

InfiniCL: Elastic Continual Learning for Resource-Constrained Edge Devices

Chenyu Lu¹, Mengyang Liu², Fang Dong¹, Borui Li¹, Ruiting Zhou¹, Shiyao Ji¹

¹Southeast University, ²The Hong Kong University of Science and Technology

{cylu, fdong, libr, ruitingzhou, shiyao.ji}@seu.edu.cn, csemyliu@ust.hk

Abstract—On-device continual learning (CL) enables lifelong and privacy-preserving learning for various edge intelligent applications. Increasing the number of model parameters as new learning tasks emerge is effective in ensuring learning quality but inefficient in memory cost, especially for resource-constrained devices. In this paper, we introduce InfiniCL, the first on-device CL system that dynamically balances memory cost and learning quality. A key idea behind InfiniCL is *elastic continual learning*: selectively freezing layers in the expanding model and periodically distilling the model, preventing unbounded memory growth while preserving learning quality for new tasks. This novel CL paradigm opens a new challenging problem: how to decide the memory allocation of the model and data to achieve better learning Quality of Service (QoS) under the limited memory budget? To alleviate this challenge, we further propose a Bayesian Optimization-driven algorithm to jointly optimize layer freezing selection and data-model memory allocation. Evaluations show that InfiniCL outperforms state-of-the-art methods on diverse memory constraints, achieving 5.34–7.15% and 2.72–9.36% higher accuracy on CIFAR-100 and ImageNet-100, respectively.

Index Terms—continual learning, lifelong learning, on-device learning, machine learning system

I. INTRODUCTION

Combining continual learning (CL) with on-device learning presents a promising approach for edge applications that require streaming data processing and personalization capabilities [1], [2]. For example, robots and automated systems continually learn from their environment, leveraging ongoing sensor data to enable lifelong learning and enhance learning Quality of Service (QoS) [3]. Existing CL methods reinforce learned knowledge through data replay [4]–[6] and manage growing new knowledge by dynamically expanding model capacity [7]–[10].

However, on resource-constrained Internet of Things (IoT) devices, infinite model expanding is infeasible due to the physical memory limitations. As illustrated in Figure 1, although model expanding enhances CL performance, continuously expanding the model results in a rapid increase in memory footprint, leading to unsustainable costs.

The key idea of this work is developing elastic continual learning, dynamically scaling up or down the model capacity to fit the model and data status. To achieve this, we face two challenges: **(1) Incremental Learning Cost**. As the model expands, its memory consumption incrementally increases. Taking ResNet32 [11] as an example, in the current state-of-the-art (SOTA) memory-efficient model-expand strategy [9],

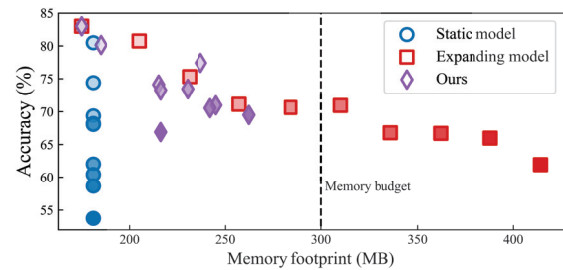


Fig. 1. Accuracy performance and memory consumption of different CL methods on CIFAR-100 using ResNet32 on a Jetson Xavier NX. The dataset is divided into 10 tasks. The color intensity of the markers in the figure represents the number of tasks learned by the CL system. Darker colors indicate that the model has learned more tasks. Expanding the model improves learning quality but causes out-of-memory (OOM) issues under a tight memory budget.

the runtime cost during training nearly doubles after expanding 10 times. Freezing layers [12], [13] is a common approach to reducing the memory cost in traditional model training. However, the existing freezing-based on-device CL system [12] has not considered model architecture change during continual learning and offers a static freeze plan before the training. This static freeze plan cannot satisfy system running requirements due to the incremental memory cost from the model expanding. Besides, the freezing strategy only reduces the runtime cost during CL without reducing the expanding memory cost from increasing parameters. In addition, excessive model expanding under a fixed memory budget will gradually squeeze the space left for trainable layers, ultimately resulting in bad learning quality. **(2) Data-Model Memory Contention**. In devices with unified memory architectures [14], the restricted memory needs to store the newly incoming data, history data and train the model together. While the freezing mechanism helps control the memory cost of the model expanding, the memory usage of both data and model inevitably affects learning QoS. Miro [15] proposes a data allocation strategy that optimizes the balance between newly incoming data and replay data through a random sampling approach. However, when integrating a freezing mechanism with model-expand strategy, jointly optimizing both freezing and data allocation becomes significantly more difficult. The existing random sampling strategy is insufficient to address this challenge.

In this paper, we propose InfiniCL, the first on-device CL system to enable sustainable lifelong learning on resource-constrained devices. InfiniCL proposes the following two

novel techniques for solving the above two challenges, respectively. **(1) Elastic Continual Learning with Layer Freezing.** InfiniCL avoids the incremental memory costs problem during CL by selective freezing layers and distilling the expanding model periodically. When a new task comes, the system first expands the model to enhance the learning capacity for learning new knowledge and then generates a freeze plan to reduce the incremental memory consumption. In the generation of the freeze plan, the system scores the learning performance of each layer, and layers with lower learning efficiency are more likely frozen. By freezing layers with low learning efficiency, InfiniCL smooths the infinite increase in memory cost with optimal learning quality. To mitigate the expanding parameter size, InfiniCL designs an automatic distillation method to shrink the model capacity when the freezing starts to sacrifice too much learning quality. **(2) Continual Balancing.** Building on the proposed elastic continual learning paradigm, we formulate the balance problem of allocating data-model memory budget as an Integer Linear Programming (ILP) problem. To efficiently solve this problem within a large solution space and profiling cost, InfiniCL proposes a planning algorithm based on Bayesian Optimization (BO) [16] with a Gaussian Process (GP) surrogate model. Moreover, InfiniCL proposes a dynamic programming algorithm to solve the freeze planning problem based on the allocated model memory budget. Continual balancing can quickly find a high-performance training configuration before each learning task, and balance the memory usage of different parts to achieve QoS-aware continual learning on resource-constraint devices.

Our contributions are summarized as follows:

- We introduce a novel *elastic continual learning* paradigm to address the deployment challenges of developing life-long CL on resource-constrained devices.
- We identify the resource contention issue between data and model in CL system. We formalize this problem as an ILP problem and propose a continual balancing algorithm to solve it.
- We implement and deploy InfiniCL on the NVIDIA Jetson Xavier NX platform. Under varying datasets and memory constraints, InfiniCL achieves 2.72%–9.36% higher accuracy than SOTA methods by dynamically balancing learning quality and memory consumption.

II. BACKGROUND AND MOTIVATION

A. Background

Continual learning. Traditional deep learning trains the model in a fixed dataset once and then deploys it into the system. Once the new data emerge, the model must be trained again to master new knowledge [17], [18]. Continual learning (CL) is a more efficient training paradigm to train the model on ever-growing streaming data to continuously evolve the model and avoid repeated pre-training. One problem in the CL is the catastrophic forgetting [19] that the model will forget the old knowledge gradually when continuously learning from the new data. The data-replay method [4]–[6], also known as the

rehearsal-based method, mitigate forgetting by storing subsets of past data in episodic memory (EM) and interleaving them with incoming data from the stream buffer (SB). This hybrid training regime balances the model’s ability to generalize across both old and new tasks. Another problem in the CL is the loss of plasticity [20]: the learning capacity of the model is limited and the model cannot handle knowledge when the size exceeds the capacity. Model-expand strategies [7]–[10] are proposed to continuously increase the learning capacity of the model by dynamically growing network architectures. For memory-efficiency, Some model-expand strategy decomposes models into general modules (shared across tasks) and adaptive modules (task-specific expansions). Once a new task comes, these methods expand a new adaptive module from the general module to scale up the learning capacity for mastering new knowledge.

On-device learning. Training the model on-device at the edge offers a more secure and cost-effective way to democratize intelligence than training from the cloud. The most problem of on-device learning is the memory and computation limits of edge devices. Lots of efficient training algorithms [21] and systems [12], [22] have been proposed to solve this problem. Continual learning integrated with on-device learning offers a promising way toward lifelong and privacy-preserving learning, especially for edge applications requiring personalization such as intelligent robots [3], on-device recommender system [23] and health monitoring [24].

B. Motivation and challenge

When deploying the CL methods on resource-constrained devices, limited memory becomes the primary challenge. Existing on-device continual learning system deploys hierarchy episodic memory (HEM) [25] to regularly exchange the samples in EM between memory and storage, significantly improving the performance of data-replay methods. Building on HEM, Miro [15] dynamically adjusts the size of the SB and EM during training to balance memory usage and learning performance. For further memory savings, LifeLearner [12] introduces a hardware-aware approach that reuses intermediate model outputs (e.g., layer activations) as EM while freezing earlier layers. While these systems improve data-replay efficiency, they are not designed to support model-expand strategies. We aim to develop a sustainable on-device CL system that employs model-expand and data-replay methods while being adaptable to varying memory requirements.

To achieve this goal, we identify two challenges in developing this system:

Challenge 1: Incremental Learning Cost. We analyze the memory usage of model-expand strategy on PyTorch Framework. As shown in Fig. 2, after 10 Tasks, the total memory footprint increases nearly 2x than the initial state. The main contributor to memory cost is intermediate activations (the main part of runtime memory) accumulated during the forward pass for computing gradients in the backward pass. Existing works [26] have demonstrated that freezing layers

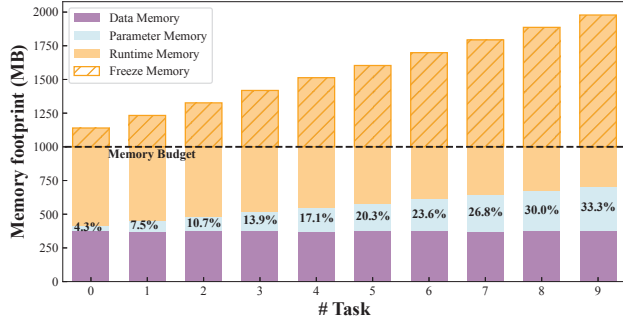


Fig. 2. The analysis of memory consumption of model-expand CL method on ImageNet-100 using ResNet18. Each task in this CL process contains 10 classes. The batch size is 32.

during training can significantly reduce memory consumption by avoiding the cache of intermediate activations.

Using model freezing to reduce learning costs is common in on-device learning systems [22] and has also been introduced into CL systems [12], [27] to improve efficiency. However, there is no existing work to develop a freezing method both considering model-expand strategy and dynamic tailoring the freeze plan for each new task. LifeLearner [12] only plans the freezing strategy once before the learning, which is unsuitable for dynamic model changing in model-expand. Memo [9] and DER [8] naively freeze previously expanded modules, meaning that the newly expanded modules are restricted to training within the current task, which limits their potential learning capability. L-SAR [27] proposes a dynamic freezing strategy that optimizes for computational efficiency. Nevertheless, it overlooks runtime memory overhead, which is a more critical bottleneck than computation in on-device CL systems. Additionally, for continuously growing models, apart from the increase in runtime memory, their parameter size will also increase. Taking the example in Fig. 2, the proportion of the model's parameters in the memory budget has grown from 4.3% to 33.3%. Freezing layers cannot address the issues arising from the growth of model parameters, while the increase in model parameters will compress the number of trainable parameters, leading to degraded training performance.

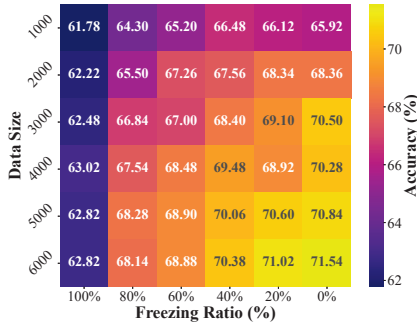


Fig. 3. Accuracy performance on CIFAR-100 after 5 tasks under different data sizes and model freeze ratios. Both the data size and the proportion of the trainable model affect accuracy.

Challenge 2: Data-Model Contention. Most edge devices employ a unified memory architecture in which all processors (such as CPUs and GPUs) share unified physical memory. Resource-constrained devices must undertake both the mem-

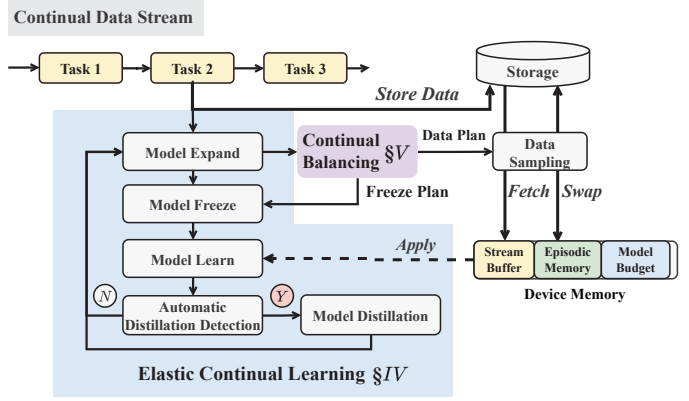


Fig. 4. InfiniCL Overview

ory consumption of model training (runtime memory and parameter memory) and data caching (including SB and EM). This results in a resource contention between data and model.

As shown in Fig. 3, increasing either the number of training examples or the number of trainable weights can enhance accuracy, but both demand more memory. Under the constraint of system memory, freezing more layers of the model reduces memory usage, allowing more data to be trained, but it may limit the learning ability of the model. Conversely, freezing fewer layers of the model enhances the learning ability of the model but reduces the amount of trainable data, which may also sacrifice the learning ability. This motivates us to achieve the optimal balance between model and data memory usage in one CL process.

Existing works are not focusing on this contention problem and most of them use a fixed budget for the data [9] in the CL process. L-SAR [27] uses a dynamic budget for the data to optimize the learning ability rather than considering memory costs. Miro [15] uses a dynamic budget for the data and balances memory proportions of EM and SB to optimize the learning QoS. However, Miro has not considered data-model contention.

The contention relationship is challenging to balance. First, offline grid search and random sampling are impractical due to time-consuming profiling. During CL, the data stream evolves continuously, and the model must quickly adapt to an increasing amount of knowledge to retain and maintain. Second, as the model structure expands and new tasks continue to arrive, decisions that were optimal for previous tasks may no longer perform well on newer tasks. This requires us to use as few samples as possible. In addition, we introduce the freeze planning problem into this contention problem, which further increases the difficulty of solving it.

III. SYSTEM OVERVIEW

In this section, we illustrate the system overview and workflow of InfiniCL. As shown in Figure 4, InfiniCL's workflow includes elastic continual learning (§IV) for model management and continual balancing (§V) for training plan optimization. Given a new task from continual data stream,

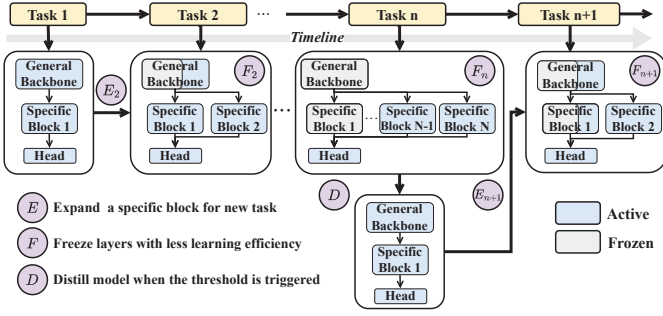


Fig. 5. Elastic Continual Learning with Layer Freezing

the system stores the new data in the storage and puts it into the elastic continual learning processing. In elastic continual learning, the model expands to increase model capacity for the growing knowledge to be learned. Then a continual balancing process will be executed to offer an optimal balancing plan including data and freeze plan. The data plan determines how much data the system should be trained on and will be input to the data sampling process to control the fetching and swapping data size of SB and EM. The freeze plan specifies which layers should be frozen and will be input to the model freeze process to freeze part of the layers in the model. After freezing, the model learning can be processed to train the model with sampled data in the memory. Once the learning of the new task is completed, automatic distillation detection is triggered to judge whether the model needs to be distilled and outputs the model (distilled or not distilled) to the model-expand process.

IV. ELASTIC CONTINUAL LEARNING

In this section, we delve into InfiniCL's elastic continual learning (ECL) mechanism in detail, explaining how the model expands, how it freezes, and how it is distilled.

Expand the model at each new task. When a new task arrives, InfiniCL is triggered to expand the model to integrate new knowledge. At each model-expanding step, only a portion of the model is expanded to ensure memory efficiency. Following Memo [9], we divide the model into classifier module $\mathcal{H}$ and feature extractor module Φ , as

$$y = \mathcal{H}(\Phi(x)). \quad (1)$$

In InfiniCL, the feature extractor of the model consists of one shared general part Φ_g and multiple specialized parts Φ_s . Let the expanded module at t -th training task as Φ_s^t . The task-specific module Φ_s^t represents the last stage of the model. The model can be expressed as

$$y = \mathcal{H}'(\Phi_s^0(\Phi_g(x)), \dots, \Phi_s^{t-1}(\Phi_g(x)), \Phi_s^t(\Phi_g(x))). \quad (2)$$

In fact, our approach does not impose any requirements on the model-expand method. Any model-expand methods can be optimized via InfiniCL.

Train selectively at runtime. InfiniCL leverages *continual freezing* during training by selecting several layers to train and freezing others. The key consideration of InfiniCL is to determine which layers to freeze. To achieve this, it is essential

to evaluate the contribution and associated costs of each layer. To automatically adapt to different model architectures, a trace profiler is implemented by incorporating hooks into each layer in the model. These hooks facilitate the detection of layer characteristics, such as kernel type, input and output shapes, and parameter size. The profiler also records the shapes of intermediate activations between layers to obtain the size of activations. This information forms the basis for computing eq. (4) and (5). On the other hand, inspired by EWC [28], InfiniCL designs a contribution-based layer selector that leverages the Fisher Information Matrix (FIM) to measure the accuracy contribution of each layer. By holistically considering the cost-contribution trade-offs of each layer, we propose a layer selection strategy for training in §V-B.

Distill the model after every t task. By leveraging the layer freezing mechanism, InfiniCL reduces computation and memory consumption during model expanding. However, as tasks incrementally increase, the expanding of the model parameters will consume more memory, resulting in a reduction of available resources for training and potentially lower overall performance. To determine when to trigger the distillation, we define two thresholds based on a minimum training ratio ρ . 1) *Memory-related threshold*. As illustrated in eq.(4), freezing the least contributory portion of the model reduces memory costs for activations and optimizer states but not for parameters. The distillation timing is defined as the point when freezing the $(1 - \rho)$ portion still fails to reduce the total memory consumption of the expanded model to the level of the initial state. 2) *Computation-related threshold*. We also consider the computation-related threshold in the same way, defining the distillation timing as the point when the $(1 - \rho)$ flops of the expanded model equals the total forward and backward flops of the original model. We finally determine the distillation iteration t as the smaller of the two thresholds.

V. CONTINUAL BALANCING

In this section, we present the design principles and technical implementation of continual balancing. First, we model the data-model allocation problem as an Integer Linear Programming (ILP) problem (§V-A). As shown in Fig. 6, to effectively search the optimal configuration, we implement a plan generator (§V-B) to handle memory allocation planning and freeze planning.

A. Problem Formulation

Continual Learning and Data Replay. When a new task arrives, InfiniCL allocates a memory space called *stream buffer* (SB) to receive the new data in the current task. To maintain the model's performance on previous classes, InfiniCL also caches a small set of previous tasks' data as *episodic memory* (EM). The training data plan $\mathbf{d} = (\text{SB}^t, \text{EM}^t)$ has a significant influence on task accuracy. Let $\mathcal{M}$ denote the number of classes in the new task, $\mathcal{N}$ the amount of sampled data in each new class, $\mathcal{Q}$ the number of classes in previous tasks,

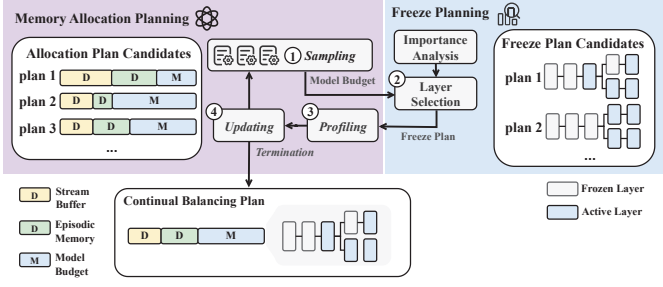


Fig. 6. Continual Balancing

$\mathcal{K}$ the amount of sampled data in each previous class. The training plan $\mathbf{d}$ of system at task t is

$$\mathbf{SB}^t = \sum_{i=0}^{\mathcal{M}} \sum_{j=0}^{\mathcal{N}} D_{i,j}, \mathbf{EM}^t = \sum_{i=0}^{\mathcal{Q}} \sum_{j=0}^{\mathcal{K}} D_{i,j}, \quad (3)$$

where $D_{i,j}$ is the random sample of class i . By adjusting the amount of $\mathbf{SB}^t$ and $\mathbf{EM}^t$ in $\mathbf{d}$, InfiniCL can refine the data distribution and improve the overall system performance.

Optimization Formulation. Due to the incremental memory consumption in continual learning systems using the model-expand strategy, we introduce a freezing strategy to reduce memory usage during training.

The memory usage of a trainable layer l_i consists of four components, including the weight size m_p^i , activation size m_a^i , gradient size m_g^i and optimizer state size m_o^i . We define the adaptive freeze plan of a model with L layers as $\mathbf{f} = (f_0, f_1, \dots, f_L)$, where each element f_i in $\mathbf{f}$ is a binary variable that determines whether the layer l_i is frozen or not. For layer l_i , the decision variable f_i determines whether m_a^i , m_g^i and m_o^i are allocated to memory or not. Specifically, as long as the earlier layers participate in training, the gradients of activation will propagate back through them. Thus, the m_a^i does not need to be stored only if all the previous layers are frozen. Given a freeze plan $\mathbf{f}$, the memory consumption of training runtime $Mem(\mathbf{f})$ can be calculated as

$$Mem(\mathbf{f}) = \sum_{i=1}^L \left(m_p^i + \left(1 - \prod_{j=1}^i f_j \right) \cdot m_a^i + (1 - f_i) \cdot m_o^i \right). \quad (4)$$

We use FLOPs as the proxy of computation cost, so that the computation cost under a continual learning configuration $(\mathbf{f}, \mathbf{d})$ can be expressed as

$$Cost(\mathbf{f}, \mathbf{d}) = \sum_{b \in \mathbf{d}} \sum_{i=1}^L \left(\text{FLOPs}_{\text{fw}}^i + \left(1 - \prod_{j=1}^i f_j \right) \cdot \text{FLOPs}_{\text{input}}^i + (1 - f_i) \cdot \text{FLOPs}_{\text{param}}^i \right), \quad (5)$$

where b is represented a batch of data in data plan, $\text{FLOPs}_{\text{fw}}^i$ is represented the forward FLOPs for layer i . We divide the computational cost of backpropagation into the gradient computation of the parameters $\text{FLOPs}_{\text{param}}^i$ and the gradient

Algorithm 1: Select the Optimal Freeze Plan

Input : Freeze Ratio ρ , Maximum memory $M_{\max}$, Maximum FLOPs $C_{\max}$, Model ϕ

Output: Freezing strategy $\mathbf{f} = (f_1, \dots, f_L)$

```

1 Initialize DP table:  $\text{dp} \leftarrow \{(0, 0) : (0, \emptyset)\}$ 
  // Calculate Require Resource Saving
2 Initialize  $R_c \leftarrow \rho * C_{\max}$ ,  $R_m \leftarrow \rho * M_{\max}$ 
3 Profile layers information  $\mathcal{L} \leftarrow \{(\Delta c_i, \Delta m_i, v_i)\}_{i=1}^L$ 
4 foreach layer  $i \in \{1, \dots, L\}$  do
5    $(\Delta c_i, \Delta m_i, v_i) \leftarrow \mathcal{L}[i]$ 
  // Keep current layer unfrozen
6   Initialize  $\text{dp}_{\text{tmp}} \leftarrow \text{dp.copy}()$ 
7   foreach  $(c, m) \in \text{dp}$  do
8     // Freeze current layer
9     if freezing all remaining layers is insufficient
10    then
11       $\text{dp}_{\text{tmp}}.\text{remove}((c, m))$ 
12      continue
13    end
14    Calculate  $c'$  and  $m'$  by eq.(2) and eq.(3)
15     $\ell' \leftarrow \text{dp}[c, m].\text{loss} + v_i$ 
16     $\mathbf{f}' \leftarrow \text{dp}[c, m].\mathbf{f} \cup \{i\}$ 
17    if  $(c', m') \notin \text{dp}_{\text{tmp}}$  or  $\ell' < \text{dp}_{\text{tmp}}[c', m'].loss$ 
18    then
19       $\text{dp}_{\text{tmp}}[c', m'] \leftarrow (\ell', \mathbf{f}')$ 
20    end
21  end
22   $\text{dp} \leftarrow \text{dp}_{\text{tmp}}$ 
  // Prune dominated states
23  Remove state  $\text{dp}[c, m]$  if  $\exists \text{dp}[c', m']$  s.t.  $c' \geq c$ ,
   $m' \geq m$ , and  $\text{dp}[c', m'].\ell' \leq \text{dp}[c, m].\ell$ .
24 return  $\text{dp}[c^*, m^*].\mathbf{f}$ 

```

computation of input activations $\text{FLOPs}_{\text{input}}^i$. We can regard the model accuracy as the accumulation of each layer's accuracy contribution [22]. We use the $\Delta \text{acc}_{i, \mathbf{d}}$ to represent the evaluate contribution of layer l_i under data plan $\mathbf{d}$. The model accuracy can be expressed as

$$Acc(\mathbf{f}, \mathbf{d}) = \sum_{i=0}^L f_i \Delta \text{acc}_{i, \mathbf{d}}. \quad (6)$$

The optimization objective of InfiniCL is to maximize the accuracy performance under memory budget B_{mem} and computation budget B_{comp} . On the one hand, freezing more layers or using a smaller data plan can reduce the computation cost of the current task, but lead to lower accuracy. On the other hand, training more layers or using a larger data plan can achieve better accuracy performance, but higher computation cost. Thus, we formulate our problem as an ILP optimization

problem which can be expressed as

$$\mathbf{P1:} \max_{\mathbf{f}, \mathbf{d}} \text{Acc}(\mathbf{f}, \mathbf{d}) \quad (7)$$

$$\text{s.t. } \text{Mem}(\mathbf{d}) + \text{Mem}(\mathbf{f}) < B_{\text{mem}} \quad (8)$$

$$\text{Cost}(\mathbf{f}, \mathbf{d}) < B_{\text{comp}}. \quad (9)$$

Solving this problem is challenging from both algorithmic and system-level perspectives. First, this problem is an NP-hard problem with a large search space for solution. For instance, with a l -layer model and m candidate data plans, the optimization problem has $m \times 2^l$ possible solutions. Second, the accuracy metric is difficult to obtain through simple proxies like FLOPs for computation cost. While prior works rely on a train-based profiler to measure accuracy, this approach is non-trivial for our system due to the large solution space.

B. Plan Generator

The plan generator focuses on optimizing **P1** and aims at allocating the memory and computation budget to data and model, respectively. It aims to find the best configuration of the data plan and freeze plan under the specific budget to optimize the system performance.

As the ILP problem formulated in §V-A, optimizing **P1** remains computationally expensive. First, the feasible solution space for this problem is high-dimensional. Second, obtaining the accuracy of a candidate configuration $(\mathbf{f}, \mathbf{d})$ requires training, which is non-trivial and resource-intensive.

A potential approach to estimate accuracy is to use a lightweight training-based profiler that samples a small portion from all data plan candidates and trains the model for a few epochs [15]. However, this approach also introduces overhead into the system, and the overhead is positively correlated with the number of candidate configurations.

To efficiently search for the best configuration with minimal overhead, InfiniCL employs *Bayesian Optimization* (BO) with a *Gaussian Process* (GP) surrogate model as the central module of the plan generator. Our design is based on two considerations. First, BO uses a surrogate model to approximate the objective function and predict the values of other configurations based on a small set of observations, thereby reducing overhead and effectively searching the optimal configurations of **P1**. Second, the surrogate GP will generate a mean μ and variance σ^2 for the objective function, representing the predicted value of the objective function at the specific configuration and the uncertainty in the prediction. This characteristic aligns with the randomness and uncertainty during the training process.

However, the convergence speed of BO critically depends on the scale and dimensionality of the solution space. To improve the search efficiency of the plan generator, we introduce a dynamic programming-based *layer selector* that rapidly prunes layers with lower learning efficiency to narrow down the exponential search space introduced by the model. As illustrated in Algorithm 1, the selector takes a freeze ratio ρ and current memory/computation budgets as inputs and returns a freeze plan $\mathbf{f}$ to determine the frozen and active layers. Lines 1-2

Algorithm 2: Optimize the Balancing Plan

Input: Model ϕ , Memory Budget B_{mem} , Computation Budget B_{comp} , Data Plan Pool $\mathcal{D}$, Freeze Ratio List $\mathcal{R}$, Profiler P , Initial point T_{init} , Total iteration T_{total}

Output: Optimal Configuration $(\mathbf{f}, \mathbf{d})$

```

1 Initialize GP surrogate model  $\mathcal{S}$  with prior mean and
  kernel;
2 Initialize observed configurations  $\mathcal{O} \leftarrow \emptyset$ ;
3 Initialize acquisition function  $\mathcal{A}$ ;
4 for  $i = 1$  to  $T_{\text{total}}$  do
5   if  $i \leq T_{\text{init}}$  then
6     Randomly sample  $(\mathbf{r}, \mathbf{d})$  from  $\mathcal{R} \times \mathcal{D}$ ;
7   else
8     Optimize  $(\mathbf{r}, \mathbf{d})$  by maximizing  $\mathcal{A}$  based on  $\mathcal{S}$ ;
9   end
10  Get  $\mathbf{f}$  by Algorithm 1 with  $(\mathbf{r}, \mathbf{d})$ ;
11  if  $(\mathbf{f}, \mathbf{d})$  satisfies  $B_{\text{mem}}$  and  $B_{\text{comp}}$  then
12     $\mathcal{F}(\mathbf{r}, \mathbf{d}) \leftarrow \text{Profiler } P(\phi, \mathbf{f}, \mathbf{d})$ ;
13     $\mathcal{O} \leftarrow \mathcal{O} \cup \{(\mathbf{r}, \mathbf{d}, \mathcal{F}(\mathbf{r}, \mathbf{d}))\}$ ;
14    Update  $\mathcal{S}$  with  $\mathcal{O}$ ;
15  end
16 end
17 return  $\arg \max_{(\mathbf{r}, \mathbf{d}) \in \mathcal{O}} \mathcal{F}(\mathbf{f}, \mathbf{d})$ ;

```

initialize the dynamic programming (DP) state table, where each DP state is represented as a tuple $(c_{\text{save}}, m_{\text{save}}) : (\ell, \mathbf{f})$. Here, c_{save} denotes computational savings of the current plan, m_{save} indicates memory savings of the current plan, and ℓ represents the contribution loss when freezing selected layers in the current plan. We aim to find a plan $\mathbf{f}$ that minimizes the contribution loss while satisfying the resource saving. We profile the FIM value of each layer as the learning efficiency with a part of data sampled from the training dataset. We iterate through each layer and calculate the current cost and efficiency loss based on whether it is frozen or not (Line 5-19). To speed up the execution, we aim to prune the suboptimal states. During the iteration of the DP table, we remove the status when freezing all remaining layers is insufficient to satisfy the required resource saving (Line 8-10). After the updating of the DP table for the current layer, the layer selector will prune the dominated states, which refer to solutions with less resource savings and higher loss in contribution, to avoid excessive time and space overhead (Line 22).

By leveraging the layer selector in BO, the plan generator performs an automated configuration search as illustrated in Algorithm 2. InfiniCL employs a training-based profiler P to evaluate candidate configurations, using the profiled accuracy as the objective function value $\mathcal{F}(\mathbf{f}, \mathbf{d})$. The algorithm initializes a GP surrogate model with a Matérn(5/2) kernel $\kappa(\cdot, \cdot)$ for modeling the objective function. This kernel strikes an effective balance between smoothness and flexibility, enabling it to capture configuration-performance relationships

accurately. During subsequent iterations, we employ Expected Improvement (EI) as the acquisition function $\mathcal{A}$ to balance exploration and exploitation, mathematically expressed as:

$$\text{EI}(\mathbf{r}, \mathbf{d}) = \mathbb{E} [\max (\mathcal{S}(\mathbf{r}, \mathbf{d}) - f^+, 0)], \quad (10)$$

where f^+ denotes the best-observed objective value. The GP posterior is iteratively updated during T_{total} optimization epochs, with each iteration either exploring random configurations (during the warm-up phase) or exploiting the surrogate model's predictions to propose promising candidates.

VI. IMPLEMENTATION

We implement InfiniCL on top of PyTorch 2.1 [29]. Our system follows Miro [15] to build based on HEM [25] architecture. HEM supports swapping episodic memory between memory and disk to replay more historical data to overcome catastrophic forgetting. All of experiments presented in this paper are evaluated on the Nvidia Jetson Xavier NX platform, which is equipped with 16GB unified memory and a 384-core NVIDIA GPU with 48 Tensor Cores.

VII. EXPERIMENT

A. Configuration

Baselines. We compare InfiniCL with the following baselines. **1) Miro.** Miro [15] is the state-of-the-art on-device CL system that presents a cost-effective data management approach without consideration of model-expand. **2) Memo.** Memo [9] is a CL method with model-expand that statically allocates the system resource between data and model. **3) Memo with naive freezing (Memo-F).** To reduce computation and memory consumption during the CL process, Memo proposed a naive freezing strategy that freezes all previous task-specific modules. All baselines are evaluated on our implemented HEM [25] architecture.

Models, Datasets and Tasks. We evaluate the performance of InfiniCL under various datasets and models. 1) CIFAR-100 [30] consists of 100 classes and each class includes 500 images. ResNet32 [11] is used for this dataset. 2) ImageNet-100 [31] consists of 100 randomly selected classes from ImageNet-1000, and each class includes 1300 images. ResNet18 [11] is used for this dataset. For continual learning, each dataset is divided into 10 tasks and each task includes 10 classes.

Training Configuration. For all training tasks, the optimization details (number of epochs, learning rate, optimizer) are set according to Miro's [15] design. We use a batch size of 64 for CIFAR-100, 32 for ImageNet-100. Each task is trained for 70 epochs.

B. Overall Performance on Various Memory Constraints

In this subsection, we comprehensively evaluate the overall performance of InfiniCL and compare it with other baseline methods. Since memory is the primary factor limiting the performance of the CL method, we first analyze the accuracy performance under different memory budgets.

When a new task arrives, Memo and Memo-F will only expand the model if there is sufficient remaining memory for the expansion. Under a tight memory budget, Memo and Memo-F will exhibit poor performance. As shown in Figure 8(a) and Figure 8(b), the ResNet32 model can be expanded once or twice, respectively, under the corresponding budget. It can be observed that when the model cannot expand, the imbalance in the remaining memory allocation of data and model causes the rapid accuracy decrease of the naive model-expand method.

To evaluate the effectiveness of our method more comprehensively, we conduct various experiments on ImageNet-100 as shown in Fig. 7, with memory budgets ranging from 1000 MB to 2000 MB. We highlight three critical performance metrics in Table I during the CL process, including average accuracy, accuracy after Task 5, and accuracy after Task 10, offering a holistic evaluation of our method's robustness and adaptability across different stages of the learning process. Overall, InfiniCL outperforms current SOTA, Miro, with 2.92-9.32% in average accuracy and 2.72-9.36% in final accuracy. The advantage of InfiniCL is more significant under a tight memory budget, as it effectively balances the memory allocation between data and the model through elastic continual learning.

C. Energy-Accuracy Trade-off

The incremental training costs include not only the memory consumption but also the computation consumption. In this subsection, we evaluate the energy-accuracy trade-off in Fig. 9. For a faithful comparison, we relaxed the memory constraints to allow Memo and Memo-F to fully extend the model in this experimental setting. In this experimental setting, the computation budget across different datasets corresponds to the computational cost at the model's initial size for training 10k and 15k samples in Fig.9(a) and Fig.9(c) for CIFAR-100, as well as 6k and 8k samples in Fig.9(b) and Fig.9(d) for ImageNet-100.

Since model-expand also leads to a linear increase in computational cost, we can observe that the performance of Memo and Memo-F is still inferior to that of Miro. Although our method does not explicitly optimize for energy efficiency, in InfiniCL, the benefits of freezing are reflected not only in memory but also in reduced computation. Compared with Miro, InfiniCL achieves a 0.29-2.47% increase in average accuracy while reducing 23.8-50.9% of the energy.

D. Design Validation

In this subsection, we validate the effectiveness of our design choices in elastic continual learning and continual balancing.

Design of Elastic Continual Learning with Layer Freezing. By distilling, we shrink the expanded model back to its original size. Based on this, we aim to verify whether the freezing strategy is essential. We employ the distill-only method to InfiniCL to figure out whether expand-and-distill can maintain

TABLE I
MEMORY-ACCURACY PERFORMANCE ON IMAGENET

Constraint	1000 MB			1250 MB			1500 MB			1750 MB			2000 MB		
	A_{avg}	A_{mid}	A_{last}	A_{avg}	A_{mid}	A_{last}	A_{avg}	A_{mid}	A_{last}	A_{avg}	A_{mid}	A_{last}	A_{avg}	A_{mid}	A_{last}
Memo	37.77	42.16	11.76	52.15	56.36	38.36	61.78	65.40	38.92	69.95	68.28	61.83	73.42	73.20	68.74
Memo-F	43.05	43.2	32.01	55.71	57.19	47.82	67.87	67.32	60.41	69.49	69.36	65.34	72.71	71.72	68.88
Miro	55.81	44.51	47.52	61.07	59.88	57.86	68.76	67.16	65.88	72.87	70.19	68.44	74.20	73.00	69.78
InfiniCL	65.13	64.00	56.88	70.54	65.80	63.78	72.37	72.0	67.74	75.13	73.80	71.82	77.12	77.60	72.50

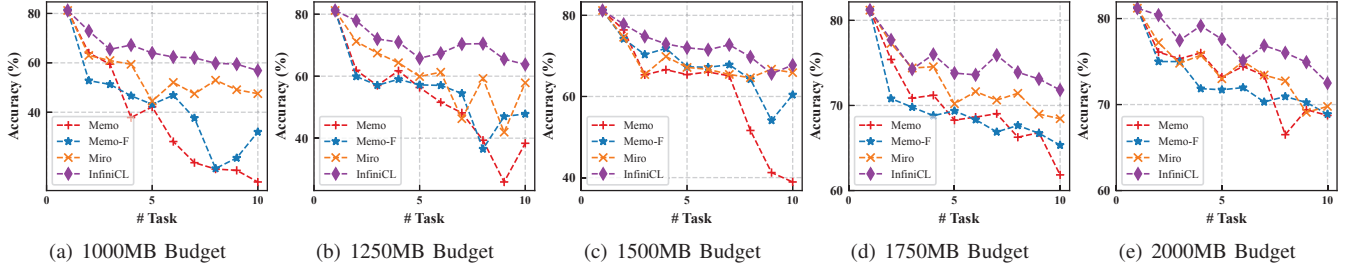


Fig. 7. Accuracy performance on ImageNet-100 with dynamic memory constraints. InfiniCL outperforms the baselines in various memory constraint, highlighting its effectiveness in boosting model performance while maintaining memory efficiency.

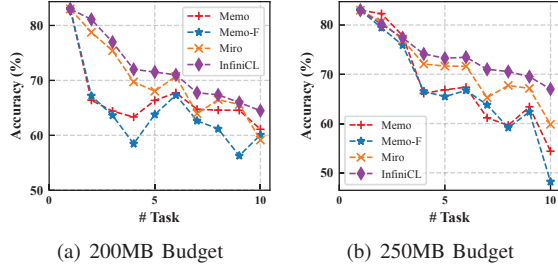


Fig. 8. Accuracy performance on CIFAR-100 with dynamic memory constraints. InfiniCL outperforms the baselines with a 2.34-3.61% increase in average accuracy and 5.34-7.15% increase in final accuracy, highlighting its effectiveness in boosting model performance while maintaining memory efficiency.

the CL performance. We conduct experiments on ImageNet-100 under a memory budget of 1000MB. InfiniCL distills the model after 6 tasks based on the least training ratio, 40%. Without freezing, InfiniCL with distill-only distills the model after every 3 tasks. Figure 10 compares the accuracy variations of the two methods during the CL process. InfiniCL with distill-only performs poorly because it neglects the allocation of data and model during training, especially when the runtime overhead of expanding the model approaches the budget (Task 3, Task 6, Task 9). This demonstrates that although the expand-and-distill approach can address the rapid growth of the model, it fails to maintain better CL performance.

Design of Bayesian Optimization. To evaluate the effectiveness of BO, we use the random sampling method proposed in Miro within InfiniCL, replacing the core component of the plan generator from BO with random sampling. We conduct the experiments on Task 4 in CIFAR-100, testing the performance of different methods under varying time budgets with 5 random seeds. As shown in Fig.11, the issue with random sampling is that when the profiling space is large and the

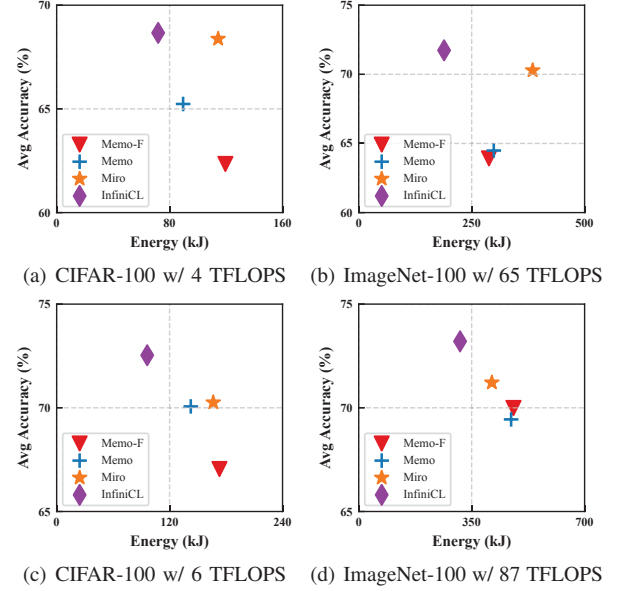


Fig. 9. Energy-accuracy trade-off on CIFAR-100 and ImageNet-100. The memory budget is 400MB for CIFAR-100 and 2000MB for ImageNet-100. InfiniCL achieves better cost-efficiency than other baselines.

search time for sampling is limited, it is difficult to obtain a good solution. InfiniCL, on the other hand, uses a BO with a surrogate model to iteratively explore potential optimal values based on the profiling results, leading to better solutions over time.

E. Ablation Study

In this subsection, we evaluate the contributions of the different modules within InfiniCL to the results. We conduct an ablation study on ImageNet-100 with memory budgets 1250MB and 1500MB and evaluate the overall accuracy performance and accuracy on the last task. InfiniCL without

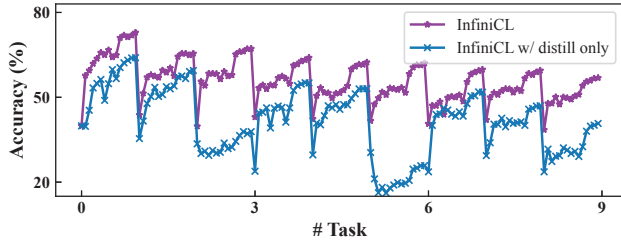


Fig. 10. Effectiveness of Elastic Continual Learning

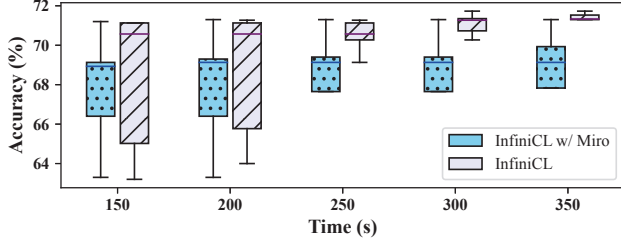


Fig. 11. Effectiveness of Bayesian Optimization

ECL adds BO to the original replay-and-expand method to optimize the allocation of the SB and EM sizes. In contrast, InfiniCL without BO adopts a static memory allocation strategy, allocating 50% of the memory to data and 50% to the model on ImageNet-100. Since the BO module cannot reduce the overhead brought by model expanding, its performance is not as effective as the static ECL. Our method first reduces the overhead using ECL and then optimizes the configuration of SB, EM, and layer freezing strategies through BO, allowing for better adaptation to different scenarios.

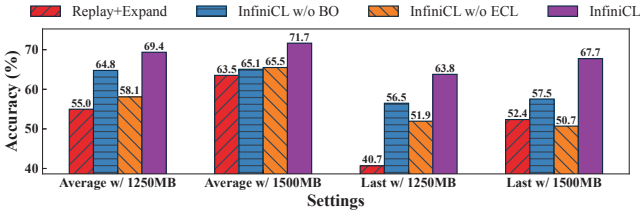


Fig. 12. Ablation study on ImageNet-100 under different memory budgets.

F. Overhead analysis

We analyze the overhead of various components in InfiniCL, including both the layer selector and the BO-based plan generator. The overhead of the layer selector mainly consists of the FIM profiling cost and the execution time of the DP algorithm. For the BO-based plan generator, the overhead is primarily due to the profiling time of the iteratively found candidates. We present an analysis of the average time spent on these additional overheads for tasks under the CL setting on the device in Table II.

TABLE II
OVERHEAD ANALYSIS

CL Setting	Layer Selection FIM Prof.	DP	BO	Train
ResNet32 on CIFAR-100	0.51%	1.41%	9.54%	88.53%
ResNet18 on ImageNet-100	0.35%	0.01%	5.87%	93.75%

VIII. RELATED WORK AND DISCUSSION

On-device continual learning systems. On-device continual learning system designs the CL methods under restricted memory and computation constraints. BudgetCL [32] analyzes various continual learning under a given memory and computation budget. SIESTA [33] designs a wake-sleep system, that involves performing lightweight and backpropagation-free training during the wake phase while conducting offline training in the sleep phase. SparCL [34] improves the efficiency by leveraging task-aware dynamic masking, dynamic data removal, and dynamic gradient masking into continual learning system. Our system is the first to implement model-expand methods for on-device continual learning.

Model-expand continual learning. Model-expand strategy dynamically expands the model architecture to enhance the representation ability of the model. Various studies focus on model-expand methods for both CNNs [8]–[10] and Transformers [35], [36]. In this paper, we have not put much effort into designing model expanding and distillation methods and have just used them straightforwardly. We position our work as a fundamental system for supporting various expanding and distillation methods in continual learning study.

Freezing-based on-device learning system. On-device training systems aim to train the deep learning model locally. ElasticTrainer [37] sparsifies the trainable tensor at training runtime to speed up the training process on devices. TinyEngine [22], [38] implements sparse backpropagation to reduce the memory usage of intermediate representations in backpropagation, thus the deep learning training can be deployed on weak devices. Both of the above works adopt a finer granularity (tensor-level) than ours (layer-level) and can achieve better results in theory. However, utilizing this fine-grained freezing strategy to really reduce memory costs needs special framework support.

IX. CONCLUSION

In this work, we design and implement InfiniCL, the first on-device continual learning system enabling lifelong learning for resource-constrained edge devices. By introducing elastic continual learning and continual balancing, InfiniCL achieves sustainable model expending while smoothing the incremental memory cost. Our extensive experiments show that InfiniCL achieves better continual learning QoS compared with SOTA works.

ACKNOWLEDGMENT

This work is supported by Jiangsu Provincial Frontier Technology Research and Development Program under Grant BF2024070, National Natural Science Foundation of China under Grants, No. 62232004, Shenzhen Science and Technology Program under Grant KJZD20240903100814018, Jiangsu Provincial Key Laboratory of Network and Information Security under Grants No.BM2003201, Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under Grants No.93K-9, and partially supported by Collaborative Innovation Center of Novel Software

Technology and Industrialization. We also thank the Big Data Computing Center of Southeast University for providing the experiment environment and computing facility.

REFERENCES

- [1] E. Verwimp, R. Aljundi, S. Ben-David, M. Bethge, A. Cossu, A. Gepperth, T. L. Hayes, E. Hüllermeier, C. Kanan, D. Kudithipudi, C. H. Lampert, M. Mundt, R. Pascanu, A. Popescu, A. S. Tolias, J. van de Weijer, B. Liu, V. Lomonaco, T. Tuytelaars, and G. M. van de Ven, “Continual learning: Applications and the road forward,” *Transactions on Machine Learning Research*, 2024. [Online]. Available: <https://openreview.net/forum?id=axBIMcGZn9>
- [2] T. L. Hayes and C. Kanan, “Online continual learning for embedded devices,” in *Proceedings of The 1st Conference on Lifelong Learning Agents*, ser. Proceedings of Machine Learning Research, S. Chandar, R. Pascanu, and D. Precup, Eds., vol. 199. PMLR, 22–24 Aug 2022, pp. 744–766. [Online]. Available: <https://proceedings.mlr.press/v199/hayes22a.html>
- [3] U. Michieli and M. Ozy, “Online continual learning for robust indoor object recognition,” in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 3849–3856.
- [4] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, “icarl: Incremental classifier and representation learning,” in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 2001–2010.
- [5] D. Rolnick, A. Ahuja, J. Schwarz, T. Lillicrap, and G. Wayne, “Experience replay for continual learning,” *Advances in neural information processing systems*, vol. 32, 2019.
- [6] Y. Wu, Y. Chen, L. Wang, Y. Ye, Z. Liu, Y. Guo, and Y. Fu, “Large scale incremental learning,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 374–382.
- [7] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, “Progressive neural networks,” *arXiv preprint arXiv:1606.04671*, 2016.
- [8] S. Yan, J. Xie, and X. He, “Der: Dynamically expandable representation for class incremental learning,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 3014–3023.
- [9] D.-W. Zhou, Q.-W. Wang, H.-J. Ye, and D.-C. Zhan, “A model or 603 exemplars: Towards memory-efficient class-incremental learning,” in *ICLR*, 2023.
- [10] T.-Y. Wu, G. Swaminathan, Z. Li, A. Ravichandran, N. Vasconcelos, R. Bhotika, and S. Soatto, “Class-incremental learning with strong pre-trained models,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 9601–9610.
- [11] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [12] Y. D. Kwon, J. Chauhan, H. Jia, S. I. Venieris, and C. Mascolo, “Life-learner: Hardware-aware meta continual learning system for embedded computing platforms,” in *Proceedings of the 21st ACM Conference on Embedded Networked Sensor Systems*, 2023, pp. 138–151.
- [13] Y. Wu, L. Li, C. Tian, T. Chang, C. Lin, C. Wang, and C.-Z. Xu, “Heterogeneity-aware memory efficient federated learning via progressive layer freezing,” in *2024 IEEE/ACM 32nd International Symposium on Quality of Service (IWQoS)*. IEEE, 2024, pp. 1–10.
- [14] S. Jiang, L. Ran, T. Cao, Y. Xu, and Y. Liu, “Profiling and optimizing deep learning inference on mobile gpus,” in *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems*, 2020, pp. 75–81.
- [15] X. Ma, S. Jeong, M. Zhang, D. Wang, J. Choi, and M. Jeon, “Cost-effective on-device continual learning over memory hierarchy with miro,” in *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, 2023, pp. 1–15.
- [16] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, “Taking the human out of the loop: A review of bayesian optimization,” *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.
- [17] Y. Zeng, R. Zhou, L. Jiao, Z. Han, J. Yu, and Y. Ma, “Efficient online dnn inference with continuous learning in edge computing,” in *2024 IEEE/ACM 32nd International Symposium on Quality of Service (IWQoS)*. IEEE, 2024, pp. 1–10.
- [18] T. Li, Q. Li, M. Zhang, Z. Yuan, and Y. Jiang, “Adaptive streaming continuous learning system for video analytics,” in *2024 IEEE/ACM 32nd International Symposium on Quality of Service (IWQoS)*. IEEE, 2024, pp. 1–10.
- [19] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska et al., “Overcoming catastrophic forgetting in neural networks,” *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.
- [20] S. Dohare, J. F. Hernandez-Garcia, Q. Lan, P. Rahman, A. R. Mahmood, and R. S. Sutton, “Loss of plasticity in deep continual learning,” *Nature*, vol. 632, no. 8026, pp. 768–774, 2024.
- [21] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, “Mcnety2: Memory-efficient patch-based inference for tiny deep learning,” in *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.
- [22] L. Zhu, L. Hu, J. Lin, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, “Pockengine: Sparse and efficient fine-tuning in a pocket,” in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 1381–1394.
- [23] H. Yin, T. Chen, L. Qu, and B. Cui, “On-device recommender systems: A tutorial on the new-generation recommendation paradigm,” in *Companion Proceedings of the ACM on Web Conference 2024*, 2024, pp. 1280–1283.
- [24] J. S. Filho, J. Xu, M. Kanchwala, G. O’Leary, J. Zariffa, and R. Genov, “Design methodology for energy-constrained ai edge inference in implantable medical devices,” in *2023 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 2023, pp. 1–5.
- [25] S. Lee, M. Weerakoon, J. Choi, M. Zhang, D. Wang, and M. Jeon, “Carm: Hierarchical episodic memory for continual learning,” in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1147–1152.
- [26] Y. Wu, L. Li, C. Tian, T. Chang, C. Lin, C. Wang, and C.-Z. Xu, “Heterogeneity-aware memory efficient federated learning via progressive layer freezing,” in *2024 IEEE/ACM 32nd International Symposium on Quality of Service (IWQoS)*. IEEE, 2024, pp. 1–10.
- [27] M. Seo, H. Koh, and J. Choi, “Budgeted online continual learning by adaptive layer freezing and frequency-based sampling,” 2024.
- [28] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska et al., “Overcoming catastrophic forgetting in neural networks,” *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.
- [29] “Pytorch deep learning framework,” <https://pytorch.org/>, 2016, accessed: 2024-03-13.
- [30] A. Krizhevsky, G. Hinton et al., “Learning multiple layers of features from tiny images,” 2009.
- [31] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.
- [32] A. Prabhu, H. A. Al Kader Hammoud, P. K. Dokania, P. H. Torr, S.-N. Lim, B. Ghanem, and A. Bibi, “Computationally budgeted continual learning: What does matter?” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 3698–3707.
- [33] M. Y. Harun, J. Gallardo, T. L. Hayes, R. Kemker, and C. Kanan, “Siesta: Efficient online continual learning with sleep,” *arXiv preprint arXiv:2303.10725*, 2023.
- [34] Z. Wang, Z. Zhan, Y. Gong, G. Yuan, W. Niu, T. Jian, B. Ren, S. Ioannidis, Y. Wang, and J. Dy, “Sparcl: Sparse continual learning on the edge,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 20 366–20 380, 2022.
- [35] A. Douillard, A. Ramé, G. Couairon, and M. Cord, “Dytox: Transformers for continual learning with dynamic token expansion,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 9285–9295.
- [36] D.-W. Zhou, H.-L. Sun, H.-J. Ye, and D.-C. Zhan, “Expandable subspace ensemble for pre-trained model-based class-incremental learning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 23 554–23 564.
- [37] K. Huang, B. Yang, and W. Gao, “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*, 2023, pp. 56–69.
- [38] H. Cai, C. Gan, L. Zhu, and S. Han, “Tinytl: Reduce activations, not trainable parameters for efficient on-device learning,” *arXiv preprint arXiv:2007.11622*, 2020.