2027 Design, Automation & Test in Europe Conference (DATE 2027)

FlyCL: Memory-efficient Sparse Forward-only Continual Learning

Yihang Zuo[†], Wanhao Yu[‡], Asmer Ali[†], Li Yang[‡], Deliang Fan[†,*]
[†]Arizona State University, Phoenix, USA [‡]University of North Carolina at Charlotte, Charlotte, USA
{yhzuo,aali115,dfan}@asu.edu, {wyu6,lyang50}@charlotte.edu [*]Corresponding author

***Abstract*—Continual learning (CL) enables AI models to incrementally adapt to new tasks, but remains challenging to deploy efficiently on edge platforms due to the high memory and computational demands of backpropagation. In this work, we propose FlyCL, the first memory-efficient, sparse, forward-only CL framework for pre-trained models (PTMs), which optimizes adapters purely through forward passes and updates the second multi-layer perceptron (MLP) projections and the classifier using closed-form solutions. For the first session, FlyCL computes the Gram matrix of the hidden activations, uses it to perform structured pruning on the backbone, and trains adapters with zeroth-order (ZO) optimization. For the next tasks, FlyCL updates the Gram matrix, the compensated projections, and the classifier in closed form to achieve task-specific adaptation. Extensive experiments show that FlyCL achieves up to $11.7\times$ training memory reduction over state-of-the-art (SOTA) CL methods while pruning 30% of the backbone MLP channels; it stays within 0.8% of the most accurate baseline on CIFAR-100, surpasses it by 3.1% on ImageNet-R, and reduces the inference time by 17%–69%. On the same edge hardware with limited memory resources, FlyCL extends the maximum supported trainable model from ViT-base to a $\sim 3.6\times$ larger ViT-large model, whereas traditional CL methods fail due to much larger memory requirements.**

***Index Terms*—Zeroth-order optimization, continual learning, efficient edge AI**

I. Introduction

Pre-trained models (PTMs) built on the transformer architecture [1], [2] have become the standard starting point for vision and language tasks, and are adapted to downstream tasks by fine-tuning [3]. However, no pre-training stage can anticipate every future scenario, and further training a PTM on new tasks causes catastrophic forgetting, where newly acquired knowledge overwrites previously learned information [4]. Continual learning (CL) addresses this problem by acquiring new knowledge while retaining previously learned representations.

Although CL offers considerable performance improvements over vanilla static training paradigms, deploying it on edge devices remains very challenging. To address this, SparCL [5], LightCL [6], and FAMP [7] have been proposed to reduce the memory footprint of CL by pruning the backbone model. Unfortunately, as shown in Table I, despite being designed for edge devices, these approaches still suffer from several notable limitations. First, they do not freeze the backbone and are all developed for convolutional neural network (CNN) architectures, with only LightCL [6] freezing some early layers. Second, using a buffer to replay previous samples introduces additional memory and computational overhead, as it requires repeatedly saving and retraining old samples. Sometimes, the size of the stored samples is even larger than the model. Third, the proven hardware-friendly pruning strategy should be structured [8] to enable end-to-end system inference speedups [8] without requiring a custom hardware redesign. Lastly, all of them rely on first-order (FO) backpropagation and therefore leave the dominant source of training memory, the activations cached for the backward pass, untouched.

TABLE I Different Memory-efficient CL Methods Comparison.

Method	No-Replay buffer	Structured Pruning	Frozen Backbone	Memory Reduction
SparCL [5]	✗	✗	✗	2×
LightCL [6]	✗	✗	✓*	1.6×
FAMP [7]	✗	✓	✗	3-5×
FlyCL (Ours)	✓	✓	✓	**11.7**×

* indicates it only freezes partial layers.

Actually, the primary source of memory consumption is the cached backward activations (i.e., FO backpropagation). In contrast to FO methods that compute explicit gradients via backpropagation, zeroth-order (ZO) optimization estimates gradients *purely from forward passes*, thereby avoiding the need to store intermediate activations during the backward pass. More recently, ZO-FC [9] and RISE [10] train the adapters of every task with ZO optimization and the classifier with FO optimization; however, their training time grows with the number of tasks, and their accuracy lags behind FO-trained adapters. Another method is a closed-form solution such as FoRo [11], which optimizes input prompts with an evolution strategy and updates the classifier via recursive least squares. However, it repeats the evolutionary search with a population of forward passes per sample on every task, so its training time grows with the number of tasks. In this paper, we propose FlyCL, a sparse forward-only CL framework that structurally prunes a transformer-based model for edge deployment, targeting both training and inference memory and compute efficiency.

To summarize, our work makes the following contributions:

- We propose FlyCL, a memory-efficient sparse forward-only continual learning framework. To the best of our knowledge, this is the first work to combine structured backbone pruning with forward-only training for continual learning on edge devices.
- We design a two-stage forward-only CL flow. In the first stage, we compute the Gram matrix of the hidden activations, use it to perform structured pruning on the backbone, and train adapters with ZO optimization. In the second stage, we only update the Gram matrix, the compensated MLP projections, and the classifier in closed

form to achieve task-specific adaptation.

- Compared with other first-order CL methods, FlyCL achieves up to 11.7× memory reduction during training while pruning 30% of the backbone MLP channels; it stays within 0.8% of the most accurate baseline on CIFAR-100, surpasses it by 3.1% on ImageNet-R, and reduces the inference time by 17%–69%.

The rest of the paper is organized as follows: In Section II, we provide background on CL and ZO optimization. FlyCL is described comprehensively in Section III. Experimental analysis and conclusion are illustrated in Section IV and Section V, respectively.

II. Background

A. Continual Learning Methods

In real-world deployments, data arrive as a stream, and privacy or storage constraints prevent retaining past samples, which calls for a paradigm that adapts continuously, known as CL [12]. Numerous strategies have been proposed to mitigate catastrophic forgetting and improve the stability and efficiency of CL [13]–[16].

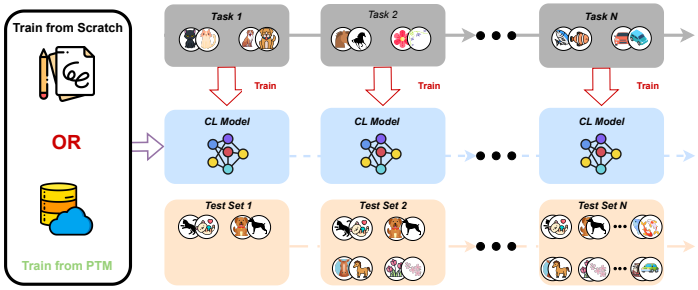


Fig. 1 Two types of continual learning. Tasks must be learned incrementally. Traditional CL starts with random initialization, whereas PTM-based CL leverages pre-trained weights to improve generalization and efficiency.

Fig. 1 illustrates the general process of CL and its variations. In CL, tasks arrive sequentially, and the model must learn each new task while retaining knowledge from prior tasks. After completing the learning process for a given task, the model is evaluated across all previously seen tasks to assess its ability to preserve and transfer knowledge. This paper targets the *class-incremental* setting [17]: each task introduces a disjoint set of new classes, the task identity is unknown at test time, so a single classifier must discriminate among all classes seen so far, and no exemplar of past tasks may be stored. Based on the model initialization strategy, CL is divided into two main scenarios: traditional training from scratch and recent training from PTMs [17]. Conceptually, this distinction resembles starting from a blank model versus adapting a knowledgeable model to new information, with the latter providing a stronger starting point and greater learning efficiency.

B. Optimization with Hessian-Informed ZO Gradient Estimation

To optimize parameters without explicit gradients, we adopt the HiZOO optimizer [18], a diagonal Hessian-informed ZO method. Unlike conventional ZO approaches such as simultaneous perturbation gradient approximation (SPSA), which approximate the gradient via symmetric function perturbations [19], HiZOO further estimates second-order curvature information through additional perturbations, achieving more efficient and stable updates. Given a model with trainable parameters $\theta \in \mathbb{R}^d$ and loss function $\mathcal{L}$, HiZOO applies a second-order Taylor expansion around θ in the directions $\pm\Sigma^{1/2}u$:

$$\mathcal{L}(\theta \pm \mu\Sigma^{1/2}u) = \mathcal{L}(\theta) \pm \mu\nabla\mathcal{L}(\theta)^\top\Sigma^{1/2}u + \frac{\mu^2}{2}u^\top\Sigma^{1/2}\nabla^2\mathcal{L}(\theta)\Sigma^{1/2}u + O(\mu^3), \tag{1}$$

where $u \in \mathbb{R}^d$ is sampled from $\mathcal{N}(0, I_d)$ and μ is the positive perturbation scale. Summing the two expansions eliminates the first-order term:

$$\begin{aligned}\Delta\mathcal{L}(\theta, u) &= \mathcal{L}(\theta + \mu\Sigma^{1/2}u) + \mathcal{L}(\theta - \mu\Sigma^{1/2}u) - 2\mathcal{L}(\theta) \\ &= \mu^2 u^\top\Sigma^{1/2}\nabla^2\mathcal{L}(\theta)\Sigma^{1/2}u + O(\mu^4).\end{aligned} \tag{2}$$

This yields an estimate of the diagonal Hessian $\mathbf{H} = \nabla^2\mathcal{L}(\theta)$ [18]:

$$\hat{\mathbf{H}}_t = \frac{1}{n}\sum_{i=1}^{n}\frac{\Delta\mathcal{L}(\theta, u_i)}{2\mu^2}\left(\Sigma_t^{-1/2}u_i u_i^\top\Sigma_t^{-1/2} - \Sigma_t^{-1}\right), \tag{3}$$

where n is the number of queries used to reduce the noise of a single estimate. Following HiZOO, we omit the $-\Sigma_t^{-1}$ term in Eq. 3 in practice and utilize an exponential moving average (EMA) to denoise the diagonal Hessian estimate as

$$\Sigma_t^{-1} = (1 - \alpha_t)\Sigma_{t-1}^{-1} + \alpha_t|\operatorname{diag}(\hat{\mathbf{H}}_t)|, \tag{4}$$

where $\alpha_t \in (0, 1)$ is the smoothing coefficient. Σ_t therefore approximates the inverse magnitude of the diagonal Hessian and preconditions both the perturbation direction and the gradient estimate. Through this procedure, HiZOO efficiently infers curvature information without backpropagation, allowing adaptive preconditioning in purely ZO settings.

III. Proposed FlyCL Framework

A. Overview of the Forward-Only Continual Learning Flow

As illustrated in Fig. 2 (a), FlyCL is a class-incremental CL framework in which every learning step is performed with forward passes only. The backbone is a frozen ViT initialized from a PTM (pre-trained on ImageNet-21k [20]). We select two types of forward-only methods to update three groups of parameters: the second projection of every MLP block and the classifier, both of which are solved in closed form from forward statistics, and a lightweight parallel adapter per block is trained by ZO optimization.

Fig. 2 (b) shows the architecture of one transformer block. The MLP of block k applies two projections, $\mathbf{W}_1^k$ and $\mathbf{W}_2^k$, with a GELU in between. Let $\mathbf{X}_k$ denote the hidden activations of block k, i.e., the GELU outputs of the first projection, with one row per token; the MLP output is then $\mathbf{X}_k\mathbf{W}_2^k$. A binary channel mask $\mathbf{m}_k$ sits between the two projections, and a bottleneck adapter runs in parallel with the MLP. All statistics used by FlyCL are derived from a single quantity, the *Gram matrix* of the hidden activations of block k,

$$\mathbf{G}_k = \mathbf{X}_k^\top\mathbf{X}_k, \tag{5}$$

accumulated over the tokens of a set of *unlabeled* images. $\mathbf{G}_k$ is additive over data, so it can be updated whenever a new task arrives without revisiting old samples.

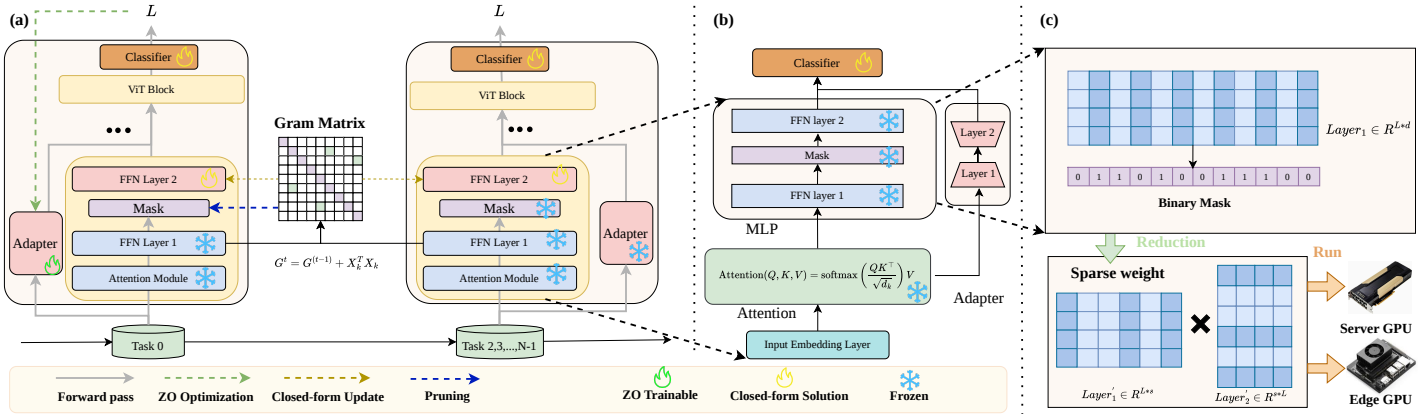


Fig. 2 The overview of FlyCL. (a) The forward-only continual learning flow: on the first task, the backbone is pruned from Gram statistics, the second MLP projections are compensated in closed form, and the adapters (about 0.35% of the backbone parameters) are trained by ZO optimization; on every task, only the accumulated Gram matrices, the compensated projections, and the classifier heads are updated in closed form. (b) The architecture of one transformer block with the parallel adapter. (c) The structured mask reduction into dense sparse weights for lightweight deployment.

For the first task only, FlyCL collects $\mathbf{G}_k$, removes the low-energy channels with a depth-dependent budget, compensates the pruned $\mathbf{W}_2^k$ in closed form, and trains the adapters with the HiZOO optimizer under the pruned backbone, after which the adapters are frozen (Section III-D). For every subsequent task $t \geq 1$, FlyCL accumulates $\mathbf{G}_k$ with unlabeled images of the new task and re-solves $\mathbf{W}_2^k$ from the accumulated matrix. The closed-form classifier head is updated with the new-class features on every task, including the first, as shown in Section III-C. Fig. 2 (c) shows how the mask is finally reduced into dense sparse weights that run on server and edge GPUs without custom kernels.

B. Gram-Based Structured Pruning

Gram-based Channel Pruning. The diagonal entries of $\mathbf{G}_k$ are the squared ℓ_2 norms of the channel activations over the calibration tokens, i.e., the activation energy of each channel, which is an established saliency for post-training pruning [21]. FlyCL defines the importance of channel c in block k as its root-mean-square activation over the N calibration tokens,

$$s_{k,c} = \sqrt{[\mathbf{G}_k]_{cc}/N}, \tag{6}$$

which requires only $N_{\text{cal}} = 1024$ unlabeled images and one forward pass per block. Channels with the smallest $s_{k,c}$ contribute the least energy to the block output and are pruned first.

Depth-decreasing sparsity schedule. Visual representations of a PTM become increasingly task-specific with depth [22], [23]: shallow blocks learn generic low-level features that are largely redundant for a downstream task, whereas the deep blocks produce the class-discriminative features consumed by the head. Our ablation in Section IV-D confirms this: at the same total sparsity, pruning the deep blocks the least is consistently more accurate than uniform pruning. Given a global channel sparsity ρ and L blocks, FlyCL therefore leaves the last K blocks unpruned and assigns each of the remaining $n = L - K$ blocks $k \in \{1, \ldots, n\}$ the pruning ratio

$$r_k = \rho \frac{L\,(n-k+1)^p}{\sum_{j=1}^{n}(n-j+1)^p}, \tag{7}$$

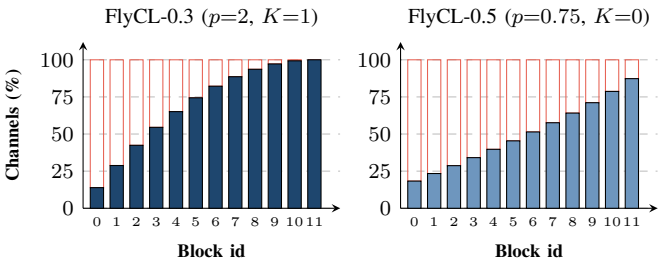


Fig. 3 Per-block MLP hidden channels of the CIFAR-100 models in TABLE II for a global sparsity of 30% (left, $p = 2$, $K = 1$) and 50% (right, $p = 0.75$, $K = 0$) on ViT-B/16 with $L = 12$ blocks. Solid bars are the retained channels and the hollow part on top is the pruned ratio r_k; each column sums to 100%.

so that the ratio decreases with depth as a power of the distance to the last pruned block while the mean ratio over all L blocks equals ρ. The mask of block k keeps the fraction $1 - r_k$ of channels with the largest saliency; the corresponding columns of $\mathbf{W}_1^k$ and rows of $\mathbf{W}_2^k$ are removed in place as shown in Fig. 2 (c). Since pruning is confined to the hidden dimension, the input and output sizes of every block are unchanged, so the residual connections and the adapters are unaffected. Fig. 3 shows the resulting per-block pruning ratios of the two CIFAR-100 models evaluated in Section IV.

C. Closed-Form Compensation and Continual Recalibration

Gram matrix and ridge regression. Both the compensation of the pruned MLP, in the spirit of layer-wise reconstruction pruning [24], and the classifier head of FlyCL, adopted from RanPAC [25], are ridge regressions solved from Gram statistics. For a linear map $\mathbf{W}$ from inputs $\mathbf{X}$ to targets $\mathbf{T}$ (one row per sample), the ridge objective $\|\mathbf{X}\mathbf{W} - \mathbf{T}\|_F^2 + \lambda\|\mathbf{W}\|_F^2$ has the closed-form minimizer

$$\mathbf{W}^\star = \left(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^\top\mathbf{T}, \tag{8}$$

where the penalty $\lambda\|\mathbf{W}\|_F^2$ keeps the system positive definite when the inputs are correlated or few. Eq. 8 depends on the data only through the Gram matrix $\mathbf{X}^\top\mathbf{X}$ and the cross term

$\mathbf{X}^\top\mathbf{T}$; both are sums over samples, so the statistics of a new task can be added to the stored ones and the solution re-solved without keeping any past sample.

For the pruned MLP, let $\mathcal{K}_k$ be the index set of the channels retained in block k, and let $\mathbf{X}_{k,\mathcal{K}_k}$ be the corresponding columns of $\mathbf{X}_k$. We seek a projection $\mathbf{W}$ acting on the retained channels only, such that the pruned block reproduces the dense output over the calibration tokens:

$$\min_{\mathbf{W}} \left\|\mathbf{X}_{k,\mathcal{K}_k}\mathbf{W} - \mathbf{X}_k\mathbf{W}_2^k\right\|_F^2 + \lambda\|\mathbf{W}\|_F^2. \tag{9}$$

Since the target is the dense projection of the same activations, both statistics of Eq. 8 are sub-matrices of the Gram matrix $\mathbf{G}_k$ in Eq. 5, and the solution is

$$\mathbf{W}^\star = \left([\mathbf{G}_k]_{\mathcal{K}_k\mathcal{K}_k} + \lambda\mathbf{I}\right)^{-1}[\mathbf{G}_k]_{\mathcal{K}_k,:}\,\mathbf{W}_2^k, \tag{10}$$

where $[\mathbf{G}_k]_{\mathcal{K}_k,:}$ and $[\mathbf{G}_k]_{\mathcal{K}_k\mathcal{K}_k}$ are the row and principal sub-matrices of $\mathbf{G}_k$ indexed by the retained channels. Eq. 10 needs neither the raw activations nor any label. The ridge λ is set to 10^{-2} times the mean diagonal of $[\mathbf{G}_k]_{\mathcal{K}_k\mathcal{K}_k}$, and the system is solved by a Cholesky factorization.

Continual update. When a new task $t \geq 1$ arrives, FlyCL draws N_{cal} unlabeled images of the task, updates $\mathbf{G}_k \leftarrow \mathbf{G}_k + \mathbf{X}_k^\top\mathbf{X}_k$ with the new activations (the yellow arrows in Fig. 2 (a)), and re-solves Eq. 10 with the accumulated matrix. The pruned backbone thus adapts to the accumulated activation statistics of *all* tasks seen so far, while the frozen dense PTM remains the regression target, so each recalibration stays functionally anchored to the original pre-trained model. The accumulated statistics are the only cross-task state of the backbone. Since $\mathcal{K}_k$ and $\mathbf{W}_2^k$ are fixed after the first task, only the two sub-matrices that enter Eq. 10, $[\mathbf{G}_k]_{\mathcal{K}_k\mathcal{K}_k}$ and $[\mathbf{G}_k]_{\mathcal{K}_k,:}\mathbf{W}_2^k$, are accumulated, in fp32 in host memory (0.28 GB for the 30% ViT-B model); only the block being refit is moved to the GPU.

The classifier head follows the random-projection head of RanPAC [25]: it operates on the frozen feature $\mathbf{f}$ (the class token after the final layer normalization) and is updated by the same rule. Features are lifted by a fixed random-projection (RP) matrix $\mathbf{W}_r$ with $M = 5000$ outputs followed by a ReLU, $\mathbf{h} = \mathrm{ReLU}(\mathbf{f}\mathbf{W}_r)$, and a linear classifier is fit by ridge regression to the one-hot labels $\mathbf{Y}$ of the current task:

$$\mathbf{W}_o = \left(\mathbf{G}_h + \lambda_h\mathbf{I}\right)^{-1}\mathbf{Q}_h, \quad \mathbf{G}_h \mathrel{+}= \mathbf{H}^\top\mathbf{H},\ \mathbf{Q}_h \mathrel{+}= \mathbf{H}^\top\mathbf{Y}, \tag{11}$$

where $\mathbf{H}$ stacks the lifted features of the current task and the accumulators $\mathbf{G}_h, \mathbf{Q}_h$ carry the statistics of all previous tasks. The ridge λ_h is selected per task from the grid $\{10^{-8}, \ldots, 10^{8}\}$ on a random 80/20 split of the current task's training features. Only the labels of the current task are needed, no old feature is stored, and the head grows by one column of M entries per new class.

D. Zeroth-Order First-Session Adapter Training

As shown in Fig. 2 (b), each block receives a bottleneck adapter in parallel with its MLP, whose output is added to the MLP output before the residual connection. The twelve adapters total about 0.3 M parameters ϕ, which is only 0.35% of the backbone. The loss is the cross-entropy over a cosine prototype head, $\mathcal{L}(\phi) = \mathrm{CE}\big(\tau\cos(\mathbf{f}_\phi(\mathbf{x}), \mathbf{P}),\ y\big)$, where $\mathbf{f}_\phi(\mathbf{x})$ is the class token after the final layer normalization, $\mathbf{P}$ holds the class means of the first task under the current ϕ (recomputed every epoch), and $\tau = 8$ is the cosine scale. At each step, HiZOO samples q Gaussian directions $\mathbf{u}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and estimates the preconditioned gradient from symmetric loss differences,

$$\hat{\mathbf{g}}(\phi) = \frac{1}{2\mu q}\sum_{i=1}^{q}\left[\mathcal{L}(\phi + \mu\boldsymbol{\Sigma}^{1/2}\mathbf{u}_i) - \mathcal{L}(\phi - \mu\boldsymbol{\Sigma}^{1/2}\mathbf{u}_i)\right]\boldsymbol{\Sigma}^{1/2}\mathbf{u}_i, \tag{12}$$

where μ is the perturbation scale and $\boldsymbol{\Sigma}$ is the diagonal Hessian estimate maintained by Eqs. 3 and 4 from the same three forward evaluations. The update is $\phi \leftarrow \phi - \eta\,\mathrm{clip}(\hat{\mathbf{g}}(\phi))$ with global-norm clipping. Each step costs $2q + 1$ forward passes and no backward pass, so the activation memory of backpropagation, which dominates FO training, is never allocated. After the first task, ϕ is frozen; later tasks only touch the closed-form components.

Algorithm 1: FlyCL

Input : Frozen PTM with blocks $\{\mathbf{W}_1^k, \mathbf{W}_2^k\}_{k=1}^L$, task stream $\{\mathcal{D}_t\}_{t=0}^{T-1}$, sparsity ρ, exponent p, ridge λ, queries q, scale μ, step size η
Output : Pruned backbone, adapter ϕ, RP head $\mathbf{W}_o$

```
// Task 0: prune, compensate, adapt (once)
G_k ← 0 for all k;  X ← N_cal unlabeled images of D_0
for k = 1 to L do
    G_k ← Gram of block k on X                          ▷ Eq. 5, forward only
    K_k ← top (1 − r_k) fraction of channels by s_{k,c}  ▷ Eqs. 6 and 7
    W_2^k ← ridge solution on K_k                       ▷ Eq. 10
for each epoch and mini-batch (x, y) ~ D_0 do
    φ ← φ − η clip(HiZOO(q, φ))                         ▷ Eq. 12
Freeze φ
F, Y ← features and labels of D_0
Update G_h, Q_h; solve W_o                              ▷ Eq. 11
Evaluate on the classes of D_0
// Every later task t ≥ 1: closed-form updates only
for t = 1 to T − 1 do
    X ← N_cal unlabeled images of D_t
    for k = 1 to L do
        G_k ← G_k + Gram of block k on X;  refit W_2^k  ▷ Eq. 10
    F, Y ← features and labels of D_t                   ▷ one forward pass
    Update G_h, Q_h with F, Y
    solve W_o                                           ▷ Eq. 11
```

IV. Experiment

A. Experiment Setting

As shown in the TABLE I, all previous memory-efficient CL works do not freeze the whole backbone and are designed for CNN, which is significantly smaller than the scale of the vision transformer. We select seven SOTA CL methods for training from PTMs and group them in TABLE II by what is trained and how: (i) prompt-based methods, CODA-Prompt [14] and L2P [15], which keep the backbone frozen and learn input prompts with FO optimization on every task; (ii) adapter-based FO methods, LAE [13], EASE [16], APER [26], and RanPAC [25], which insert lightweight adapters into the frozen backbone and train them by backpropagation; and (iii) the ZO adapter method ZO-FC [9], which trains adapters with forward passes only and is the closest to FlyCL. RISE [10] and FoRo [11] are not included because their code is not publicly available, so

TABLE II Class-incremental performance on CIFAR-100 (20 tasks) and ImageNet-R (40 tasks) without task identity or exemplars. Peak Memory is the peak GPU memory of the whole training run; Parameters is the size of the final inference model on ImageNet-R. ↑ means higher is the better, and ↓ means lower is the better. (Note, for fair comparison, we only compare to efficient CL with pre-trained frozen transformer-based AI model approaches)

Method	Peak Memory (GB) (↓)	Structured Sparsity	Parameters (M) (↓)	CIFAR-100			ImageNet-R		
				Avg (%) (↑)	Last (%) (↑)	Fgt (%) (↓)	Avg (%) (↑)	Last (%) (↑)	Fgt (%) (↓)
CODA-Prompt [14]	7.35	0%	89.8	89.12	81.82	7.29	66.77	58.08	5.60
L2P [15]	5.58	0%	172.0	86.02	79.38	9.13	70.16	63.23	6.44
LAE [13]	4.48	0%	86.0	89.29	84.34	8.60	73.57	66.35	10.14
EASE [16]	4.53	0%	139.5	89.04	82.93	7.33	70.81	61.20	7.66
APER [26]	4.31	0%	173.1	90.20	84.66	5.18	65.30	57.30	8.21
RanPAC [25]	4.25	0%	96.7	**93.57**	**89.43**	**4.24**	75.36	69.00	7.06
ZO-FC [9]	0.73	0%	86.0	87.95	81.65	5.73	72.00	66.52	**4.71**
FlyCL (ours)	**0.63**	50%	**62.6**	90.45	84.84	7.03	75.81	68.75	8.02
		30%	73.9	92.79	88.05	4.89	**78.48**	**71.92**	6.88

their reported numbers cannot be reproduced under our protocol.

Dataset. We evaluate FlyCL on CIFAR-100 [27] and ImageNet-R [28]. To compare with other CL methods, we strictly follow a split of CIFAR-100 and ImageNet-R into 20 and 40 tasks, respectively, with 5 classes per task, without task identity or exemplars. The adapters are trained once on the first task for 20 epochs on CIFAR-100 and 60 epochs on ImageNet-R and all subsequent tasks are handled by the closed-form updates in Section III. We denote the models pruned by 30% and 50% as FlyCL-0.3 and FlyCL-0.5, respectively.

Experiment Details. Peak memory in TABLE II is the peak GPU memory of the whole training run, captured with `torch.cuda.max_memory_allocated()` for every method on an NVIDIA Ada5000 GPU. For FlyCL, it covers the task-0 training, every recalibration and head update, and the bf16 copy of the dense $\mathbf{W}_2$ that serves as the regression target. The only state FlyCL keeps in host memory between tasks is the compact Gram statistics of Section III-C and the RP head accumulators (0.38 GB). We further verify the practical effectiveness of our method on an edge device, NVIDIA Jetson TX2, in Section IV-C. For a fair comparison, all other methods employ the same ViT-Base [2] backbone pre-trained on ImageNet-21k [20]. We insert a bottleneck adapter with hidden width 16 in parallel with the MLP of every transformer block. All other baselines use the same batch size of 48. For HiZOO optimization, we set the perturbation magnitude to $\mu = 0.1$, the query budget to $Q = 8$, the step size to $\eta = 1$, and the batch size to 64.

B. Main Result

1) Performance Comparison for the Training Phase

Classification accuracy. As shown in TABLE II, FlyCL achieves substantial backbone sparsity while maintaining competitive accuracy. On CIFAR-100, FlyCL-0.3 stays within 0.8% of the FO-trained RanPAC and outperforms all other FO baselines. On ImageNet-R, FlyCL outperforms every baseline, by 3.1% over RanPAC and 6.5% over ZO-FC. Pruning 30% of the channels even yields higher accuracy on ImageNet-R than the unpruned variant, which we attribute to the continual unlabeled recalibration of the compensated projections across tasks. Raising the sparsity to 40% and 50% degrades accuracy monotonically, yet even FlyCL-0.5 still exceeds several baselines (Fig. 4).

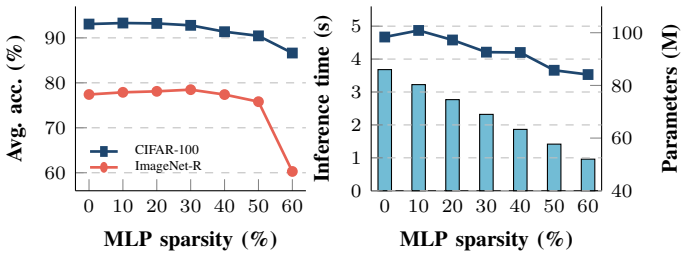


Fig. 4 Sweep of the MLP channel sparsity from 0% to 60%.

Forgetting. Because the adapters are trained only on the first task and every later update is closed-form, the forgetting of FlyCL at 30% sparsity is low (4.9% on CIFAR-100 and 6.9% on ImageNet-R). On CIFAR-100, only RanPAC forgets less, at 7× the memory; on ImageNet-R, only the prompt-based methods and ZO-FC forget less, at the cost of lower accuracy. Forgetting grows mildly with the pruning ratio, mainly because the final accuracy drops.

Peak memory usage. As shown in TABLE II, FlyCL reduces the peak GPU memory of training by up to 11.7× over the conventional CL methods. The saving comes from never storing the activations for a backward pass, which account for the largest share of training memory [29]; even the adapter-based baselines (LAE, EASE, APER, and RanPAC) with lightweight parameter-efficient fine-tuning (PEFT) modules exceed 4 GB.

Model size. The final model of FlyCL consists of the pruned backbone, 12 parallel adapters (0.3 M parameters, shared across all tasks), and the random-projection head, without any per-task state. As shown in Fig. 4, the backbone size decreases linearly with the pruning ratio. The other methods keep the dense backbone and add their tailored modules, so their total size in TABLE II is never below the dense ViT: L2P and APER keep a second frozen copy of the backbone, EASE stores one adapter set per task, and RanPAC needs a large random projection. At 30% sparsity, FlyCL is 24% smaller than RanPAC with higher accuracy.

2) Training and Inference Time

Training time. Fig. 5 (left) reports the wall-clock time to learn all 20 CIFAR-100 tasks. A ZO step is slower than an FO step [9], but FlyCL runs ZO training only once, on the first task, so its total training time is comparable to that of the FO methods

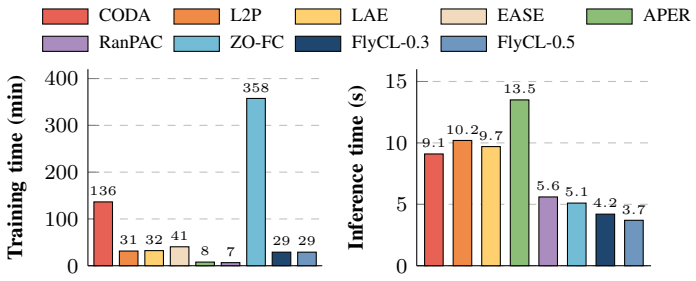


Fig. 5 Total training time over the 20 CIFAR-100 tasks (left) and inference time of the final model over the 10k test images (right). EASE (111 s) is omitted from the right panel for readability.

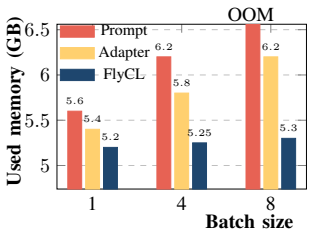


Fig. 6 Used memory for different CL methods on the edge GPU

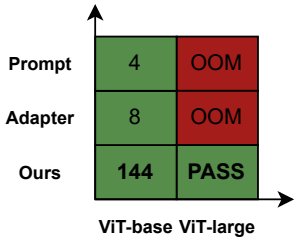


Fig. 7 Shmoo plot on the edge GPU

instead of 12× higher as for ZO-FC. The first session takes about 9 min, and every later task takes under one minute of forward passes. CODA-Prompt, EASE, LAE, and L2P (FO) and ZO-FC (ZO) update their modules on every task and therefore train longer. Only APER and RanPAC, which fine-tune the adapter with FO on the first task alone, train faster, at the cost of 6–7× higher peak memory (TABLE II).

Inference time. Fig. 5 (right) compares the time to classify the 10k test images with the final model. FlyCL is the fastest method: FlyCL-0.3 is 17% faster than ZO-FC and 25% faster than RanPAC, and FlyCL-0.5 widens the gap over ZO-FC to 28%. The prompt-based methods need an extra query forward pass and are 2.2–2.4× slower, APER runs both the original and the adapted backbone, and EASE passes every image through all 20 task-specific adapters, so its inference time grows linearly with the number of tasks. The structured sparsity of FlyCL thus turns the reduced computation into an end-to-end runtime gain.

C. Experiments on the Edge Device

As illustrated in Fig. 6, we evaluate the memory consumption of different CL methods on the NVIDIA Jetson TX2 (maximum usable memory is about 7.8 GB) using the ViT-Base model. Used memory is read from `tegrastats`, i.e., the total occupied unified memory, so it includes the host-side state of FlyCL. **Prompt** and **Adapter** denote the prompt-based and the FO adapter-based methods, which behave alike within each group. FlyCL has the lowest footprint at every batch size. When the batch size increases to 8, the prompt-based method encounters out-of-memory (OOM) failures, whereas the memory of FlyCL grows by only 0.1 GB from batch size 1 to 8.

To assess scalability, Fig. 7 reports the largest batch size each method supports. FlyCL trains ViT-Base with a batch size of 144, against 4 and 8 for the prompt-based and adapter-based methods, and it is the only method that trains the 3.6× larger ViT-Large on the device, where the FO methods run out of memory at any batch size.

D. Ablation Study

Component analysis. TABLE III adds the components of FlyCL one at a time at 30% MLP channel sparsity, starting from uniform saliency-based pruning without correction. The closed-form compensation (Comp.) of Section III-C is the most effective component, recovering 7.5% and 3.7% on CIFAR-100 and ImageNet-R without any training. Continual recalibration (Recal) adds 1.0% and 0.5% by re-solving the regression with

TABLE III Ablation of the FlyCL components at 30% sparsity (average accuracy $\bar{A}$ in %, ↑). FO: the same adapters trained by back-propagation instead of ZO (1.75 vs. 0.63 GB peak memory).

Comp.	Recal.	Sched.	Adapter	$\bar{A}_{\text{CIFAR}}$	$\bar{A}_{\text{INR}}$
				81.98	68.88
✓				89.46	72.62
✓	✓			90.46	73.13
✓	✓	✓		91.36	76.18
✓	✓	✓	✓	**92.79**	**78.48**
✓	✓	✓	FO	93.12	80.86

the Gram statistics of all tasks. The depth-decreasing schedule (Sched.) of Eq. 7 gains a further 0.9% and 3.1% at the same total sparsity, and the ZO first-session adapters (Adapter) of Section III-D add 1.4% and 2.3%. The complete flow ends within 0.3% of the unpruned backbone with the same adapters on CIFAR-100 and above it on ImageNet-R.

Cost of the ZO estimate. The last row of TABLE III isolates the price of estimating the adapter gradient from forward passes. Exact gradients raise the average accuracy by 0.3% on CIFAR-100 and 2.4% on ImageNet-R, so the forward-only estimate recovers most of the accuracy of back-propagation. However, the backward pass raises the peak training memory from 0.63 GB to about 2 GB, which is exactly the cost that FlyCL avoids.

V. Conclusion

We presented FlyCL, a forward-only continual learning framework that prunes a pre-trained transformer from unlabeled forward statistics, adapts it with one-shot ZO training, and handles later tasks in closed form, so no backpropagation is ever required. With 30% of the MLP channels pruned, FlyCL trains within 0.63 GB, up to 11.7× less than existing CL methods, while matching the most accurate baseline on CIFAR-100 and ImageNet-R, and it is the only method that trains ViT-Large on a Jetson TX2. Future work will accelerate the first-session ZO training and extend FlyCL beyond image classification.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, 2017.

[2] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021.

[3] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *International Conference on Learning Representations*, 2022.

[4] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of learning and motivation*. Elsevier, 1989, vol. 24, pp. 109–165.

[5] 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*, 2022.

[6] Z. Wang, F. Cheng, K. Ji, and B. Huang, "Lightcl: Compact continual learning with low memory footprint for edge device," in *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, 2025.

[7] J.-J. Hung, Y.-J. Chen, H.-Y. Cheng, H. Kao, and C.-L. Yang, "Filter-based adaptive model pruning for efficient incremental learning on edge devices," in *2025 Design, Automation & Test in Europe Conference (DATE)*, 2025.

[8] Y. He and L. Xiao, "Structured pruning for deep convolutional neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[9] W. Yu, Z. Wang, S. Niu, S. Lin, and L. Yang, "More than memory savings: zeroth-order optimization mitigates forgetting in continual learning," in *2026 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2026.

[10] Y. Shu, J. Mu, and Z. Dai, "Why zeroth-order adaptation may forget less: A randomized shaping theory," *arXiv preprint arXiv:2605.10658*, 2026.

[11] J. Chen, J. He, F. Chen, Z. Lv, and J. Tang, "Forward-only continual learning," in *Proceedings of the 33rd ACM International Conference on Multimedia*, 2025.

[12] S. Lin, L. Yang, D. Fan, and J. Zhang, "TRGP: Trust region gradient projection for continual learning," in *International Conference on Learning Representations*, 2022.

[13] Q. Gao, C. Zhao, Y. Sun, T. Xi, G. Zhang, B. Ghanem, and J. Zhang, "A unified continual learning framework with general parameter-efficient tuning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

[14] J. S. Smith, L. Karlinsky, V. Gutta, P. Cascante-Bonilla, D. Kim, A. Arbelle, R. Panda, R. Feris, and Z. Kira, "Coda-prompt: Continual decomposed attention-based prompting for rehearsal-free continual learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023.

[15] Z. Wang, Z. Zhang, C.-Y. Lee, H. Zhang, R. Sun, X. Ren, G. Su, V. Perot, J. Dy, and T. Pfister, "Learning to prompt for continual learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022.

[16] 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.

[17] D.-W. Zhou, H.-L. Sun, J. Ning, H.-J. Ye, and D.-C. Zhan, "Continual learning with pre-trained models: A survey," in *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, 2024.

[18] Y. Zhao, S. Dang, H. Ye, G. Dai, Y. Qian, and I. Tsang, "Second-order fine-tuning without pain for llms: A hessian informed zeroth-order optimizer," in *International Conference on Learning Representations*, vol. 2025, 2025.

[19] J. C. Spall, "Multivariate stochastic approximation using a simultaneous perturbation gradient approximation," *IEEE Transactions on Automatic Control*, 1992.

[20] T. Ridnik, E. Ben-Baruch, A. Noy, and L. Zelnik-Manor, "Imagenet-21k pretraining for the masses," in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

[21] M. Sun, Z. Liu, A. Bair, and Z. Kolter, "A simple and effective pruning approach for large language models," in *International Conference on Learning Representations*, vol. 2024, 2024.

[22] M. Raghu, T. Unterthiner, S. Kornblith, C. Zhang, and A. Dosovitskiy, "Do vision transformers see like convolutional neural networks?" *Advances in neural information processing systems*, 2021.

[23] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," in *2021 IEEE/CVF international conference on computer vision (ICCV)*, 2021.

[24] E. Frantar and D. Alistarh, "Sparsegpt: Massive language models can be accurately pruned in one-shot," in *International Conference on Machine Learning*, 2023.

[25] M. D. McDonnell, D. Gong, A. Parvaneh, E. Abbasnejad, and A. van den Hengel, "Ranpac: Random projections and pre-trained models for continual learning," in *Advances in Neural Information Processing Systems*, vol. 36, 2023.

[26] D.-W. Zhou, Z.-W. Cai, H.-J. Ye, D.-C. Zhan, and Z. Liu, "Revisiting class-incremental learning with pre-trained models: Generalizability and adaptivity are all you need," *International Journal of Computer Vision*, 2025.

[27] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[28] D. Hendrycks, S. Basart, N. Mu, S. Kadavath, F. Wang, E. Dorundo, R. Desai, T. Zhu, S. Parajuli, M. Guo *et al.*, "The many faces of robustness: A critical analysis of out-of-distribution generalization," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[29] S. Malladi, T. Gao, E. Nichani, A. Damian, J. D. Lee, D. Chen, and S. Arora, "Fine-tuning language models with just forward passes," *Advances in Neural Information Processing Systems*, 2023.