SparCL: Sparse Continual Learning on the Edge

Zifeng Wang[1,*], **Zheng Zhan**[1,*], **Yifan Gong**[1], **Geng Yuan**[1], **Wei Niu**[2], **Tong Jian**[1], **Bin Ren**[2], **Stratis Ioannidis**[1], **Yanzhi Wang**[1], **Jennifer Dy**[1]
[1] Northeastern University, [2] College of William and Mary
{zhan.zhe, gong.yifa, geng.yuan, yanz.wang}@northeastern.edu,
{zifengwang, jian, ioannidis, jdy}@ece.neu.edu,
wniu@email.wm.edu, bren@cs.wm.edu

Abstract

Existing work in continual learning (CL) focuses on mitigating catastrophic forgetting, *i.e.*, model performance deterioration on past tasks when learning a new task. However, the training efficiency of a CL system is under-investigated, which limits the real-world application of CL systems under resource-limited scenarios. In this work, we propose a novel framework called Sparse Continual Learning (SparCL), which is the first study that leverages sparsity to enable cost-effective continual learning on edge devices. SparCL achieves both training acceleration and accuracy preservation through the synergy of three aspects: *weight sparsity*, *data efficiency*, and *gradient sparsity*. Specifically, we propose task-aware dynamic masking (TDM) to learn a sparse network throughout the entire CL process, dynamic data removal (DDR) to remove less informative training data, and dynamic gradient masking (DGM) to sparsify the gradient updates. Each of them not only improves efficiency, but also further mitigates catastrophic forgetting. SparCL consistently improves the training efficiency of existing state-of-the-art (SOTA) CL methods by at most $23\times$ less training FLOPs, and, surprisingly, further improves the SOTA accuracy by at most 1.7%. SparCL also outperforms competitive baselines obtained from adapting SOTA sparse training methods to the CL setting in both efficiency and accuracy. We also evaluate the effectiveness of SparCL on a real mobile phone, further indicating the practical potential of our method.

1 Introduction

The objective of Continual Learning (CL) is to enable an intelligent system to accumulate knowledge from a sequence of tasks, such that it exhibits satisfying performance on both old and new tasks [32]. Recent methods mostly focus on addressing the *catastrophic forgetting* [43] problem – learning model tends to suffer performance deterioration on previously seen tasks. However, in the real world, when CL applications are deployed in resource-limited platforms [48] such as edge devices, the learning efficiency, w.r.t. both training speed and memory footprint, are also crucial metrics of interest, yet they are rarely explored in prior CL works.

Existing CL methods can be categorized into regularization-based [2, 32, 37, 68], rehearsal-based [8, 12, 50, 61], and architecture-based [31, 42, 52, 58, 59, 70]. Both regularization- and rehearsal-based methods directly train a dense model, which might be over-parameterized even for the union of all tasks [19, 39]. Though several architecture-based methods [51, 57, 64] start with a sparse sub-network from the dense model, they still grow the model size progressively to learn emerging tasks. The aforementioned methods, although striving for greater performance with less forgetting, still introduce significant memory and computation overhead during the whole CL process.

[*]Both authors contributed equally to this work

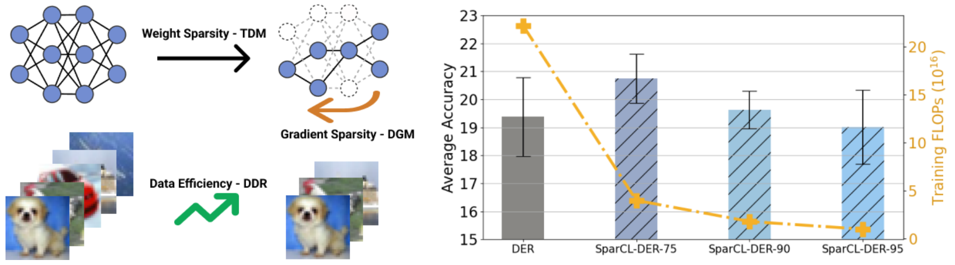

Figure 1: **Left:** Overview of SparCL. SparCL consists of three complementary components: task-aware dynamic masking (TDM) for weight sparsity, dynamic data removal (DDR) for data efficiency, and dynamic gradient masking (DGM) for gradient sparsity. **Right:** SparCL successfully preserves the accuracy and significantly improves efficiency over DER++ [8], one of the SOTA CL methods, with different sparsity ratios on the Split Tiny-ImageNet [16] dataset.

Recently, another stream of work, sparse training [4, 20, 35], has emerged as a new training trend to achieve training acceleration, which embraces the promising training-on-the-edge paradigm. With sparse training, each iteration takes less time with the reduction in computation achieved by sparsity. Inspired by these sparse training methods, under the traditional i.i.d. learning setting, we naturally think about introducing sparse training to the field of CL. A straightforward idea is to directly combine existing sparse training methods, such as SNIP [35], RigL [20], with a rehearsal buffer under the CL setting. However, these methods fail to consider key challenges in CL to mitigate catastrophic forgetting, for example, properly handling transition between tasks. As a result, these sparse training methods, though enhancing training efficiency, cause significant accuracy drop (see Section 5.2). Thus, we would like to explore a general strategy, orthogonal to existing CL methods, that not only leverages the idea of sparse training for efficiency, but also addresses key challenges in CL to preserve (or even improve) accuracy.

In this work, we propose *Sparse Continual Learning* (SparCL), a general framework for cost-effective continual learning, aiming at enabling practical CL on edge devices. As shown in Figure 1 (left), SparCL achieves both learning acceleration and accuracy preservation through the synergy of three aspects: *weight sparsity*, *data efficiency*, and *gradient sparsity*. Specifically, to maintain a small dynamic sparse network during the whole CL process, we develop a novel task-aware dynamic masking (TDM) strategy to keep only important weights for both the current and past tasks, with special consideration during task transitions. Moreover, we propose a dynamic data removal (DDR) scheme, which progressively removes "easy-to-learn" examples from training iterations, which further accelerates the training process and also improves accuracy of CL by balancing current and past data and keeping more informative samples in the buffer. Finally, we provide an additional dynamic gradient masking (DGM) strategy to leverage gradient sparsity for even better efficiency and knowledge preservation of learned tasks, such that only a subset of sparse weights are updated. Figure 1 (right) demonstrates that SparCL successfully preserves the accuracy and significantly improves efficiency over DER++ [8], one of the SOTA CL methods, under different sparsity ratios.

SparCL is simple in concept, compatible with various existing rehearsal-based CL methods, and efficient under practical scenarios. We conduct comprehensive experiments on multiple CL benchmarks to evaluate the effectiveness of our method. We show that SparCL works collaboratively with existing CL methods, greatly accelerates the learning process under different sparsity ratios, and even sometimes improves upon the state-of-the-art accuracy. We also establish competing baselines by combining representative sparse training methods with advanced rehearsal-based CL methods. SparCL again outperforms these baselines in terms of both efficiency and accuracy. Most importantly, we evaluate our SparCL framework on real edge devices to demonstrate the practical potential of our method. We are not aware of any prior CL works that explored this area and considered the constraints of limited resources during training.

In summary, our work makes the following contributions:

- We propose *Sparse Continual Learning* (SparCL), a general framework for cost-effective continual learning, which achieves learning acceleration through the synergy of *weight sparsity*, *data effi-*

ciency, and *gradient sparsity*. To the best of our knowledge, our work is the first to introduce the idea of sparse training to enable efficient CL on edge devices. Our code is publicly available[*].

- SparCL shows superior performance compared to both conventional CL methods and CL-adapted sparse training methods on all benchmark datasets, leading to at most $23\times$ less training FLOPs and, surprisingly, an 1.7% improvement over SOTA accuracy.
- We evaluate SparCL on a real mobile edge device, demonstrating the practical potential of our method and also encouraging future research on CL on-the-edge. The results indicate that our framework can achieve at most $3.1\times$ training acceleration.

2 Related work

2.1 Continual Learning

The main focus in continual learning (CL) has been mitigating catastrophic forgetting. Existing methods can be classified into three major categories. *Regularization-based* methods [2, 32, 37, 68] limit updates of important parameters for the prior tasks by adding corresponding regularization terms. While these methods reduce catastrophic forgetting to some extent, their performance deteriorates under challenging settings [40], and on more complex benchmarks [50, 61]. *Rehearsal-based* methods [13, 14, 25] save examples from previous tasks into a small-sized buffer to train the model jointly with the current task. Though simple in concept, the idea of rehearsal is very effective in practice and has been adopted by many state-of-the-art methods [8, 11, 49]. *Architecture-based* methods [42, 51, 57, 59, 63] isolate existing model parameters or assign additional parameters for each task to reduce interference among tasks. As mentioned in Section 1, most of these methods use a dense model without consideration of efficiency and memory footprint, thus are not applicable to resource-limited settings. Our work, orthogonal to these methods, serves as a general framework for making these existing methods efficient and enabling a broader deployment, *e.g.*, CL on edge devices.

A limited number of works explore sparsity in CL, however, for different purposes. Several methods [41, 42, 53, 57] incorporate the idea of weight pruning [24] to allocate a sparse sub-network for each task to reduce inter-task interference. Nevertheless, these methods still reduce the full model sparsity progressively for every task and finally end up with a much denser model. On the contrary, SparCL maintains a sparse network throughout the whole CL process, introducing great efficiency and memory benefits both during training and at the output model. A recent work [15] aims at discovering lottery tickets [21] under CL, but still does not address efficiency. However, the existence of lottery tickets in CL serves as a strong justification for the outstanding performance of our SparCL.

2.2 Sparse Training

There are two main approaches for sparse training: fixed-mask sparse training and dynamic sparse training. Fixed-mask sparse training methods [35, 54, 56, 60] first apply pruning, then execute traditional training on the sparse model with the obtained fixed mask. The pre-fixed structure limits the accuracy performance, and the first stage still causes huge computation and memory consumption. To overcome these drawbacks, dynamic mask methods [4, 17, 20, 45, 46] adjust the sparsity topology during training while maintaining low memory footprint. These methods start with a sparse model structure from an untrained dense model, then combine sparse topology exploration at the given sparsity ratio with the sparse model training. Recent work [67] further considers incorporating data efficiency into sparse training for better training accelerations. However, all prior sparse training works are focused on the traditional training setting, while CL is a more complicated and difficult scenario with inherent characteristics not explored by these works. In contrast to prior sparse training methods, our work explores a new learning paradigm that introduces sparse training into CL for efficiency and also addresses key challenges in CL, mitigating catastrophic forgetting.

3 Continual Learning Problem Setup

In supervised CL, a model f_θ learns from a sequence of tasks $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_T\}$, where each task $\mathcal{D}_t = \{(\boldsymbol{x}_{t,i}, y_{t,i})\}_{i=1}^{n_t}$ consists of input-label pairs, and each task has a disjoint set of classes. Tasks

[*] https://github.com/neu-spiral/SparCL

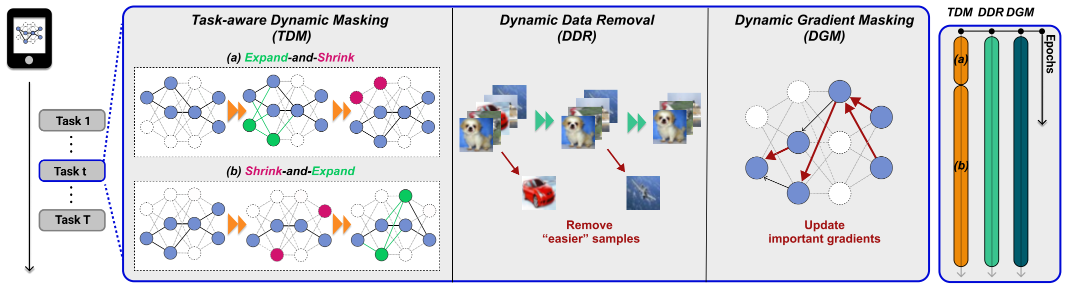
Task 1
Task t
Task T
Task-aware Dynamic Masking (TDM)
(a) Expand-and-Shrink
(b) Shrink-and-Expand
Dynamic Data Removal (DDR)
Remove "easier" samples
Dynamic Gradient Masking (DGM)
Update important gradients
TDM DDR DGM
Epochs
(a)
(b)

Figure 2: Illustration of the SparCL workflow. Three components work synergistically to improve training efficiency and further mitigate catastrophic forgetting for preserving accuracy.

arrive sequentially, and the model must adapt to them. At the t-th step, the model gains access to data from the t-th task. However, a small fix-sized rehearsal buffer $\mathcal{M}$ is allowed to save data from prior tasks. At test time, the easiest setting is to assume task identity is known for each coming test example, named task-incremental learning (Task-IL). If this assumption does not hold, we have the more difficult class-incremental learning (Class-IL) setting. In this work, we mainly focus on the more challenging Class-IL setting, and only report Task-IL performance for reference.

The goal of conventional CL is to train a model sequentially that performs well on all tasks at test time. The main evaluation metric is average test accuracy on all tasks. In real-world resource-limited scenarios, we should further consider *training efficiency* of the model. Thus, we measure the performance of the model more comprehensively by including training FLOPs and memory footprint.

4 Sparse Continual Learning (SparCL)

Our method, Sparse Continual Learning, is a unified framework composed of three complementary components: *task-aware dynamic masking* for weight sparsity, *dynamic data removal* for data efficiency, and *dynamic gradient masking* for gradient sparsity. The entire framework is shown in Figure 2. We will illustrate each component in detail in this section.

4.1 Task-aware Dynamic Masking

To enable cost-effective CL in resource limited scenarios, SparCL is designed to maintain a dynamic structure when learning a sequence of tasks, such that it not only achieves high efficiency, but also intelligently adapts to the data stream for better performance. Specifically, we propose a strategy named *task-aware dynamic masking* (TDM), which dynamically removes less important weights and grows back unused weights for stronger representation power periodically by maintaining a single binary weight mask throughout the CL process. Different from typical sparse training work, which only leverages the weight magnitude [45] or the gradient w.r.t. data from a single training task [20, 67], TDM considers also the importance of weights w.r.t. data saved in the rehearsal buffer, as well as the switch between CL tasks.

Specifically, TDM strategy starts from a randomly initialized binary mask $M_\theta = M_0$, with a given sparsity constraint $\|M_\theta\|_0/\|\theta\|_0 = 1 - s$, where $s \in [0, 1]$ is the sparsity ratio. Moreover, it makes different intra- and inter-task adjustments to keep a dynamic sparse set of weights based on their continual weight importance (CWI). We summarize the process of task-aware dynamic masking in Algorithm 1 and elaborate its key components in detail below.

Continual weight importance (CWI). For a model f_θ parameterized by θ, the CWI of weight $w \subset \theta$ is defined as follows:

$$\mathrm{CWI}(w) = |w| + \alpha|\frac{\partial\tilde{\mathcal{L}}(\mathcal{D}_t;\theta)}{\partial w}| + \beta|\frac{\partial\mathcal{L}(\mathcal{M};\theta)}{\partial w}|, \quad (1)$$

where $\mathcal{D}_t$ denotes the training data from the t-th task, $\mathcal{M}$ is the current rehearsal buffer, and α, β are coefficients to control the influence of current and buffered data, respectively. Moreover, $\mathcal{L}$ represents the cross-entropy loss for classification, while $\tilde{\mathcal{L}}$ is the *single-head* [1] version of the cross-entropy loss, which only considers classes from the current task by masking out the logits of other classes.

Algorithm 1: Task-aware Dynamic Masking (TDM)

```
Input: Model weight $\theta$, number of tasks $T$, training epochs of the $t$-th task $K_t$, binary sparse
  mask $M_\theta$, sparsity ratio $s$, intra-task adjustment ratio $p_{\texttt{intra}}$, inter-task adjustment ratio
  $p_{\texttt{inter}}$, update interval $\delta k$
Initialize: $\theta$, $M_\theta$, s.t. $\|M_\theta\|_0/\|\theta\|_0 = 1 - s$
for $t = 1, \ldots, T$ do
    for $e = 1, \ldots, K_T$ do
        if $t > 1$ then
            /* Inter-task adjustment */
            Expand $M_\theta$ by randomly adding unused weights,
              s.t. $\|M_\theta\|_0/\|\theta\|_0 = 1 - (s - p_{\texttt{inter}})$
            if $e = \delta k$ then
                Shrink $M_\theta$ by removing the least important weights according to equation 1,
                  s.t. $\|M_\theta\|_0/\|\theta\|_0 = 1 - s$
            end
        end
        if $e \mod \delta k = 0$ then
            /* Intra-task adjustment */
            Shrink $M_\theta$ by removing the least important weights according to equation 1,
              s.t. $\|M_\theta\|_0/\|\theta\|_0 = 1 - (s + p_{\texttt{intra}})$
            Expand $M_\theta$ by randomly adding unused weights,
              s.t. $\|M_\theta\|_0/\|\theta\|_0 = 1 - s$
        end
        Update $\theta \odot M_\theta$ via backpropagation
    end
end
```

Intuitively, CWI ensures we keep (1) weights of larger magnitude for output stability, (2) weights important for the current task for learning capacity, and (3) weights important for past data to mitigate catastrophic forgetting. Moreover, inspired by the classification bias in CL [1], we use the single-head cross-entropy loss when calculating the importance score w.r.t. the current task to make the importance estimation more accurate.

Intra-task adjustment. When training the t-th task, a natural assumption is that the data distribution is consistent inside the task. As a result, we would like to update the sparse model in a relatively stable way while keeping its flexibility. Thus, in Algorithm 1, we choose to update the sparsity mask M_θ in a *shrink-and-expand* way every δk epochs. We first remove $p_{\texttt{intra}}$ of the weights of least CWI to retain learned knowledge so far. Then we randomly select unused weights to recover the learning capacity for the model and keep the sparsity ratio s unchanged.

Inter-task adjustment. When tasks switch, on the contrary, we assume that the data distribution shifts immediately. Ideally, we would like the model to keep the knowledge learned from old tasks as much as possible, and to have enough learning capacity to accommodate the new task. Thus, instead of the shrink-and-expand strategy for intra-task adjustment, we follow an *expand-and-shrink* scheme. Specifically, at the beginning of the $(t+1)$-th task, we expand the sparse model by randomly adding a proportion of $p_{\texttt{inter}}$ unused weights. Intuitively, the additional learning capacity facilitates fast adoption of new knowledge and reduces interference with learned knowledge. We allow our model to have smaller sparsity (*i.e.*, larger learning capacity) temporarily for the first δk epochs as a warm-up period, and then remove the $p_{\texttt{inter}}$ weights with least CWI, following the same process in the intra-task case, to satisfy the sparsity constraint.

4.2 Dynamic Data Removal

In addition to weight sparsity, decreasing the amount of training data can be directly translated into training time savings. Thus, we would also like to explore data efficiency to reduce the training workload. Some prior CL works select informative examples to construct the rehearsal buffer [3, 6, 65]. However, their main purpose is not training acceleration; thus, they either introduce excessive computational cost or consider different problem settings. By considering the features of CL, we present a simple yet effective strategy, *dynamic data removal* (DDR), to reduce training data for further acceleration.

We measure the importance of each training example by the occurrence of misclassification [55, 67] during CL. In TDM, the sparse structure of our model updates periodically every δk epochs, so we align our data removal process with weight mask updates for further efficiency and training stability. In Section 4.1, we have partitioned the training process for the t-th task into $N_t = K_t/\delta k$ stages based on the dynamic mask update. Therefore, we gradually remove training data at the end of i-th stage, based on the following policy: 1) Calculate the total number of misclassifications $f_i(x_j)$ for each training example during the i-th stage. 2) Remove a proportion of ρ_i training samples with the least number of misclassifications. Although our main purpose is to keep the "harder" examples to learn to consolidate the sparse model, we can get further benefits for better CL results. First, the removal of "easier" examples increases the probability that "harder" examples to be saved to the rehearsal buffer, given the common strategy, *e.g.* reservoir sampling [14], to buffer examples. Thus, we construct a more informative buffer in a implicit way without heavy computation. Moreover, since the buffer size is much smaller than the training set size of each task, the data from the buffer and the new task is highly imbalanced; dynamic data removal also relieves this data imbalance issue.

Formally, we set the data removal proportion for each task as $\rho \in [0, 1]$, and a cutoff stage, such that:

$$\sum_{i=1}^{\texttt{cutoff}} \rho_i = \rho, \qquad \sum_{i=\texttt{cutoff}+1}^{N_k} \rho_i = 0 \tag{2}$$

The cutoff stage controls the trade-off between efficiency and accuracy: when we set the cutoff stage earlier, we reduce the training time for all the following stages; however, when the cutoff stage is set too early, the model might underfit the removed training data. Note that when we set $\rho_i = 0$ for all $i = 1, 2, \ldots, N_t$ and $\texttt{cutoff} = N_t$, we simply recover the vanilla setting without any data efficiency considerations. In our experiments, we assume $\rho_i = \rho/\texttt{cutoff}$, i.e., removing equal proportion of data at the end of every stage, for simplicity. We also conduct a comprehensive exploration study of ρ and the selection of the cutoff stage in Section 5.3 and Appendix B.3.

4.3 Dynamic Gradient Masking

With TDM and DDR, we can already achieve weight efficiency and data efficiency during training. To further boost training efficiency, we explore gradient sparsity and propose dynamic gradient masking (DGM) for CL. Our method focuses on reducing computational costs by only applying the most important gradients onto the corresponding unpruned model parameters via a gradient mask. The gradient mask is also dynamically updated along with the weight mask defined in Section 4.1. Intuitively, while targeting for better training efficiency, DGM also promotes the preservation of past knowledge by preventing a fraction of weights from updating.

Formally, our goal here is to find a subset of unpruned parameters (or, equivalently, a gradient mask M_G) to update over multiple training iterations. For a model f_θ parameterized by θ, we have the corresponding gradient matrix G calculated during each iteration. To prevent the pruned weights from updating, the weight mask M_θ will be applied onto the gradient matrix G as $G \odot M_\theta$ during backpropagation. Besides the gradients of pruned weights, we in addition consider to remove less important gradient coefficients for faster training. To achieve this, we introduce the continual gradient importance (CGI) based on the CWI to measure the importance of weight gradients:

$$\text{CGI}(w) = \alpha \left| \frac{\partial \tilde{\mathcal{L}}(\mathcal{D}_t; \theta)}{\partial w} \right| + \beta \left| \frac{\partial \mathcal{L}(\mathcal{M}; \theta)}{\partial w} \right|. \tag{3}$$

We remove a proportion q of non-zero gradients from G with less importance measured by CGI and we have $\|M_G\|_0/\|\theta\|_0 = 1 - (s + q)$. The gradient mask M_G is then applied onto the gradient matrix G. During the entire training process, the gradient mask M_G is updated with a fixed interval.

5 Experiment

5.1 Experiment Setting

Datasets. We evaluate our SparCL on two representative CL benchmarks, Split CIFAR-10 [33] and Split Tiny-ImageNet [16] to verify the efficacy of SparCL. In particular, we follow [8, 68] by

Table 1: Comparison with CL methods. SparCL consistently improves training efficiency of the corresponding CL methods while preserves (or even improves) accuracy on both class- and task-incremental settings.

Method	**Sparsity**	**Buffer size**	**Split CIFAR-10**			**Split Tiny-ImageNet**		
			Class-IL (↑)	Task-IL (↑)	FLOPs Train $\times 10^{15}$ (↓)	Class-IL (↑)	Task-IL (↑)	FLOPs Train $\times 10^{16}$ (↓)
EWC [32]	0.00	–	19.49±0.12	68.29±3.92	8.3	7.58±0.10	19.20±0.31	13.3
LwF [37]			19.61±0.05	63.29±2.35	8.3	8.46±0.22	15.85±0.58	13.3
PackNet [42]	0.50†	–	-	93.73±0.55	5.0	–	61.88±1.01	7.3
LPS [57]			-	94.50±0.47	5.0	–	63.37±0.83	7.3
A-GEM [13]	0.00	200	20.04±0.34	83.88±1.49	11.1	8.07±0.08	22.77±0.03	17.8
iCaRL [50]			49.02±3.20	88.99±2.13	11.1	7.53±0.79	28.19±1.47	17.8
FDR [5]			30.91±2.74	91.01±0.68	13.9	8.70±0.19	40.36±0.68	22.2
ER [14]			44.79±1.86	91.19±0.94	11.1	8.49±0.16	38.17±2.00	17.8
DER++ [8]			64.88±1.17	91.92±0.60	13.9	10.96±1.17	40.87±1.16	22.2
SparCL-ER$_{75}$	0.75	200	46.89±0.68	92.02±0.72	2.0	8.98±0.38	39.14±0.85	3.2
SparCL-DER++$_{75}$			66.30±0.98	94.06±0.45	2.5	12.73±0.40	42.06±0.73	4.0
SparCL-ER$_{90}$	0.90		45.81±1.05	91.49±0.47	0.9	8.67±0.41	38.79±0.39	1.4
SparCL-DER++$_{90}$			65.79±1.33	93.73±0.24	1.1	12.27±1.06	41.17±1.31	1.8
SparCL-ER$_{95}$	0.95		44.59±0.23	91.07±0.64	0.5	8.43±0.09	38.20±0.46	0.8
SparCL-DER++$_{95}$			65.18±1.25	92.97±0.37	0.6	10.76±0.62	40.54±0.98	1.0
A-GEM [13]	0.00	500	22.67±0.57	89.48±1.45	11.1	8.06±0.04	25.33±0.49	17.8
iCaRL [50]			47.55±3.95	88.22±2.62	11.1	9.38±1.53	31.55±3.27	17.8
FDR [5]			28.71±3.23	93.29±0.59	13.9	10.54±0.21	49.88±0.71	22.2
ER [14]			57.74±0.27	93.61±0.27	11.1	9.99±0.29	48.64±0.46	17.8
DER++ [8]			72.70±1.36	93.88±0.50	13.9	19.38±1.41	51.91±0.68	22.2
SparCL-ER$_{75}$	0.75	500	60.80±0.22	93.82±0.32	2.0	10.48±0.29	50.83±0.69	3.2
SparCL-DER++$_{75}$			74.09±0.84	95.19±0.34	2.5	20.75±0.88	52.19±0.43	4.0
SparCL-ER$_{90}$	0.90		59.34±0.97	93.33±0.10	0.9	10.12±0.53	49.46±1.22	1.4
SparCL-DER++$_{90}$			73.42±0.95	94.82±0.23	1.1	19.62±0.67	51.93±0.36	1.8
SparCL-ER$_{95}$	0.95		57.75±0.45	92.73±0.34	0.5	9.91±0.17	48.57±0.50	0.8
SparCL-DER++$_{95}$			72.14±0.78	94.39±0.15	0.6	19.01±1.32	51.26±0.78	1.0

†PackNet and LPS actually have a decreased sparsity after learning every task, we use 0.50 to roughly represent the average sparsity.

splitting CIFAR-10 and Tiny-ImageNet into 5 and 10 tasks, each of which consists of 2 and 20 classes respectively. Dataset licensing information can be found in Appendix A.

Comparing methods. We select several CL methods including regularization-based (EWC [32], LwF [37]), architecture-based (PackNet [42], LPS [57]), and rehearsal-based (A-GEM [13], iCaRL [44], FDR [5], ER [14], DER++ [8]) methods. Note that PackNet and LPS are only compatible with task-incremental learning. We also adapt representative sparse training methods (SNIP [35], RigL [20]) to the CL setting by combining them with DER++ (SNIP-DER++, RigL-DER++).

Variants of our method. To show the generality of SparCL, we combine it with DER++ (one of the SOTA CL methods), and ER (simple and widely-used) as *SparCL-DER++* and *SparCL-ER*, respectively. We also vary the weight sparsity ratio $(0.75, 0.90, 0.95)$ of SparCL for a comprehensive evaluation.

Evaluation metrics. We use the average accuracy on all tasks to evaluate the performance of the final model. Moreover, we measure the training FLOPs [20], and memory footprint [67] (including feature map pixels and model parameters during training) to demonstrate the efficiency of each method. Please see Appendix B.1 for detailed definitions of these metrics.

Experiment details. For fair comparison, we strictly follow the settings in prior CL work [8, 29]. We set the per task training epochs to 50 and 100 for Split CIFAR-10 and Tiny-ImageNet, respectively, with a batch size of 32. For the model architecture, we follow [8, 50] and adopt the ResNet-18 [26] without any pre-training. We also use the best hyperparameter setting reported in [8, 57] for CL methods, and in [20, 35] for CL-adapted sparse training methods. For SparCL and its competing CL-adapted sparse training methods, we adopt a uniform sparsity ratio for all convolutional layers. Please see Appendix B for further details.

5.2 Main Results

Comparison with CL methods. Table 1 summarizes the results on Split CIFAR-10 and Tiny-ImageNet, under both class-incremental (Class-IL) and task-incremental (Task-IL) settings. From Table 1, we can clearly tell that SparCL significantly improves upon ER and DER++, while also

Table 2: Comparison with CL-adapted sparse training methods. All methods are combined with DER++ with a 500 buffer size. SparCL outperforms all methods in both accuracy and training efficiency, under all sparsity ratios. All three methods here can save 20% $\sim$ 51% memory footprint, please see Appendix B.2 for details.

Method	**Spasity**	**Split CIFAR-10** Class-IL (↑)	**Split CIFAR-10** FLOPs Train $\times 10^{15}$ (↓)	**Split Tiny-ImageNet** Class-IL (↑)	**Split Tiny-ImageNet** FLOPs Train $\times 10^{16}$ (↓)
DER++ [8]	0.00	72.70$\pm$1.36	13.9	19.38$\pm$1.41	22.2
SNIP-DER++ [35]	0.90	69.82$\pm$0.72	1.6	16.13$\pm$0.61	2.5
RigL-DER++ [20]		69.86$\pm$0.59	1.6	18.36$\pm$0.49	2.5
SparCL-DER++$_{90}$		73.42$\pm$0.95	1.1	19.62$\pm$0.67	1.8
SNIP-DER++ [35]	0.95	66.07$\pm$0.91	0.9	14.76$\pm$0.52	1.5
RigL-DER++ [20]		66.53$\pm$1.13	0.9	15.88$\pm$0.63	1.5
SparCL-DER++$_{95}$		72.14$\pm$0.78	0.6	19.01$\pm$1.32	1.0

Table 3: Ablation study on Split-CIFAR10 with 0.75 sparsity ratio. All components contributes to the overall performance, in terms of both accuracy and efficiency (training FLOPs and memory footprint).

TDM	**DDR**	**DGM**	Class-IL (↑)	FLOPs Train $\times 10^{15}$ (↓)	Memory Footprint (↓)
✗	✗	✗	72.70	13.9	247MB
✓	✗	✗	73.37	3.6	180MB
✓	✓	✗	73.80	2.8	180MB
✓	✗	✓	73.97	3.3	177MB
✓	✓	✓	**74.09**	**2.5**	**177MB**

outperforming other CL baselines, in terms of training efficiency (measured in FLOPs). With higher sparsity ratio, SparCL leads to fewer training FLOPs. Notably, SparCL achieves 23$\times$ training efficiency improvement upon DER++ with a sparsity ratio of 0.95. On the other hand, our framework also improves the average accuracy of ER and DER++ consistently under all cases with a sparsity ratio of 0.75 and 0.90, and only a slight performance drop when sparsity gets larger as 0.95. In particular, SparCL-DER++ with 0.75 sparsity ratio sets new SOTA accuracy, with all buffer sizes under both benchmarks. The outstanding performance of SparCL indicates that our proposed strategies successfully preserve accuracy by further mitigating catastrophic forgetting with a much sparser model. Moreover, the improvement that SparCL brings to two different existing CL methods shows the generalizability of SparCL as a unified framework, *i.e.*, it has the potential to be combined with a wide array of existing methods.

We also take a closer look at PackNet and LPS, which also leverage the idea of sparsity to split the model by different tasks, a different motivation from training efficiency. Firstly, they are only compatible with the Task-IL setting, since they leverage task identity at both training and test time. Moreover, the model sparsity of these methods reduces with the increasing number of tasks, which still leads to much larger overall training FLOPs than that of SparCL. This further demonstrates the importance of keeping a sparse model without permanent expansion throughout the CL process.

Comparison with CL-adapted sparse training methods. Table 2 shows results under the more difficult Class-IL setting. SparCL outperforms all CL-adapted sparse training methods in both accuracy and training FLOPs. The performance gap between SparCL-DER++ and other methods gets larger with a higher sparsity. SNIP- and RigL-DER++ achieve training acceleration at the cost of compromised accuracy, which suggests that keeping accuracy is a non-trivial challenge for existing sparse training methods under the CL setting. SNIP generates the static initial mask after network initialization which does not consider the structure suitability among tasks. Though RigL adopts a dynamic mask, the lack of task-awareness prevents it from generalizing well to the CL setting.

5.3 Effectiveness of Key Components

Ablation study. We provide a comprehensive ablation study in Table 3 using SparCL-DER++ with 0.75 sparsity on Split CIFAR10. Table 3 demonstrates that all components of our method contribute to both efficiency and accuracy improvements. Comparing rows 1 and 2, we can see that the majority of FLOPs decrease results from TDM. Interestingly, TDM leads to an increase in accuracy, indicating

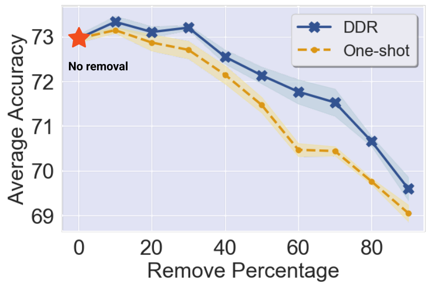

Figure 3: Comparison between DDR and One-shot [67] data removal strategy w.r.t. different data removal proportion ρ. DDR outperforms One-shot and also achieves better accuracy when $\rho \leq 30\%$.

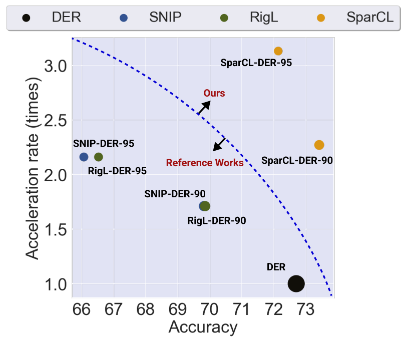

Figure 4: Comparison with CL-adapted sparse training methods in training acceleration rate and accuracy results. The radius of circles are measured by memory footprint.

TDM generates a sparse model that is even more suitable for learning all tasks than then full dense model. Comparing rows 2 and 3, we can see that DDR indeed further accelerates training by removing less informative examples. As discussed in Section 4.2, when we remove a certain number of samples (30% here), we achieve a point where we keep as much informative samples as we need, and also balance the current and buffered data. Comparing rows 2 and 4, DGM reduce both training FLOPs and memory footprint while improve the performance of the network. Finally, the last row demonstrates the collaborative performance of all components. We also show the same ablation study with 0.90 sparsity in Appendix B.4 for reference. Details can be found in Appendix B.1.

Exploration on DDR. To understand the influence of the data removal proportion ρ, and the `cutoff` stage for each task, we show corresponding experiment results in Figure 3 and Appendix B.3, respectively. In Figure 3, we fix `cutoff` $= 4$, *i.e.*, gradually removing equal number of examples every 5 epochs until epoch 20, and vary ρ from 10% to 90%. We also compare DDR with One-shot removal strategy [67], which removes all examples at once at `cutoff`. DDR outperforms One-shot consistently with different ρ in average accuracy. Also note that since DDR removes the examples gradually before the `cutoff` stage, DDR is more efficient than One-shot. When $\rho \leq 30\%$, we also observe increased accuracy of DDR compared with the baseline without removing any data. When $\rho \geq 40\%$, the accuracy gets increasingly lower for both strategies. The intuition is that when DDR removes a proper amount of data, it removes redundant information while keeps the most informative examples. Moreover, as discussed in Section 4.2, it balances the current and buffered data, while also leave informative samples in the buffer. When DDR removes too much data, it will also lose informative examples, thus the model has not learned these examples well before removal.

Exploration on DGM. We test the efficacy of DGM at different sparsity levels. Detailed exploratory experiments are shown in Appendix B.5 for reference. The results indicate that by setting the proportion q within an appropriate range, DGM can consistently improve the accuracy performance regardless of the change of weight sparsity.

5.4 Mobile Device Results

The training acceleration results are measured on the CPU of an off-the-shelf Samsung Galaxy S20 smartphone, which has the Qualcomm Snapdragon 865 mobile platform with a Qualcomm Kryo 585 Octa-core CPU. We run each test on a batch of 32 images to denote the training speed. The detail of on-mobile compiler-level optimizations for training acceleration can be found in Appendix C.1.

The acceleration results are shown in Figure 4. SparCL can achieve approximately $3.1\times$ and $2.3\times$ training acceleration with 0.95 sparsity and 0.90 sparsity, respectively. Besides, our framework can also save 51% and 48% memory footprint when the sparsity is 0.95 and 0.90. Furthermore, the obtained sparse models save the storage consumption by using compressed sparse row (CSR) storage and can be accelerated to speed up the inference on-the-edge. We provide on-mobile inference acceleration results in Appendix C.2.

6 Conclusion

This paper presents a unified framework named SparCL for efficient CL that achieves both learning acceleration and accuracy preservation. It comprises three complementary strategies: task-aware dynamic masking for weight sparsity, dynamic data removal for data efficiency, and dynamic gradient masking for gradient sparsity. Extensive experiments on standard CL benchmarks and real-world edge device evaluations demonstrate that our method significantly improves upon existing CL methods and outperforms CL-adapted sparse training methods. We discuss the limitations and potential negative social impacts of our method in Sections 7 and 8, respectively.

7 Limitations

One limitation of our method is that we assume a rehearsal buffer is available throughout the CL process. Although the assumption is widely-accepted, there are still situations that a rehearsal buffer is not allowed. However, as a framework targeting for efficiency, our work has the potential to accelerate all types of CL methods. For example, simply removing the terms related to rehearsal buffer in equation 1 and equation 3 could serve as a naive variation of our method that is compatible with other non-rehearsal methods. It is interesting to further improve SparCL to be more generic for all kinds of CL methods. Moreover, the benchmarks we use are limited to vision domain. Although using vision-based benchmarks has been a common practice in the CL community, we believe evaluating our method, as well as other CL methods, on datasets from other domains such as NLP will lead to a more comprehensive and reliable conclusion. We will keep track of newer CL benchmarks from different domains and further improve our work correspondingly.

8 Potential Negative Societal Impact

Although SparCL is a general framework to enhance efficiency for various CL methods, we still need to be aware of its potential negative societal impact. For example, we need to be very careful about the trade-off between accuracy and efficiency when using SparCL. If one would like to pursue efficiency by setting the sparsity ratio too high, then even SparCL will result in significant accuracy drop, since the over-sparsified model does not have enough representation power. Thus, we should pay much attention when applying SparCL on accuracy-sensitive applications such as healthcare [66]. Another example is that, SparCL as a powerful tool to make CL methods efficient, can also strengthen models for malicious applications [7]. Therefore, we encourage the community to come up with more strategies and regulations to prevent malicious use of artificial intelligence.

9 Acknowledgement

The authors gratefully acknowledge support by the National Science Foundation under grants CCF-1937500, CCF-1919117 and CNS-2112471.

References

[1] Hongjoon Ahn, Jihwan Kwak, Subin Lim, Hyeonsu Bang, Hyojun Kim, and Taesup Moon. Ss-il: Separated softmax for incremental learning. In *CVPR*, 2021.

[2] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *ECCV*, 2018.

[3] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *NeurIPS*, 2019.

[4] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. In *ICLR*, 2018.

[5] Ari S Benjamin, David Rolnick, and Konrad Kording. Measuring and regularizing networks in function space. *arXiv preprint arXiv:1805.08289*, 2018.

[6] Zalán Borsos, Mojmir Mutny, and Andreas Krause. Coresets via bilevel optimization for continual learning and streaming. *NeurIPS*, 2020.

[7] Miles Brundage, Shahar Avin, Jack Clark, Helen Toner, Peter Eckersley, Ben Garfinkel, Allan Dafoe, Paul Scharre, Thomas Zeitzoff, Bobby Filar, et al. The malicious use of artificial intelligence: Forecasting, prevention, and mitigation. *arXiv preprint arXiv:1802.07228*, 2018.

[8] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. In *NeurIPS*, 2020.

[9] Pietro Buzzega, Matteo Boschini, Angelo Porrello, and Simone Calderara. Rethinking experience replay: a bag of tricks for continual learning. In *ICPR*, pages 2180–2187. IEEE, 2021.

[10] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

[11] Hyuntak Cha, Jaeho Lee, and Jinwoo Shin. Co2l: Contrastive continual learning. In *ICCV*, 2021.

[12] Arslan Chaudhry, Albert Gordo, Puneet Kumar Dokania, Philip Torr, and David Lopez-Paz. Using hindsight to anchor past knowledge in continual learning. *arXiv preprint arXiv:2002.08165*, 2(7), 2020.

[13] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

[14] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019.

[15] Tianlong Chen, Zhenyu Zhang, Sijia Liu, Shiyu Chang, and Zhangyang Wang. Long live the lottery: The existence of winning tickets in lifelong learning. In *ICLR*, 2020.

[16] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*. Ieee, 2009.

[17] Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

[18] Peiyan Dong, Siyue Wang, Wei Niu, Chengming Zhang, Sheng Lin, Zhengang Li, Yifan Gong, Bin Ren, Xue Lin, and Dingwen Tao. Rtmobile: Beyond real-time mobile acceleration of rnns for speech recognition. In *DAC*, pages 1–6. IEEE, 2020.

[19] Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. In *NeurIPS*, pages 759–770, 2019.

[20] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *ICML*, pages 2943–2952. PMLR, 2020.

[21] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *ICLR*, 2019.

[22] Yifan Gong, Geng Yuan, Zheng Zhan, Wei Niu, Zhengang Li, Pu Zhao, Yuxuan Cai, Sijia Liu, Bin Ren, Xue Lin, et al. Automatic mapping of the best-suited dnn pruning schemes for real-time mobile acceleration. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 27(5):1–26, 2022.

[23] Yifan Gong, Zheng Zhan, Zhengang Li, Wei Niu, Xiaolong Ma, Wenhao Wang, Bin Ren, Caiwen Ding, Xue Lin, Xiaolin Xu, et al. A privacy-preserving-oriented dnn pruning and mobile acceleration framework. In *GLSVLSI*, pages 119–124, 2020.

[24] Song Han, Jeff Pool, et al. Learning both weights and connections for efficient neural network. In *NeurIPS*, pages 1135–1143, 2015.

[25] Tyler L Hayes, Nathan D Cahill, and Christopher Kanan. Memory efficient experience replay for streaming learning. In *ICRA*, 2019.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[27] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018.

[28] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *CVPR*, pages 4340–4349, 2019.

[29] Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. *arXiv preprint arXiv:1810.12488*, 2018.

[30] Tong Jian, Yifan Gong, Zheng Zhan, Runbin Shi, Nasim Soltani, Zifeng Wang, Jennifer G Dy, Kaushik Roy Chowdhury, Yanzhi Wang, and Stratis Ioannidis. Radio frequency fingerprinting on the edge. *IEEE Transactions on Mobile Computing*, 2021.

[31] Zixuan Ke, Bing Liu, and Xingchang Huang. Continual learning of a mixed sequence of similar and dissimilar tasks. *NeurIPS*, 2020.

[32] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *PNAS*, 114(13):3521–3526, 2017.

[33] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *https://www.cs.toronto.edu/ kriz/learning-features-2009-TR.pdf*, 2009.

[34] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[35] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *ICLR*, 2019.

[36] Tuanhui Li, Baoyuan Wu, Yujiu Yang, Yanbo Fan, Yong Zhang, and Wei Liu. Compressing convolutional neural networks via factorized convolutional filters. In *CVPR*, pages 3977–3986, 2019.

[37] Zhizhong Li and Derek Hoiem. Learning without forgetting. *TPAMI*, 40(12):2935–2947, 2017.

[38] Xiaolong Ma, Fu-Ming Guo, Wei Niu, Xue Lin, Jian Tang, Kaisheng Ma, Bin Ren, and Yanzhi Wang. Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices. In *AAAI*, pages 5117–5124, 2020.

[39] Xiaolong Ma, Wei Niu, Tianyun Zhang, Sijia Liu, Sheng Lin, Hongjia Li, Wujie Wen, Xiang Chen, Jian Tang, Kaisheng Ma, et al. An image enhancing pattern-based sparsity for real-time inference on mobile devices. In *ECCV*, pages 629–645. Springer, 2020.

[40] Zheda Mai, Ruiwen Li, Jihwan Jeong, David Quispe, Hyunwoo Kim, and Scott Sanner. Online continual learning in image classification: An empirical survey. *arXiv preprint arXiv:2101.10423*, 2021.

[41] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *ECCV*, pages 67–82, 2018.

[42] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *CVPR*, 2018.

[43] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[44] Sanket Vaibhav Mehta, Darshan Patil, Sarath Chandar, and Emma Strubell. An empirical investigation of the role of pre-training in lifelong learning. *ICML Workshop*, 2021.

[45] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):1–12, 2018.

[46] Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *ICML*, pages 4646–4655. PMLR, 2019.

[47] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. *arXiv preprint arXiv:2001.00138*, 2020.

[48] Lorenzo Pellegrini, Vincenzo Lomonaco, Gabriele Graffieti, and Davide Maltoni. Continual learning at the edge: Real-time training on smartphone devices. *arXiv preprint arXiv:2105.13127*, 2021.

[49] Quang Pham, Chenghao Liu, and Steven Hoi. Dualnet: Continual learning, fast and slow. *NeurIPS*, 2021.

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

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

[52] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *ICML*, 2018.

[53] Ghada Sokar, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Spacenet: Make free space for continual learning. *Neurocomputing*, 439:1–11, 2021.

[54] Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *NeurIPS*, 33:6377–6389, 2020.

[55] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. An empirical study of example forgetting during deep neural network learning. *arXiv preprint arXiv:1812.05159*, 2018.

[56] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *ICLR*, 2019.

[57] Zifeng Wang, Tong Jian, Kaushik Chowdhury, Yanzhi Wang, Jennifer Dy, and Stratis Ioannidis. Learn-prune-share for lifelong learning. In *ICDM*, 2020.

[58] Zifeng Wang, Zizhao Zhang, Sayna Ebrahimi, Ruoxi Sun, Han Zhang, Chen-Yu Lee, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, et al. Dualprompt: Complementary prompting for rehearsal-free continual learning. *ECCV*, 2022.

[59] Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Learning to prompt for continual learning. *CVPR*, 2022.

[60] Paul Wimmer, Jens Mehnert, and Alexandru Condurache. Freezenet: Full performance by reduced storage costs. In *ACCV*, 2020.

[61] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *CVPR*, pages 374–382, 2019.

[62] Yushu Wu, Yifan Gong, Pu Zhao, Yanyu Li, Zheng Zhan, Wei Niu, Hao Tang, Minghai Qin, Bin Ren, and Yanzhi Wang. Compiler-aware neural architecture search for on-mobile real-time super-resolution. *ECCV*, 2022.

[63] Shipeng Yan, Jiangwei Xie, and Xuming He. Der: Dynamically expandable representation for class incremental learning. In *CVPR*, pages 3014–3023, 2021.

[64] Li Yang, Sen Lin, Junshan Zhang, and Deliang Fan. Grown: Grow only when necessary for continual learning. *arXiv preprint arXiv:2110.00908*, 2021.

[65] Jaehong Yoon, Divyam Madaan, Eunho Yang, and Sung Ju Hwang. Online coreset selection for rehearsal-based continual learning. *arXiv preprint arXiv:2106.01085*, 2021.

[66] Kun-Hsing Yu, Andrew L Beam, and Isaac S Kohane. Artificial intelligence in healthcare. *Nature biomedical engineering*, 2(10):719–731, 2018.

[67] Geng Yuan, Xiaolong Ma, Wei Niu, Zhengang Li, Zhenglun Kong, Ning Liu, Yifan Gong, Zheng Zhan, Chaoyang He, Qing Jin, et al. Mest: Accurate and fast memory-economic sparse training framework on the edge. *NeurIPS*, 34, 2021.

[68] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *ICML*, 2017.

[69] Zheng Zhan, Yifan Gong, Pu Zhao, Geng Yuan, Wei Niu, Yushu Wu, Tianyun Zhang, Malith Jayaweera, David Kaeli, Bin Ren, et al. Achieving on-mobile real-time super-resolution with neural architecture and pruning search. In *ICCV*, pages 4821–4831, 2021.

[70] Tingting Zhao, Zifeng Wang, Aria Masoomi, and Jennifer Dy. Deep bayesian unsupervised lifelong learning. *Neural Networks*, 149:95–106, 2022.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] The claims match the experimental results and it is expected to generalize according to the diverse experiments stated in our paper. We include all of our code, data, and models in the supplementary materials, which can reproduce our experimental results.
   (b) Did you describe the limitations of your work? [Yes] See Section 6 and Appendix 7.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 6 and Appendix 8.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] We have read the ethics review guidelines and ensured that our paper conforms to them.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A] Our paper is based on the experimental results and we do not have any theoretical results.
   (b) Did you include complete proofs of all theoretical results? [N/A] Our paper is based on the experimental results and we do not have any theoretical results.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Section 5.1, Section 5.4 and we provide code to reproduce the main experimental results.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 5.1 and Section 5.4.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Table 1, Table 2, fig 1, fig 3.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 5.1, Section 5.4.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] We mentioned and cited the datasets (Split CIFAR-10 and Tiny-ImageNet), and all comparing methods with their paper and github in it.

(b) Did you mention the license of the assets? [Yes] The licences of used datasets/models are provided in the cited references and we state them explicitly in Appendix A.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We provide code for our proposed method in the supplement.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Dataset Licensing Information

- CIFAR-10 [33] is licensed under the MIT license.
- The licensing information of Tiny-ImageNet [34] is not available. However, the data is available for free to researchers for non-commercial use.

B Additional Experiment Details and Results

We set $\alpha = 0.5, \beta = 1$ in equation 1 and equation 3. We also set $\delta k = 5$, $p_{\texttt{inter}} = 0.01$, $p_{\texttt{intra}} = 0.005$. We also match different weight sparsity with gradient sparsity for best performance. We sample 20% data from Split CIFAR-10 training set for validation, and we use grid-search on this validation set to help us select the mentioned best hyperparameters. We use the same set of hyperparameters for both datasets. For accurate evaluation, we repeat each experiments 3 times using different random seeds and report the average performance. During our experiments, we adopt unstructured sparsity type and uniform sparsity ratio $(0.75, 0.90, 0.95)$ for all convolutional layers in the models.

B.1 Evaluation Metrics Explanation

Training FLOPs The FLOPs of a single forward pass is calculated by taking the sum of the number of multiplications and additions in each layer l for a given layer sparsity s_l. Each iteration in the training process is composed of two phases, i.e., the forward propagation and backward propagation.

The goal of the forward pass is to calculate the loss of the current set of parameters on a given batch of data. It can be formulated as $a_l = \sigma(z_l) = \sigma(w_l * a_{l-1} + b_l)$ for each layer l in the model. Here, w, b, and z represent the weights, biases, and output before activation, respectively; $\sigma(.)$ denotes the activation function; a is the activations; $*$ means convolution operation. The formulation indicates that the layer activations are calculated in sequence using the previous activations and the parameters of the layer. Activation of layers are stored in memory for the backward pass.

As for the backward propogation, the objective is to back-propagate the error signal while calculating the gradients of the parameters. The two main calculation steps can be represented as:

$$\delta_l = \delta_{l+1} * \text{rotate180°}(w_l) \odot \sigma'(z_l), \tag{4}$$

$$G_l = a_{l-1} * \delta_l, \tag{5}$$

where δ_l is the error associated with the layer l, G_l denotes the gradients, $\odot$ represents Hadamard product, $\sigma'(.)$ denotes the derivative of activation, and rotate180°$(.)$ means rotating the matrix by 180° is the matrix transpose operation. During the backward pass, each layer l calculates two quantities, i.e., the gradient of the activations of the previous layer and the gradient of its parameters. Thus, the backward passes are counted as **twice** the computation expenses of the forward pass [20]. We omit the FLOPs needed for batch normalization and cross entropy. In our work, the total FLOPs introduced by TDM, DDR, and DGM on split CIFAR-10 is approximately 4.5×10^9 which is less than 0.0001% of total training FLOPs. For split Tiny-ImageNet, the total FLOPs of them is approximately 1.8×10^{10}, which is also less than 0.0001% of total training FLOPs. Therefore, the computation introduced by TDM, DDR, and DGM is negligible.

Memory Footprints Following works [10, 67], the definition of memory footprints contain two parts: 1) activations (feature map pixels) during training phase, and 2) model parameters during training phase. For experiments, activations, model weights, and gradients are stored in 32-bit floating-point format for training. The memory footprint results are calculated with an approximate summation of them.

B.2 Details of Memory Footprint

The memory footprint is composed of three parts: activations, model weights, and gradients. They are all represented as b_w-bit numbers for training.

The number of activations in the model is the sum of the activations in each layer. Suppose that the output feature of the l-th layer with a batch size of B is represented as $a_l \in \mathcal{R}^{B \times O_l \times H_l \times W_l}$, where

O_l is the number of channels and $H_l \times W_l$ is the feature size. The total number of activations of the model is thus $B \sum_l O_l H_l W_l$.

As for the model weights, our SparCL training a sparse model with a sparsity ratio $s \in [0, 1]$ from scratch. The sparse model is obtained from a dense model with a total of N weights. A higher value of s indicates fewer non-zero weights in the sparse model. Compressed sparse row (CSR) format is commonly used for sparse storage, which greatly reduces the number of indices need to be stored for sparse matrices. As our SparCL adopt only one sparsity type and we use a low-bit format to store the indices, we omit the indices storage here. Therefore, the memory footprint for model representation is $(1 - s)Nb_w$.

Similar calculations can be applied for the gradient matrix. Besides the sparsity ratio s, additional q gradients are masked out from the gradient matrix, resulting a sparsity ratio $s + q$. Therefore, the storage of gradients can be approximated as $(1 - (s + q))Nb_w$.

Combining the activations, model representation, and gradients, the total memory footprint in SparCL can be represented as $(2B \sum_l O_l H_l W_l + (1 - s)N + (1 - (s + q))N)b_w$.

DDR requires store indices for the easier examples during the training process. The number of training examples for Split CIFAR-10 and Split Tiny-ImageNet on each task is 10000. In our work, we only need about 3KB (remove 30% training data) for indices storage (in the int8 format) and the memory cost is negligible compared with the total memory footprint.

B.3 Effect of Cutoff Stage

Table A1: Effect of `cutoff`.

`cutoff`	1	2	3	4	5	6	7	8	9
Class-IL (↑)	71.54	72.38	72.74	73.20	73.10	73.32	73.27	73.08	73.23

To evaluate the effect of the `cutoff` stage, we use the same setting as in Figure 3 by setting the sparsity ratio to 0.90. We keep the data removal proportion $\rho = 30\%$, and only change `cutoff`. Table A1 shows the relationship between `cutoff` and the Class-IL average accuracy. Note that from the perspective of efficiency, we would like the `cutoff` stage as early as possible, so that the remaining epochs will have less examples. However, from Table A1, we can see that if we set it too early, *i.e.*, `cutoff` ≤ 3, the accuracy drop is significant. This indicate that even for the "easy-to-learn" examples, removing them too early results in underfitting. As a balance point between accuracy and efficiency, we choose `cutoff` $= 4$ in our final version.

B.4 Supplementary Ablation Study

Table A2: Ablation study on Split-CIFAR10 with 0.90 sparsity.

TDM	**DDR**	**DGM**	Class-IL (↑)	FLOPs Train $\times 10^{15}$ (↓)	Memory Footprint (↓)
✗	✗	✗	72.70	13.9	247MB
✓	✗	✗	72.98	1.6	166MB
✓	✓	✗	73.20	1.2	166MB
✓	✗	✓	73.30	1.5	165MB
✓	✓	✓	**73.42**	**1.1**	**165MB**

Similar to Table 3, we show ablation study with 0.90 sparsity ratio in Table A2. Under a larger sparsity ratio, the conclusion that all components contribute to the final performance still holds. However, we can observe that the accuracy increase that comes from DDR and DGM is less than what we show in Table 3. We assume that larger sparsity ratio makes it more difficult for the model to retain good accuracy in CL. Similar results has also been observed in [67] under the usual i.i.d. learning setting.

Table A3: Ablation study of the gradient sparsity ratio on Split-CIFAR10.

weight sparsity	**gradient sparsity**	Class-IL (↑)	FLOPs Train $\times 10^{15}$ (↓)	Memory Footprint (↓)
0.75	0.78	74.08	3.4	178MB
0.75	0.80	73.97	3.3	177MB
0.75	0.82	73.79	3.3	177MB
0.75	0.84	73.26	3.2	176MB
0.90	0.91	73.33	1.6	166MB
0.90	0.92	73.30	1.5	165MB
0.90	0.93	72.64	1.5	165MB

B.5 Exploration on DGM

We conduct further experiments to demonstrate the influence of gradient sparsity, and the results are shown in Table A4. There are two sets of the experiments with different weight sparsity settings: 0.75 and 0.90. Within each set of the experiments (the weight sparsity is fixed), we vary the gradient sparsity values. From the results we can see that increasing the gradient sparsity can decrease the FLOPs and memory footprint. However, the accuracy performance degrades more obvious when the gradient sparsity is too much for the weight sparsity. The results indicate that suitable gradient sparsity setting can bring further efficiency to the training process while boosting the accuracy performance. In the main results, the gradient sparsity is set as 0.80 for 0.75 weight sparsity, and set as 0.92 for 0.90 weight sparsity.

C On-Mobile Compiler Optimizations and Inference Results

C.1 Compiler Optimizations

Each iteration in the training process is composed of two phases, i.e., the forward propagation and backward propagation. Prior works [18, 23, 27, 28, 30, 36, 38, 69] have proved that sparse weight matrices (tensors) can provide inference acceleration via reducing the number of multiplications in convolution operation. Therefore, the forward propagation phase, which is the same as inference, can be accelerated by the sparsity inherently. As for backward pass, both of the calculation steps are based on convolution, i.e., matrix multiplication. Equation 4 uses sparse weight matrix (tensor) as the operand, thus can be accelerated in the same way as the forward propagation. Equation 5 allows a sparse output result since the gradient matrix is also sparse. Thus, both two steps have reduced computations, which are roughly proportional to the sparsity ratio, providing the acceleration for the backward propagation phase.

Compiler optimizations are used to accelerate the inference in prior works [22, 47, 62]. In this work, we extend the compiler optimization techniques for accelerating the forward and backward pass during training on the edge devices. Our compiler optimizations are general, support both sparse model training and inference accelerations on mobile platforms. The optimizations include 1) the supports for sparse models; 2) an auto-tuning process to determine the best-suited configurations of parameters for different mobile CPUs. The details of our compiler optimizations are presented as follows.

C.1.1 Supports for Sparse Models

Our framework supports sparse model training and inference accelerations with unstructured pruning. For the sparse (pruned) model, the framework first compacts the model storage with a compression format called Compressed Sparse Row (CSR) format, and then performs computation reordering to reduce the branches within each thread and eliminates the load imbalance among threads.

A row reordering optimization is also included to further improve the regularity of the weight matrix. After this reordering, the continuous rows with identical or similar numbers of non-zero weights are processed by multi-threads simultaneously, thus eliminating thread divergence and achieving load balance. Each thread processes more than one rows, thus eliminating branches and improving instruction-level parallelism. Moreover, a similar optimization flow (i.e., model compaction and computation reorder and other optimizations) is employed to support all compiler optimizations for sparsity as PatDNN [47].

C.1.2 Auto-Tuning for Different Mobile CPUs

During DNN sparse training and inference execution, there are many tuning parameters, e.g., matrix tiling sizes, loop unrolling factors, and data placement on memory, that influence the performance. It is hard to determine the best-suited configuration of these parameters manually. To alleviate this problem, our compiler incorporates an auto-tuning approach for sparse (pruned) models. The Genetic Algorithm is leveraged to explore the best-suited configurations automatically. It starts the parameter search process with an arbitrary number of chromosomes and explores the parallelism better. Acceleration codes for different DNN models and different mobile CPUs can be generated efficiently and quickly through this auto-tuning process.

C.2 Inference Acceleration Results On Mobile

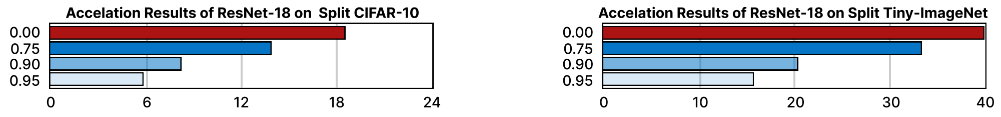

Figure 5: Inference results of sparse models obtained from SparCL under different sparsity ratio compared with dense models obtained from traditional CL methods (sparsity ratio 0.00).

Besides accelerating the training process, SparCL also possesses the advantages of providing a sparse model as the output for faster inference. To demonstrate this, we show the inference acceleration results of SparCL with different sparsity ratio settings on mobile in Figure 5. The inference time is measured on the CPU of an off-the-shelf Samsung Galaxy S20 smartphone. Each test takes 50 runs on different inputs with 8 threads on CPU. As different runs do not vary greatly, only the average time is reported. From the results we can see that the obtained sparse model from SparCL can significantly accelerate the inference on both Split-CIFAR-10 and Tiny-ImageNet dataset compared to the model obtained by traditional CL training. For ResNet-18 on Split-CIFAR-10, the model obtained by traditional CL training, which is a dense model, takes 18.53ms for inference. The model provided by SparCL can achieve an inference time of 14.01ms, 8.30ms, and 5.85ms with sparsity ratio of 0.75, 0.90, and 0.95, respectively. The inference latency of the dense ResNet-18 obtained by traditional CL training on Tiny-ImageNet is 39.64 ms. While the sparse models provided by SparCL with sparsity ratio settings as 0.75, 0.90, and 0.95 reach inference speed of 33.06ms, 20.37ms, and 15.49ms, respectively, on Tiny-ImageNet.

D Comparison with Buffer Selection Methods

In this section, we compare DDR and GSS [3] or Loss-Aware Reservoir Sampling (LARS) [9].

DDR aims at removing training examples for efficiency, while GSS and LARS put the focus on selecting more informative examples that are saved in the buffer. Technically, DDR removes less informative training examples at certain epochs (and thus indirectly affects samples saved in the buffer), while GSS and LARS directly replaces less informative buffered examples in the buffer. Thus, the original GSS and LARS are not directly comparable to DDR. However, we can actually use the example importance criteria used in GSS and LARS to remove less informative training examples as well. We replace the misclassification rate in DDR by the gradient-based (GSS) and loss-based criteria (LARS) objectives and get two variants of our approach, DDR-GSS and DDR-LARS, respectively. For fair comparison, we fix all other parameters used in DDR the same for all methods (sparisity 0.75, remove 30% training data, with TDM only). Since all variants of DDR already remove training examples for efficiency, we mainly focus on their accuracy performance here. The final results on Split-CIFAR10 is shown in the table below:

Table A4: Comparison with Buffer Selection Methods.

Method	Importance	Accuracy
DDR	Misclassification	73.80
DDR-GSS	Gradient	73.45
DDR-LARS	Loss	73.67

E Exploration on Pruning Pattern

In this work, we conduct uniform pruning (i.e., each layer has the same pruning ratio) across different CONV layers as mentioned before in experimental details. The usage of uniform pruning ratio is to match the single-instruction multiple-data (SIMD) [47] architecture of embedded CPU/GPU processors for efficient hardware accelerations.

To observe the pruning pattern, we also experimented with setting an overall pruning ratio as 95% for the entire network, allowing each layer to have different pruning ratios by ranking CWI for the whole model. According to the results, earlier CONV layers tend to have a smaller pruning ratio, which is only around $25\% - 30\%$, while the pruning ratios for the latter CONV layers can reach 99%. The results are reasonable, as latter layers are more redundant with a larger amount of parameters. In addition, the weights in earlier layers might be more important for keeping high accuracy, but take a large portion of the computation. Therefore, though slightly improving the accuracy performance to 72.45% compared to the uniform pruning ratio setting, allowing different pruning ratios across different layers yields worse acceleration (drop to $2.2\times$ compared with $3.1\times$ when adopting the uniform pruning ratio) on the hardware. As our purpose is to facilitate the efficiency of the CL-system, we adopt the uniform pruning ratio setting.