A. Casual Tracking for Localization

In this paper, the weight selection is localized to the first MLP layer within transformer blocks. We discussed such localization in prior model editing and probing works in Section 4. We further perform casual tracking to validate the localization.

Vig et al. [54] quantifies the contribution of intermediate variables in causal graphs for causal mediation analysis. Based on this, Meng et al. [42] proposed casual tracking for identifying neuron activations that are decisive in a language model's factual predictions. Casual tracking identifies specific locations that contribute to the input's recognition by computing the average effects of restoring activations at these locations on a corrupted input. We adapt the casual tracking to CLIP models and formulate the computation of average indirect effect (AIE) in the following.

In CLIP model with ViT backbone, we freeze one tower of visual or text and perform casual tracking on the other. Take the casual tracking on image tower for example, we do the next three runs:

- **Clean run.** We pass an image-text pair into the model and store activations of the visual tower $\{a_i^\ell | i \in [1, T], \ell \in [1, L]\}$, and get the similarity score S. Here T is the number of tokens, and L is the number of layers.
- **Corrupted run.** We then pass the image into visual tower by adding noises to the image embeddings of patches related to the corresponding text and get a corrupted visual output feature. We compute the similarity score S_c between this feature and the clean text features.
- **Corrupted-with-restoration run.** Finally, we follow the corrupted run to add noises on image embeddings, and replace the activation of layer ℓ token i with the clean activation a_i^ℓ, and get corrupted-with-restoration visual output features. We compute the similarity score ${S_r}_i^\ell$ between these features and the clean text features.

The average indirect effect (AIE) is computed by the average difference between the similarity scores of corrupted run and corrupted-with-restoration runs, *i.e.*

$$\text{AIE}^\ell = \frac{1}{T} \sum_{i \in [1,T]} \frac{\left| {S_r}_i^\ell - S_c \right|}{S}. \tag{7}$$

Here $\left| {S_r}_i^\ell - S_c \right|$ measures the change of similarity scores when we restore one single state, *i.e.*, activation, back to the clean activation. In CLIP model, we observe that this restoration often does not lead to positive effect to the similarity score; thus we compute the absolute change here. As we need to aggregate AIE over multiple image-text pairs, we normalize the change of similarity by the score from the clean run. In casual tracking of text tower, we freeze the CLIP visual tower and apply the same procedure on the text tower.

Intuitively, higher AIE^ℓ means activations or states of layer ℓ are more important to the final classification. We further compute AIE over MLP layers $\text{AIE}^\ell_{\text{mlp}}$ or Attention layers $\text{AIE}^\ell_{\text{attn}}$ by restoring activation values outputted from MLP or Attention layers among all the transformer blocks.

In practice, we perform casual tracking on the validation set of COCO [35] since it provides detailed information on objects in images. We decide the object-related image patch by the bounding box information. We use the prompt `a photo of {class name}` as text input, and the image-related tokens are those that represent the class name. The casual tracking results are in Fig. 2. Here we show the effect of restoring states (activations) after full layer in blue, the effect of restoring states after Attention layers in orange, and the effect of restoring states after MLP layers in green.

The figure demonstrates higher $\text{AIE}^\ell_{\text{mlp}}$ values compared to $\text{AIE}^\ell_{\text{attn}}$ values in both visual and text tower, with a larger contrast in the visual tower. This implies the change of MLP layers contributes more to the final classification, which further validates our choice in performing selection on the MLP layers.

B. Visualization for Parameter Selection

In addition to section 5.3, we further validate the parameter selection qualitatively from two perspectives. Firstly, we utilize gScoreCAM [13] to visualize the attention of selected neurons on original images, illustrating their representativeness to the features. Secondly, we visualize the correlation of selected weights from different tasks. This is to demonstrate the task-wise separation in the selection process, which aids in mitigating forgetting.

gScoreCAM [13] follows the idea of ScoreCAM [56] to perturb the input image with the upsampled activation map, and aggregate the CAM scores. The importance of neuron activations to specific input features is derived from the aggregated scores. gScoreCAM selects only 10% of the activations in regard to their gradient values to perturb the input image, and shows the selected activations are effective in localizing the features. This is in agreement with our selection strategy and modularity hypothesis (section 4). We applied gScoreCAM on the neurons of the first MLP layers in CLIP visual tower and selected the top 10% activation values to perturb the input image. We show the highlighted regions by selected activations of images from the CUB dataset in Fig. 3. We perform the visualization on the neurons of the first MLP layers of the 9th transformer layers.

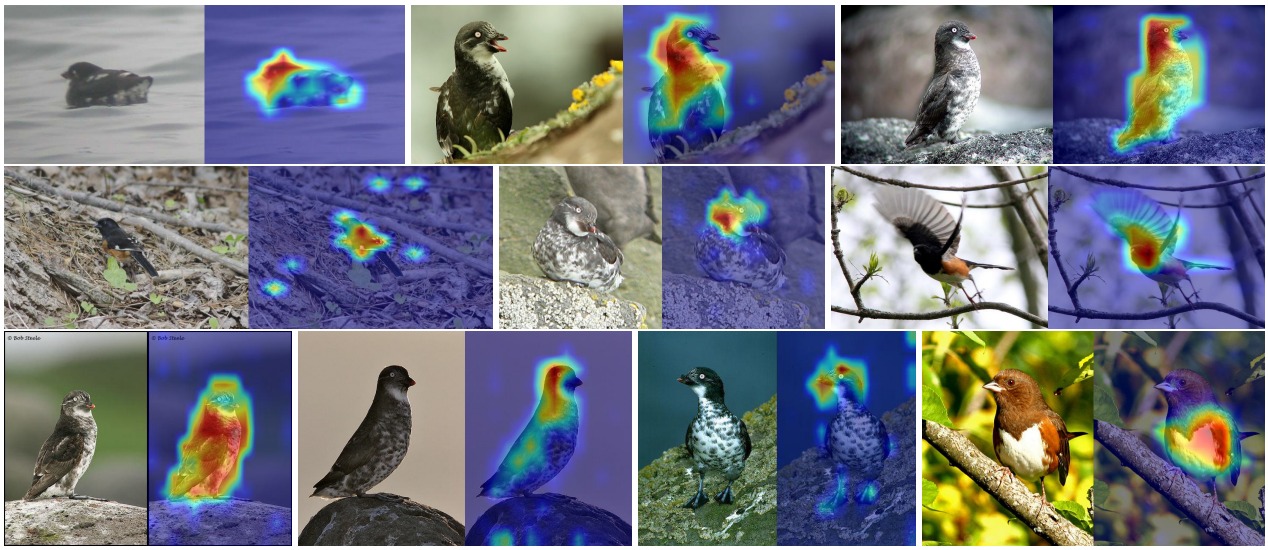

Figure 3. Highlighted regions by activations of selected neurons in the first MLP layers of the 9th transformer block in gScoreCAM visualization. Selected neurons represent meaningful features in the input image.

In Fig. 4, we analyze the correlation between selected weights across different tasks in various layers. Patch with row label task i and column label task j shows the percentage of the shared weights selected in task i and task j. We observe that the frequency of repeated weight selection for different tasks seldom exceeds 50%. This pattern suggests that while our scoring function occasionally identifies common weights across tasks, it predominantly selects task-specific weights. Notably, the incidence of repeated selection decreases in shallower layers, as demonstrated in layer 5 (first row), which implies a higher occurrence of modulation in these layers. Thus, our approach of selecting weights across all transformer layers is further validated.

C. Learnable Scoring Function

In section 5.3, we perform different scoring functions to validate the effectiveness of our proposed gradient-based scoring function, including the Mask baseline. Here, we describe the Mask baseline.

Although the gradient is an efficient approximation of the parameters' relevance to the task at hand, we suspect selecting parameters independently based on their gradient magnitude might not consider the contribution of the parameters together when updated, and can potentially cause redundancy in the selection. To explore this, we propose to involve an optional optimization stage to adjust the scoring function based on the initial gradient values. Specifically, for parameters $\theta^l \in \mathbb{R}^{m\times n}$, we define $\boldsymbol{S} \in \mathbb{R}^{m\times n}$ to be the learnable parameters scores. We initialize $\boldsymbol{S}$ with the gradients computed on the current task, where $\boldsymbol{S}_{ij} = \frac{1}{N_t'}\sum_{k=1}^{N_t'} g_{ij}(x_k)$. We consider the estimated gradient as the basis for a target update of the model parameters and construct an imaginary update:

$$\theta^{l\prime} = \theta^l - \mu \cdot \boldsymbol{S}, \tag{8}$$

where μ is the update step size (learning rate). We then optimize $\boldsymbol{S}$ by minimizing the task loss $\mathcal{L}$ and an additional L_1 loss ($\|\boldsymbol{S}\|_1$)

$$\boldsymbol{S}' = \arg\min_{\boldsymbol{S}} \mathcal{L}(\theta^{l\prime}; D^t) + \lambda\|\boldsymbol{S}\|_1, \tag{9}$$

where λ is a hyperparameter that weighs the contribution of L_1 loss. L_1 loss is introduced to encourage sparsity in the estimated scores, guiding the optimization to tolerate parameters with large gradient magnitude (and hence large initial scores) when proven relevant to the minimization of the task loss while zeroing out gradients of irrelevant or redundant parameters.

We optimize $\boldsymbol{S}$ for a few epochs. Then, we define $\mathcal{S}(\theta^l_{ij}, D^t) = \boldsymbol{S}'_{i,j}$ and select top r parameters as the most relevant parameters for the task at hand. Note that here we estimate parameter scores for one selected layer θ^l, but the formulation can generalize to an arbitrary number of layers.

The learnable scoring function requires more computation due to the additional optimization phase of $\boldsymbol{S}$ compared to the gradient scores. We present the efficacy of this optional stage in Table 3 of the main paper.

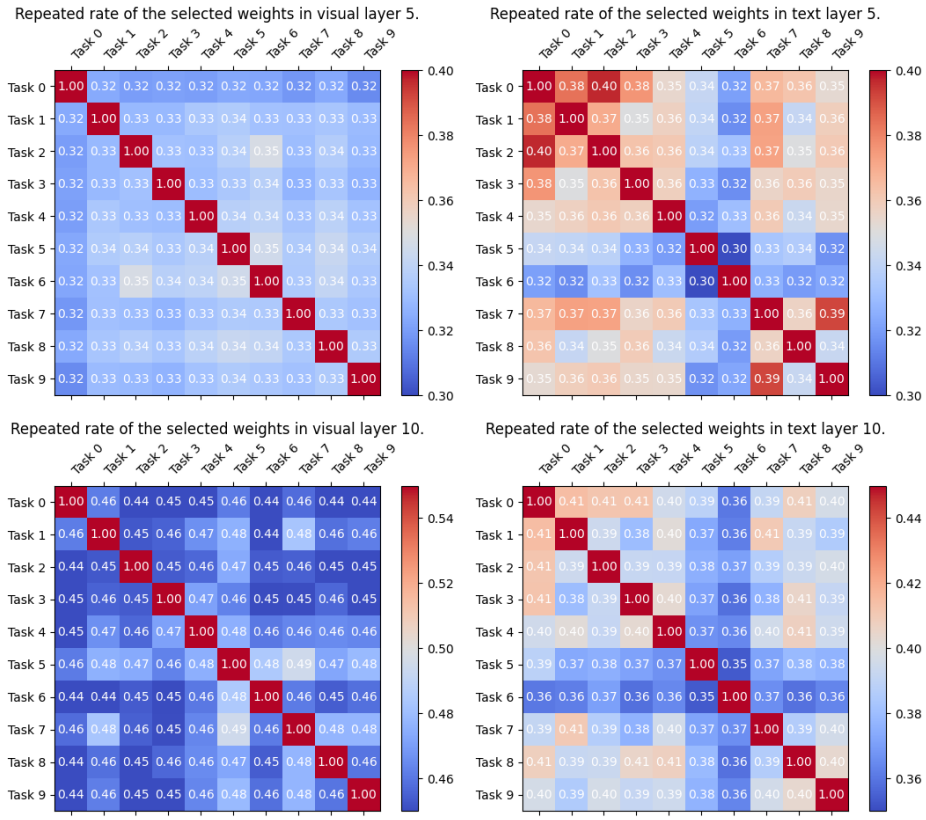

Figure 4. Repeat rate of the selected weight in visual and text tower of layer 5 and layer 10 in CLIP. The shared weight selected two different tasks only counts a small amount of total selected weight.

D. Relations to other sparse update works

Gradient-based selection. Gradient-based parameter attribution is a robust metric that has been widely used in CL and other fields like model compression and multi-task learning; however, when and where to use it is the key point to stand our method out. HAT [50], SupSup [62], and SPG [30] leverage the *gradients after training* of previous tasks to penalize the change of previously learned important parameters, which are more similar to EWC [29] and MAS [2]. We applied SPG in our setting and show significant improvements by our method in all metrics in **??** More importantly, in our case, gradients of previous data (pre-training data) are inefficient or unavailable to obtain and noisy. Distinctly, we achieve knowledge retention via sparse updates by two key steps, localization and parameter *pre-selection*. Our gradient-based pre-selection serves as an approximation to identify the specialized parameters of the upcoming task, which is crucial to allow large decrease in the loss function with the smallest change in the selected parameters.

Sparse network. PiggyBack [40] learns a task-specific binary mask for every task, requiring task identifiers which we assume inaccessible. SparseCL [58] compresses the model by 75%-95% for on-device CL, and would fail to fit in our problem in two ways. 1) During training of a task, SparseCL combines the magnitude of the parameters and their gradients as a score to omit unimportant parameters in Equation (1). The magnitude of the parameter $\|w\|_1$ is relevant when training a network from scratch and in SparseCL it is the dominant factor in parameter selection; after reproducing SparseCL experiment, we found that the magnitude of weights $\|w\|_1$ is on average 6.93e-3 while the gradient $\alpha\|\frac{\partial\tilde{\mathcal{L}}(\mathcal{D}_t;\theta)}{\partial w}\|_1$ is on average 2.02e-4. However, in a pre-trained foundation model, the magnitude of weights is mostly relevant to the knowledge learned during pre-training. Our gradient-based selection is to measure the relevance of parameters to the *upcoming task*, and the sparse update is not to compress the network but to discourage the unrelated parameters to be modified. 2) SparseCL dynamically selects parameters to be updated every several epochs; selecting additional parameters and omitting from already changed parameters leads, at the end of the task training, to many parameters changed. This would incur more forgetting of generic knowledge. In Tab. 1 SparseCL fails to improve Acc. and causes drop in generic knowledge (C.) specially when learning generic datasets.

E. Implementation Details

E.1. Dataset

Here are the statistics of our six experimental benchmarks.

Birdsnap [5] Birdsnap is a large bird dataset originally consisting of 49,829 images from 500 bird species with 47,386 images used for training and 2,443 images used for testing. We download the dataset from the official link. We follow the official train-test split. We use a fixed buffer of size 1,500 for this dataset.

CUB-200-2011 [55] The Caltech-UCSD Birds-200-2011 (CUB-200-2011) dataset is for fine-grained visual categorization task. It contains 11,788 images of 200 subcategories belonging to birds, 5,994 for training and 5,794 for testing. We use the Hugging Face implementation of the dataloader. We use a fixed buffer of size 240 for this dataset.

CIFAR100 [32] This dataset has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. We use the PyTorch implementation of the dataloader. We used a fixed buffer of 2,000 for this dataset.

FGVC-Aircraft [39] The dataset contains 10,200 images of aircraft, with 100 images for each of 102 different aircraft model variants, most of which are airplanes. The data is divided into three equally sized training, validation, and test subsets. We use the PyTorch implementation of the dataloader, where train and valid set are used for training, and the test set is used for testing. We use a fixed buffer of size 250 for this dataset.

Stanford Cars [31] The Cars dataset contains 16,185 images of 196 classes of cars. The data is split into 8,144 training images and 8,041 testing images, where each class has been split roughly in a 50-50 split. Classes are typically at the level of Make, Model, Year, *e.g.*, 2012 Tesla Model S or 2012 BMW M3 coupe. We use the Hugging Face implementation of the dataloader. We use a fixed buffer of size 240 for this dataset.

GTSRB [52] This dataset is designed for recognition of traffic signs. By the time we download it, it contains 43 classes with 26,640 training samples and 12,630 testing samples. We use the PyTorch implementation of the dataloader. We used a fixed buffer of 1,000 for this dataset.

For each dataset, during the training, we use the prompt `a photo of {}` with class name as text inputs. We evaluate each baseline on the test set using the original prompts and ensembling strategy provided by Radford et al. [48].

E.2. Hyper-parameters

For our algorithm, we use PyTorch implemented AdamW optimizer [38] and learning rate scheduler of Cosine Annealing with Warmup [37] for our algorithm, as well as FLYP combined with ER and other CL regularization methods. We use a learning rate of 7.5e-6 and train for 10 epochs for all datasets. We report results based on an average of 5 different random seeds. We run all our experiments on one single Nvidia A100 GPU.

E.3. Baseline Details

Here are the implementations for other baselines in Table 1.

FLYP [21] For all FLYP based baselines, we tuned the learning rate in [2.5e-6, 5e-6, 7.5e-6] and training epochs in [5,10,15] and report the best results

FLYP+ER [11] For ER-based baselines, we apply balanced sampling, where at each step, we sample a balanced batch, half from the current task and half from the previous tasks.

FLYP + MAS [2] We follow the avalanche [36] to implement MAS regularizer with FLYP. To cope with the large-scale architecture, we normalize the estimated weights' importance by their maximum value. We tuned the scaling factor of MAS loss in [0.01, 0.05, 0.1] and report the best results.

FLYP + ER + LwF/PRD [4, 34] For LwF, we follow the implementation of avalanche. For PRD, we follow the official implementation. We further tuned temperature in [0.01,0.1,1.0,5.0] and loss scaling factor in [0.01, 0.05, 0.1] and report the best results.

L2P, DualPrompt [59, 60] These two methods were originally designed for ViT backbone with linear classifier. They proposed to freeze the feature extractor and only train the classifier. We adopt the idea to the backbone of CLIP architecture, where we freeze the visual and text feature extractors and only train the linear projection layers. We applied the prompt techniques on the visual tower of CLIP. We deploy them with the CLIP pre-trained weights provided by `timm` library. We further applied a class balanced buffer to them as what we did for FLYP + ER. These methods are highly tailored for ImageNet pretrained transformers and do not scale to other pretrained weights, leading to surprisingly bad performance when combined with CLIP, in spite of our best efforts to tune the hyperparameters carefully.

SLCA [65] We adopted the slow learning rate and classifier alignment to the CLIP backbone. In the first training phase, we applied the learning of $1.5e-6$ to the backbone, and $7.5e-6$ to the projection layers. In the section training phase of

classifier alignment, we only train the projection layers.

LoRA-EWC [63] We compute fisher information by CC12m [9], and apply the EWC loss on every task. We applied the LoRA architecture on both visual and text tower. We further modified it with different ranks in LoRA and applied a replay buffer.

Here are other baselines in Table 2.

Random This baseline mainly follows our method SPU, except for Equation 2 in the main paper. In this baseline, we use random values for the scoring function.

Mask We described this method in **??** in the supplement. We optimize the learnable score matrix $\boldsymbol{S}$ for 5 epochs, with the learning rate of $5e-4$ and step size $\mu = 5e-4$. We set the L_1 loss coefficient $\lambda = 1e-3$

PiggyBack [40] This baseline mainly follows the Mask baseline, except for the format of the imaginary update in Appendix C. We applied the PiggyBack mask learning format, where

$$\theta^{l\prime} = \theta^{l} + \mu \cdot \mathrm{m}(\boldsymbol{S}). \tag{10}$$

Here μ is a scaling factor, and $\mathrm{m}(\cdot)$ is a binary mask. We uniformly initialized $\boldsymbol{S}$ and applied AdamW optimizer as proposed in PiggyBack. We optimize the score matrix $\boldsymbol{S}$ for 5 epochs, with the learning rate of 1e-4 and the scaling factor μ of 1e-5.

F. More Details in Ablation Study

	Aircraft			Birdsnap			Cars			CIFAR100			CUB			GTSRB			Average		
	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc. In.	Avg. F.	C. Drop
w/ Loc.	44.43	14.42	63.48	55.35	12.78	61.94	77.51	3.26	63.42	83.99	-0.39	61.38	71.51	4.84	62.87	94.25	-7.87	62.55	**21.34**	**4.51**	**0.94**
w/o Loc.	43.35	16.12	63.54	54.56	14.37	61.08	76.83	4.30	63.26	84.43	-0.26	59.95	71.64	4.56	62.42	93.81	-7.75	61.53	20.93	5.22	1.59

Table 7. Comparison between w/ localization and w/o localization. The localization improves the retention ability.

Knowledge Retention. Our good retention ability is dually contributed by the localization and selective update. We localize the change to the first MLP layer, and keep other model components unchanged for knowledge retention. In Tab. 7, selective update without localization (w/o Loc.) results in less retention. Besides localization's role in retention and the sparse updates, the parameters randomly selected generally have smaller gradients magnitudes regarding the task-in-hand; thus under same learning rate and number of epochs, the magnitude of parameters change can be smaller, which helps in retention.

	Aircraft			Birdsnap			Cars			CIFAR100			CUB			GTSRB			Average		
	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc. In.	Avg. F.	C. Drop
Weight	44.43	14.42	63.48	55.35	12.78	61.94	77.51	3.26	63.42	83.99	-0.39	61.38	71.51	4.84	62.87	94.25	-7.87	62.55	**21.34**	4.51	0.94
Neuron	44.13	14.02	63.60	55.32	12.66	62.77	77.46	3.33	63.62	83.98	-0.79	61.84	71.14	5.16	63.23	93.54	-8.15	63.22	21.09	**4.37**	**0.50**

Table 8. Comparison between weight-based selection and neuron-based selection. Our method employs weight selection and has better learning ability.

Neuron-based Selection. We propose to compute the element-wise importance scores by Equation 2 in the main paper to facilitate weight-based selection. Whereas, Aljundi et al. [3] put forth a technique to calculate row-wise importance scores to perform neuron-based selection.

Tab. 8 shows the full results of variants of selection strategy, where the gray row represents our strategy. In the baseline named "Weight" we compute an element-wise scoring function by Equation 2, and select the top 10% entries of each weight matrix to update. In the baseline named "Neuron", we compute a row-wise scoring function based on the row summation of the element-wise scoring function by Equation 2. Then we select the 10% rows of each weight matrix to update.

We find that weight-based selection yields slightly improved learning performance while exhibiting a marginal decrease in hold-out accuracy. Nonetheless, the overall performance trends remain comparable between the two strategies. This observation highlights the robustness of our localization and importance scoring methods to any of the selection strategy.

Selection Rate. In Section 5.3, we present the average results of our method under varying selection rates. Tab. 9 shows the full results, where the gray row represents our reported results. Our main results select the top 10% elements localized layer. We compare to the baselines where the top 1% or the top 50% are selected for update. All other configurations are kept the same.

	Aircraft			Birdsnap			Cars			CIFAR100			CUB			GTSRB			Average		
	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc. In.	Avg. F.	C. Drop
0.01	37.64	11.45	63.54	53.49	9.87	62.25	74.62	2.20	63.40	83.79	-1.64	60.91	66.79	4.39	63.01	88.89	-7.71	61.53	17.70	**3.10**	1.11
0.10	44.43	14.42	63.48	55.35	12.78	61.94	77.51	3.26	63.42	83.99	-0.39	61.38	71.51	4.84	62.87	94.25	-7.87	62.55	21.34	4.51	**0.94**
0.50	46.73	20.74	63.56	53.96	17.98	61.72	77.64	6.12	63.48	83.47	1.51	61.53	71.89	8.06	62.85	95.74	-7.85	62.43	**21.73**	7.76	0.95

Table 9. Full results of ablation on selection rate. Our method select 10% weights, achieving better trade-off in learning and forgetting.

Buffer Size. In Section 5.3, we present the average results of our method and FLYP + ER under varying buffer size. We study buffer sizes of 1%, 2% and 4% of the total dataset size. Tab. 10 shows the full results of buffer size ablation. We report our method with 4% buffer size of the total dataset size in Table 1 in the main paper, highlighted in gray.

Method	Buffer Size / Total Size	Aircraft			Birdsnap			Cars			CIFAR100			CUB			GTSRB			Average		
		Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc. In.	Avg. F.	C. Drop
ER	1%	27.76	44.49	49.22	43.76	33.59	55.90	61.22	21.19	55.34	73.70	13.34	40.14	53.39	25.26	48.81	93.03	-4.22	16.79	8.97	22.27	19.18
ER	2%	33.42	41.37	49.74	49.96	29.20	56.35	62.83	21.45	57.35	78.72	7.94	41.74	57.90	22.84	50.83	95.64	-6.68	15.82	13.24	19.35	18.24
ER	4%	41.42	31.48	50.41	56.22	21.63	56.72	69.08	16.42	58.07	82.86	3.41	42.10	64.07	17.72	51.30	96.28	-7.40	17.34	18.48	13.88	17.56
SPU	1%	37.82	21.96	63.56	47.54	23.61	61.65	73.68	6.84	63.36	80.44	4.87	61.49	66.06	9.32	62.47	90.55	-4.93	62.76	16.18	10.28	1.00
SPU	2%	40.65	20.31	63.44	51.33	18.97	61.90	75.00	6.17	63.41	82.45	2.03	61.36	68.39	8.42	62.81	92.99	-7.04	62.63	18.63	8.14	0.96
SPU	4%	44.43	14.42	63.48	55.35	12.78	61.94	77.51	3.26	63.42	83.99	-0.39	61.38	71.51	4.84	62.87	94.25	-7.87	62.55	21.34	4.51	0.94

Table 10. Full results of ablation on the buffer size. Our method shows superior performance over ER even in smaller buffer scenarios.

Method	Task Length	Aircraft			Birdsnap			Cars			CIFAR100			CUB			GTSRB			Average		
		Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc. In.	Avg. F.	C. Drop
ZSCL	10	30.96	15.65	**65.53**	49.85	13.28	63.13	67.79	8.27	62.90	80.50	1.05	**61.90**	61.09	7.69	62.78	62.92	13.54	62.92	9.01	9.91	**0.36**
ER	10	41.42	31.48	50.41	56.22	21.63	56.72	69.08	16.42	58.07	82.86	3.41	42.10	64.07	17.72	51.30	96.28	-7.40	17.34	18.48	13.88	17.56
SPU	10	44.43	14.42	63.48	55.35	12.78	61.94	77.51	3.26	63.42	83.99	-0.39	61.38	71.51	4.84	62.87	94.25	-7.87	62.55	21.34	4.51	0.94
ZSCL	20	28.23	28.81	62.92	43.23	20.23	62.83	69.67	11.21	62.56	68.05	21.21	55.17	60.55	16.15	62.15	15.40	33.43	55.82	-2.32	21.84	3.31
ER	20	5.67	37.93	47.99	53.53	28.12	55.80	65.71	22.60	52.98	81.73	9.74	32.58	61.58	23.25	47.21	94.80	-0.33	9.72	15.66	20.22	22.50
SPU	20	39.60	13.95	63.70	54.52	13.12	62.41	75.13	6.40	63.01	83.77	3.43	61.33	68.57	8.32	62.78	92.53	-2.37	62.27	19.18	7.14	0.97

Table 11. Full results of ablation on the task length. Our method shows superior performance over ER even in longer task scenarios.

Task Length We perform experiments on 20-split datasets and compare our method with ER (second-best Acc. In.) and ZSCL (best C.) in Tab. 11. The gap between SPU and ER/ZSCL becomes larger, as shown in blue value, than that in 10-split experiments. With 20 tasks, SPU has almost no drop in performance compared to 10 tasks, while ER and ZSCL have negative overall performance (Acc. In. - C. Drop).

G. More Details in Efficiency

	Aircraft			Birdsnap			Cars			CIFAR100			CUB			GTSRB			Average		
	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc.	F.	C.	Acc. In.	Avg. F.	C. Drop
one batch	44.42	14.40	63.50	55.02	13.33	62.04	77.41	3.36	63.40	83.99	-0.38	61.36	71.48	4.90	62.86	94.14	-7.79	62.52	21.24	4.64	**0.94**
0.25	44.43	14.42	63.48	55.35	12.78	61.94	77.51	3.26	63.42	83.99	-0.39	61.38	71.51	4.84	62.87	94.25	-7.87	62.55	21.34	4.51	**0.94**
0.50	44.33	14.48	63.48	55.31	12.73	61.88	77.54	3.16	63.44	84.03	-0.41	61.37	71.67	4.63	62.87	94.24	-7.82	62.58	21.35	4.46	**0.94**
1.00	44.40	14.40	63.47	55.28	12.61	61.85	77.61	3.12	63.44	84.05	-0.40	61.35	71.66	4.64	62.87	94.27	-7.81	62.58	**21.37**	**4.43**	0.96

Table 12. Full results of ablation on the number of samples to compute the gradient approximation. Our scoring function can efficiently cope with only one-batch gradient accumulation.

Number of samples to compute gradient approximation. In Equation 2, we accumulate the gradients of N'_t samples to approximate the importance. Here we ablate the effect of accumulating gradients with one batch (128 data points), 25% 50%, and 100% of the current set. We compute the importance score by the accumulated gradients before the training of every task, and the computational cost per task gets reduced with fewer samples to approximate the scoring function. With more samples, the accuracy is slightly increased, with also slight decrease in forgetting. Our algorithm is robust to all different configurations in general. Full results are shown in Tab. 12. We choose to report our main results by accumulate gradients of 25% samples of the current set, highlighted in gray.