Advance Sustainable Science, Engineering and Technology (ASSET)
Vol. 8, No.2, April 2026, pp. 02602020-01 ~ 02602020-012
ISSN: 2715-4211DOI:https://doi.org/10.26877/asset.v8i2.2787

Comparative Evaluation of Parameter-Efficient Fine-Tuning Strategies for Continual Image Classification

Nancy Agarwal[1], Alok Singh Chauhan[1*], Patrick Bours[2]

[1]School of Computer Applications and Technology, Galgotias University, Greater Noida 203201, India

[2]Department of Information Security and Communication Technology, Norwegian University of Science and Technology (NTNU), 2815 Gjøvik, Norway

*alok.chauhan@galgotiasuniversity.edu.in

Abstract: Catastrophic forgetting remains a key challenge in continual transfer learning, where a model's performance on earlier tasks degrades after sequential adaptation to a new task. While full fine-tuning updates all parameters and may achieve robust performance on the new task, it is computationally expensive and prone to true forgetting. In this work, we compare prominent parameter-efficient fine-tuning (PEFT) strategies, namely, adapters, additive learning, side-tuning, LoRA, and zero-initialized layers, against full fine-tuning on CIFAR-100 using a two-stage sequential protocol with a fixed class split: task-A (classes 0-49) followed by task-B (classes 50-99), evaluated on both Resnet-18 and Resnet-50. We report mean ± standard deviation over three independent runs (n = 3) and quantify retention using a Swapback-based recall procedure that separates apparent forgetting from true forgetting (Δ).Across both backbones, all PEFT methods preserve task-A knowledge with Δ = 0.00, whereas full fine-tuning shows non-zero true forgetting (Δ = 0.31 on Resnet-18 and Δ = 0.20 on Resnet-50). PEFT achieves competitive task-B performance while training only 0.22–4.49% of the parameters (versus 100% for full fine-tuning). Notably, on Resnet-50, LoRA attains the best task-B accuracy (Acc_B= 0.82) while updating 0.93% parameters and maintaining zero forgetting, slightly outperforming full fine-tuning (Acc_B= 0.81). These results show that PEFT methods provide a favorable accuracy–efficiency–stability trade-off and can outperform full fine-tuning in deeper architectures, making them strong candidates for scalable continual transfer learning.

Keywords: Catastrophic forgetting, Parameter-Efficient Fine-Tuning (PEFT), Continual Learning, Transfer Learning, Adapters, LoRA, Resnet-18, Resnet-50, CIFAR-100

(Received 2025-09-20, Revised 2025-11-04, Accepted 2026-02-24, Available Online by 2026-03-18)

1. Introduction

Transfer learning is one of the dominant approaches in modern machine learning, enabling pre-trained models to be adapted efficiently to new related domains or sub-tasks[1], [2]. In conventional transfer learning, a large-scale deep learning model trained on a source dataset is fine-tuned on a target dataset, often yield superior performance compared to training from scratch. However, when tasks arrive sequentially, the problem extends beyond transfer learning into the domain of continual learning, where a model must acquire new knowledge without forgetting the previously learned information[3], [4]. This sequential continual learning paradigm introduces the well-known challenge of catastrophic forgetting[5], [6]. Here, performance on earlier tasks significantly deteriorates as the model adapts to later tasks.

In continual learning, several families of methods have been explored to mitigate catastrophic forgetting, including rehearsal-based approaches (storing and replaying prior samples, or using generative replay), regularization-based approaches (penalizing changes to parameters deemed important for earlier tasks), and architecture-based methods (expanding capacity or isolating task-specific components to reduce interference) [7], [8], [9], [10]. While these baselines are well studied, they often impose added memory, computing, or system-level complexity, which can be undesirable in practical transfer and deployment pipelines.

Parameter-efficient fine-tuning (PEFT) offers an attractive alternative. Its methods are developed primarily in the context of transfer learning aimed at reducing the cost of adapting large backbones, by keeping most parameters frozen and training only lightweight modules (e.g., adapters, side-tuning blocks) or structured low-rank updates (e.g., LoRA)[11], [12], [13]. These techniques have been comparatively under-explored in continual settings where retaining task-A knowledge after learning task-B is a first-class requirement. In this paper, we study PEFT from a continual-learning perspective, highlighting its dual potential: (i) substantially reducing the number of trainable parameters and training cost, and (ii) mitigating catastrophic forgetting by constraining updates and/or isolating task-specific adaptations.

We provide a controlled comparison of multiple PEFT strategies under a two-stage continual transfer protocol, and we evaluate knowledge retention using a swapback-based recall analysis that separates apparent forgetting from true forgetting (Δ). Unlike common evaluations that report only post-training accuracy, our analyses directly test whether task-A knowledge is preserved after sequential adaptation.

The work compares five PEFT strategies, namely, adapters, additive learning, side-tuning, LoRA, and zero-initialized layers against full fine-tuning on CIFAR-100, using a fixed sequential split: task-A (classes 0-49) followed by task-B (classes 50-99). Experiments are conducted on Resnet-18 and Resnet-50[14]. The comparison is structured along three key dimensions: (i) susceptibility to catastrophic forgetting, (ii) computational efficiency in terms of trainable parameters, and (iii) accuracy on the subsequent task (task-B). The objective is to quantify the trade-offs between efficiency and performance across different PEFT methods and to demonstrate their superiority over full fine-tuning in a continual learning environment.

We report mean ± standard deviation over three runs and evaluate task-A accuracy before task-B training (Acc_A^{before}), task-A accuracy after task-B without recall alignment (Acc_A^{now}), task-A recall accuracy after Swapback (Acc_A^{recall}), apparent forgetting, true forgetting (Δ), and task-B accuracy (Acc_B). Efficiency is reported via trainable parameter counts and percentages. This work is guided by the following hypotheses:

- **H1:** PEFT approaches avoid catastrophic forgetting, keeping task-A information while adapting to task-B.
- **H2:** PEFT techniques show trade-offs between parameter efficiency and task-B accuracy.
- **H3:** Full fine-tuning is parameter-inefficient and prone to catastrophic forgetting.
- **H4:** PEFT methods can surpass full fine-tuning in efficiency and, in deeper backbones, even in task-B accuracy, making them strong candidates for continual transfer learning.

1.1 PEFT Strategies

1.1.1 Adapters

Adapters were introduced by Houls by et al. as a PEFT approach in natural language processing (NLP)[15]. Instead of updating all model weights, small bottleneck modules are inserted within transformer layers. In adapter tuning, the pre-trained weights *w*are kept frozen, and a small set of new parameters *v* are introduced. The modified function is expressed by Equation 1.

$$f_{w,v}(x) = f_w(x) + g_v(x) \quad (1)$$

Here, $f_w(x)$ represents the original pretrained model, and $g_v(x)$denotes the adapter module with trainable parameters v. At initialization, $g_v(x) \approx 0$, ensuring that the new function behaves like the original model. During fine-tuning, only the adapter parameters *v*are updated, while *w* remains unchanged. Following the original work, a range of adapter designs were proposed, including parallel adapters, adapter fusion for multi-task learning, and adapters for continual learning. In each case, the aim was to adapt large language models to new tasks or domains while minimizing catastrophic forgetting and computational overhead. Inspired by their success in NLP, adapters were extended to vision backbones such as Vision Transformers (ViTs)[16] and Contrastive Language–Image Pre-training (CLIP) [17]. In generative modeling, adapter-style modules (e.g., T2I-Adapter) have been employed to condition pre-trained diffusion models like stable diffusion on additional inputs such as depth maps, sketches, or segmentation masks. In these cases, the adapters serve as lightweight conditioning pathways that can be fine-tuned without touching the main diffusion backbone, enabling domain- or style-specific control.

1.1.2 Additive Learning

The Additive Learning strategy works by attaching new trainable components and parameters (e.g., masks, pruning modules, or attention heads) outside the original network while keeping the base model frozen. Unlike adapter modules, which are inserted inside existing layers of the network and rely on predefined insertion points inside the architecture, additive learning operates externally, making it more flexible in terms of extending a model to multiple tasks. Mathematically, if the pre-trained model is represented as$f_w(x)$where *w* is the frozen parameters, then Additive Learning modifies the output by Equation 2.

$$f_{w,v}(x) = f_w(x) + h_v(x) \quad (2)$$

Here, $f_w(x)$ is the frozenbase model, $h_v(x)$ is the additive module with trainable parameters v, $f_{w,v}(x)$ is the extended model that combines both. At initialization, $h_v(x) \approx 0$, ensuring the behaviour of the new model is initially aligned with the pretrained network. During training, only *v* is updated, while *w* remains unchanged.

In computer vision, piggyback is a classic additive learning method. It adapts the base frozen network to new tasks by learning binary masks externally that selectively activate pre-trained weights. This avoids catastrophic forgetting and enables multi-task reuse[18]. One more recent example is Self-Masking Networks (SMNs) which train binary masks in a self-supervised manner to adapt to new domain with limited labeled data[19]. In the domain of generative models, particularly diffusion models, SaRA (Sparsity-Aware Reuse Adaptation) applies additive learning which focuses on reusing weights that were previously inactive by using sparse weight mask[20]. This strategy is particularly advantageous in multi-task learning, since multiple additive modules can be attached for different tasks without interfering with the base model or with each other, providing a scalable and parameter-efficient alternative to full fine-tuning. However, this incurs storage overhead, as new masks or modules are required per task.

1.1.3 Side-Tuning

Side-Tuning is a light-weight method that changes a base model by adding and training a distinct side network, while keeping the original pre-trained model the same[21]. During prediction, the outputs of the base network and the side network are linearly combined through a weighted summation by Equation 3.

$$f(x) = \alpha B(x) + (1 - \alpha)S(x) \qquad (3)$$

Here, $B(x)$ is an output of the frozen base model, $S(x)$ is the output of the side network (trainable),$\alpha$is the mixing coefficient ($0 \leq \alpha \leq 1$) that controls the contribution of each network, and $f(x)$ is the final prediction after combining both outputs.

Although side-tuning preserves pre-trained knowledge and prevents forgetting, it requires an additional network leading to an extra inference cost. It was initially adopted in NLP for a question answering task with BERT as base model[22]. The study extended the idea by using hierarchical (ladder-like) side networks that receive intermediate activations as input by building shortcut connections (called ladders) from the network of base models. The study shows a reduction in the memory requirements during training by substantial amounts.

1.1.4 Low-Rank Adaptation (LoRA)

LoRA injects the trainable rank decomposition matrices into the backbone architecture[14], [23]. The weight update in a neural network is approximated as the product of two low-rank matrices by Equation 4.

$$\Delta W = A \times B^T, \text{ with rank } r \ll d \qquad (4)$$

Here, A and B represent the trainable matrices, r is the rank, chosen to be much smaller than the original dimension d and W is the weight matrix of the pre-trained model which remains frozen.
This requires training of less than 1% of the parameters as compared to full fine-tuning. This makes it possible for individuals and labs with limited resources (e.g., using a single GPU) to adapt large models such as GPT-3 for chatbots and domain-specific tasks. LoRA is later extended to Vision Transformers (ViTs) and multimodal LLaMA variants (e.g., LLaVA, LLaMA-adapted for Vision + Language)[24]. This approach has proved especially useful for tasks like image captioning, visual question answering, and multimodal reasoning. In generative AI, stable diffusion models are widely adopted using LoRA. Artists and developers can use LoRA adapters to personalize diffusion models, e.g., training a LoRA on just a few images of a person, style, or object, then plugging it into stable diffusion to generate personalized outputs. This has become a community standard because it is much faster and lighter than training or fine-tuning the full diffusion model.

1.1.5 Zero-Initialized Layers

ZIL is a PEFT technique where new task-specific layers are added to a pre-trained model. The parameters are initialized to zero at the start of training. The pre-trained model's weights remain frozen. Since the added layers begin at zero, the model initially behaves exactly like the base model. During fine-tuning, only the zero-initialized layers learn task-specific adaptations, avoiding interference with the original pre-trained knowledge. Let $f(x,W)$ represent the frozen prê-trained model with parameters W.A zero-initialized layer introduces a residual update $\Delta W(x, \theta)$, where θ isa new trainable parameter initialized at zero. The relationship is shown in Equation 5[25].

$$y = f(x, W) + \Delta W(x, \theta) \qquad (5)$$

Here $f(x,W)$ is the output of the frozen pre-trained model with parameters W,$\Delta W(x,\theta)$, is the task-specific update learned from scratch, and $\theta = 0$ at initialization sets the new parameters to zero. So, the model initially behaves exactly like the pre-trained model.

The study proposed a zero-initialized attention mechanism and a gating factor in the LLaMA model for instruction and prompt tuning to control the importance of word tokens. Stable diffusion and other

large diffusion models commonly use Zero-Convolution (ZeroConv) layers in their LoRA extensions. These are convolutional layers with zero initialization that adapt features for new styles or domains without disrupting the pretrained generative ability. For example, in ControlNet, the base model and the trainable part are bridged through ZeroConv layers to ensure that noise in the downstream task does not affect the pre-trained features.

2. Methods

2.1. Data Preparation

CIFAR-100 is used as a benchmark dataset to study the catastrophic forgetting in PEFT strategies for computer vision tasks. The dataset consists of 60,000 color images (32×32 resolution) across 100 classes, with 500 training and 100 test samples per class. To construct a two-task continual-learning setting, we partition the label space into two disjoint tasks, i.e., task-A comprising classes 0-49 and task-B comprising classes 50-99. This split yields two related yet mutually exclusive classification problems, enabling a direct evaluation of forgetting when the model is trained sequentially across tasks. Each task contains 30,000 images in total. For each task, we follow the official dataset splits by restricting the original training and testing sets to the task's classes. Finally, for each task, the size of the training and testing datasets are 25000 and 5000 images respectively. All images are normalized using the mean and standard deviation statistics of ImageNet.

2.2. Replication of PEFT Approaches

To ensure a fair and reproducible comparison, we replicate five discussed PEFT strategies, i.e., adapters, LoRA, additive learning, side-tuning, and zero-initialized residuals. In all cases, the pre-trained backbone of Resnet-18 and Resnet-50 were kept frozen, and only lightweight task-specific modules were trained. The following subsections describe the replication details of each method in the experiments.

Adapter: In our setup, the adapter operates in a feature space where given a backbone feature vector $x \in R^d$, the adapter applies a bottleneck transformation that first compresses the representation to a smaller dimension $b = 64$, then applies a nonlinearity, and finally projects it back to d. With a residual connection, the adapted feature is given by Equation (6)

$$x' = x + W_{up}.\sigma(W_{down}\cdot x+b_{down}) + b_{up}, \qquad (6)$$

where $W_{down} \in R^{b\times d}$, $W_{up} \in R^{d\times b}$, $\sigma(\cdot)$ = ReLU(·), and b = 64. For stable optimization, the down-projection W_{down} is initialized using Kaiming initialization, while the up-projection W_{up} (and its bias) is zero-initialized. This ensures that at the beginning of task-B training, the adapter contributes nearly zero ($x' \approx x$), so the network initially behaves exactly like the frozen backbone. During task-B adaptation, only the adapter parameters and the task-B classifier head are updated. The output of the bottleneck network (adapter module) is added to the original feature vector through a residual connection, so the final representation becomes the sum of the original features and adapter features.

LoRA (Low-Rank Adaptation) – Low-rank adaptation (LoRA) means learning only the most important directions of change instead of changing everything which introduces task-specific updates in the weight space of neural network. For a linear layer with frozen weights, LoRA parameterizes the update asin Equation 7. Here $A \in R^{d_{out}\times r}$ and $B \in R^{r\times d_{in}}$, A, and B are trainable. The factors α/r controls the effective update magnitude. In our case, α=1 and $\Delta W \approx 0$ at the starting point.

$$W = W_0 + \Delta W, \quad \Delta W = \frac{\alpha}{\beta} AB, \qquad (7)$$

Here, LoRA is inserted at two locations namely, convolutional blocks and task-B classifier head. For the convolutional layers, each frozen 3×3 convolution is expanded with a low-rank branch as two 1×1 convolutions. These are applied in an order, down projection ($C_{in}\rightarrow r$) and then up projection ($r\rightarrow C_{out}$). Then output becomes $y = Conv_{3\times3}(x; W_0) + {}^{\alpha}/_{r}\, Up(Down(x))$.

Additive Learning: Additive learning here is realized as a channel-wise learnable mask applied to the frozen baseline feature vector. Let $x \in R^d$, where d denotes the d-dimensional feature vector extracted by the baseline (kept fixed during task-B). It introduces a trainable logit vector $g \in R^d$, and converts it into a gating vector $m \in (0,1)^d$,using a sigmoid function, as $m = sigmoid(g)$. The adapted representation is obtained by element-wise modulation of the baseline features as x′ = x ⊙ m, where ⊙ denotes element-wise multiplication. The logits are initially zero, which gives m = 0.5 for all channels at the start of task-B training. During task-B adaptation, only the mask parameters g and the task-B classifier head are updated, while the baseline remains frozen. This mechanism aims to learn how much to "pass through" or "suppress" each feature channel for the new task, without changing the backbone weights.

Side-Tuning: Side-tuning here adapts to task-B by training a small auxiliary network in parallel to the frozen task-A classifier and then combining both predictions. Let $x \in R^d$, be the feature vector extracted by the frozen baseline. A task-A classifier head produces base logits as$y_{base} = f_{base}(x)$, where f_{base} is the frozen linear classifier trained on task-A. In parallel, we train a lightweight side network (SideNet), implemented as a two-layer MLP with one hidden layer of size h = 256, as $y_{side} = f_{side}(x) = W2 \cdot ReLU(\text{W1}\cdot\text{x+b1})+\text{b2}$,with $W1 \in R^{h \times d}, W2 \in R^{50 \times d}$.The final task-B prediction is obtained by convex combination of the two logit vectors as $y = \alpha y_{base} + (1-\alpha) y_{side}$. In our case, α is a mixing coefficient set to 0.5. This design preserves the previously learned decision boundaries via frozen base head, while side-network learns task-specific adjustments.

Zero-Initalized layers: Zero-initialized layers here are used as a residual adaptation module that starts with no effect on the frozen backbone features and gradually learns task-specific corrections during task-B. In this work, we now introduce a small bottleneck MLP, denoted as g, and apply it through a residual connection as $x' = x + g(x)$. The residual branch g is defined as a two-layer MLP with bottleneck size (b) of 64. Then, $g(x) = W2 \cdot ReLU(W1 \cdot x + b1) + b2$,where $W1 \in R^{b \times d}$ and $W2 \in R^{d \times b}$. Here, the final projection (W2 and b2) is initialized to zero. Therefore, at the start of task-B training, $g(x) = 0$ and the model behaves exactly like the frozen baseline (x′ = x).

2.3. Evaluation Metrics

We track multiple metrics to evaluate both adaptation and forgetting. Algorithm 1 in Figure 1 provides a stepwise procedure for evaluating these metrics. Let Task-A (classes 0–49) be trained in Stage 1, and Task-B (classes 50-99) be learned in Stage 2 using either full fine-tuning or a PEFT method. After Stage 1, we compute the baseline Task-A accuracy, Acc_A^{before}which is before adapting to task-B. Afterwards, we move to stage B and adapt the model on task-B (classes 50–99) using different PEFT methods and full fine-tuning. After task-B training, we again evaluate the adapted model on the test set of task-A, Acc_A^{now}.

A drop in accuracy is expected here because the model is configured with task-B modules and has shifted its focus to the new classes. Thus, Acc_A^{now}reflects how much task-A ability seems to remain after task-B training, but this drop may not represent true forgetting. To check whether task-A knowledge is truly lost or only overshadowed, we apply a swapback procedure, where task-B specific modules (e.g., new head, adapters, masks, side net) are temporarily removed and the original task-A modules trained in stage A are reattached. The recovered accuracy, Acc_A^{recall}, indicates how much of task-A knowledge can be restored. If $Acc_A^{recall}$is close to Acc_A^{before}, the drop in $Acc_A^{now}$is only apparent forgetting; if it is much lower, then true forgetting has occurred. Finally, we quantify True Forgetting (Δ) as the difference between $Acc_A^{before}$and Acc_A^{recall} given by Equation 8.

$$\Delta = Acc_A^{before} - Acc_A^{recall} \qquad (8)$$

In addition, we report the final task-B accuracy (Acc_B) and measure parameter efficiency in terms of the number of trainable parameters during task-B training (Trainable Params B) and their ratio to full fine-tuning (% Params).

2.4. Experimental Setup

Our experiments follow a two-stage continual-learning algorithm given in Figure 1. Stage 1 (training phase) trains the base model on task-A to learn initial visual representations. We use Resnet-18 and Resnet-50as the backbone for all experiments. In Stage 1, full fine-tuning is applied to optimize all backbone and head parameters on the first 50 classes, producing a task-A trained model. In the algorithm, h^A represents the final prediction layer trained for task-Aby mapping backbone features totask-A labels (0-49)**.**Stage 2 (adaptation phase) adapts the task-A trained model to task-B by reusing the backbone and updating it using one of five parameter-efficient fine-tuning (PEFT) strategies, i.e., adapters, additive mask, side-tuning, zero-init, and LoRA. Each method is trained on the task-B training split to learn the new 50 classes while aiming to preserve task-A knowledge and reduce catastrophic forgetting. Here, h^B is prediction layer for task-B and maps the same backbone features to labels (50-99).To ensure fair comparison, we keep optimization settings identical across strategies: Adam optimizer with learning rate 1×10^{-3} and batch size 128, using cross-entropy loss. We train for 10 epochs on task-A and 20 epochs on task-B. The longer task-B training is used because PEFT updates only lightweight modules (rather than the full backbone), requiring additional iterations to better adapt to the new task.

Input:
Task-A: D_A^{train}, D_A^{test}, Task-B: D_B^{train}, D_B^{test}
Backbone feature extractor (Resnet18, Resnet50): f_θ
Task-A head: h^A, Task-B head: h^B
PEFT module (adapter, LoRa, side-tuning etc): g_ϕ.
Epochs: E_A (Task A), E_B (Task B)
Metric: Accuracy ()

Output:
Acc_A^{before}, Acc_B, Acc_A^{now}, Acc_A^{recall}, Δ_{app}, Δ_{true}

Stepwise Procedure

#Stage 1: Train Task A (full fine-tuning)

1. Train θ and h^A on D_A^{train} for E_A epochs (cross-entropy).
2. Acc_A^{before} = Accuracy (f_θ, h^A; D_A^{test}) *# Computing baseline Task-A accuracy.*
3. $\theta_A \leftarrow \theta$ and $h_A \leftarrow h^A$ *# Saving checkpoint for Swapback*

#Stage 2: Adapt to Task B using PEFT (freeze backbone)

4. Restore $\theta \leftarrow \theta_A$ and freeze θ.
5. Initialize Task-B head h^B and PEFT parameters ϕ in g_ϕ.
6. Train only ϕ and h^B on D_B^{train} for E_B epochs.
7. Acc_B = Accuracy (f_θ, g_ϕ, h^A; D_A^{test}) *#Computing Task-B accuracy.*

#Stage 3: Accuracy on Task-A after Task-B training (No Swapback)

8. Acc_A^{now} = Accuracy (f_θ, g_ϕ, h^A; D_A^{test}) # used the Task-A head (h^A), not (h^B)
9. Δ_{app} = max (0, $Acc_A^{before} - Acc_A^{now}$) *#calculating apparent forgetting*

#Stage 4: Accuracy on Task-A after Task-B training (with Swapback)

10. $\theta = \theta_A$, $h_A = h_A^{saved}$ #restoring parameters
11. Acc_A^{Recall} = Accuracy ($f_\theta + h^A$, D_A^{test})
12. Δ_{true} = max (0, $Acc_A^{Before} - Acc_A^{Recall}$) *#calculating true forgetting*

Figure 1. Two-Stage Training with Swapback Recall Procedure

3. Results and Discussion

We evaluated continual learning on CIFAR-100 using a fixed class split: task-A (classes 0–49) followed by task-B (classes 50–99). For each backbone (Resnet-18 and Resnet-50), we reported the

mean ± standard deviation over three independent runs for the following metrics: task-A accuracy before task-B training (Acc_A^{before}), task-A accuracy after task-B training without recall alignment (Acc_A^{now}), task-A recall accuracy after Swapback (Acc_A^{recall}), apparent forgetting, true forgetting (Δ), and task-B accuracy (Acc_B). In addition, we report efficiency measures for PEFT strategies, including the number of trainable parameters and the percentage of trainable parameters.

Table 1. Continual learning results on (Resnet-18) across fine-tuning strategies with Mean ± Std (n = 3)

Strategy	Acc_A^{before}	Acc_A^{now} (no swap)	Acc_A^{recall} (swapback)	Apparent Forgetting	True Forgetting (Δ)	Acc_B
Side-Tuning	0.78 ± 0.002	0.01 ± 0.001	0.78 ± 0.002	0.76 ± 0.003	0.00 ± 0.000	0.63 ± 0.002
Additive Learning	0.78 ± 0.002	0.01 ± 0.003	0.78 ± 0.002	0.77 ± 0.006	0.00 ± 0.000	0.64 ± 0.003
Zero-Init	0.78 ± 0.002	0.01 ± 0.002	0.78 ± 0.002	0.76 ± 0.001	0.00 ± 0.000	0.63 ± 0.007
Adapters	0.78 ± 0.002	0.01 ± 0.004	0.78 ± 0.002	0.76 ± 0.002	0.00 ± 0.000	0.63 ± 0.003
LoRA	0.78 ± 0.002	0.01 ± 0.002	0.78 ± 0.002	0.77 ± 0.003	0.00 ± 0.000	0.68 ± 0.003
Full Fine-Tuning	0.78 ± 0.002	0.01 ± 0.004	0.46 ± 0.012	0.77 ± 0.006	**0.31 ± 0.012**	0.77 ± 0.003

Table 2. Continual learning results on (Resnet-50) across fine-tuning strategies with Mean ± Std (n = 3)

Strategy	Acc_A^{before}	Acc_A^{now}(no swap)	Acc_A^{recall}(s wapback)	Apparent Forgetting	True Forgetting (Δ)	Acc_B
Side-Tuning	0.81 ± 0.009	0.01 ± 0.003	0.81 ± 0.009	0.80 ± 0.011	0.00 ± 0.000	0.78 ± 0.008
Additive Learning	0.81 ± 0.009	0.01 ± 0.002	0.81 ± 0.009	0.80 ± 0.010	0.00 ± 0.000	0.80 ± 0.007
Zero-Init	0.81 ± 0.009	0.01 ± 0.002	0.81 ± 0.009	0.80 ± 0.009	0.00 ± 0.000	0.77 ± 0.005
Adapters	0.81 ± 0.009	0.01 ± 0.002	0.81 ± 0.009	0.80 ± 0.008	0.00 ± 0.000	0.78 ± 0.003
LoRA	0.81 ± 0.009	0.01 ± 0.003	0.81 ± 0.009	0.80 ± 0.013	0.00 ± 0.000	0.82 ± 0.005
Full Fine-Tuning	0.81 ± 0.009	0.01 ± 0.002	0.61 ± 0.002	0.80 ± 0.009	0.20 ± 0.009	0.81 ± 0.002

3.1. Catastrophic Forgetting

As shown in Table 1 and Table 2, across both backbones (Resnet-18 and Resnet-50), all PEFT strategies (Side-Tuning, Additive Learning, Zero-Init, Adapters, LoRA) avoided catastrophic forgetting. Unlike apparent forgetting $(Acc_A^{before} - Acc_A^{now})$, true forgetting $(Acc_A^{before} - Acc_A^{recall})$ measures the actual loss of task-A knowledge when model configurations are aligned with task-A after fine-tuning on task-B .Without applying the Swapback or recall mechanism, all the parameter-efficient strategies (SideNet, additive learning, zero-init, adapters, LoRA) show near-complete forgetting on task-A (Acc_A^{now}~1%, apparent forgetting ~ 0.81). However, once Swapback is applied, they fully restore task-A accuracy, reducing true forgetting (Δ) to ~0. This demonstrated that PEFT strategies are resistant to catastrophic forgetting. In contrast, full fine-tuning exhibits substantial true forgetting on both backbones (Resnet-18: Δ ~ 0.314 and Resnet-50: Δ ~ 0.197), i.e., even after recall alignment, task-A performance remains significantly below the pretask-B level.

3.2. Accuracy on task-B

In this section, we evaluatedthe performance of different PEFT strategies on task-B together with the percentage of trainable parameters required to achieve this performance. Tables 3 and 4 report the trainable parameter statistics for Resnet-50 and Resnet-18, respectively. It is important to note that a lower percentage of trainable parameters generally leads to faster training and improved efficiency.

As shown in Table 3, all PEFT strategies demonstrate strong task-B performance on the deeper Resnet-50 architecture while effectively mitigating catastrophic forgetting. In particular, LoRA achieves the highest task-B accuracy (Acc_B= 0.82), slightly outperforming full fine-tuning (Acc_B= 0.81), while updating less than 1% of the model parameters and exhibiting zero true forgetting. Additive learning and side-tuning also achieve competitive task-B accuracies, reaching up to 0.80 and 0.78, respectively, while requiring only 0.41% and 4.49% of the parameters to be trained. In contrast, full fine-tuning updates all model parameters, suffer from measurable true forgetting, and do not provide a consistent advantage in task-B performance. This highlights an unfavourable trade-off between accuracy, stability, and computational cost for deeper architectures.

For the shallower Resnet-18 architecture, full fine-tuning achieves the highest task-B accuracy (Acc_B= 0.77); however, this improvement comes at the cost of substantial true forgetting (Δ = 0.31). In comparison, all PEFT strategies preserved task-A performance with zero true forgetting, while achieving moderate task-B accuracies in the range of 0.63-0.68. Among these, LoRA again provides the best performance, demonstrating the most favourable balance between accuracy and efficiency.

Table 3. Percentage of trainable parameters in PEFT and Full Fine-tuning - Resnet-50

Strategy	**Side-Tuning**	**Additive Learning**	**Zero-Init**	**Adapters**	**LoRA**	**Full FT**
Trainable parameter count	1,153,330	104,498	366,706	366,706	237,618	25,651,802
Trainable Params (%)	4.49	0.41	1.43	1.43	0.93	100

Table 4. Percentage of trainable parameters in PEFT and Full Fine-tuning - Resnet-18

Strategy	**Side-Tuning**	**Additive Learning**	**Zero-Init**	**Adapters**	**LoRA**	**Full FT**
Trainable parameter count	288,562	26,162	91,762	91,762	52,786	11,707,482
Trainable Params (%)	2.47	0.22	0.78	0.78	0.45	100

Table 5 summarizes the overall results across both backbones. Notably, LoRA with Resnet-50 achieves the best task-B performance with zero forgetting and only 0.93% trainable parameters. Taken together, these results indicate that as model depth increases, PEFT strategies shift the trade-off frontier, enabling high task-B accuracy, strong parameter efficiency, and complete knowledge retention simultaneously, an outcome that full fine-tuning fails to achieve.

Table 5. Summary of the results

Backbone	**Method**	**Params (%)**	Acc_B	**Δ**
R18	Best PEFT (LoRA)	0.45	0.68	0.00
R18	Full FT	100	0.77	0.31
R50	Best PEFT (LoRA)	0.93	0.82	0.00
R50	Full FT	100	0.81	0.20

3.3 Hypothesis Validation

Findings of the experiments strongly support all four hypotheses:

Support for H1: This hypothesis is strongly supported by the experimental results. All PEFT approaches exhibit no significant forgetting (Δ = 0.00) in both Resnet-18 and Resnet-50 architectures, indicating that task-A knowledge is entirely retained. In contrast, full fine-tuning consistently shows non-zero true forgetting, with approximately 20% forgetting for Resnet-50 and 31% for Resnet-18.

These findings confirm that the PEFT method effectively prevents catastrophic forgetting while still enabling successful adaptation to task-B.

Support for H2: The second hypothesis, indicating that different PEFT approaches exhibit trade-offs between accuracy and parameter efficiency, is unequivocally supported by the data. Additive learning is the most parameter-efficient approach, requiring about 0.22 to 0.41% of trainable parameters while achieving commendable task-B accuracy (around 0.64 on Resnet-18 and roughly 0.80 on Resnet-50).In comparison, LoRA achieves the best task-B accuracy (≈0.68 on Resnet-18 and ≈0.82 on Resnet-50) with a slightly higher parameter update budget (0.45–0.93%). These results highlight a favorable balance between accuracy and parameter efficiency. In contrast, zero-init, side-tuning, and adapters require a larger fraction of trainable parameters (≈0.78–4.49%) while yielding moderate task-B performance in the range of 0.63–0.78.This analysis provides a practical basis for selecting PEFT methods under different constraints. For example, when memory is limited, additive learning proves to be highly efficient and maintains accuracy effectively. LoRA is superior at achieving the most precise task-B outcomes without much fine-tuning.

Support for H3: The third hypothesis posits that comprehensive fine-tuning is both parameter-inefficient and prone to induce catastrophic amnesia. The most costly method was complete fine-tuning, which entailed modifying all parameters. It also exhibited the issue of catastrophic amnesia. This demonstrates its inefficiency regarding parameter utilization and knowledge retention.

Support for H4: The findings provide significant corroboration for Hypothesis 4, particularly with the utilization of the more advanced Resnet-50 backbone. LoRA demonstrates superior accuracy and efficiency compared to comprehensive fine-tuning. It gets a higher task-B accuracy (Acc_B= 0.82) than full fine-tuning (Acc_B= 0.81) while only changing 0.93% of the parameters instead of 100% of them. LoRA has no actual forgetting (Δ = 0.00), which is an important difference from full fine-tuning, which has measurable forgetting (Δ = 0.20). This shows that PEFT can help the model generalize better to the new task while keeping what it learned before and at a far lesser cost.

4. Conclusion

This work shows that parameter-efficient fine-tuning (PEFT) is an effective alternative to full fine-tuning in continual learning. A comprehensive comparison of adapters, additive learning, side-tuning, LoRA, and zero-initialized residual layers elucidates their benefits regarding catastrophic forgetting, computing efficiency, and task-B performance. All PEFT methods substantially reduced forgetting, whereas full fine-tuning lost nearly 37% accuracy on task-A. By updating fewer than 2% of Resnet-50 parameters, PEFT techniques achieved equivalent or greater task-B accuracy with much lower computing cost. Additive learning produced the best parameter efficiency (75% with 0.41%), zero-init achieved the highest task-B accuracy (76% with 1.41% trainable parameters), and adapters/side-tuning supplied balanced performance. In comparison, LoRA demonstrates poorer accuracy despite excellent parameter efficiency, with full fine-tuning updating all parameters while attaining just 53% accuracy. No single PEFT approach prevails across all criteria. The option relies on the intended accuracy-efficiency trade-off. Future work will explore broader benchmarks and hybrid PEFT techniques to further increase scalability and performance.

Declaration of AI and AI assisted technologies in the writing process
During the preparation of this work the author(s) used Grammarly, Quillbot and ChatGPT to check grammar, spelling and summarize referenced papers. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

Declaration of Competing Interest
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] M. Gholizade, H. Soltanizadeh, M. Rahmanimanesh, and S. S. Sana, "A review of recent advances and strategies in transfer learning," *Int J SystAssur Eng Manag*, vol. 16, no. 3, pp. 1123–1162, Mar. 2025, doi: 10.1007/s13198-024-02684-2.

[2] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *J Big Data*, vol. 3, no. 1, p. 9, May 2016, doi: 10.1186/s40537-016-0043-6.

[3] E. Holton, L. Braun, J. A. Thompson, J. Grohn, and C. Summerfield, "Humans and neural networks show similar patterns of transfer and interference during continual learning," *Nat Hum Behav*, pp. 1–15, Oct. 2025, doi: 10.1038/s41562-025-02318-y.

[4] Z. Ke, B. Liu, N. Ma, H. Xu, and L. Shu, "Achieving Forgetting Prevention and Knowledge Transfer in Continual Learning," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. W. Vaughan, Eds., Curran Associates, Inc., 2021, pp. 22443–22456. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/bcd0049c35799cdf57d06eaf2eb3cff6-Paper.pdf

[5] M. Perkonigg*et al.*, "Dynamic memory to alleviate catastrophic forgetting in continual learning with medical imaging," *Nat Commun*, vol. 12, no. 1, p. 5678, Sep. 2021, doi: 10.1038/s41467-021-25858-z.

[6] Z. Chen and B. Liu, "Continual Learning and Catastrophic Forgetting," in *Lifelong Machine Learning*, Z. Chen and B. Liu, Eds., Cham: Springer International Publishing, 2018, pp. 55–75. doi: 10.1007/978-3-031-01581-6_4.

[7] M. Serra-Perello and A. Ortiz, "Incremental Learning Methodologies for Addressing Catastrophic Forgetting: Analysis and Experimental Evaluation," *Journal of Artificial Intelligence Research*, vol. 83, Aug. 2025, doi: 10.1613/jair.1.18405.

[8] M. De Lange *et al.*, "A Continual Learning Survey: Defying Forgetting in Classification Tasks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 7, pp. 3366–3385, Jul. 2022, doi: 10.1109/TPAMI.2021.3057446.

[9] L. Wang, X. Zhang, H. Su, and J. Zhu, "A Comprehensive Survey of Continual Learning: Theory, Method and Application," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 8, pp. 5362–5383, Aug. 2024, doi: 10.1109/TPAMI.2024.3367329.

[10] B. Wickramasinghe, G. Saha, and K. Roy, "Continual Learning: A Review of Techniques, Challenges, and Future Directions," *IEEE Transactions on Artificial Intelligence*, vol. 5, no. 6, pp. 2526–2546, Jun. 2024, doi: 10.1109/TAI.2023.3339091.

[11] L. Wang *et al.*, "Parameter-efficient fine-tuning in large language models: a survey of methodologies," *ArtifIntell Rev*, vol. 58, no. 8, p. 227, May 2025, doi: 10.1007/s10462-025-11236-4.

[12] M. Weyssow, X. Zhou, K. Kim, D. Lo, and H. Sahraoui, "Exploring Parameter-Efficient Fine-Tuning Techniques for Code Generation with Large Language Models," *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 7, p. 204:1-204:25, Aug. 2025, doi: 10.1145/3714461.

[13] M. H. Zahweh, H. Nasrallah, M. Shukor, G. Faour, and A. J. Ghandour, "Empirical Study of PEFT Techniques for Winter-Wheat Segmentation," *Environmental Sciences Proceedings*, vol. 29, no. 1, Nov. 2023, doi: 10.3390/ECRS2023-15833.

[14] K. Kansal, T. B. Chandra, and A. Singh, "ResNet-50 vs. EfficientNet-B0: Multi-Centric Classification of Various Lung Abnormalities Using Deep Learning," *Procedia Computer Science*, vol. 235, pp. 70–80, Jan. 2024, doi: 10.1016/j.procs.2024.04.007.

[15] N. Houlsby*et al.*, "Parameter-Efficient Transfer Learning for NLP," in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., in Proceedings of Machine Learning Research, vol. 97. PMLR, Jun. 2019, pp. 2790–2799. [Online]. Available: https://proceedings.mlr.press/v97/houlsby19a.html

[16] M. Wei, M. Shi, and T. Vercauteren, "Enhancing surgical instrument segmentation: integrating vision transformer insights with adapter," *Int J CARS*, vol. 19, no. 7, pp. 1313–1320, Jul. 2024, doi: 10.1007/s11548-024-03140-z.

[17] Z. Ye, F. Jiang, Q. Wang, K. Huang, and J. Huang, "IDEA: Image description enhanced CLIP-adapter for image classification," *Pattern Recognition*, vol. 171, p. 112224, Mar. 2026, doi: 10.1016/j.patcog.2025.112224.

[18] L. Bogensperger, M. J. Ehrhardt, T. Pock, M. S. Salehi, and H. S. Wong, "An Adaptively Inexact Method for Bilevel Learning Using Primal–Dual-Style Differentiation," *J Math Imaging Vis*, vol. 67, no. 5, p. 49, Aug. 2025, doi: 10.1007/s10851-025-01262-w.

[19] S. Li, G. Yuan, J. Chen, C. Tan, and H. Zhou, “Self-Supervised Learning for Solar Radio Spectrum Classification,” *Universe*, vol. 8, no. 12, Dec. 2022, doi: 10.3390/universe8120656.
[20] H. Wang *et al.*, “A Context-Awareness and Hardware-Friendly Sparse Matrix Multiplication Kernel for CNN Inference Acceleration,” *IEEE Transactions on Computers*, vol. 74, no. 4, pp. 1182–1195, Apr. 2025, doi: 10.1109/TC.2024.3517745.
[21] H. Yu *et al.*, “Efficient Side-Tuning for Remote Sensing: A Low-Memory Fine-Tuning Framework,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 18, pp. 11908–11925, 2025, doi: 10.1109/JSTARS.2025.3563641.
[22] N. M. Gardazi, A. Daud, M. K. Malik, A. Bukhari, T. Alsahfi, and B. Alshemaimri, “BERT applications in natural language processing: a review,” *ArtifIntell Rev*, vol. 58, no. 6, p. 166, Mar. 2025, doi: 10.1007/s10462-025-11162-5.
[23] Y. Mao *et al.*, “A survey on LoRA of large language models,” *Front. Comput. Sci.*, vol. 19, no. 7, p. 197605, Dec. 2024, doi: 10.1007/s11704-024-40663-9.
[24] S. Chen *et al.*, “Enhancing Chinese comprehension and reasoning for large language models: an efficient LoRA fine-tuning and tree of thoughts framework.,” *Journal of Supercomputing*, vol. 81, no. 1, p. 1, Jan. 2025, doi: 10.1007/s11227-024-06499-7.
[25] C. Cao, Q. Dong, and Y. Fu, “ZITS++: Image Inpainting by Improving the Incremental Transformer on Structural Priors,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 10, pp. 12667–12684, Oct. 2023, doi: 10.1109/TPAMI.2023.3280222.