RECAST: REPARAMETERIZED, COMPACT WEIGHT ADAPTATION FOR SEQUENTIAL TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Incremental learning aims to adapt to new sets of categories over time with minimal computational overhead. Prior work often addresses this task by training efficient task-specific adapters that modify frozen layer weights or features to capture relevant information without affecting predictions on any previously learned categories. While these adapters are generally more efficient than finetuning the entire network, they still can require tens or hundreds of thousands of task-specific trainable parameters even for relatively small networks, making it challenging to operate on resource-constrained environments with high communication costs like edge devices or mobile phones. Thus, we propose **Re**parameterized, **C**ompact weight **A**daptation for **S**equential **T**asks (**RECAST**), a novel method that dramatically reduces the number of task-specific trainable parameters to fewer than 50 – several orders of magnitude less than competing methods like LoRA. RECAST accomplishes this efficiency by learning to decompose layer weights into a soft parameter-sharing framework consisting of a set of shared weight templates and very few module-specific scaling factor coefficients. This soft parameter-sharing framework allows for effective task-wise reparameterization by tuning only these coefficients while keeping the templates frozen. A key innovation of RECAST is the novel weight reconstruction pipeline called Neural Mimicry, which eliminates the need for training in our framework from scratch. Extensive experiments across six diverse datasets demonstrate RECAST outperforms the state-of-the-art by up to 3% across various scales, architectures, and parameter spaces. Moreover, we show that RECAST's architecture-agnostic nature allows for seamless integration with existing methods, further boosting performance.

1 INTRODUCTION

Incremental Learning (IL) aims to learn from a continuous stream of data or tasks over time. Generally speaking, the goal of IL methods is to avoid catastrophic forgetting (McCloskey & Cohen, 1989; French, 1999) while minimizing computational overhead. As summarized in Figure 1(a), prior work in IL can be separated into three themes. First, *Rehearsal* methods that learn what samples to retain to use on subsequent iterations to ensure their task-specific knowledge is retained (Caccia et al., 2020; Rebuffi et al., 2017; Chaudhry et al., 2019; 2021; Shin et al., 2017), but they can require large amounts of storage as the number of tasks grows. Second, *Regularization* methods that require no extra storage (Kirkpatrick et al., 2017; Kim et al., 2021; Aljundi et al., 2018; Nguyen et al., 2018; Zhang et al., 2020), but can result in network instability due, in part, to the plasticity-stability dilemma (Grossberg, 1982). Finally, third, *Reconfiguration/Adaptation* methods (Yoon et al., 2018; Rusu et al., 2022; Hung et al., 2019; Mallya & Lazebnik, 2018; Singh et al., 2020; Verma et al., 2021; Jin & Kim, 2022; Ge et al., 2023; Douillard et al., 2022; Wang et al., 2023) that learn task-specific parameters, but can result in large numbers of new parameters per-task that may not scale well. Even methods designed to be efficient, like those based on LoRA (Hu et al., 2021), still require tens or hundreds of thousands of task-specific trainable parameters for each task (*e.g.*, (Wu et al., 2024; Wang et al., 2024; Zhang et al., 2024; Erkoç et al., 2023)) making them ill-suited for some applications in resource-constrained environments like mobile phones or edge devices.

To address this, we propose **Re**parameterized, **C**ompact weight **A**daptation for **S**equential **T**asks (**RECAST**), which provides an extremely parameter-efficient approach for IL via reparameterization of a model's pretrained weights. As illustrated in Figure 1(b), RECAST achieves its compact

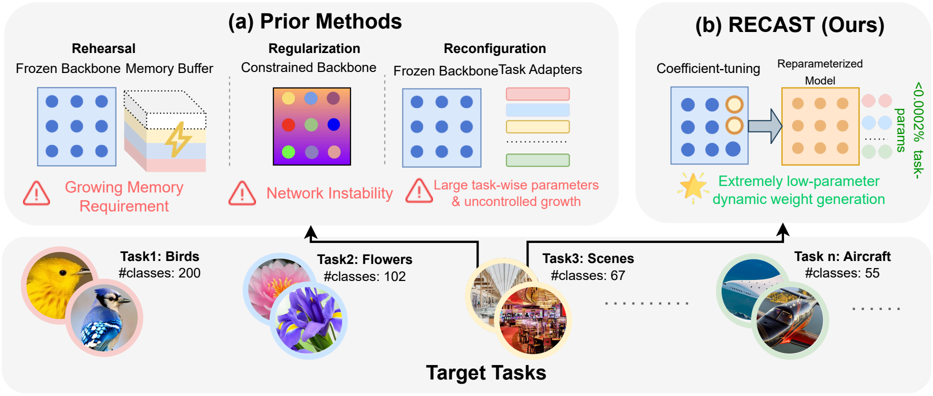


Figure 1: (a) Existing IL methods,*i.e.* Rehearsal, Regularization, Reconfiguration - exhibit various limitations in terms of model complexity, memory requirements, and training overheads. In comparison our proposed method, (b) RECAST can be used as a frozen backbone, allowing efficient reparameterization of any target module with a negligible number of parameter updates (order of 10^{-6}) and accommodate any number of disjoint tasks.

task-specific representation by finetuning only the coefficients for a set of shared weight templates, enabling us to reconfigure an entire network's weights using fewer than 50 trainable parameters. This framework shares some principles with prior Template Mixing methods (*e.g.*, (Plummer et al., 2022; Savarese & Maire, 2019)). However, these methods did not explore applications to incremental learning, or the reparameterization paradigms but rather evaluated their models on tasks like hierarchy discovery and parameter budget allocation. In addition, these methods required that a model is trained in the Template Mixing framework from scratch, making them unable to leverage large pre-trained models like DINO (Caron et al., 2021).

To enable RECAST to use large pretrained models that would be prohibitively expensive to retrain in our desired framework, we propose **Neural Mimicry**, a weight reconstruction approach that learns a decomposition of a pretrained layer into our framework such that it emulates the original network. This enables us to leverage pretrained weights from any model with minimal additional processing (*i.e.*, just a few minutes). Few works in the reparameterization theme of IL have explored reconstruction methods. Still, they focus primarily on generating weights for LoRA matrices typically through diffusion (Wu et al., 2024; Wang et al., 2024; Zhang et al., 2024; Erkoç et al., 2023) or used pretrained weights to initialize their LoRA components (Liu et al., 2024a; Si et al., 2024). Prior work learned additional predictor networks from scratch to generate weights for some subset of modules (Salimans & Kingma, 2016). In contrast to these methods, our work directly changes the backbone weights rather than learning an adapter like LoRA-based methods, while also being several orders of magnitude more parameter efficient.

We evaluate RECAST's effectiveness on diverse datasets using Task-incremental IL settings. We compare against 12 baselines comprising state-of-the-art IL methods on both CNN and Transformer architectures, where RECAST reports up to a 3% gain over prior work. When combined with adapter methods with higher parameter budgets, RECAST still boosts them by up to 2%.

Our key contributions can thus be summarized as follows:

- We propose *RECAST*, A novel framework that dynamically generates module weights using shared basis matrices & module-specific coefficients. This weight-decomposed architecture enables task-specific adaptations optimizing only coefficients, leveraging cross-layer knowledge.
- We introduce *Neural Mimicry*, a novel weight-reconstruction pipeline, that reproduces network weights without resource-intensive pretraining. This scale and architecture-agnostic approach enables easy upgrades to existing architectures. We demonstrate that it builds stronger, more expressive backbones for transfer learning.
- RECAST's fine-tuned resource-performance control suits efficient continual learning with bounded resource constraints. Its compact task-specific parameters enable easier distribution across large-scale systems and edge devices.

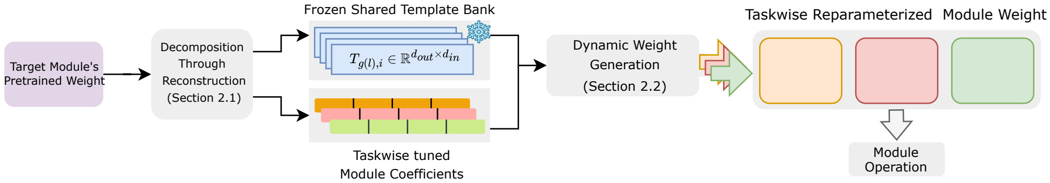

Frozen Shared Template Bank
$T_{g(l),i} \in \mathbb{R}^{d_{out}\times d_{in}}$
Target Module's Pretrained Weight
Decomposition Through Reconstruction (Section 2.1)
Taskwise tuned Module Coefficients
Dynamic Weight Generation (Section 2.2)
Taskwise Reparameterized Module Weight
Module Operation

Figure 2: RECAST decomposes pretrained weights into templates and coefficients. During transfer-learning the templates are kept frozen and only coefficients get tuned to generate module-specific task-wise reparameterized weights

2 REPARAMETERIZED, COMPACT WEIGHT ADAPTATION FOR SEQUENTIAL TASKS (RECAST)

In Task Incremental Learning (TIL), a model sequentially learns D tasks $\{d_1, d_2, d_3, \dots d_D\}$, where each task $d_i = (X_i, Y_i)$ introduces new categories, domains, or output spaces. The objective is to maximize classification performance across all tasks while minimizing resource usage, *i.e.*, $\max \sum_{i=1}^{D} P(d_i)$ s.t. $R \leq R_{max}$ where $P(d_i)$ is the performance on task d_i, R is the total resource usage, and R_{max} is the resource constraint. This resource-awareness is crucial in practice, as it enables deployment on edge devices with limited memory, impacts cloud infrastructure costs and real-time processing, and ensures accessibility in resource-limited environments. Thus, TIL poses a complex optimization problem: how can we design a system that adapts to new tasks without significantly increasing its computational footprint or compromising performance on existing tasks?

We address this multifaceted challenge with our novel framework**Re**parameterized, **C**ompact weight **A**daptation for **S**equential **T**asks (**RECAST**), that strategically balances model plasticity and stability while adhering to the resource constraints in TIL scenarios. As illustrated in figure 2 RECAST introduces a flexible way to adapt neural network layers through the dynamic combination of sets of shared weight blueprints (*i.e.*, **Template Banks**) and module-wise calibration factors or **Coefficients**. Our work builds upon and extends several key areas in neural network adaptation. In (Section 2.1) we present our Weight Decomposition scheme. It has been previously demonstrated that such modular networks are particularly suited for task adaptation (Mendez & Eaton, 2021; Ostapenko et al., 2021). Older works like, Weight Normalization (Salimans & Kingma, 2016) decouples the length of target weight vectors from their direction to reparameterize the network weights. Decomposition has also been used to make DNNs more interpretable Zhou et al. (2018) and to discover hierarchical structures Patil et al. (2023). Works by Savarese & Maire (2019) and Plummer et al. (2022) demonstrate the effectiveness of weight sharing through decomposition, similar to our approach. However, these methods either require training models from scratch in their decomposed form or learning additional task-driven priors to accommodate new tasks Veniat et al. (2021). RECAST introduces Neural Mimicry to enable the direct reconstruction of pretrained weights into reusable templates and sparse coefficients (Section 2.2). Prior approaches to generating neural network weights have explored various strategies: HyperNetwork (Ha et al., 2017) uses smaller networks to predict weights of larger networks. However, while we use static templates to reconstruct weights, HyperNetworks learns to predict weights from input vectors and require end-to-end training. NeRN (Ashkenazi et al., 2023) builds upon this idea of subnetworks by using positional embeddings of weight coordinates, but like HyperNetworks, doesn't leverage pretrained knowledge. A recent work, DoRA (Liu et al., 2024b) shares some similarity with our approach by decomposing pretrained weights into magnitude vectors and LoRA matrices. However, in their case, the weight is utilized only during initialization and requires significant parameter tuning. Another contemporary work, Wang et al. (2024) attempts novel weight generation through autoencoders and latent diffusion. Again, their approach requires complex training of generative models. In contrast, RECAST offers a direct and efficient approach to simple decomposition (Algorithm 1) and novel weight generation, formally described in Algorithm 2. Our simple implementation allows the pipeline to finish executing in < 5 minutes in our experiments, making the added overhead negligible (See Appendix Table 5). These two techniques also combine to establish RECAST as a parameter-efficient model adaptation scheme like LoRA (Chen et al., 2022; Liang & Li, 2024; Zhu et al., 2024; Nikdan et al., 2024) which is now popularly used to adapt to new tasks through low-rank updates. In fact, in Section 3, we empirically show, how RECAST can enhance the variants of LoRA providing a substantial boost in performance across a range of parameters.

Algorithm 1 RECAST Framework

```
Require:
    $L$: Total number of layers in network
    $G$: Number of template bank groups
    $n$: Number of templates per bank
    $K$: Number of coefficient sets
    $d$: Dimension of weight matrices
    $M_l$: Number of modules in layer $l$
procedure INITIALIZE($L, G, n, K, d, M_l$)
    for $g = 1$ to $G$ do
        $\tau_g = T_{g,i} \in \mathbb{R}^{d \times d} | i = 1, \dots, n$        ▷ Template bank for group $g$
    end for
    for $l = 1$ to $L$ do
        for $m = 1$ to $M_l$ do
            $g = \lceil l/(L/G) \rceil$        ▷ Map layer to group
            $C_{l,m} = C^k_{l,m,i} \in \mathbb{R} | k = 1, \dots, K; i = 1, \dots, n$        ▷ Module coefficients
        end for
    end for
end procedure
procedure FORWARDPASS($x, l, m$)
    $g = \lceil l/(L/G) \rceil$        ▷ Get group index
    $W_{l,m} = \frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{n} C^k_{l,m,i} \cdot T_{g,i}$        ▷ Generate weights
    $y = \text{ModuleOperation}(x, W_{l,m})$        ▷ Apply layer operation
    return $y$
end procedure
```

2.1 WEIGHT DECOMPOSITION

RECAST maintains overall architecture (ResNet, Vision Transformer etc.) while replacing specific modules with components that can dynamically generate weights. A target module has access to a bank of templates which is shared with some other layers according to a specific grouping scheme (See Section 3.2). Each module has its own set of coefficients which are used to generate the weights. The process can be mathematically formalized as follows: Let L be the total number of layers in the network organized into G groups, where each group shares a template bank. For each group $g \in 1, \dots, G$, we define a template bank $\tau_g = T_{g,1}, T_{g,2}, \dots, T_{g,n}$, where n is the number of templates per bank. Each template $T_{g,i} \in \mathbb{R}^{d_{out} \times d_{in} \times \cdots}$ has the same shape as the layer being parameterized. For each target module $m \in 1, \dots, M_l$ in layer $l \in 1, \dots, L$, where M_l is the number of target modules in layer l, we generate K sets of coefficient vectors. The k-th set of coefficients for module m in layer l is defined as $C^k_{l,m} = C^k_{l,m,1}, C^k_{l,m,2}, \dots, C^k_{l,m,n}$, where $C^k_{l,m,i} \in \mathbb{R}$ is the learned coefficient for template $T_{g,i}$ in group $g = \lceil l/(L/G) \rceil$. The weight matrix for module m in layer l is computed via:

$$W^k_{l,m} = \sum_{i=1}^{n} T_{g,i} \cdot C^k_{l,m,i} \tag{1}$$

Here, $\cdot$ represents element-wise multiplication. The final weight matrix for module m in layer l is the average of these K parameter matrices.

$$W_{l,m} = \frac{1}{K} \sum_{k=1}^{K} W^k_{l,m} \tag{2}$$

Algorithm 1 describes how template banks for each group are created and as well as the orthogonal initialization of coefficients for each module m in layer l. The forward pass computes the final weight matrix using these components before applying the module operation. The target module then generates the appropriate output (*e.g.*, convolutional feature map for CNN, attention feature vectors for attention module). See Appendix A.4.1 for additional architecture-specific details.

Algorithm 2 Neural Mimicry

```
Require:
    $M$: Original pretrained model
    $M^*$: Reconstructed Model
    $L$: Number of layers
    $M_l$: Number of modules in layer $l$
    $T_{g(l)}$: Template bank for group $g$ containing layer $l$
    $C_{l,m}$: Coefficients for module $m$ in layer $l$
 1: procedure NEURALMIMICRY($M, M^*$, max_epochs)
 2:     for epoch $= 1$ to max_epochs do
 3:         total_loss $\leftarrow 0$
 4:         for $l = 1$ to $L$ do                                          ▷ Layer iteration
 5:             for $m = 1$ to $M_l$ do                                    ▷ Module iteration
 6:                 $W^*_{l,m} \leftarrow$ GenerateWeights$(T_{g(l)}, C_{l,m})$          ▷ Weight reconstruction
 7:                 $L_{l,m} \leftarrow$ LossFunction$(W^*_{l,m}, W_{l,m})$
 8:                 total_loss $\leftarrow$ total_loss $+ L_{l,m}$
 9:                 $T_{g(l)}, C_{l,m} \leftarrow$ Update$(T_{g(l)}, C_{l,m}, L_{l,m})$
10:             end for
11:         end for
12:     end for
13:     return $M^*$                                                    ▷ Return reconstructed model
14: end procedure
```

2.2 NEURAL MIMICRY

We introduce "Neural Mimicry" as a foundational technique for efficiently emulating complex pretrained neural networks. This end-to-end pipeline enables the recreation of a target model within our framework, bypassing resource-intensive pretraining. The input to this pipeline is a pretrained target model M, whose module weights we aim to emulate. The output is a set of template banks and coefficients that, when combined, approximate the target module weights of M. The core of Neural Mimicry lies in its iterative refinement process, which progressively shapes the custom model M^* to closely mirror the behavior of the target model M. The pretrained weights of M are obtained from official timm repository (See Appendix A.3). To generate weights for each module in each layer, we employ Equations 1 and 2 as described in Section 2.1, and then aim to reduce the difference between the original pretrained weight ($W_{l,m}$) and RECAST-generated weight ($W^*_{l,m}$). The main update step uses gradient descent (Algorithm 2 line 9) on the coefficients and templates (η is the learning rate): $C^k_{l,m,i} \leftarrow C^k_{l,m,i} - \eta \frac{\partial \mathcal{L}}{\partial C^k_{l,m,i}}; T_{g(l),i} \leftarrow T_{g(l),i} - \eta \frac{\partial \mathcal{L}}{\partial T_{g(l),i}}$. The process can be formulated as the following optimization problem:

$$\min_{C,T} \mathbb{E}_{\epsilon \sim \mathcal{N}(0,\sigma^2)} \left[\sum_{l=1}^{L} \sum_{m=1}^{M_l} \mathcal{L}(W^*_{l,m}(\epsilon), W_{l,m}) \right] \tag{3}$$

where $\epsilon = (\epsilon_1, \ldots, \epsilon_K)$ is a set of K-noise vectors sampled from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$, where $\epsilon_i \in \mathbb{R}^{n \times 1 \times \cdots}$. The noise vectors are added with the coefficients only during the reconstruction training process. Here, L denotes total layers, M_l target modules in layer l, n templates per bank, and $\mathcal{L}$ the weight discrepancy loss. The choice of $\mathcal{L}$ is an important design decision(see Section 3.2). We found Smooth L1 loss most effective, balancing L1 and L2 properties. It offers smooth gradients for small differences and stability for larger errors, crucial for diverse weights across layers and stable fine-tuning performance (Figures 5a, 3). This outperforms L2/MSE (sensitive to outliers) and KL divergence (computationally heavy) (Hinton et al., 2015; Ashkenazi et al., 2023). After reconstruction, we calculate the layerwise feature similarity between M and M^* through Cosine Similarity metric to evaluate the quality(See Table 6).

2.3 PARAMETRIC SCALING STRATEGY

RECAST's design is governed by three parameters: the number of groups (G), templates per group (n), and coefficient sets (K), balancing model expressiveness and efficiency. Increasing G enhances

Table 1: Average accuracy@top1 over six image classification datasets using ResNet-34 in TIL setting (standard deviation over 3 seeds). RECAST surpasses non-adapter methods and enhances adapter-based approaches when combined. Parameters: M: model size, R: buffer, P: total parameters, U: unit attention mask, G: ghost modules The Learnt Parameters and Storage metrics are reported in a Per Task manner.

Method	**Learnt Params**	**Storage**	**Acc@top1(%)**
ResNet-34 (He et al., 2015)	0.0	M	65.2
EWC (Lee et al., 2019)	$P + FIM$	$M + FIM$	22.1
LWF (Li & Hoiem, 2016)	P	M	19.3
GDUMB (Prabhu et al., 2020)	P	$M + R$	34.8
ER-ACE (Caccia et al., 2021)	P	$M + R$	36.1
DER++ (Buzzega et al., 2020)	P	$M + R$	16.2
RECAST (ours)	$3 \times 10^{-6}P$	M	66.3±0.05
HAT (Serra et al., 2018)	P	U	40.0
Piggyback (Mallya & Lazebnik, 2018)	P	M	82.9
CLR (Ge et al., 2023)	$4 \times 10^{-3}P$	$M + G$	79.9
RECAST (ours) + Piggyback	P	M	**83.9**±0.1
RECAST (ours) + CLR	$4 \times 10^{-3}P$	$M + G$	80.9±0.3

specialization, while larger n boosts expressiveness within groups. The parameter-sharing scheme allows K to expand weight configurations without significantly increasing parameters. This design achieves substantial memory savings, approximated as: $\mathbb{S} = L \times M_l \times d^2 - (G \times n \times d^2 + L \times M_l \times n \times K)$. Here, M_l refers to the number of target modules per layer, and we can approximate the savings as: $\mathbb{S} \approx L \times M_l \times d^2 - G \times n \times d^2 = d^2(L \times M_l - G \times n)$ .The savings primarily stem from replacing $L \times M_l$ weight matrices with $G \times n$ templates. For experiments, G and n are calibrated to match the baseline model's parameter count but can be adjusted for desired compression. Additionally, multiple coefficient sets enable a combinatorially large number of weight matrices (n^K) per module, covering extensive parameter spaces even with small n. The total learnable coefficients are $\sum_{l=1}^{L} M_l \times n \times K$, providing flexibility in controlling tunable parameters.

3 EXPERIMENTS AND ANALYSIS

Baselines To evaluate our proposed architecture, we have compared the performance against 15 baselines in total, representing the aforementioned IL strategy categories. Among the popular We evaluate our architecture against 16 baselines across multiple IL categories. From *Regularization* approaches, we include **EWC** (Kirkpatrick et al., 2017), which penalizes changes to critical network parameters, and **LwF** (Li & Hoiem, 2016), which uses previous model outputs as soft labels. For *Replay Methods*, we implement **GDumb** (Prabhu et al., 2020) with a 4400-sample memory buffer, and ECR-ACE Caccia et al. (2021) and DER++ (Buzzega et al., 2020), both using 120 samples per class. *Reconfiguration* methods are our primary focus, particularly adapter-based schemes that learn extra task-specific components. Among them, **PiggyBack** Mallya & Lazebnik (2018) uses binary masking schemes within a fixed architecture, while **HAT** Serra et al. (2018) learns soft attention mask. **CLR** Ge et al. (2023) is a purely adapter-based method, that introduces depthwise separable Ghost modules (Han et al., 2020) after each convolutional layer. We evaluate RECAST-ViT against LoRA-based approaches, that apply LoRA to various ViT components (Attention (Zhu et al., 2024) or Fully-connected layers (Chen et al., 2022)). We also compare with **InfLoRA** (Liang & Li, 2024), a hybrid between regularization, **RoSA** (Nikdan et al., 2024), which combines LoRA with sparse adaptation, **DoRA** (Liu et al., 2024b) that decomposes pretrained weight into magnitude and direction matrices, and a prompt-based adapter technique: **L2P** Wang et al. (2022). Finally, we include finetuned classifier layers as naive baselines.

Dataset Following Aljundi et al. (2019); Ge et al. (2023) we employ **six** diverse benchmarking datasets covering fine-grained to coarse image classification tasks across various domain (see Table 3 in Appendix A.1) including flowers (Nilsback & Zisserman, 2008), scenes (Quattoni & Torralba, 2009), birds (Wah et al., 2011), animals (Krizhevsky, 2009), vehicles (Maji et al., 2013), and other man-made objects (Krizhevsky, 2009). These datasets capture the diversity and inter-task

Table 2: Average accuracy@top1 over six image classification datasets using ViT-Small backbone in TIL setting (standard deviation over 3 seeds). RECAST improves baseline performance by 3% and enhances existing methods by$> 1\%$. M: model parameters, P: total parameters, r: LoRA rank, Pl: prompt pool size, To: tokens per prompt, D: embedding dimension, SpA : Sparse Adaptation. Learnt Parameters and Storage metric are reported in a Per Task manner (except L2P and InfLoRA)

Method	**Learnt Params**	**Storage**	**Acc@top1(%)**
ViT-Small (Dosovitskiy et al., 2020)	0.0	M	82.4
RoSA (Nikdan et al., 2024)(SpA MLP)	$6 \times 10^{-6}P$	M	84.5
RECAST (ours) + RoSA(SpA MLP)	$6 \times 10^{-6}P$	M	85.4±0.05
RECAST (ours)	$2 \times 10^{-6}P$	M	85.0±0.1
L2P (Wang et al., 2022)	$To * D * M_{pool}$	$M + Pl$	35.9
InfLoRA (Liang & Li, 2024)	$P + 2 \times 10^{-3}P(T-1)$	M	88.5
AdaptFormer (Chen et al., 2022)	$6 \times 10^{-4}Pr$	M	89.0
MeLo (Zhu et al., 2024)	$8 \times 10^{-4}Pr$	M	88.7
DoRA (Liu et al., 2024b)	$1.9 \times 10^{-3}Pr$	M	89.3
RoSA (Nikdan et al., 2024)(Full - Att.)	$7 \times 10^{-3}Pr$	M	88.6
RECAST (ours) + Adaptformer	$6 \times 10^{-4}Pr$	M	**90.1**±0.2
RECAST (ours) + MeLo	$8 \times 10^{-4}Pr$	M	89.6±0.2
RECAST (ours) + RoSA(Full - Att.)	$7 \times 10^{-3}Pr$	M	88.9 ±0.08

variance essential for evaluating IL tasks. Images are standardized to 224×224 pixels, applying basic transformations and dataset-specific normalization. The datasets are then pickled to ensure consistent comparisons across various experiments. More details are in Appendix A.1.

Implementation Details We picked ResNet (He et al., 2015) and Vision Transformer (ViT) (Dosovitskiy et al., 2020) - to demonstrate the versatility of our framework as they represent two different branches of image classification architectures (CNN and Transformer). Both architectures achieve SOTA results on image classification tasks and are well-studied - making it easier to compare with existing methods. Specifically, we use ResNet-34 and ViT-Small variants for all our methods, both of which have approximately $21mil.$ parameters. However, average best performance across the six datasets for larger models are also presented in the Appendix section 4. For fair evaluation, we compare the ResNet dependent baselines, against our RECAST-Resnet models and compare ViT-based baselines against RECAST-ViT model. We trained GDUMB (Prabhu et al., 2020), EWC (Lee et al., 2019), LWF (Li & Hoiem, 2016), and L2P (Wang et al., 2022) using the *avalanche* library (Carta et al., 2023). Official PyTorch implementations of other methods were modified for TIL settings. We evaluated two RoSA (Nikdan et al., 2024) variants: sparse MLP adaptation and full RoSA (LoRA-based attention). A sparsity of 0.001% ensured active parameters remained under 150, matching RECAST's efficiency. LoRA methods used rank=1 and task-specific configurations: DoRA (Liu et al., 2024b) (Q,K,V matrices), MeLo (Q,V matrices), and Adaptformer (MLP layers). All experiments ran on an RTX8000 GPU. RECAST, MeLo (Zhu et al., 2024), and AdaptFormer (Chen et al., 2022), RoSA (Nikdan et al., 2024), DoRA (Liu et al., 2024b) used *AdamW* (Loshchilov & Hutter, 2017) with $2e{-}3$–$5e{-}3$ learning rates, $1e{-}6$ weight decay, and stepwise LR scheduling (decay by 0.1 every 33 epochs) for 100 epochs. Default hyperparameters were used for *avalanche* models and methods like HAT (Serra et al., 2018), Piggyback (Mallya & Lazebnik, 2018), and CLR (Ge et al., 2023), trained for 100 epochs. InfLoRA (Liang & Li, 2024) used class increments of 90 and 10 epochs per task. For RECAST-ViT, we used a group size of 6, 2 templates per bank, and 2 coefficient sets. Integrated models used RECAST as the backbone, with adapter layers modifying features while RECAST generated core layer weights and coefficients adapted per task (See Appendix A.4.2).

3.1 RESULTS

RECAST demonstrates substantial improvements in continual learning across diverse neural network architectures. We evaluate its performance on ResNet-34 (Table 1) and ViT-Small (Table 2). Task-wise classification results are presented in Appendix Table 4, with ImageNet1k results across model scales and reconstruction schemes in Appendix 6. The results highlight RECAST's ability

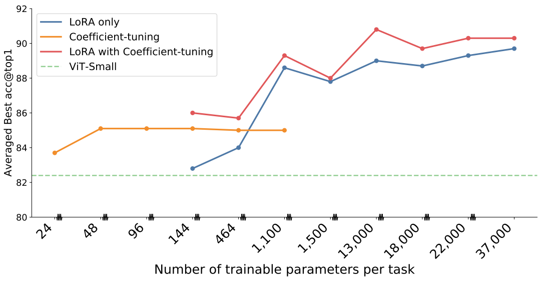


Figure 3: Performance comparison on Small ViT backbone: RECAST operates effectively in extremely few parameters range (24-96) where LoRA doesn't function fruitfully. It also enhances LoRA's performance when combined across all parameter ranges providing complementary benefits.

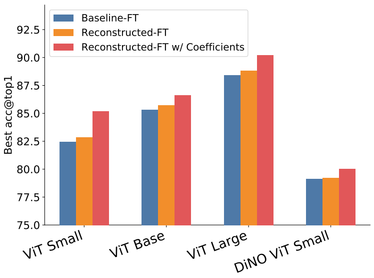


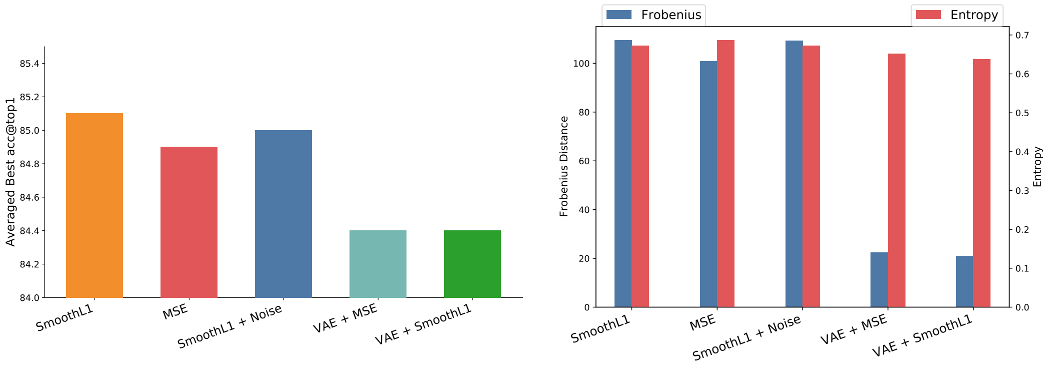
Figure 4: Best accuracy@top1 averaged over six diverse image classification datasets, reported for ViT models of different scales. Performances are shown for Baseline Small-ViT fine-tuning, RECAST adapted ViT fine-tuning and RECAST adapted ViT finetuning with Coefficient finetuning

to maintain high task accuracy while enhancing the performance of existing IL methods. Below, we compare RECAST with both non-adapter and adapter-based methods, focusing on parameter efficiency and storage requirements.

Table 1 showcases the performance of different methods using a ResNet-34 backbone. Traditional IL methods such as EWC (Lee et al., 2019) and LWF (Li & Hoiem, 2016) face drastic degradation as the number of tasks increases, demonstrating significant catastrophic forgetting with accuracies below 25%. GDUMB (Prabhu et al., 2020), which utilizes a large memory buffer, performs marginally better at 34.8%. Interestingly, HAT (Serra et al., 2018), despite learning unit-wise masks over the entire network, struggles to achieve high accuracy (40.0%). This is likely because HAT was designed to allocate a fixed amount of network capacity to each task, and our diverse task suite challenges its ability to budget sufficient resources per dataset. In contrast, RECAST-FT attains 66.2%, surpassing the baseline ResNet-34 (65.2%) and outperforming other non-adapter methods, while finetuning only 3×10^{-6} of the original parameters. Among adapter methods, Piggyback (Mallya & Lazebnik, 2018) and CLR (Ge et al., 2023) show strong performance (82.9% and 79.9% respectively). However, RECAST combined with these methods pushes the performance even further. RECAST + Piggyback achieves 84.1%, and RECAST + CLR reaches 80.9%, demonstrating RECAST's ability to consistently improve adapter-based approaches. **Table 2** summarizes results for methods using the ViT-Small (Dosovitskiy et al., 2020) backbone, highlighting RECAST's strong performance. The baseline achieves 82.4% accuracy, while RECAST-FT improves it to 85.2%—a 2.8 percentage point gain with only 2×10^{-6} additional parameters. Adapter and reparameterization methods show strong performance with the ViT-Small backbone. InfLoRA (Liang & Li, 2024), MeLo (Zhu et al., 2024), AdaptFormer (Chen et al., 2022), RoSA (Nikdan et al., 2024), and DoRA (Liu et al., 2024b) all exceed 88% accuracy. However, combining RECAST with these methods once again yields performance improvements. Combining RECAST with these enhances performance further, with RECAST + MeLo achieving 89.6% and RECAST + AdaptFormer reaching 90.1%. In contrast, prompting methods like L2P (Wang et al., 2022), which rely on instance-level prompts, struggle in diverse domains, yielding a low accuracy of 35.9%.

Comparison of Resource Efficiency Tables 1 and 2 compare various incremental learning methods for ResNet-34 and ViT-Small backbones, respectively. We see that, baseline models (ResNet-34, ViT-Small) and parameter-free methods (EWC, LWF) maintain minimal storage (only the backbone model parameters) but suffer from performance degradation as task diversity increases. Rehearsal-based methods (GDUMB, DER++ & ER-ACE) and HAT offer improved performance at the cost of increased storage requirements, which may limit scalability for long-term learning. Recent parameter-efficient methods balance performance and resource usage. CLR learns 0.3% of backbone parameters, requiring backbone storage plus task-specific depthwise modules. InfLoRA trains the full model for the first dataset and only 0.202% for subsequent tasks. Other LoRA variants train less than 1% per rank. While masking requires training a large number of parameters, both LoRA updates and binary masks (PiggyBack, RoSA Sparse updates) can be mathematically merged to

Figure 5: The designing process of reconstruction is crucial to the IL performance of models. The plot above provides concrete evidence that Neural Mimicry with SmoothL1 is capable of generating backbones with most balanced template diversity and flexibility for Computer Vision Tasks



(a) Plot showing average classification accuracy of models reconstructed in different ways. VAE models performed slightly worse than the rest.

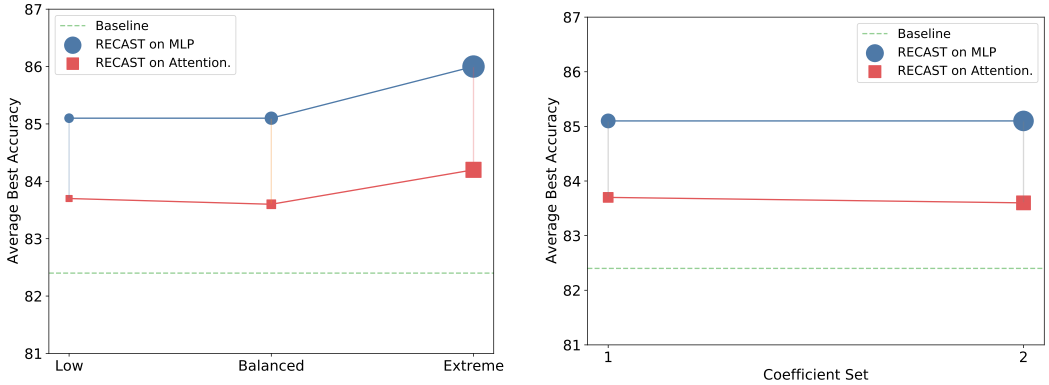
(b) Plot showing Diversity and adaptability measurements of models reconstructed in different ways. VAE models have shallow template diversity, and their internal information content is also less than the rest

prevent extra storage and communication costs during deployment. Our proposed RECAST method demonstrates unprecedented parameter efficiency, learning only 0.0002% of the total backbone parameters while maintaining only the backbone model in storage. When combined with other efficient methods like MeLo or Adaptformer, RECAST achieves state-of-the-art performance (90.1% Acc@top1 for ViT-Small) with the base storage requirement. This reveals that RECAST not only enhances existing adapter methods but does so in ways that are complementary to their strengths. The impact is delineated clearly in Figure 3, in a range of taskwise parameter counts. It compares three adaptation approaches using ViT-Small backbone (82.4% baseline). Traditional LoRA adaptation (blue) becomes effective only above 464 parameters/task. RECAST (orange) achieves consistent 85% accuracy with just 24-96 parameters, showing marginal improvement beyond this range. The combined RECAST+LoRA (red) peaks at 91% accuracy, outperforming across all parameter ranges, including areas where LoRA alone falters. To scale traditional LoRA implementations to a negligibly small (< 150) parameters, we utilized three techniques: (1) lowering LoRA rank, (2) sharing LoRA matrices across layers, and (3) binary masking for pruning. The figure reveals that RECAST can be used effectively in parameter ranges where other reparameterization methods might struggle.

3.2 MODEL ANALYSIS

How does the number of coefficients impact performance? We analyze the effect of hyperparameters G, n, and K (Subsection 2.3) through two experiments. First, we vary n while keeping $K = 1$, adjusting G to maintain constant parameters ($G \propto \frac{1}{n}$), comparing three sharing schemes: Low (two-layer sharing), Balanced (four-layer sharing), and Extreme (all-layer sharing). Results are shown in Figure 6a. Second, we fix G and n while varying K (since $|C| \propto K$) as depicted in Figure 6b. Both experiments use RECAST on MLP and Attention layers of a small ViT model, with marker sizes indicating relative coefficient sizes. Results reveal that adjusting template numbers and sharing schemes has a greater impact than linearly increasing K, emphasizing the importance of template diversity and specificity for task adaptation. Additional linear combinations show diminishing returns once sufficient coefficients are available (see the orange line in Figure 3).

Where do we apply RECAST? RECAST is applicable to various layers or modules (Appendix A.4.1), with notable insights highlighted in Figures 6a and 6b. These figures illustrate the differing performance boosts when applying the method to attention layers versus MLP layers. MLP layers, though simpler, have more parameters that may introduce redundancy, making them more amenable to sharing. In contrast, attention layers are complex, involving multiple computational steps to model relationships between input parts rather than directly transforming representations. This complexity leads to a challenging optimization space, hindering effective template sharing and



(a) Increasing the number of templates, i.e: increasing the number of layers in a group, - both increases the number of coefficients and the intensity of template sharing across the architecture. In the plot, for *Low* sharing we set, $G = 6$, for *Balanced* we set $G = 3$ and for $Extreme$, $G = 1$(one template bank is shared by all the layers of the network) This setting provides substantial performance boost.

(b) The number of coefficients can also change by increasing the number of coefficient sets, K. However, the gain in this case is not as pronounced - indicating that the model performance largely depends on the diversity of the templates.

Figure 6: Plots comparing the strategies of changing coefficients. The size of the markers corresponds to the relative number of coefficients. The impact of increasing coefficients by varying the number of coefficient sets K, is much less prominent than changing it through grouping schemes.

fine-tuning-based reparameterization. Empirical evidence in Figure 3 supports this: LoRA implementations for data points 1100 and $13,000$ from AdaptFormer (Chen et al., 2022) show a performance spike when applied to MLP layers, indicating their greater impact compared to attention layers with fewer parameters (Zhu et al., 2024).

What is the best way to reconstruct weights? Section 2.2 described the weight reconstruction methodology for RECAST. Earlier we noted that model expressivity depends more on template bank diversity. To investigate this, we generated five models with the same configurations (6 groups, 2 layers each), varying the reconstruction objective: 1) SmoothL1 loss, 2) MSE loss, 3) SmoothL1 loss with coefficient noise, 4) VAE with MSE loss, and 5) VAE with SmoothL1 loss. All models achieved at least 98.5% reconstruction similarity (Table 6).Figure 5a shows that SmoothL1 loss (with or without perturbation) provides the most benefit during reparameterization & VAE models provide the least. To understand this, we calculated template diversity within groups using *Frobenius Norm* (diversity metric) and *Entropy* (versatility metric) (See Appendix A.3) Just like their classification performance, SmoothL1 loss shows roughly similar internal patterns with or without noise. The high Frobenius norm for Neural Mimicry (Algo. 2) with SmoothL1 loss indicates greater template diversity than the rest of the approaches. MSE models showed slightly lower Frobenius norms but higher entropy, suggesting templates are versatile and not overly dependent on dominant components. In contrast, VAEs showed shallow template diversity, limiting adaptability, and lower entropy, indicating reduced versatility. Figure 8 (Appendix A.3) illustrates higher intra-group coefficient similarity in VAE-based models, suggesting multiple layers perform similar operations.

4 CONCLUSION

RECAST represents a significant advancement in neural network architecture design, addressing the challenge of optimizing parameter efficiency while maintaining model expressivity. By decomposing network layers into templates and coefficients, it facilitates dynamic, task-specific weight generation with a reduced parameter footprint. Our comprehensive evaluations demonstrate RECAST's efficacy across diverse image classification tasks and vision architectures. Based on our model analysis, future research directions may include enhancing model diversity and coefficient versatility, exploring applications beyond computer vision, and investigating alternative approaches for reconstructing novel parameters to overcome the limitations of existing pretrained weights. Additionally, further research is necessary to elucidate the relationship between model complexity and performance, particularly in the context of model compression techniques.

5 ETHICS STATEMENT

While our work on efficient reparameterization aims to improve model robustness and adaptability, we acknowledge the potential ethical implications of such advancements. First, the incremental learning capabilities enabled by our approach could be beneficial in scenarios like robotics, where systems need to constantly adapt to new categories or environments, as well as provide potential benefits in terms of environmental sustainability by reducing computational footprint. While we intend to democratize access to powerful AI models, we recognize that easier deployment of these models could also lower barriers to potential misuse, such as in surveillance applications that could infringe on privacy rights. Additionally, as with any machine learning model, there are inherent biases and limitations in the training data and model architecture that users should be aware of. As we reconstruct from existing weights, our reconstructed pretrained weights also likely inherits the implicit biases in those existing weights. Thus, we caution against over-reliance on model predictions without human oversight, especially in high-stakes decision-making processes. As researchers, we emphasize the importance of responsible development and deployment of AI technologies, and we encourage ongoing discussions about the ethical use of incremental learning and efficient deep learning models in various applications.

6 REPRODUCIBILITY

To ensure the reproducibility of our work, we have taken several key steps. The implementation details of our custom ResNet and Vision Transformer architectures are fully described in the main text, with complete code provided in the supplementary materials. We have also provided the full codes for our framework integrated with other adapter and reparameterization methods we have discussed in the main text. Our reconstruction scripts for both ResNet and ViT models, including the loss functions and training procedures, are also included in the supplementary material. We have specified all hyperparameters, including learning rates, scheduling, and template bank configurations. The base models used for comparison (ResNet34 and ViT-Small) are from widely available libraries (torchvision and timm), ensuring consistent baselines. All experiments were conducted using PyTorch, with specific versions and dependencies listed in the supplementary materials. We have also included our data preprocessing steps and evaluation metrics to facilitate accurate replication of our results. The supplementary material also includes descriptions of how to use the codebase to both reproduce our results, as well as to extend or use by users in a plug-and-play manner.

REFERENCES

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

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

Maor Ashkenazi, Zohar Rimon, Ron Vainshtein, Shir Levi, Elad Richardson, Pinchas Mintz, and Eran Treister. Nern - learning neural representations for neural networks. January 2023. Publisher Copyright: © 2023 11th International Conference on Learning Representations, ICLR 2023. All rights reserved.; 11th International Conference on Learning Representations, ICLR 2023 ; Conference date: 01-05-2023 Through 05-05-2023.

Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Lucas Caccia, Eugene Belilovsky, Massimo Caccia, and Joelle Pineau. Online learned continual compression with adaptive quantization modules, 2020.

Lucas Caccia, Rahaf Aljundi, Nader Asadi, Tinne Tuytelaars, Joelle Pineau, and Eugene Belilovsky. New insights on reducing abrupt representation change in online continual learning. *ArXiv*, abs/2203.03798, 2021. URL `https://api.semanticscholar.org/CorpusID:247315339`.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.

Antonio Carta, Lorenzo Pellegrini, Andrea Cossu, Hamed Hemati, and Vincenzo Lomonaco. Avalanche: A pytorch library for deep continual learning. *Journal of Machine Learning Research*, 24(363):1–6, 2023. URL `http://jmlr.org/papers/v24/23-0130.html`.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem, 2019.

Arslan Chaudhry, Albert Gordo, Puneet K. Dokania, Philip Torr, and David Lopez-Paz. Using hindsight to anchor past knowledge in continual learning, 2021.

Shoufa Chen, Chongjian Ge, Zhan Tong, Jiangliu Wang, Yibing Song, Jue Wang, and Ping Luo. Adaptformer: Adapting vision transformers for scalable visual recognition, 2022. URL `https://arxiv.org/abs/2205.13535`.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020. URL `https://arxiv.org/abs/2010.11929`.

Arthur Douillard, Alexandre Ramé, Guillaume Couairon, and Matthieu Cord. Dytox: Transformers for continual learning with dynamic token expansion, 2022. URL `https://arxiv.org/abs/2111.11326`.

Ziya Erkoç, Fangchang Ma, Qi Shan, Matthias Nießner, and Angela Dai. Hyperdiffusion: Generating implicit neural fields with weight-space diffusion, 2023.

R. M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3: 128–135, 1999. doi: 10.1016/s1364-6613(99)01294-2.

Yunhao Ge, Yuecheng Li, Shuo Ni, Jiaping Zhao, Ming-Hsuan Yang, and Laurent Itti. Clr: Channel-wise lightweight reprogramming for continual learning, 2023.

S. Grossberg. Studies of mind and brain. *Boston Studies in the Philosophy of Science*, 1982. doi: 10.1007/978-94-009-7758-7.

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=rkpACe1lx`.

Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet: More features from cheap operations, 2020. URL `https://arxiv.org/abs/1911.11907`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL `https://arxiv.org/abs/1503.02531`.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL `https://arxiv.org/abs/2106.09685`.

Steven C. Y. Hung, Cheng-Hao Tu, Cheng-En Wu, Chien-Hung Chen, Yi-Ming Chan, and Chu-Song Chen. Compacting, picking and growing for unforgetting continual learning, 2019.

Hyundong Jin and Eunwoo Kim. Helpful or harmful: Inter-task association in continual learning. In *European Conference on Computer Vision*, pp. 519–535. Springer, 2022.

Chris Dongjoo Kim, Jinseo Jeong, Sangwoo Moon, and Gunhee Kim. Continual learning on noisy data streams via self-purified replay, 2021.

J. Kirkpatrick, R. Pascanu, N. C. Rabinowitz, J. Veness, G. Desjardins, A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwińska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114:3521–3526, 2017. doi: 10.1073/pnas.1611835114.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. URL `https://api.semanticscholar.org/CorpusID:18268744`.

K. Lee, K. Lee, J. Shin, and H. Lee. Overcoming catastrophic forgetting with unlabeled data in the wild. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019. doi: 10.1109/iccv.2019.00040.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *CoRR*, abs/1606.09282, 2016. URL `http://arxiv.org/abs/1606.09282`.

Yan-Shuo Liang and Wu-Jun Li. Inflora: Interference-free low-rank adaptation for continual learning, 2024. URL `https://arxiv.org/abs/2404.00228`.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. *ArXiv*, 2024a. URL `arxiv.org/abs/2402.09353`.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32100–32121. PMLR, 21–27 Jul 2024b. URL `https://proceedings.mlr.press/v235/liu24bn.html`.

Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017. URL `http://arxiv.org/abs/1711.05101`.

S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. Technical report, 2013.

Arun Mallya and Svetlana Lazebnik. Piggyback: Adding multiple tasks to a single, fixed network by learning to mask. *CoRR*, abs/1801.06519, 2018. URL `http://arxiv.org/abs/1801.06519`.

Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pp. 109–165. Academic Press, 1989. doi: https://doi.org/10.1016/S0079-7421(08)60536-8. URL `https://www.sciencedirect.com/science/article/pii/S0079742108605368`.

Jorge A. Mendez and Eric Eaton. Lifelong learning of compositional structures, 2021. URL `https://arxiv.org/abs/2007.07732`.

Cuong V. Nguyen, Yingzhen Li, Thang D. Bui, and Richard E. Turner. Variational continual learning, 2018.

Mahdi Nikdan, Soroush Tabesh, Elvir Crnčević, and Dan Alistarh. RoSA: Accurate parameter-efficient fine-tuning via robust adaptation. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 38187–38206. PMLR, 21–27 Jul 2024. URL `https://proceedings.mlr.press/v235/nikdan24a.html`.

Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.

Oleksiy Ostapenko, Pau Rodriguez, Massimo Caccia, and Laurent Charlin. Continual learning via local module composition, 2021. URL `https://arxiv.org/abs/2111.07736`.

Shreyas Malakarjun Patil, Loizos Michael, and Constantine Dovrolis. Neural sculpting: Uncovering hierarchically modular task structure in neural networks through pruning and network analysis, 2023. URL `https://arxiv.org/abs/2305.18402`.

Bryan A. Plummer, Nikoli Dryden, Julius Frost, Torsten Hoefler, and Kate Saenko. Neural parameter allocation search, 2022. URL `https://arxiv.org/abs/2006.10598`.

A. Prabhu, P. H. S. Torr, and P. K. Dokania. Gdumb: a simple approach that questions our progress in continual learning. *Computer Vision – ECCV 2020*, pp. 524–540, 2020. doi: 10.1007/978-3-030-58536-5_31.

Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 413–420, 2009. doi: 10.1109/CVPR.2009.5206537.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning, 2017.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks, 2022.

Tim Salimans and Diederik P. Kingma. Weight normalization: a simple reparameterization to accelerate training of deep neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 901–909, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Pedro Savarese and Michael Maire. Learning implicitly recurrent cnns through parameter sharing, 2019. URL `https://arxiv.org/abs/1902.09701`.

Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4548–4557. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/serra18a.html`.

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay, 2017.

Chongjie Si, Xiaokang Yang, and Wei Shen. See further for parameter efficient fine-tuning by standing on the shoulders of decomposition. *arXiv preprint arXiv:2407.05417*, 2024.

Pravendra Singh, Vinay Kumar Verma, Pratik Mazumder, Lawrence Carin, and Piyush Rai. Calibrating cnns for lifelong learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 15579–15590. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper_files/paper/2020/file/b3b43aeeacb258365cc69cdaf42a68af-Paper.pdf`.

Tom Veniat, Ludovic Denoyer, and Marc'Aurelio Ranzato. Efficient continual learning with modular networks and task-driven priors, 2021. URL `https://arxiv.org/abs/2012.12631`.

V. K. Verma, K. J. Liang, N. Mehta, P. Rai, and L. Carin. Efficient feature transformations for discriminative and generative continual learning. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. doi: 10.1109/cvpr46437.2021.01365.

C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

Kaili Wang, Zhaopan Xu, Yukun Zhou, Zelin Zang, Trevor Darrell, Zhuang Liu, and Yang You. Neural network parameter diffusion. 2024. URL `https://api.semanticscholar.org/CorpusID:267759999`.

Yabin Wang, Zhiwu Huang, and Xiaopeng Hong. S-prompts learning with pre-trained transformers: An occam's razor for domain incremental learning, 2023. URL `https://arxiv.org/abs/2207.12819`.

Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Learning to prompt for continual learning, 2022. URL `https://arxiv.org/abs/2112.08654`.

Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

Yujia Wu, Yiming Shi, Jiwei Wei, Chengwei Sun, Yuyang Zhou, Yang Yang, and Heng Tao Shen. Difflora: Generating personalized low-rank adaptation weights with diffusion. *ArXiv*, abs/2408.06740, 2024. URL `https://api.semanticscholar.org/CorpusID:271859971`.

Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks, 2018.

Baoquan Zhang, Chuyao Luo, Demin Yu, Xutao Li, Huiwei Lin, Yunming Ye, and Bowen Zhang. Metadiff: Meta-learning with conditional diffusion for few-shot learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(15):16687–16695, Mar. 2024. doi: 10.1609/aaai.v38i15.29608. URL `https://ojs.aaai.org/index.php/AAAI/article/view/29608`.

J. Zhang, J. Zhang, S. Ghosh, D. Li, S. Tasci, L. Heck, H. Zhang, and C. J. Kuo. Class-incremental learning via deep model consolidation. *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2020. doi: 10.1109/wacv45572.2020.9093365.

Bolei Zhou, Yiyou Sun, David Bau, and Antonio Torralba. Interpretable basis decomposition for visual explanation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

Yitao Zhu, Zhenrong Shen, Zihao Zhao, Sheng Wang, Xin Wang, Xiangyu Zhao, Dinggang Shen, and Qian Wang. Melo: Low-rank adaptation is better than fine-tuning for medical image diagnosis. In *2024 IEEE International Symposium on Biomedical Imaging (ISBI)*, pp. 1–5, 2024. doi: 10.1109/ISBI56570.2024.10635615.

A APPENDIX

A.1 DATASET

In Table 3 we have summarized the class variations and number of samples for the **six** datasets we have used in our TIL experiments. All the datasets were split with $75\%-15\%-15\%$ train-validation-test split. Simple augmentations like resizing, centercropping, horizontal flips, and normalization have been applied for each.

A.2 TASKWISE PERFORMANCE BREAKDOWN

Impact of Task-order The sensitivity to task ordering varies significantly across different incremental learning approaches. Methods that rely on knowledge distillation or parameter importance estimation, such as LwF and EWC, exhibit cumulative degradation as knowledge gets diluted across sequential tasks. In particular, LwF's distillation-based approach becomes less effective for later tasks as the knowledge transfer chain grows longer. EWC's reliance on Fisher Information Matrix to protect earlier tasks' parameters can lead to representation bias, where earlier tasks disproportionately influence the learned feature space. Rehearsal-based methods like GDUMB and DER++

Table 3: Details on the Benchmarking Datasets

Dataset	Classes	#Sample
Oxford Flowers (Nilsback & Zisserman, 2008)	102	6553
FGVC Aircrafts (Maji et al., 2013)	55	10001
MIT Scenes (Quattoni & Torralba, 2009)	67	15614
CIFAR100 (Krizhevsky, 2009)	100	60000
CIFAR10 (Krizhevsky, 2009)	10	60000
CUBs (Wah et al., 2011)	200	11789

face challenges in maintaining balanced representative memory buffers across tasks, though sample selection strategies help mitigate but don't eliminate these order effects. Even masked-based approaches like HAT, despite maintaining unit-wise masks over the entire model, show order sensitivity due to the constrained mask allocation being influenced by earlier tasks. In contrast, both RECAST and several adapter-based baselines (AdaptFormer, MeLo, RoSA, DoRA) maintain absolute parameter inference through strict parameter isolation. As shown in Table 4, these methods achieve consistent performance across tasks, with no significant degradation in earlier task performance as new tasks are learned. For instance, RECAST + AdaptFormer maintains high accuracy across all domains (99.2% on Flowers, 84.7% on Aircraft, 87.1% on Scene), demonstrating that task-specific parameters can effectively adapt while the shared feature space remains stable. This architectural design choice effectively eliminates task interference, as each task's performance remains stable once learned, regardless of the order in which subsequent tasks are introduced. The empirical results suggest that methods employing strict parameter isolation are inherently robust to task ordering effects compared to approaches that modify shared parameters or rely on knowledge transfer between tasks.

Table 4: Model Performance Comparison Across Different Datasets. For InfLoRA the 6 tasks are generated by taking a chunk of 90 classes from the combined dataset - so, the classification result doesn't exactly correspond to each dataset.

Model Type	Flowers	Aircraft	Scene	CIFAR100	CIFAR10	Birds
Small ViT	97.4	66.6	82.6	72.5	94.96	81.1
L2P	0.4	5.7	0.8	56.4	75.0	77.6
InfLoRA	99.86	89.07	88.25	84.86	84.69	84.5
AdaptFormer	99.2	79.6	85.7	87.6	98.1	84.3
MeLO	99.0	84.2	84.0	87.3	98.1	80.0
Rosa (Sparse MLP 0.001%)	97.6	69.1	84.1	78.9	96.2	81.6
RoSA Full	99.4	79.1	85.2	87.4	97.9	82.6
DoRA	99.02	86.4	84.7	87.6	98.2	80.0
RECAST (Small ViT)	98.6	72.5	84.3	79.1	95.7	80.5
RECAST + AdaptFormer	99.2	84.7	87.1	88.8	98.3	84.4
RECAST + MeLO	99.4	86.3	84.3	87.1	97.9	81.0
RECAST (Sparse MLP 0.001%)	98.4	72.9	83.9	80.0	96.3	81.0
RECAST + RoSA Full	99.2	81.9	84.7	87.5	97.9	82.4
Resnet34	87.6	51.9	66.9	59.8	83.0	42.0
RECAST (Resnet34)	89.8	52.7	65.8	61.4	83.8	44.2
EWC	2.6	12.1	9.0	1.7	42.2	65.4
LWF	1.4	23.1	5.6	1.7	41.1	42.9
GDUMB	38.9	25.6	33.2	4.5	69.7	36.9
DER++	1.3	12.50	26.10	53.90	0.76	2.55
HAT	49.1	71.9	46.5	9.9	30.6	39.8
PiggyBack	93.9	85.4	74.4	82.6	96.6	65.0
RECAST + PiggyBack	95.7	87.3	77.0	81.7	96.3	66.3
CLR	93.9	83.3	73.3	75.9	94.3	59.0
RECAST + CLR	94.7	83.7	72.1	77.4	95.6	62.1

A.3 RECONSTRUCTION ANALYSIS

Pre-trained Model Details We obtained the ViT and DiNO pretrained weights from the official timm repository Wightman (2019). Resnet weights are obtained from Pytorch Hub. The specific models and their corresponding weight URLs are:

- ResNet-34: `https://download.pytorch.org/models/resnet34-b627a593.pth`
- ViT-Small: `https://storage.googleapis.com/vit_models/augreg/S_16-i21k-300ep-lr_0.001-aug_light1-wd_0.03-do_0.0-sd_0.0--imagenet2012-steps_20k-lr_0.03-res_224.npz`
- ViT-Base: `https://storage.googleapis.com/vit_models/augreg/B_16-i21k-300ep-lr_0.001-aug_medium1-wd_0.1-do_0.0-sd_0.0--imagenet2012-steps_20k-lr_0.01-res_224.npz`
- ViT-Large: `https://storage.googleapis.com/vit_models/augreg/L_16-i21k-300ep-lr_0.001-aug_medium1-wd_0.1-do_0.1-sd_0.1--imagenet2012-steps_20k-lr_0.01-res_224.npz`
- DiNO-Small Caron et al. (2021): `https://dl.fbaipublicfiles.com/dino/dino_deitsmall16_pretrain/dino_deitsmall16_pretrain.pth`

Table 5: Required runtime and hardware utilization to run Neural Mimicry (Section 2.2)

Model Scale	Peak GPU Memory (GB)	Average CPU Utilization	Per Epoch Processing (ms)	Total Epochs	Wall Clock Time (seconds)
ViT Small ($\approx 21M$)	0.33	3.7%	36.2	1000	37.4
ViT Base ($\approx 86M$)	1.2	3.7%	112	1000	102.3
ViT Large ($\approx 203M$)	4.5	3.7%	340	1000	245.0

Resource Requirements The results in Table 5 demonstrate that Neural Mimicry has a modest memory footprint (0.33-4.5GB GPU memory) with minimal CPU overhead (3.7%). It also linearly scales with model size. Furthermore, the complete reconstruction is completed in minutes even for large models. Notably, this step achieves 98-100% reconstruction accuracy (Table 6) and requires significantly less resources than model pretraining.

RECAST is architecture and scale agnostic RECAST can reconstruct models across various architecture and scales. In Table 6, we present the results of reconstructed models and the official performance of these models on the Imagenet-1K (Deng et al., 2009) dataset. All reconstructed models in the table have the same configuration - 6 groups, 2 layers in each group and 2 sets of coefficients for each target module (except ViT-Large has 12 groups). The inference comparison on Imagenet tells us that, RECAST can completely emulate the feature generation of any pretrained weight. We also report the averaged best accuracy@top1 across the datasets in Table 3, for the varying scales of the ViT models. In all cases, RECAST perfermos slightly better than baseline without coefficient-tuning, and significantly better with coefficient-tuning.

Template diversity is important for better reparameterization In Section A.3, we empirically showed that models with higher Frobenius norm value and higher entropy provides better task-adaptable model. We further breakdown this analysis by first defining the metrics, and then analyzing their layerwise trends for various reconstruction methods.

- **Frobenius Norm:** Calculates the square root of the sum of the absolute squares of its elements. In the context of RECAST, the Frobenius norm is used to measure the diversity among templates within a template bank for each group. For a template bank $\tau_g = \{T_{g,1}, T_{g,2}, \ldots, T_{g,n}\}$ in group g, the average Frobenius norm can be calculated as: $\text{avg\_frobenius}_g = \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \|T_{g,i} - T_{g,j}\|_F$. Squaring the differences makes

Model	Imagenet Acc@top1	Reconstruction Similarity
Resnet-34	73.3	-
Recon. Resnet-34	73.0	99%
ViT-Small	74.6	-
Recon. ViT-Small (MLP)	74.6	100%
Recon. ViT-Small (Attention)	74.6	100%
Recon. ViT-Small (MLP & Attention)	74.6	100%
Recon. ViT-Small (VAE + MSE)	73.8	98.5%
Recon. ViT-Small (VAE + SmoothL1)	73.0	98.6%
ViT-Base	81.1	-
Recon. ViT-Base	81.0	100%
ViT-Large	84.4	-
Recon. ViT-Large	84.4	100%
DINO Small	77.0	-
Recon. DiNO Small	76.9	99%

Table 6: Accuracy comparison of reconstructed RECAST models over the ImageNet1k dataset, against the original pretrained models from timm (Wightman, 2019) demonstrates that both backbones are empirically the same. Despite this similarity, RECAST provides reparameterization facilities with $< 0.0002\%$ coefficients

the metric more sensitive to large deviations between templates. This property helps identify when templates are significantly different, not just slightly varied.

- **Entropy** Quantifies the average amount of information contained in a set of values. For each template bank τ_g, we calculate the entropy of singular values to measure the balance of importance among the components of the templates. The average entropy for the template bank in group g is calculated as: avg_entropy$_g = \frac{1}{n}\sum_{i=1}^{n} H(T_{g(l),i})$, where $H(T_{g,i})$ is the entropy for a single template $T_{g,i}$

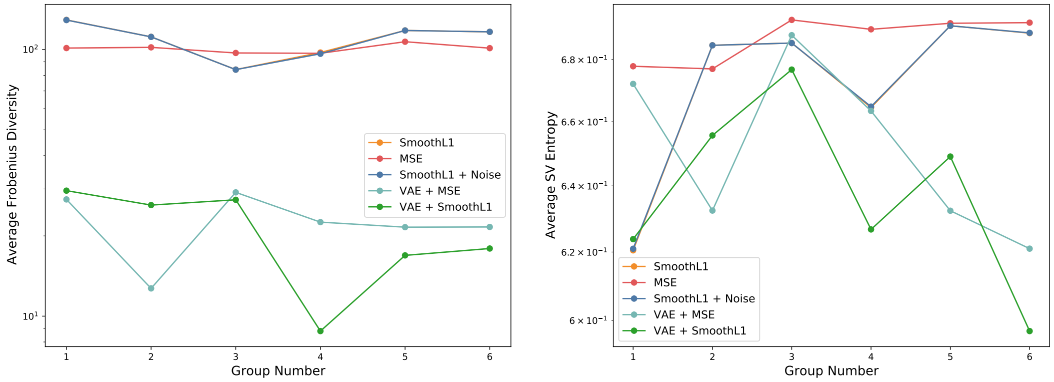
The Frobenius norm helps ensure diversity among $T_{g,i}$, allowing for a wide range of possible $W_{l,m}$. A higher score means more difference among the template. The entropy ensures that each $T_{g,i}$ is itself balanced and flexible, contributing to the adaptability of the reconstructed weights. A higher entropy corresponds to better information quality within each group. Together, these metrics provide a comprehensive view of the template characteristics in RECAST, helping to optimize both the diversity of the template bank and the flexibility of individual templates for efficient and effective weight reconstruction. In Figure 7a and 7b, we show the change of Frobenius norm and Entropy. The VAE models perform very low in terms of the Frobenius metric, although their pattern varies depending on the loss function. This is also true for the models generated through Neural Mimicry 2.2,*i.e.* models reconstructed with MSE loss and SmoothL1 loss performance almost similar on classification task but differs in entropy pattern and Frobenius norm. Another observation is that noise perturbations don't seem to impact these two metrics at all.

We define another metric the **intra-group coefficient similarity** to measure how similar the coefficient vectors are between different layers within the same group of RECAST model. For layers $l1$ and $l2$ in the same group: $sim(l1, l2) = cosine\_similarity(C_{l1}, C_{l2})$ Where C_l is the flattened and concatenated vector of all coefficients for layer l. If similarities are generally low within a group, it means each coefficient is using the shared templates in unique ways. This could indicate that each layer is capturing different features or transformations. The Figure 8, shows that the later groups in the VAE models share very high intra-group coefficient similarity indicating that layers within each group are using templates very similarly or there are redundancies in the network.

A.4 RECAST ADAPTATION

A.4.1 RECAST-ADAPTED COMPONENTS

Our framework can be adapted to various neural network components. Below we provide the details of how they may be implemented in Neural Networks.



(a) Change of Frobenius norm across the groups

(b) Change of Entropy across the groups

Figure 7: Analyzing the strategies of changing reconstruction schemes by quantifying their respective Frobenius Norm and Entropy metrics (as defined in A.3) across 6 groups of layers of a ViT. It can be observed that the VAE-reconstructed models generally demonstrate significantly low Group-wise Frobenius score and comparatively lower Entropy.

Fully-connected Layers We use RECAST here to generate the weights for the full-connected layers of a network. Here, $T_i \in \mathbb{R}^{d_{out} \times d_{in}}$. d_{out} is the number of output features and d_{in} is the number of input features. Coefficient shape, $C_i^j \in \mathbb{R}^{1\times1}$, represents a scalar value for each template with additional broadcasting dimensions. After generating final weight using Eq. 2, it's used as $Y = f(X \cdot W_{final} + b)$, where $X \in \mathbb{R}^{batch\_size \times d\_in}$ and $Y \in \mathbb{R}^{batch\_size \times d\_out}$

Attention QKV Matrices In attention mechanisms, RECAST generates a single matrix for the combined query, key, and value projections. This uses a template shape $T_i \in \mathbb{R}^{3d \times d}$, where d is the embedding dimension and coefficient shape, $C_i^j \in \mathbb{R}^{1\times1\times1}$. The weight generation process is similar to the FC layer case. The resulting W_{final} is then used to compute Q, K, and V. For input shape $X \in \mathbb{R}^{batch\_size \times seq\_length \times d}$ we can obtain $Q, K, V \in \mathbb{R}^{batch\_size \times seq\_length \times d}$

Convolution Kernels For convolutional layers, RECAST generates filter kernels that are applied across the input feature maps. Here, $T_i \in \mathbb{R}^{c_{out} \times c_{in} \times K \times K}$. c_{out} is the number of output channels, c_{in} is the number of input channels, and K is the kernel size. The weight generation process follows the same pattern as before. The resulting W_{final} is then used as the convolution kernel: $Y = conv2d(X, W_{final})$, where $X \in \mathbb{R}^{batch\_size \times c\_in \times h\_in \times w\_in}$ and $Y \in \mathbb{R}^{batch\_size \times c\_out \times h\_out \times w\_out}$

A.4.2 RECAST INTEGRATION WITH EXISTING METHODS

RECAST complements existing incremental learning methods as a task-adaptable weight generation backbone. For CNN architectures, RECAST generates base weights that PiggyBack's binary masks can modulate and provides primary convolution weights that CLR's Ghost modules can refine. In transformer architectures, RECAST produces core weight matrices that methods like MeLO, AdaptFormer (for query/value and MLP components), DoRA (for magnitude-direction decomposition), and RoSA (for sparse adaptations) can then modify.

For example, in standard LoRA, a weight matrix $W \in \mathbb{R}^{d_{out} \times d_{in}}$ is modified by adding a low-rank update:

$$W_{\text{LoRA}} = W_0 + BA \tag{4}$$

where $B \in \mathbb{R}^{d_{out} \times r}$, $A \in \mathbb{R}^{r \times d_{in}}$, and r is the rank.

When integrating with RECAST, instead of using a fixed W_0, we use the dynamically generated weight matrix from RECAST. From Equation 1

$$W_{\text{RECAST}} = \frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{n} T_{g,i} \cdot C_{l,m,i}^{k} \tag{5}$$

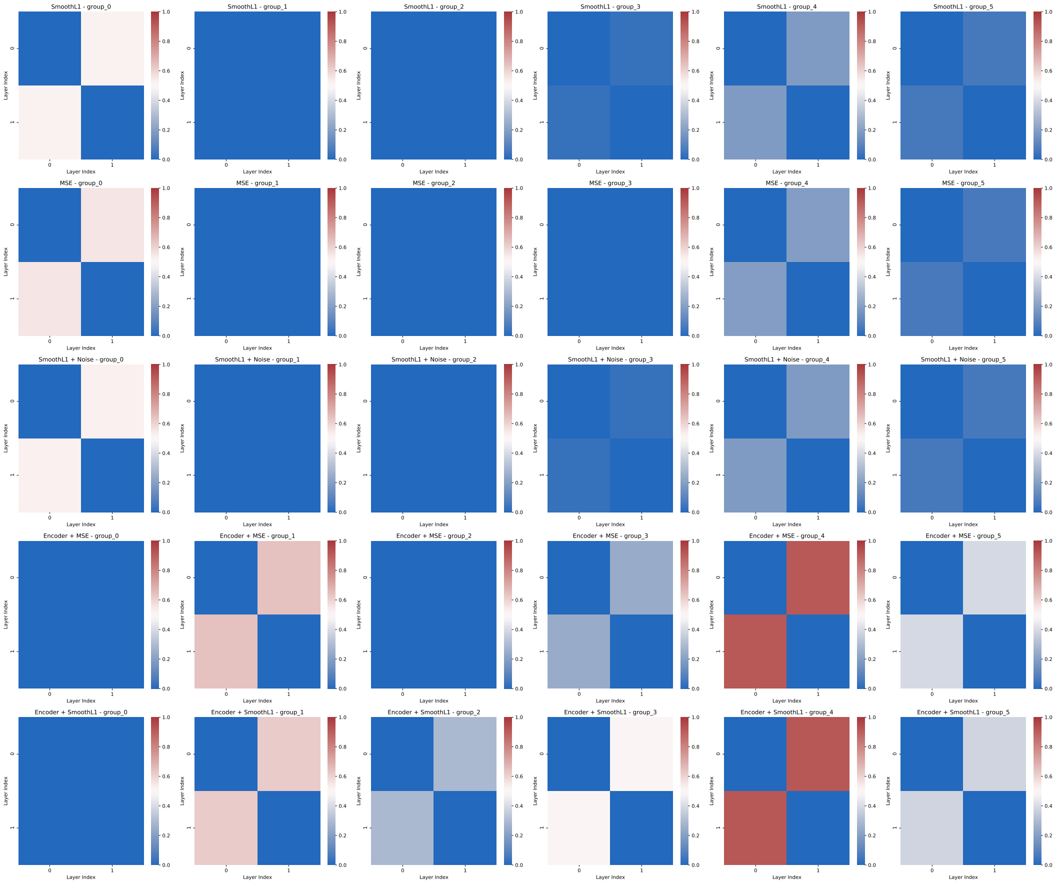


Figure 8: Intra-group Coefficient Similarity across the groups of similar configuration models, that have been reconstructed in different ways. The later groups in the VAE/Encoder models share very high intra-group coefficient similarity and that layers within each group are using templates very similarly or there are redundancies in the network.

where $T_{g,i}$ are the templates and $C^k_{l,m,i}$ are the coefficients. The final integrated approach combines both methods:

$$W_{\text{combined}} = W_{\text{RECAST}} + BA \tag{6}$$

The total trainable parameters for the combined approach are:

- RECAST: $O(nK)$ coefficients per module
- LoRA: $O(r(d_{in} + d_{out}))$ parameters per module

where n is the number of templates, K is the number of coefficient sets, and r is the LoRA rank.

Similar principles apply when integrating with other methods:

CLR (Channel-wise Lightweight Reprogramming):

$$Y = \text{Ghost}(\text{Conv}(X, W_{\text{RECAST}})) \tag{7}$$

where Ghost($\cdot$) represents the lightweight channel-wise transformations through depthwise-separable convolutions.

PiggyBack:

$$W_{\text{combined}} = W_{\text{RECAST}} \odot M \tag{8}$$

where M is a binary mask matrix.

DoRA:

$$W_{\text{combined}} = \|W_{\text{RECAST}}\| \cdot \frac{W_{\text{RECAST}} + BA}{\|W_{\text{RECAST}} + BA\|} \tag{9}$$

where $\|\cdot\|$ denotes the magnitude and BA is the LoRA update.

RoSA:

$$W_{\text{combined}} = W_{\text{RECAST}} + S \odot W_{\text{RECAST}} + (BA) \tag{10}$$

where S is a sparse binary mask applied to the LoRA update.