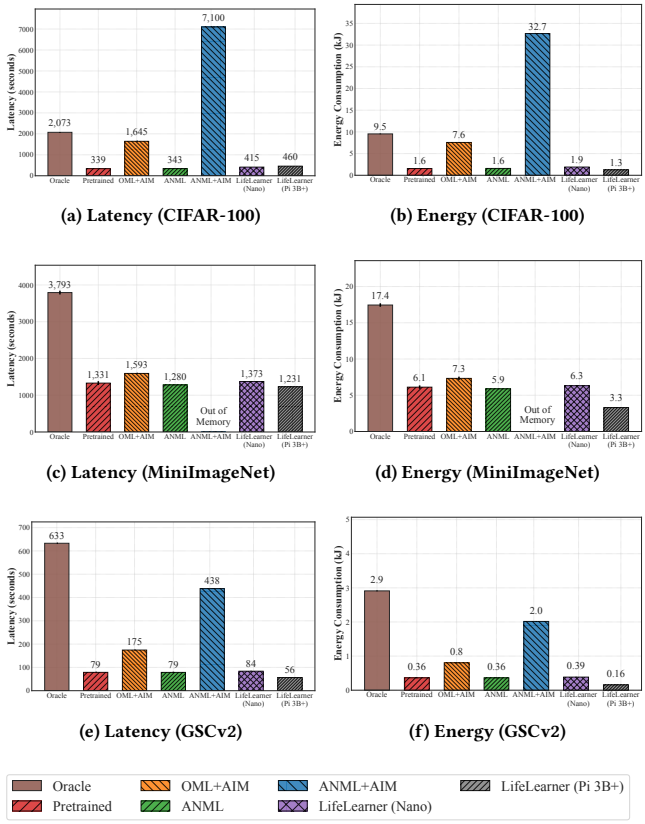


Figure 5: The end-to-end latency and energy consumption of the baselines and LifeLearner to perform CL over all the given classes. All results are averaged over three runs with standard deviations.

End-to-end Latency & Energy Consumption. We now examine the run-time system efficiency, i.e., end-to-end latency and energy consumption for the entire CL process, of our system and the baselines when deployed on the two embedded devices - Jetson Nano and Pi 3B+ as shown in Figure 5. To obtain the end-to-end latency, we include: (1) the time to load a pretrained model, (2) the time to train the model continually over all the given classes one by one, and (3) the time to compress and decompress the latent representations using our compression method (i.e., sparse bitmap compression and PQ).

We first measure the end-to-end latency of our system and the baselines on Jetson Nano CPU to perform CL over all the given classes with 30 samples per class. As shown in Figures 5a, 5c, and 5e, LifeLearner enables a fast end-to-end latency (415 seconds for CIFAR-100, 1,373 seconds for MiniImageNet, and 84 seconds for GSCv2), which is 80.8-94.2% reduction of latency compared to ANML+AIM (e.g., 7,100 seconds for CIFAR-100 and 438 seconds for GSCv2). Note that ANML+AIM often crashes from running out of memory on Jetson Nano due to its excessive memory requirements (as shown in Figures 5c and 5d). Furthermore, compared to ANML which shares the same network architecture, LifeLearner introduces

negligible overheads in terms of the overall latency (343s vs. 415s for CIFAR-100, 1,280s vs. 1,373s for MiniImageNet, and 79s vs. 84s for GSCv2). It is because although there exist some overheads on LifeLearner to perform the compression techniques like the sparse bitmap compression and PQ, the speed gains derived from using quantized neural weights and activations offset the overheads of compression techniques (refer to Section 5.3 for details). After having demonstrated the efficiency of LifeLearner on the Jetson Nano, we deployed our system on an even more resource-constrained device, Pi 3B+ (600-700 MB available memory). The end-to-end latency on Pi 3B+ largely stays similar to that on Jetson Nano as shown in Figure 5.

To measure the energy consumption, we first use Tegrastats on Jetson Nano to measure the power consumption. Then, we calculate the energy consumption by multiplying power consumption and the elapsed time for each end-to-end CL trial. Similar to the latency results, Figures 5b, 5d, and 5f show that LifeLearner remarkably reduces the energy consumption by 80.9-94.2% (1.9kJ vs. 32.7kJ for CIFAR-100 and 0.4kJ vs. 2.0kJ for GSCv2) compared to ANML+AIM. Moreover, compared to ANML, LifeLearner shows small overheads of the additional energy consumption (1.6kJ vs. 1.9kJ for CIFAR-100, 5.9kJ vs. 6.3kJ for MiniImageNet, and 0.36kJ vs. 0.39kJ for GSCv2). In the case of Pi 3B+, it consistently consumes less energy than Jetson Nano. It is because while the end-to-end latency of the two embedded devices is similar, the power consumption profile on Pi 3B+ is lower than that on Jetson Nano, making Pi 3B+ a more energy-efficient option. A YOTINO USB power meter is used to obtain the power consumption on Pi 3B+.

Summary. *Our result demonstrates that LifeLearner can effectively learn new classes in a continual manner based on only a few samples without experiencing catastrophic forgetting, i.e., it generalizes well to new samples of many classes unseen during the offline learning phase. Moreover, LifeLearner enables fast and energy-efficient CL on edge devices with significantly reduced memory footprint.*

5.3 Ablation Study

We perform an ablation study to investigate the role of each component of our system by incrementally adding our proposed components on top of the baseline system (ANML): (1) rehearsal strategy with inner-and outer-loop optimization (Latent), (2) sparse bitmap compression (Latent+Bit), (3) PQ (Latent+PQ), and (4) quantization (LifeLearner).

Effect of Rehearsal with Double-Loop Optimization. As shown in Table 2, we find that our proposed rehearsal strategy with double-loop optimization drastically improves the accuracy (compare ANML vs Latent). For example, Latent increases the accuracy of ANML by 10.6-28.4% across all the datasets. Yet, Latent causes resource overheads on memory footprint, latency, and energy consumption compared to ANML, as Latent is a baseline CL system without our Compression Module.

Effect of Compression and Hardware-aware Implementation. The results of various CL systems such as Latent+Bit, Latent+PQ, and Latent+Bit+PQ show that the proposed compression techniques for latent representations do not sacrifice the accuracy of the CL systems but reduce the overall memory footprint compared to Latent. Moreover, our Compression Module incurs small resource overheads in end-to-end latency and energy. Then, LifeLearner, which combines quantization of weights and activations accelerating the CL execution on hardware by exploiting efficient integer-based operations, shows excellent performance in all aspects: (1) outperforms ANML by a large margin (8.4-22.7%) with a minor accuracy drop compared to Latent (0.9-5.7%), (2) drastically reduces the memory footprint by 61.0-71.2% compared to ANML and by 71.2-73.3% compared to Latent, and (3) incurs minimal overheads of latency and energy over ANML (costs additional 56.6s and 0.3kJ on average, respectively) but still shows lower latency and energy than Latent (saves 47.9s and 0.2kJ on average, respectively).

Overall, the ablation study reveals that the co-utilization of the rehearsal strategy with double-loop optimization, Compression Module, and hardware-friendly implementation effectively makes LifeLearner more accurate and efficient.

Table 2: The comparison of LifeLearner and variants of rehearsal-based Meta CL methods for ablation study.

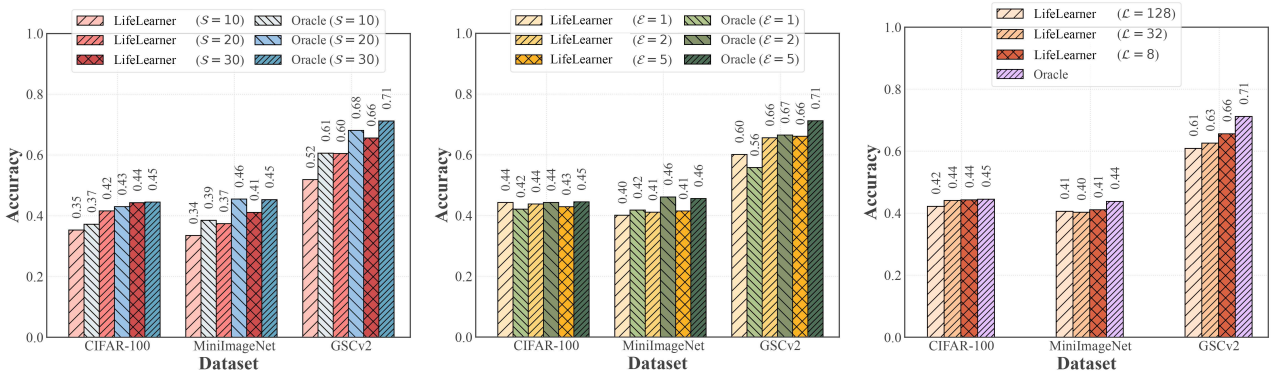
Dataset	System	Accuracy	Memory	Latency	Energy
CIFAR-100	ANML	0.272	39.7 MB	343.2s	1.58kJ
	Latent	0.452	53.9 MB	432.5s	1.99kJ
	Latent+Bit	0.452	41.2 MB	466.9s	2.15kJ
	Latent+PQ	0.448	41.8 MB	437.1s	2.01kJ
	Latent+Bit+PQ	0.446	40.4 MB	471.4s	2.17kJ
	LifeLearner	**0.443**	**15.5 MB**	**414.7s**	**1.91kJ**
Mini-ImageNet	ANML	0.327	474.5 MB	1,280s	5.89kJ
	Latent	0.433	512.5 MB	1,492s	6.86kJ
	Latent+Bit	0.433	477.7 MB	1,551s	7.14kJ
	Latent+PQ	0.430	483.0 MB	1,501s	6.90kJ
	Latent+Bit+PQ	0.423	476.4 MB	1,560s	7.18kJ
	LifeLearner	**0.411**	**136.7 MB**	**1,373s**	**6.32kJ**
GSCv2	ANML	0.429	10.2 MB	78.6s	0.36kJ
	Latent	0.713	12.0 MB	90.6s	0.42kJ
	Latent+Bit	0.713	10.4 MB	90.8s	0.42kJ
	Latent+PQ	0.708	11.0 MB	95.0s	0.44kJ
	Latent+Bit+PQ	0.707	10.3 MB	95.2s	0.44kJ
	LifeLearner	**0.656**	**3.40 MB**	**83.8s**	**0.39kJ**

5.4 Parameter Analysis

Next, we study the impact of the various hyper-parameters that could affect the performance of our system (see Figure 6).

The Number of Given Samples. We first examine the accuracy of LifeLearner according to the number of given samples per class (ranging from 10 to 30) as it would directly affect labeling effort of users (see Figure 6a). Apparently, the more samples are given for training, the higher the accuracy, which holds for both LifeLearner and Oracle. Even when only 10 samples per class are given to conduct training, the accuracy degradation of LifeLearner is relatively low (7-14%), indicating that LifeLearner can still perform reasonably well under extreme data scarcity. Also, the accuracy differences between LifeLearner and Oracle are small (e.g., 1-2% for CIFAR-100, 1-3% for MiniImageNet, and 5-9% for GSCv2), demonstrating that LifeLearner achieves the similar accuracy of Oracle. With 30 given samples, the accuracy difference is minimal: 2.8% on average (ranging from 1 to 5%).

The Number of Replay Epochs. We study to what extent the number of replay epochs affects the CL performance as more epochs



(a) The Number of Samples per Class ($\mathcal{S}$) **(b) The Number of Replay Epochs ($\mathcal{E}$)** **(c) The Sub-Vector Length ($\mathcal{L}$)**

Figure 6: The parameter analysis of LifeLearner for all the datasets according to the three parameters.

incur larger latency and energy consumption. Figure 6b shows that the accuracy of LifeLearner converges after the first or the second replay epoch. However, Oracle requires at least two to five epochs to reach the convergence accuracy, which consumes much more training time and energy than our system (see Figure 5). This result benefits us since replaying the rehearsal samples over one or two epochs is enough for LifeLearner to reach the converging accuracy, which helps decrease the system overheads.

PQ Codebook's Sub-vector Length. We investigate the accuracy of LifeLearner according to the sub-vector length of the PQ codebook (the number of values per index) ranging from 8 to 128 as it affects the compression ratio of rehearsal samples. For CIFAR-100 and MiniImageNet, there is little difference according to the sub-vector length. In contrast, for GSCv2, we observe that the shorter the length of the sub-vector (i.e., lower compression rate), the higher the accuracy. These results inform us to select the largest sub-vector length that does not degrade accuracy.

These results show that with only 10-30 samples per class, LifeLearner achieve similar CL performance to Oracle, exhibit rapid convergence with small replay epochs (at most two), and accomplish a high compression rate for rehearsal samples.

5.5 MCU Deployment

TinyANML Architecture. For the extremely resource constrained IoT devices like MCUs where on-chip memory of SRAM and Flash are typically a few hundred KB or 1 MB at most (an order of magnitude smaller than Jetson Nano and Pi 3B+ in terms of memory), the memory requirements of the MetaCL methods, including LifeLearner, are prohibitively large. Thus, we propose a small and accurate TinyANML architecture designed for MCUs with tiny memory by experimenting with various width modifiers [56, 57, 83]. We identified widths of 0.2, 0.05, and 0.4 for the ANML architecture of CIFAR-100, MiniImageNet, and GSCv2, respectively.

MCU Implementation and Results. Backbone represents an inference-only feature extractor based on TFLM. On top of that, our hardware-aware systems are added incrementally: (1) Backpropagation Engine (Tiny ANML) and (2) Compression Module (Tiny LifeLearner). Table 3 shows the MCU deployment results based on STM32H747 in terms of accuracy, SRAM, Flash, latency, and energy consumption to learn a class with ten samples when continually learning ten classes.

Table 3: MCU deployment of the Backbone, tiny ANML, and tiny LifeLearner on STM32H747.

Dataset	System	Accuracy	SRAM	Flash	Latency	Energy
CIFAR-100	Backbone	-	75kB	428kB	561ms	128mJ
	Tiny ANML	0.176	185kB	691kB	579ms	134mJ
	Tiny LifeLearner	0.393	236kB	825kB	832ms	195mJ
Mini-ImageNet	Backbone	-	119kB	329kB	926ms	221mJ
	Tiny ANML	0.112	224kB	591kB	944ms	218mJ
	Tiny LifeLearner	0.301	281kB	725kB	1204ms	282mJ
GSCv2	Backbone	-	81kB	475kB	956ms	218mJ
	Tiny ANML	0.209	181kB	738kB	968ms	223mJ
	Tiny LifeLearner	0.534	212kB	806kB	1160ms	271mJ

Backpropagation Engine. As shown with Tiny ANML compared to inference-only Backbone, our Backpropagation Engine enables on-device CL with extremely small latency/energy overheads (e.g., 579ms vs. 561ms and 134mJ vs. 128mJ for CIFAR-100) while requiring only an additional 100KB SRAM and 260KB Flash.

Co-design of Our Algorithm and Hardware-aware System Implementation. Tiny LifeLearner not only largely prevents accuracy degradation compared to its original LifeLearner (see Table 2) but also maintains higher accuracy than ANML despite Tiny LifeLearner's model size being 24.1-1839× smaller than ANML. Tiny LifeLearner achieves significantly higher accuracy than Tiny ANML while having minimal resource requirements (e.g., 181-281kB SRAM, 725-825kB Flash, 832-1,204ms latency, and 195-282mJ energy consumption), demonstrating the effectiveness of our proposed algorithm and hardware-aware system implementation on such an extremely resource-constrained device.

Note that it is infeasible to perform the ablation study to quantify the benefits of our design as in Section 5.3. This is because other baselines with rehearsal strategy and prior works exhibit out-of-memory problems and only tiny LifeLearner could run on MCUs with severely limited memory.

6 Discussion

Impact on Continual Learning. We envision that LifeLearner could make CL a practical reality on embedded and IoT devices by leveraging meta-learning and rehearsal strategy with only a few samples. Such CL systems will allow DNNs to add new classes (e.g., adding new objects to an image recognition system, adding new keywords to a voice assistant) or new modalities (e.g., adding image recognition on top of a voice recognition authentication system) on the fly without relying on the cloud (i.e., no communication costs). As one future direction, further optimizing LifeLearner to use stricter quantization such as 1, 2, or 4 bits will be interesting.

Generalizability of LifeLearner. LifeLearner successfully works on three different datasets operating on two different modalities: image and audio, showing the generalizability of our framework. With the proliferation of smart spaces, such as smart homes and offices, LifeLearner can be used to learn the personal habits and preferences of users in order to control environmental conditions, such as temperature, humidity and lighting, with readings coming from thermometers, motion sensors and cameras on IoT devices. LifeLearner would enable this personalization and space adaptivity to happen in a data-efficient manner and to stay local to ensure privacy. Moreover, LifeLearner could be used on robot vacuum cleaners to enhance their adaptability, e.g., to continually learn to visually recognize new objects and thus avoid collisions.

The evaluation of other datasets and potentially other modalities, including various other sensor signals [14, 75] as mentioned above to further test the applicability of LifeLearner for learning continually for other real-world applications, is left as future work.

Scalibility over Many Classes. The sample-wise compression ratio of LifeLearner is about 30×, significantly reducing the memory overhead of adding many classes. It incurs only 1.68 MB, 6.16 MB, and 0.66 MB of memory when adding 100 classes with 30 samples per class for CIFAR-100, MiniImageNet, and GSCv2, respectively. Also, our scalar quantization and selective layer updates resolve scalability issues of latency as it incurs minimal latency overhead over ANML with fixed latency to learn new classes (see Tables 2, 3).

Feasibility of Labeling Samples. One of the key challenges of enabling realistic applications for CL is annotation difficulty by users. As conventional CL typically demands a few thousand labeled samples, it becomes almost infeasible for users to perform labeling (as discussed in Section 2.1). Instead, LifeLearner ameliorates this labeling burden by enabling data-efficient CL with 10-30 samples per class which are not impractical to label.

Other Considerations. In this work, our evaluation demonstrated that LifeLearner achieves near-optimal CL performance, falling short by only 2.8% accuracy compared to the upper bound system (Oracle). However, a higher accuracy (over 80-90%) given fewer samples (less than 10-30 samples) would be desirable. Thus, it is worth investigating larger and more advanced model architectures specializing in the target problem and task, such as Transformers [16, 94], to push the envelope of the upper bound testing accuracy of the challenging CL problem.

7 Conclusions

We proposed LifeLearner, a hardware-aware meta CL system with adaptive fast-slow weights and resource-optimized compression for embedded and IoT platforms. LifeLearner outperforms all existing Meta CL methods by a large margin (approximating the upper bound method that performs training in i.i.d. setting) and demonstrates its potential applicability in real-world deployments. Our efficient CL system opens the door to adaptive applications to run on embedded and IoT devices by allowing them to learn new tasks and adapt to the dynamics of the user and context.

ACKNOWLEDGMENTS

This work is supported by a Google Faculty Award, ERC through Project 833296 (EAR), and Nokia Bell Labs through a donation

REFERENCES

[1] Maruan Al-Shedivat, Trapit Bansal, Yura Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. 2018. Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments. In *International Conference on Learning Representations (ICLR)*.

[2] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. 2018. Memory Aware Synapses: Learning what (not) to forget. In *European Conference on Computer Vision (ECCV)*.

[3] 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 (MLSys)* (2021).

[4] Shawn Beaulieu, Lapo Frati, Thomas Miconi, Joel Lehman, Kenneth O. Stanley, Jeff Clune, and Nick Cheney. 2020. Learning to Continually Learn. In *ECAI 2020*. IOS Press, 992–1001.

[5] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. 2020. TinyTL: Reduce Memory, Not Parameters for Efficient On-Device Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[6] Han Cai, Ji Lin, Yujun Lin, Zhijian Liu, Haotian Tang, Hanrui Wang, Ligeng Zhu, and Song Han. 2022. Enable Deep Learning on Mobile Devices: Methods, Systems, and Applications. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 27, 3 (2022), 20:1–20:50.

[7] Francisco M. Castro, Manuel J. Marin-Jimenez, Nicolas Guil, Cordelia Schmid, and Karteek Alahari. 2018. End-to-End Incremental Learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[8] Jagmohan Chauhan, Young D. Kwon, Pan Hui, and Cecilia Mascolo. 2020. ContAuth: Continual Learning Framework for Behavioral-based User Authentication. *Proc. IMWUT* 4, 4 (Dec. 2020), 122:1–122:23.

[9] Jagmohan Chauhan, Young D. Kwon, and Cecilia Mascolo. 2022. Exploring On-Device Learning Using Few Shots for Audio Classification. In *2022 30th European Signal Processing Conference (EUSIPCO)*. 424–428. https://doi.org/10.23919/EUSIPCO55093.2022.9909551

[10] Jianfei Chen, Lianmin Zheng, Zhewei Yao, Dequan Wang, Ion Stoica, Michael W Mahoney, and Joseph E Gonzalez. 2021. ActNN: Reducing Training Memory Footprint via 2-Bit Activation Compressed Training. In *International Conference on Machine Learning (ICML)*.

[11] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training Deep Nets with Sublinear Memory Cost. https://doi.org/10.48550/ARXIV.1604.06174

[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 (MLSys)*.

[13] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Alex; Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. 2022. A Continual Learning Survey: Defying Forgetting in Classification Tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 7 (2022), 3366–3385. https://doi.org/10.1109/TPAMI.2021.3057446

[14] Shohreh Deldari, Hao Xue, Aaqib Saeed, Daniel V. Smith, and Flora D. Salim. 2022. COCOA: Cross Modality Contrastive Learning for Sensor Data. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 6, 3, Article 108 (sep 2022), 28 pages. https://doi.org/10.1145/3550316

[15] Shuya Ding, Zhe Chen, Tianyue Zheng, and Jun Luo. 2020. RF-Net: A Unified Meta-Learning Framework for RF-Enabled One-Shot Human Activity Recognition. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems (SenSys '20)*.

[16] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In

International Conference on Learning Representations. https://openreview.net/forum?id=YicbFdNTTy

[17] Marat Dukhan, Yiming Wu, Hao Lu, and Bert Maher. 2018. QNNPACK: Open source library for optimized mobile deep learning. https://engineering.fb.com/2018/10/29/ml-applications/qnnpack/.

[18] R David Evans and Tor Aamodt. 2021. AC-GC: Lossy Activation Compression with Guaranteed Convergence. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[19] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70 (ICML)*.

[20] 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.

[21] In Gim and JeongGil Ko. 2022. Memory-Efficient DNN Training on Mobile Devices. In *Annual International Conference on Mobile Systems, Applications and Services (MobiSys)*.

[22] Taesik Gong, Yeonsu Kim, Jinwoo Shin, and Sung-Ju Lee. 2019. MetaSense: Few-Shot Adaptation to Untrained Conditions in Deep Mobile Sensing. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems (SenSys '19)*.

[23] Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.

[24] Yunhui Guo, Noel C. Codella, Leonid Karlinsky, James V. Codella, John R. Smith, Kate Saenko, Tajana Rosing, and Rogerio Feris. 2020. A Broader Study of Cross-Domain Few-Shot Learning. In *European Conference on Computer Vision (ECCV)*.

[25] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *International Conference on Learning Representations (ICLR)*.

[26] Tyler L. Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. 2020. REMIND Your Neural Network to Prevent Catastrophic Forgetting. In *European Conference on Computer Vision (ECCV)*.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[28] Abdelrahman Hosny, Marina Neseem, and Sherief Reda. 2021. Sparse Bitmap Compression for Memory-Efficient Training on the Edge. In *2021 IEEE/ACM Symposium on Edge Computing (SEC)*.

[29] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. 2022. Meta-Learning in Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 9 (Sept. 2022), 5149–5169. https://doi.org/10.1109/TPAMI.2021.3079209 Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[30] Shell Xu Hu, Da Li, Jan Stühmer, Minyoung Kim, and Timothy M. Hospedales. 2022. Pushing the Limits of Simple Pipelines for Few-Shot Learning: External Data and Fine-Tuning Make a Difference. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[31] Chien-Chin Huang, Gu Jin, and Jinyang Li. 2020. SwapAdvisor: Pushing Deep Learning Beyond the GPU Memory Limit via Smart Swapping. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[32] Kai Huang, Boyuan Yang, and Wei Gao. 2023. ElasticTrainer: Speeding Up On-Device Training with Runtime Elastic Tensor Selection. In *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services (MobiSys '23)*.

[33] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. 2019. GPipe: Efficient Training of Giant Neural Networks Using Pipeline Parallelism. In *International Conference on Neural Information Processing Systems (NeurIPS)*.

[34] Ching-Yi Hung, Cheng-Hao Tu, Cheng-En Wu, Chien-Hung Chen, Yi-Ming Chan, and Chu-Song Chen. 2019. Compacting, Picking and Growing for Unforgetting Continual Learning. *Advances in Neural Information Processing Systems (NeurIPS)* (2019).

[35] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[36] Paras Jain, Ajay Jain, Aniruddha Nrusimha, Amir Gholami, Pieter Abbeel, Joseph Gonzalez, Kurt Keutzer, and Ion Stoica. 2020. Checkmate: Breaking the Memory Wall with Optimal Tensor Rematerialization. In *Conference on Machine Learning and Systems (MLSys)*.

[37] Khurram Javed and Martha White. 2019. Meta-Learning Representations for Continual Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[38] Joo Seong Jeong, Jingyu Lee, Donghyun Kim, Changmin Jeon, Changjin Jeong, Youngki Lee, and Byung-Gon Chun. 2022. Band: Coordinated Multi-DNN Inference on Heterogeneous Mobile Processors. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services (MobiSys '22)*.

[39] Saurav Jha, Martin Schiemer, Franco Zambonelli, and Juan Ye. 2021. Continual learning in sensor-based human activity recognition: An empirical benchmark analysis. *Information Sciences* 575 (Oct. 2021), 1–21. https://doi.org/10.1016/j.ins.2021.04.062

[40] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* (2019), 1–1.

[41] 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.

[42] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*.

[43] Marisa Kirisame, Steven Lyubomirsky, Altan Haan, Jennifer Brennan, Mike He, Jared Roesch, Tianqi Chen, and Zachary Tatlock. 2021. Dynamic Tensor Rematerialization. In *International Conference on Learning Representations (ICLR)*.

[44] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. Overcoming catastrophic forgetting in neural networks. *Proc. National Academy of Sciences* 114, 13 (March 2017), 3521–3526.

[45] Yousun Ko, Alex Chadwick, Daniel Bates, and Robert Mullins. 2021. Lane Compression: A Lightweight Lossless Compression Method for Machine Learning on Embedded Systems. *ACM Trans. Embed. Comput. Syst.* 20, 2, Article 16 (mar 2021), 26 pages. https://doi.org/10.1145/3431815

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

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

[48] 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*. Association for Computing Machinery (ACM).

[49] 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.

[50] Young D. Kwon, Jagmohan Chauhan, and Cecilia Mascolo. 2022. YONO: Modeling Multiple Heterogeneous Neural Networks on Microcontrollers. In *2022 21st ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. 285–297. https://doi.org/10.1109/IPSN54338.2022.00030

[51] Young D. Kwon, Rui Li, Stylianos I. Venieris, Jagmohan Chauhan, Nicholas D. Lane, and Cecilia Mascolo. 2023. TinyTrain: Deep Neural Network Training at the Extreme Edge. arXiv:2307.09988 [cs.LG]

[52] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. 2015. Human-level concept learning through probabilistic program induction. *Science* 350, 6266 (2015), 1332–1338. https://doi.org/10.1126/science.aab3050 arXiv:https://www.science.org/doi/pdf/10.1126/science.aab3050

[53] Guohao Lan, Bailey Heit, Tim Scargill, and Maria Gorlatova. 2020. GazeGraph: Graph-Based Few-Shot Cognitive Context Sensing from Human Visual Behavior. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems (SenSys '20)*.

[54] 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.

[55] Eugene Lee, Cheng-Han Huang, and Chen-Yi Lee. 2021. Few-Shot and Continual Learning with Attentive Independent Mechanisms. *arXiv:2107.14053 [cs]* (July 2021). http://arxiv.org/abs/2107.14053

[56] Edgar Liberis, Łukasz Dudziak, and Nicholas D. Lane. 2021. uNAS: Constrained Neural Architecture Search for Microcontrollers. In *Proceedings of the 1st Workshop on Machine Learning and Systems (EuroMLSys '21)*.

[57] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. 2020. MCUNet: Tiny Deep Learning on IoT Devices. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[58] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. 2022. On-Device Training Under 256KB Memory. In *Advances on Neural Information Processing Systems (NeurIPS)*.

[59] Neiwen Ling, Xuan Huang, Zhihe Zhao, Nan Guan, Zhenyu Yan, and Guoliang Xing. 2022. BlastNet: Exploiting Duo-Blocks for Cross-Processor Real-Time DNN Inference. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems (SenSys)*.

[60] Neiwen Ling, Kai Wang, Yuze He, Guoliang Xing, and Daqi Xie. 2021. RT-MDL: Supporting Real-Time Mixed Deep Learning Tasks on Edge Platforms. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems (SenSys '21)*.

[61] 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. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

[62] Xiaoxuan Liu, Lianmin Zheng, Dequan Wang, Yukuo Cen, Weize Chen, Xu Han, Jianfei Chen, Zhiyuan Liu, Jie Tang, Joey Gonzalez, Michael Mahoney, and Alvin Cheung. 2022. GACT: Activation Compressed Training for Generic Network Architectures. In *International Conference on Machine Learning (ICML)*.

[63] David Lopez-Paz and Marc\textquotesingle Aurelio Ranzato. 2017. Gradient Episodic Memory for Continual Learning. In *NeurIPS*.

[64] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. 2018. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In *European Conference on Computer Vision (ECCV)*.

[65] James L. McClelland, Bruce L. McNaughton, and Randall C. O'Reilly. 1995. Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological Review* 102, 3 (1995), 419–457.

[66] Michael McCloskey and Neal J. Cohen. 1989. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. In *Psychology of Learning and Motivation*. Vol. 24. 109–165.

[67] Sudhanshu Mittal, Silvio Galesso, and Thomas Brox. 2021. Essentials for Class Incremental Learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*.

[68] Anusha Nagabandi, Chelsea Finn, and Sergey Levine. 2019. Deep Online Learning Via Meta-Learning: Continual Adaptation for Model-Based RL. In *International Conference on Learning Representations (ICLR)*.

[69] Sinno Jialin Pan and Qiang Yang. 2010. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 10 (Oct. 2010), 1345–1359.

[70] Zizheng Pan, Peng Chen, Haoyu He, Jing Liu, Jianfei Cai, and Bohan Zhuang. 2021. Mesa: A Memory-saving Training Framework for Transformers. *arXiv preprint arXiv:2111.11124* (2021).

[71] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. 2019. Continual lifelong learning with neural networks: A review. *Neural Networks* 113 (May 2019), 54–71.

[72] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[73] Shishir G Patil, Paras Jain, Prabal Dutta, Ion Stoica, and Joseph Gonzalez. 2022. POET: Training Neural Networks on Tiny Devices with Integrated Rematerialization and Paging. In *International Conference on Machine Learning (ICML)*.

[74] Lorenzo Pellegrini, Gabriele Graffieti, Vincenzo Lomonaco, and Davide Maltoni. 2020. Latent Replay for Real-Time Continual Learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[75] Nhat Pham, Hong Jia, Minh Tran, Tuan Dinh, Nam Bui, Young Kwon, Dong Ma, Phuc Nguyen, Cecilia Mascolo, and Tam Vu. 2022. PROS: An Efficient Pattern-Driven Compressive Sensing Framework for Low-Power Biopotential-Based Wearables with on-Chip Intelligence. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking (MobiCom '22)*. 661–675.

[76] Ameya Prabhu, Hasan Abed Al Kader Hammoud, Puneet K. Dokania, Philip H.S. Torr, Ser-Nam Lim, Bernard Ghanem, and Adel Bibi. 2023. Computationally Budgeted Continual Learning: What Does Matter?. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 3698–3707.

[77] 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.

[78] Zhongnan Qu, Zimu Zhou, Yongxin Tong, and Lothar Thiele. 2022. P-Meta: Towards On-Device Deep Model Adaptation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Washington DC, USA) *(KDD '22)*. Association for Computing Machinery, New York, NY, USA, 1441–1451. https://doi.org/10.1145/3534678.3539293

[79] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. 2019. Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML. In *International Conference on Learning Representations (ICLR)*.

[80] 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.

[81] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. 2017. iCaRL: Incremental classifier and representation learning. In *Proc. CVPR*. 2001–2010.

[82] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2016. Progressive neural networks. *arXiv preprint arXiv:1606.04671* (2016).

[83] 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 (CVPR)*.

[84] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. 2018. Progress & Compress: A Scalable Framework for Continual Learning. In *International Conference on Machine Learning (ICML)*.

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

[86] Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical Networks for Few-shot Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[87] Nimit Sharad Sohoni, Christopher Richard Aberger, Megan Leszczynski, Jian Zhang, and Christopher Ré. 2019. Low-Memory Neural Network Training: A Technical Report. *arXiv:1904.10631 [cs, stat]* (April 2019).

[88] Pierre Stock, Angela Fan, Benjamin Graham, Edouard Grave, Rémi Gribonval, Herve Jegou, and Armand Joulin. 2020. Training with Quantization Noise for Extreme Model Compression. In *International Conference on Learning Representations (ICLR)*.

[89] Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. 2019. And the Bit Goes Down: Revisiting the Quantization of Neural Networks. In *International Conference on Learning Representations (ICLR)*.

[90] Filip Svoboda, Javier Fernandez-Marques, Edgar Liberis, and Nicholas D. Lane. 2022. Deep Learning on Microcontrollers: A Study on Deployment Costs and Challenges. In *Proceedings of the 2nd European Workshop on Machine Learning and Systems (EuroMLSys '22)*.

[91] V. Sze, Y. Chen, T. Yang, and J. S. Emer. 2017. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proc. IEEE* 105, 12 (Dec. 2017), 2295–2329. https://doi.org/10.1109/JPROC.2017.2761740 Conference Name: Proceedings of the IEEE.

[92] Jihoon Tack, Subin Kim, Sihyun Yu, Jaeho Lee, Jinwoo Shin, and Jonathan Richard Schwarz. 2023. Learning Large-scale Neural Fields via Context Pruned Meta-Learning. arXiv:2302.00617 [cs.LG]

[93] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Utku Evci, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, and Hugo Larochelle. 2020. Meta-Dataset: A Dataset of Datasets for Learning to Learn from Few Examples. In *International Conference on Learning Representations (ICLR)*.

[94] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[95] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, koray kavukcuoglu, and Daan Wierstra. 2016. Matching Networks for One Shot Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[96] Qipeng Wang, Mengwei Xu, Chao Jin, Xinran Dong, Jinliang Yuan, Xin Jin, Gang Huang, Yunxin Liu, and Xuanzhe Liu. 2022. Melon: Breaking the Memory Wall for Resource-Efficient On-Device Machine Learning. In *Annual International Conference on Mobile Systems, Applications and Services (MobiSys)*.

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

[98] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. 2019. Large Scale Incremental Learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[99] Rui Xiao, Jianwei Liu, Jinsong Han, and Kui Ren. 2021. OneFi: One-Shot Recognition for Unseen Gesture via COTS WiFi. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems (SenSys '21)*. 206–219.

[100] Shuochao Yao, Jinyang Li, Dongxin Liu, Tianshi Wang, Shengzhong Liu, Huajie Shao, and Tarek Abdelzaher. 2020. Deep Compressive Offloading: Speeding up Neural Network Inference by Trading Edge Computation for Network Latency. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems (SenSys '20)*. 476–488.

[101] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. 2018. Lifelong Learning with Dynamically Expandable Networks. In *International Conference on Learning Representations (ICLR)*.

[102] Sheng Yue, Ju Ren, Jiang Xin, Deyu Zhang, Yaoxue Zhang, and Weihua Zhuang. 2021. Efficient Federated Meta-Learning over Multi-Access Wireless Networks. arXiv:2108.06453 [cs.LG]

[103] Friedemann Zenke, Ben Poole, and Surya Ganguli. 2017. Continual Learning Through Synaptic Intelligence. In *Proc. ICML*. 3987–3995.

[104] Yu Zhang, Tao Gu, and Xi Zhang. 2020. MDLdroidLite: A Release-and-Inhibit Control Approach to Resource-Efficient Deep Neural Networks on Mobile Devices. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems (SenSys '20)*. New York, NY, USA, 463–475.