CLOSED-FORM MERGING OF PARAMETER-EFFICIENT MODULES FOR FEDERATED CONTINUAL LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Model merging has emerged as a crucial technique in Deep Learning, enabling the integration of multiple models into a unified system while preserving performance and scalability. In this respect, the compositional properties of low-rank adaptation techniques (e.g., LoRA) have proven beneficial, as simple averaging LoRA modules yields a single model that mostly integrates the capabilities of all individual modules. Building on LoRA, we take a step further by imposing that the merged model matches the responses of all learned modules. Solving this objective in closed form yields an indeterminate system with A and B as unknown variables, indicating the existence of infinitely many closed-form solutions. To address this challenge, we introduce LoRM, an alternating optimization strategy that trains one LoRA matrix at a time. This allows solving for each unknown variable individually, thus finding a unique solution. We apply our proposed methodology to Federated Class-Incremental Learning (FCIL), ensuring alignment of model responses both between clients and across tasks. Our method demonstrates state-of-the-art performance across a range of FCIL scenarios.

1 INTRODUCTION

Humans naturally excel at learning a diverse array of skills independently, effortlessly acquiring knowledge across multiple domains throughout their lives. In contrast, the traditional paradigm for artificial neural networks relies on training a unified model on a single, large dataset. While this approach facilitates the simultaneous incorporation of different skills, it lacks the capacity for specialized or incremental learning, making it less adaptable and responsive to changes in the environment. To overcome this limitation and mimic human flexibility, various paradigms have been developed to enhance neural networks' ability to manage diverse skills effectively. Multi-Task Learning (Caruana, 1997) involves training a model on several tasks simultaneously, promoting the sharing of representations across tasks, while Continual Learning (CL) (McCloskey & Cohen, 1989) focuses on enabling models to learn tasks incrementally without forgetting prior knowledge. Federated Learning (FL) (McMahan et al., 2017), on the other hand, focuses on decentralized training by distributing data and computation across separate clients, each specializing in their local task. While each of these scenarios has its own unique characteristics, they all share the common objective of integrating task-specific modules into a unified framework.

Recently, large pre-trained architectures have facilitated model editing (Ortiz-Jimenez et al., 2024) and specialization (Bowman et al., 2023), particularly for fine-tuning downstream tasks. In practice, deep models often leave their parameters fixed, leveraging Parameter-Efficient Fine-Tuning (PEFT) techniques to adapt to new tasks effectively. Among PEFT methods, Low-Rank Adaptation (LoRA) (Hu et al., 2022) has emerged as a prominent approach. LoRA introduces residual weights in the form of $\Delta \boldsymbol{W} = \boldsymbol{B}\boldsymbol{A}$, where $\boldsymbol{B}$ and $\boldsymbol{A}$ are low-rank matrices. These residuals, commonly referred to as task vectors (Ilharco et al., 2023), form the foundation of the novel model merging literature, which has introduced various approaches for their integration. For example, Zhang et al. (2023c) explore the combination of task vectors through linear arithmetic operations, while other works focus on identifying optimal coefficients for weighting these modules during aggregation (Yadav et al., 2024; Yang et al., 2024; Wu et al., 2024). In contrast to these methods, we propose a novel solution that merges LoRA modules in a *closed-form*.

To evaluate the feasibility of our approach, we base our investigation on a well-defined empirical framework, situating our work at the intersection of Federated Learning and Continual Learning. These two paradigms are ideal for assessing the merging of task vectors, as they encompass both spatial (across clients) and temporal (over tasks) aggregation. Specifically, in Federated Class-Incremental Learning (FCIL) (Yoon et al., 2021), tasks are introduced incrementally, and data is distributed across multiple clients in a decentralized manner.

We introduce a novel approach, termed Low-rank Regression Mean (LoRM), tailored to the FCIL setting. Our method builds upon RegMean (Jin et al., 2023), a model merging technique derived from an exactly solvable regression problem. Starting from RegMean's formulation, we develop a strategy to merge LoRA modules in closed form. Our derivations result in two key equations — one for merging matrix $\boldsymbol{A}$ and another one for matrix $\boldsymbol{B}$. During training, we propose an *alternating* optimization procedure, where one matrix is learned while the other remains fixed across models. In the Federated Class-Incremental Learning setting, spatial aggregation across clients is performed using this alternating procedure. Conversely, for temporal aggregation, task-specific modules are merged by applying the RegMean formulation directly to the full residual weights $\Delta\boldsymbol{W}$ of all tasks.

In summary, the key contributions of this work are as follows:

- We explore the feasibility of merging LoRA modules using a closed-form solution.
- We introduce LoRM, a novel approach for Federated Class-Incremental Learning that leverages insights from our exploration.
- We demonstrate the effectiveness of our method across diverse datasets and varying degrees of data distribution, achieving state-of-the-art results.

2 Background and Motivation

In this section, we present the theoretical foundations behind LoRM, alongside its motivation and the problem setting in which we situate our empirical evaluation.

2.1 Preliminaries

LoRA (Hu et al., 2022) was introduced to reduce the number of trainable parameters when fine-tuning pre-trained models. Formally, let $\boldsymbol{W}_0 \in \mathbb{R}^{d\times k}$ represent the matrix of pre-trained weights of a linear layer, and let $\boldsymbol{x} \in \mathbb{R}^{k\times 1}$ be the input vector for that layer. The output $\boldsymbol{h}$ is given by:

$$\boldsymbol{h} = \boldsymbol{W}_0\boldsymbol{x} + \Delta\boldsymbol{W}\boldsymbol{x} = \boldsymbol{W}_0\boldsymbol{x} + \boldsymbol{B}\boldsymbol{A}\boldsymbol{x}, \tag{1}$$

where $\Delta\boldsymbol{W} = \boldsymbol{B}\boldsymbol{A}$ is the residual weight introduced by LoRA, with matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ as the only components being trained. The efficiency of this approach stems from the low rank r of the matrices, where $\boldsymbol{B} \in \mathbb{R}^{d\times r}$ and $\boldsymbol{A} \in \mathbb{R}^{r\times k}$, with d and k representing the number of output and input features of the layer, respectively. Consequently, the number of trainable parameters is $r\cdot(d+k)$, which, since $r \ll d$, constitutes only a fraction of the $d \cdot k$ parameters required for full fine-tuning. Additionally, $\boldsymbol{B}$ is initialized to 0: *i.e.*, the first forward pass is equivalent to the absence of a LoRA residual.

RegMean (Jin et al., 2023) introduces a method for merging a collection of N linear layers $\{\boldsymbol{W}_i\}_{i=1}^N$, each corresponding to N distinct models trained on distinct inputs $\{\boldsymbol{X}_i\}_{i=1}^N$. The goal is to identify a single linear layer that produces responses that closely match those of the starting layers. Specifically, the objective function is defined as follows[1]:

$$\text{minimize } \Omega = \sum_{i=1}^{N} \|\boldsymbol{W}_M\boldsymbol{X}_i - \boldsymbol{W}_i\boldsymbol{X}_i\|_2^2. \tag{2}$$

By computing the gradient of Ω with respect to $\boldsymbol{W}_M$ and setting it equal to zero, a closed-form solution is obtained. Notably, the merged layer $\boldsymbol{W}_M$ is computed as follows:

$$\boldsymbol{W}_M = \left(\sum_{i=1}^{N} \boldsymbol{W}_i\boldsymbol{X}_i\boldsymbol{X}_i^\top\right)\left(\sum_{i=1}^{N} \boldsymbol{X}_i\boldsymbol{X}_i^\top\right)^{-1}. \tag{3}$$

[1]We rework the original formulation to ease subsequent derivations.

In this context, $\boldsymbol{X}_i \in \mathbb{R}^{k \times \text{samples}}$ represents the input to the i-th layer being merged. Hence, the Gram matrix $\boldsymbol{X}_i \boldsymbol{X}_i^\top$ has dimensions $k \times k$ features.

While RegMean finds a weight matrix that approximates the outputs of all layers considered, it does not take into account starting from pre-trained weights or the use of low-rank modules (*e.g.*, LoRA). Given these considerations, we ask whether this method can be suitably adapted to merge LoRA modules. In other words:

Can we devise a strategy to merge LoRA modules in a closed form?

To further explore this question from an experimental standpoint, we place our investigation in the context of a Federated Class-Incremental Learning scenario.

2.2 Problem Setting

In Federated Class-Incremental Learning, the dataset $\mathcal{D}$ is first divided into T tasks, each consisting of a distinct set of classes. Then, each partition $\mathcal{D}^t$ corresponding to the t-th task is further distributed among N clients, resulting in $\mathcal{D}_i^t$ for the i-th client. Similar to standard Class-Incremental Learning (Van de Ven et al., 2022), the task-specific partitions $\{\mathcal{D}^t\}_{t=1,\dots,T}$ arrive sequentially.

In this federated scenario, the training for each task is conducted over multiple communication rounds. During each round, clients are restricted to learning only from their local dataset $\mathcal{D}_i^t$. The local optimization objective for client i, based on the loss function $\mathcal{L}$, can be formally expressed as:

$$\underset{\theta_i}{\text{minimize}} \ \mathbb{E}_{(x,y)\sim\mathcal{D}_i^t} \left[\mathcal{L}(f(x;\theta_i), y)\right], \tag{4}$$

where x and y denote the inputs and corresponding labels, respectively, with θ_i representing the set of parameters for client i, and $f(\cdot;\theta_i)$ denoting the associated model.

After completing local updates, each client sends its model parameters θ_i to the central server, where they are aggregated with those from other clients. The server then sends the global aggregated model back to the clients, marking the end of a communication round. This process repeats for several rounds until the training for task t is completed. Once all rounds for task t have finished, the system progresses to the next task $t+1$ using the corresponding dataset $\mathcal{D}^{t+1}$. The ultimate objective is to obtain a global model, derived from the aggregation of local models performed by the server, that functions well across all incremental tasks and successfully integrates the distributed knowledge.

3 Methodology

3.1 A closed-form solution for LoRA merging

Using the same notation as in Equation 2, let $\boldsymbol{W}$ denote the weight matrix of a given layer. Since LoRA (see Equation 2) is applied to each linear layer across all clients, we express the weight matrix for the i-th client as $\boldsymbol{W}_i = \boldsymbol{W}_0 + \boldsymbol{B}_i\boldsymbol{A}_i$, where $\boldsymbol{W}_0$ is the shared pre-trained weight matrix, and $\boldsymbol{B}_i$, $\boldsymbol{A}_i$ represent the low-rank matrices specific to the client. This formulation is consistent across clients, as they all utilize the same model architecture. At the end of each communication round, after conducting local training on the LoRA matrices, the goal is to merge the corresponding matrices (*i.e.*, $\boldsymbol{A}$'s with $\boldsymbol{A}$'s and $\boldsymbol{B}$'s with $\boldsymbol{B}$'s) using the closed-form solution derived from RegMean. Starting from Equation 2, our objective becomes:

$$\Omega = \sum_{i=1}^{N} \|(\boldsymbol{W}_0 + \boldsymbol{B}_M\boldsymbol{A}_M)\boldsymbol{X}_i - (\boldsymbol{W}_0 + \boldsymbol{B}_i\boldsymbol{A}_i)\boldsymbol{X}_i\|_2^2. \tag{5}$$

To find the optimal $\boldsymbol{A}_M$ and $\boldsymbol{B}_M$ that minimize Ω, we differentiate Ω with respect to each variable, one at a time, and set the gradients to zero:

$$\begin{cases} \dfrac{\partial\Omega}{\partial\boldsymbol{B}} = 0 \\ \dfrac{\partial\Omega}{\partial\boldsymbol{A}} = 0 \end{cases} \tag{6}$$

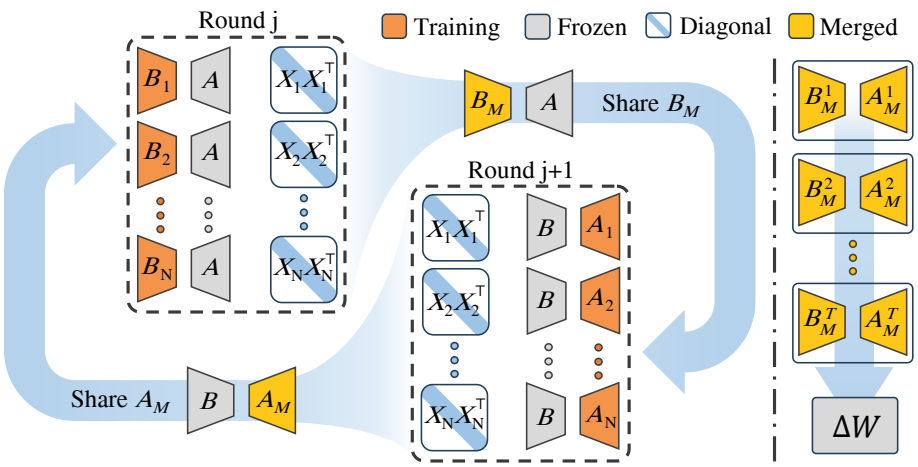

Figure 1: Training and aggregation procedure of LoRM, alternating between rounds. The figure depicts both Federated and Incremental aggregation for a generic layer.

However, the system reveals indeterminate, as the two equations exhibit linear dependence on one another during the calculations. For further mathematical derivations illustrating the infeasibility of this approach, refer to appendix A.1.

As a solution, we propose freezing one of the two matrices. This means that either $\boldsymbol{A}$ or $\boldsymbol{B}$ is shared across clients (*i.e.*, treated as a constant), and a single closed-form equation determines how to merge the trainable matrices across clients. If we choose to share $\boldsymbol{A}$, the merged $\boldsymbol{B}_M$ is obtained as:

$$\boldsymbol{B}_M = \left(\sum_{i=1}^{N} \boldsymbol{B}_i \boldsymbol{A} \boldsymbol{X}_i \boldsymbol{X}_i^\top \right) \boldsymbol{A}^\top \left(\boldsymbol{A} \sum_{i=1}^{N} \boldsymbol{X}_i \boldsymbol{X}_i^\top \boldsymbol{A}^\top \right)^{-1}. \tag{7}$$

Note that $\boldsymbol{A}$ does not have a subscript i, as it is identical for all clients. Instead, if we opt to share $\boldsymbol{B}$, the merged $\boldsymbol{A}_M$ is computed as:

$$\boldsymbol{A}_M = \left(\sum_{i=1}^{N} \boldsymbol{A}_i \boldsymbol{X}_i \boldsymbol{X}_i^\top \right) \left(\sum_{i=1}^{N} \boldsymbol{X}_i \boldsymbol{X}_i^\top \right)^{-1}. \tag{8}$$

For the formal derivation of these equations, refer to appendices A.1.1 and A.1.2.

3.2 LoRM

Having established closed-form solutions for weight merging, we now outline the full procedure of LoRM, illustrated in Figure 1. Each client i begins by optimizing its own $\boldsymbol{B}_i$, which is learned during local training, and a shared $\boldsymbol{A}$, initialized and distributed by the server. In this first round, it is essential to freeze $\boldsymbol{A}$, as LoRA's $\boldsymbol{B}$ is initialized to 0. Freezing $\boldsymbol{B} = 0$ across all clients would render the training ineffective. At the end of each round, each client i computes the Gram matrix $\boldsymbol{X}_i \boldsymbol{X}_i^\top$ with a forward pass on all examples[2]. Then, it sends $\boldsymbol{B}_i$ and $\boldsymbol{X}_i \boldsymbol{X}_i^\top$ to the server, where the merging operation (as described in Equation 7) is performed. The resulting $\boldsymbol{B}_M$ is then sent back to the clients, who begin the next communication round.

Alternated optimization. Empirically, we observe that consistently training only the matrix $\boldsymbol{B}$ while keeping $\boldsymbol{A}$ fixed can be limiting (see Section 4.3 for a detailed analysis). Therefore, to fully exploit LoRA's representational potential, we introduce an *alternating* training approach, where the matrix to be updated changes at each round. Specifically, after the first round, we freeze $\boldsymbol{B}$, which is already synchronized across clients due to the previous server aggregation, and train $\boldsymbol{A}$ instead. Then, at the end of the second round, all local $\boldsymbol{A}$'s are aggregated using Equation 8. This strategy also improves

[2]Note that this process provides a Gram matrix for every layer in the network. In this discussion, $\boldsymbol{X}_i \boldsymbol{X}_i^\top$ is related to the single generic linear layer.

Algorithm 1 – **LoRM**, for a generic linear layer Round j Round j + 1

Input: T tasks; N clients residual modules $\Delta \boldsymbol{W}_i = \boldsymbol{B}_i \boldsymbol{A}_i$; layer's pre-trained weights $\boldsymbol{W}_0$.
for each task $t \in \{1, \ldots, T\}$ **do**
  for each communication round **do**
    Clients side:
    for each client $i \in \{1, \ldots, N\}$ **in parallel do**
      $\boldsymbol{A}^t = \boldsymbol{A}_M^t$ **or** $\boldsymbol{B}^t = \boldsymbol{B}_M^t$ ▷ Fix either matrix
      for each epoch **do**
        for each input $\boldsymbol{x}$ **do**
          $\boldsymbol{h} = \boldsymbol{W}_0 \boldsymbol{x} + \boldsymbol{B}_i^t \boldsymbol{A}^t \boldsymbol{x}$ **or** $\boldsymbol{h} = \boldsymbol{W}_0 \boldsymbol{x} + \boldsymbol{B}^t \boldsymbol{A}_i^t \boldsymbol{x}$ ▷ Forward pass
          Optimize $\boldsymbol{B}_i^t$ **or** Optimize $\boldsymbol{A}_i^t$
        end for
      end for
      Send to the server $\mathrm{diag}(\boldsymbol{X}_i^t \boldsymbol{X}_i^{t\top})$
      Send $\boldsymbol{B}_i^t$ to the server **or** Send $\boldsymbol{A}_i^t$ to the server
    end for
    Server side:
    Use Equation 7 and distribute $\boldsymbol{B}_M^t$ **or** Use Equation 8 and distribute $\boldsymbol{A}_M^t$
  end for
end for
Use Equation 9 to compute the final residual module ▷ RegMean across tasks

efficiency, as only one of the two matrices needs to be communicated per round, offering a significant advantage compared to transmitting the whole LoRA module.

Merge task-specific modules. At the conclusion of the generic task t, the merged matrices $\boldsymbol{B}_M^t$ and $\boldsymbol{A}_M^t$ are multiplied to obtain the task-specific residual module: $\Delta \boldsymbol{W}^t = \boldsymbol{B}_M^t \boldsymbol{A}_M^t$. On the server side, when aggregating modules from all tasks at the end of the training, we apply the standard RegMean formulation (Equation 3) to merge the aforementioned residuals as:

$$\Delta \boldsymbol{W} = \left(\sum_{t=1}^{T} \boldsymbol{B}_M^t \boldsymbol{A}_M^t \boldsymbol{X}^t \boldsymbol{X}^{t\top} \right) \left(\sum_{t=1}^{T} \boldsymbol{X}^t \boldsymbol{X}^{t\top} \right)^{-1}. \tag{9}$$

To accomplish this, the server stores the task-specific weight $\{\Delta \boldsymbol{W}^1, \ldots, \Delta \boldsymbol{W}^T\}$ throughout the training process, along with the global Gram matrices $\{\boldsymbol{X}^t \boldsymbol{X}^{1\top}, \ldots, \boldsymbol{X}^t \boldsymbol{X}^{T\top}\}$. Due to the associative property of matrix multiplication, the computation of the latter is straightforward, achieved by summing the client-specific Gram matrices shared by the clients at the end of each task t: $\boldsymbol{X}^t \boldsymbol{X}^{t\top} = \sum_{i=1}^{N} \boldsymbol{X}_i^t \boldsymbol{X}_i^{t\top}$. It is worth noting that, for what concerns the classification layer, concatenating the task-specific classification heads is equivalent to applying RegMean. We provide a mathematical derivation of this result in appendix A.4.

The overall procedure yields a final weight residual $\Delta \boldsymbol{W}$ that promotes both generalization across tasks and robustness to knowledge distribution. The pseudocode for LoRM, applied to a generic linear layer, is presented in Algorithm 1.

3.3 LoRM Characteristics for Federated Class-Incremental Learning

In Federated Class-Incremental Learning (FCIL), data privacy, efficiency, and the rate of convergence are crucial concerns. LoRM is specifically designed to address these challenges.

Privacy-preserving. In FCIL, inputs $\boldsymbol{X}_i$ for the i-th client's generic layer cannot be transmitted to the central server, as doing so, especially for the first layer, would compromise data privacy by effectively sharing the dataset. In LoRM, instead of transmitting $\boldsymbol{X}_i$ directly, we send the Gram matrix $\boldsymbol{X}_i \boldsymbol{X}_i^\top$, which obfuscates the original data. Furthermore, only the diagonal of the Gram matrix is communicated, ensuring that no local data is exposed.

Efficiency. The use of LoRA inherently improves efficiency compared to full fine-tuning, as it necessitates the communication of only two low-rank matrices for each layer. Additionally, LoRM's alternating optimization procedure permits the transmission of only one matrix per communication round, further reducing overhead. The use of solely the diagonal of the Gram matrix mitigates communication costs even further, as it avoids the need to transmit the full matrix. As a result, at the conclusion of each communication round, each client transmits a low-rank matrix and a vector to the server for each layer.

Rate of convergence. Ultimately, LoRM demonstrates faster convergence compared to other FCIL baselines, as further discussed in Section 4.2.

4 Experimental Study

4.1 Evaluation settings

Datasets. The importance of evaluating pre-trained models on datasets that deviate significantly from their pre-training domain is well-established in the literature (Kolesnikov et al., 2020; Kornblith et al., 2019). Accordingly, we divide the evaluation of LoRM into two parts: one focusing on *in-domain* datasets and the other on an *out-of-domain* dataset. For in-domain evaluation, we use CIFAR-100 (Krizhevsky et al., 2009) and ImageNet-R (Hendrycks et al., 2021). For out-of-domain evaluation, we employ EuroSAT (Helber et al., 2018), a satellite dataset recognized by Oh et al. (2022) as one of the most challenging for domain adaptation from ImageNet-21k pre-training. CIFAR-100 and ImageNet-R are partitioned into 10 incremental tasks, with each task consisting of 10 and 20 classes, respectively. EuroSAT is divided into 5 tasks, each containing 2 classes. The data is distributed across 10 clients using the commonly adopted *distribution-based* label imbalance setting (Li et al., 2022; Yurochkin et al., 2019), where partitioning is governed by a Dirichlet distribution parameterized by β. A smaller β value corresponds to a more challenging data distribution. We evaluate all methods across three scenarios for each dataset, using $\beta \in 0.5, 0.1, 0.05$ for the in-domain datasets and $\beta \in 1, 0.5, 0.2$ for the out-of-domain one. For further details on data preprocessing, including the dataset-specific augmentations used, refer to appendix B.

Evaluated approaches. We compare LoRM against 10 competing methods spanning different fields. From Continual Learning, we evaluate EWC (Kirkpatrick et al., 2017), LwF (Li & Hoiem, 2017), DER++ (Buzzega et al., 2020), L2P (Wang et al., 2022b), and CODA-Prompt (Smith et al., 2023). Following previous studies (Zhang et al., 2023b; Guo et al., 2024), we adapt these methods to the federated domain by merging client weights using the FedAvg algorithm (McMahan et al., 2017). We also include FisherAvg (Matena & Raffel, 2022) and RegMean (Jin et al., 2023), from the model merging literature, and CCVR (Luo et al., 2021), a method from Federated Learning. These approaches are adapted for Continual Learning by applying Asymmetric Cross-Entropy (ACE) (Caccia et al., 2022), where the classification heads are optimized separately for each task. Additionally, we evaluate two algorithms specifically designed for Federated Class-Incremental Learning: TARGET (Zhang et al., 2023b) and PLoRA (Guo et al., 2024), the latter representing the current State Of The Art. Finally, we include Joint results as an upper bound, achieved by training the backbone on the entire dataset without any federated or incremental partitioning.

Implementation Details. As the backbone for LoRM and all competing approaches, we employ a pre-trained ViT-B/16 model (Dosovitskiy et al., 2021). Specifically, the model is initialized with pre-trained weights from ImageNet-21K (Ridnik et al., 2021) for all evaluated datasets. Each task is trained over 5 communication rounds, with each round consisting of 5 epochs. For a complete overview of the method-specific hyperparameters, refer to appendix D.

4.2 Results

In Tables 1 and 2, we present the results of the evaluated approaches in terms of Final Average Accuracy (FAA). For a formal definition of this metric, please refer to appendix B.4. The reported results are averaged over 3 runs, with standard deviations provided in appendix C.1.

In the context of in-domain datasets (Table 1), models tend to perform better on CIFAR-100 than on ImageNet-R, indicating the former as a simpler task, partially due to having fewer classes. Continual Learning techniques such as LwF and EWC exhibit low to moderate performance in scenarios

Table 1: *In-domain SOTA comparison*. Results are reported in terms of FAA(%). Best-performing methods are indicated in bold, while the second bests are underlined.

	CIFAR-100			ImageNet-R		
Joint	91.59			86.31		
Partition	$\beta = 0.5$	$\beta = 0.1$	$\beta = 0.05$	$\beta = 0.5$	$\beta = 0.1$	$\beta = 0.05$
EWC (Kirkpatrick et al., 2017)	71.31	71.75	63.48	56.77	55.67	53.90
LwF (Li & Hoiem, 2017)	54.18	55.12	42.44	49.78	45.03	41.60
DER++ (Buzzega et al., 2020)	67.90	61.97	55.79	55.83	50.07	45.52
L2P (Wang et al., 2022b)	<u>76.30</u>	52.82	45.41	57.58	46.82	37.57
CODA-P (Smith et al., 2023)	62.00	22.21	16.12	48.02	29.68	21.52
FisherAvg (Matena & Raffel, 2022)	70.43	70.10	67.87	57.05	54.05	51.55
RegMean (Jin et al., 2023)	71.07	64.56	61.47	61.05	57.32	49.22
CCVR (Luo et al., 2021)	73.56	<u>78.04</u>	<u>79.47</u>	<u>68.68</u>	<u>69.55</u>	**71.83**
TARGET (Zhang et al., 2023b)	69.36	69.18	64.23	57.32	58.75	55.70
PILoRA (Guo et al., 2024)	75.34	73.27	68.34	53.03	51.40	47.33
LoRM (ours)	**83.19**	**82.05**	**81.21**	**75.62**	**70.83**	<u>70.18</u>

with limited data heterogeneity. Surprisingly, employing a rehearsal buffer (DER++) offers minimal benefit. Prompt-based CL approaches similarly show rapid performance degradation as the distribution parameter β decreases, although L2P demonstrates strong performance at $\beta = 0.5$, ranking second overall. Among FL and model merging methods, which generally outperform CL approaches, CCVR emerges as the top performer, achieving state-of-the-art accuracy in one setting (ImageNet-R, $\beta = 0.05$). Interestingly, CCVR exhibits a reverse trend with respect to increasing data heterogeneity: as heterogeneity grows, its performance improves. TARGET and PILoRA, despite being designed specifically for FCIL, show lower performance relative to the adapted CCVR, a result we attribute to CCVR's server-side calibration procedure using centralized synthetic data. Finally, due to its closed-form merging technique, LoRM achieves state-of-the-art results across all settings except for one (*i.e.*, Imagenet-R, $\beta = 0.05$).

On the EuroSAT dataset (Table 2), the previous experimental findings are reversed for most methods. Specifically, DER++ emerges as the second-best performer across all levels of increasing heterogeneity, highlighting the value of utilizing a memory buffer when adapting to different distributions. L2P follows closely, benefiting from its prompt pool, which proves crucial when deviating from the pre-training weights. Conversely, CCVR's advantage vanishes in this setting, as merely calibrating the classifier fails to extract meaningful representations. Target and PILoRA perform on par with CCVR but fall short of the top performer, LoRM, which surpasses all approaches by a significant margin[3].

Rate of convergence. Figure 2 illustrates the convergence rates of LoRM and the FCIL competitors on the first task of ImageNet-R with $\beta = 0.05$. The steeper slope of the curve for LoRM, compared to that of Target and PILoRA, showcases the faster convergence of our approach. This behavior stems from the closed-form solution applied for merging LoRA at each communication round, which accelerates convergence by leveraging its optimality. In contrast, PILoRA applies simple averaging of LoRA modules, while Target does not utilize PEFT modules at all. Additionally, the slower convergence of PILoRA compared to Target can be attributed to its prototype-based classification procedure, which necessitates a higher number of training rounds to achieve well-refined prototypes.

4.3 Ablation Studies

Alternating vs. Only B. We investigate whether training the $\boldsymbol{B}$ matrix only (*i.e.*, following the same procedure as LoRM but applying solely Equation 7 to merge $\boldsymbol{B}$ matrices) could lead to improved performance compared to our alternating optimization approach. We conduct an experiment on EuroSAT with $\beta = 1.0$ (see Figure 3), showing that the *alternating* strategy outperforms the *only*

[3]Less relevant methodologies are omitted from Table 2 and are reported in Table C.

Table 2: *Out-of-domain SOTA comparison.* Results are reported in terms of FAA(%).

Joint		93.72	
Partition	$\beta = 1.0$	$\beta = 0.5$	$\beta = 0.2$
DER++	66.72	65.20	65.44
L2P	64.31	62.75	64.42
CCVR	47.41	36.07	43.90
TARGET	36.78	37.81	43.69
PILoRA	40.93	34.02	39.64
LoRM	**80.44**	**77.37**	**75.12**

Table 3: *LoRM components*. Evaluation of the different components of LoRM. Results are reported in terms of FAA (%).

	ImageNet-R	EuroSAT
Partition	$\beta = 0.5$	$\beta = 1.0$
FedAvg	57.65	41.04
FedAvg *w/* LoRA	57.00	53.14
RegMean	61.05	44.84
LoRM *w/o* Equation 9	75.08	71.40
LoRM	75.62	77.37

B approach. This result underscores the importance of leveraging the entire LoRA representation capabilities to achieve superior performance.

LoRM components. To evaluate the contribution of each component to the overall performance of LoRM, we conduct an ablation study by incrementally introducing each part and comparing it with baseline methods. These experiments are performed on EuroSAT with $\beta = 1.0$ and ImageNet-R with $\beta = 0.5$. The initial baseline for both datasets is FedAvg (McMahan et al., 2017) combined with ACE (Caccia et al., 2022). Notably, leveraging LoRA leads to a remarkable performance improvement on EuroSAT but does not affect ImageNet-R. Meanwhile, using the closed-form aggregation of RegMean, enhanced with ACE, establishes an additional baseline. Building upon this, our alternating optimization strategy, when applied without Equation 9 (*i.e.*, simply averaging task-specific residuals), substantially boosts performance on both datasets, with a more pronounced impact on the out-of-domain dataset. Finally, applying RegMean to merge task-specific residuals further enhances robustness in addressing FCIL challenges, resulting in the final form of LoRM.

5 Relation with prior works

Parameter-Efficient Fine-Tuning

Adapting deep neural networks to new tasks typically requires full re-training of all parameters, which is computationally demanding. Parameter-Efficient Fine-Tuning (PEFT) addresses this by updating only a small subset of the model parameters, leaving the rest of the network unchanged. One of the earliest methods in this domain is Adapters (Houlsby et al., 2019), which are lightweight neural modules inserted between the layers of a pre-trained model to facilitate task adaptation. More recently, a prominent class of PEFT techniques includes Prompt Tuning (Lester et al., 2021) and Prefix Tuning (Li & Liang, 2021), both of which introduce learnable embeddings, or prompts, appended to the layers' input tokens. Prompt Tuning prepends these embeddings directly to the initial input sequence (Wang et al., 2022b;a), whereas Prefix Tuning concatenates them to specific attention layers (Smith et al., 2023). An alternative approach to prompting is Low-Rank Adaptation (LoRA) (Hu et al., 2022), which injects residual parameters into the pre-trained weights using low-rank matrices, significantly reducing the number of learnable parameters. While recent works have explored various alternatives to LoRA(Kopiczko et al., 2024; Zhang et al., 2023d; Renduchintala et al., 2024; Liu et al., 2022), our research is grounded in the original LoRA framework, leveraging its efficiency and the extensive literature on model merging that goes with it.

Model merging

Task-specific modules are typically deployed to address individual tasks. However, an alternative approach involves merging these modules to create a model capable of generalizing across all tasks. A notable line of research within model merging focuses on combining task vectors (Ilharco et al., 2023), specifically those generated by LoRA-like methods. Early studies have explored the use of linear arithmetic operations (*e.g.*, addition and subtraction) to combine these modules (Zhang et al., 2023c). Expanding on this, subsequent works have focused on optimizing the coefficients that determine the contribution of each LoRA module during aggregation (Huang et al., 2023; Yang

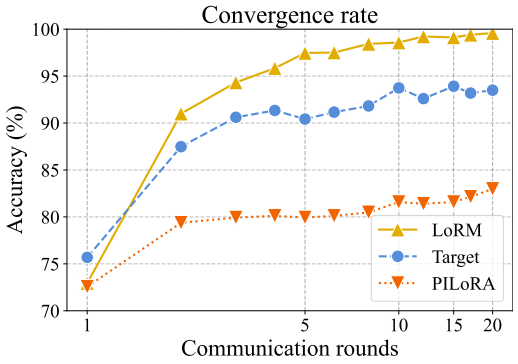

Figure 2: *Convergence rate*.

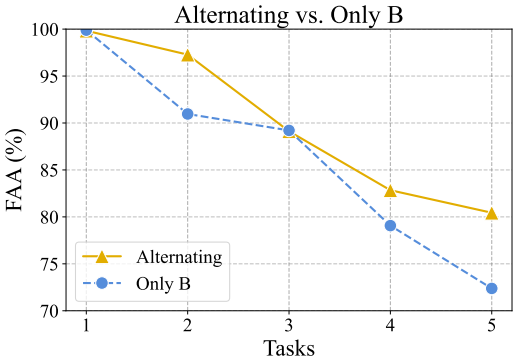

Figure 3: *Alternating* vs. *Only* $\boldsymbol{B}$.

et al., 2024; Yadav et al., 2024; Wu et al., 2024). Alternatively, RegMean (Jin et al., 2023) introduces a closed-form solution for merging model weights, though its application to LoRA module merging has not yet been explored. In this work, we present a novel methodology that leverages this closed-form solution, specifically tailored for the Federated Class-Incremental Learning setting.

Federated Class-Incremental Learning

Federated Class-Incremental Learning (FCIL), introduced by Yoon et al. (2021), addresses the challenges inherent in Continual Learning, such as Catastrophic Forgetting (Robins, 1995), and Federated Learning, such as Client Drift (Zhao et al., 2018; Karimireddy et al., 2020), within a unified framework. Although the FCIL literature is still in its early stages, several studies have adapted popular Continual Learning methods to the federated paradigm, often by employing the FedAvg (McMahan et al., 2017) technique for aggregating client model weights. A few approaches, however, are specifically designed for the FCIL setting. For instance, GLFC (Dong et al., 2022) incorporates local buffers and a regularization technique to mitigate the effects of gradient updates related to novel classes, with further refinements in a subsequent work by the same authors (Dong et al., 2023). TARGET (Zhang et al., 2023b) trains a centralized GAN (Goodfellow et al., 2014) to populate replay buffers with synthetic samples. Fed-CPrompt (Bagwe et al., 2023) combines local and global prompts to address both Continual Learning and Federated Learning challenges simultaneously. Most recently, PILoRA (Guo et al., 2024) proposed a method that integrates LoRA with prototypes (*i.e.*, average feature vectors), using the latter for classification instead of relying on the traditional classification layer. In contrast, our proposed methodology leverages LoRA modules and aggregates them by aligning their output in both Continual and Federated scenarios in a closed-form.

6 Conclusions

At the outset of our research, we explored the possibility of a closed-form solution for merging LoRA modules. Motivated by this question, we identified and validated the existence of such a solution, demonstrating its practicality within the framework of Federated Class-Incremental Learning (FCIL). We introduced LoRM, a method specifically designed for the FCIL scenario, which sets a new State of the Art in this domain.

Looking forward, we aim to broaden our exploration to encompass a wider array of PEFT modules. For modules such as VeRA (Kopiczko et al., 2024) and `(IA)`^{3} (Liu et al., 2022), we have already derived closed-form merging equations, which are detailed in appendices A.2 and A.3, respectively. Furthermore, given the generality of our derivations beyond the FCIL setting, we also intend to evaluate the robustness of our equations across various model merging scenarios. Ultimately, we hope this work will serve not only the Federated Learning and Continual Learning communities but also contribute to the growing body of research on efficient module compositionally.

REFERENCES

Gaurav Bagwe, Xiaoyong Yuan, Miao Pan, and Lan Zhang. Fed-cprompt: Contrastive prompt for rehearsal-free federated continual learning. *arXiv preprint arXiv:2307.04869*, 2023.

Benjamin Bowman, Alessandro Achille, Luca Zancato, Matthew Trager, Pramuditha Perera, Giovanni Paolini, and Stefano Soatto. a-la-carte prompt tuning (apt): Combining distinct data via composable prompting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2023.

Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *Advances in Neural Information Processing Systems*, 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. *International Conference on Learning Representations*, 2022.

Rich Caruana. Multitask learning. *Machine Learning*, 1997.

Jiahua Dong, Lixu Wang, Zhen Fang, Gan Sun, Shichao Xu, Xiao Wang, and Qi Zhu. Federated class-incremental learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2022.

Jiahua Dong, Hongliu Li, Yang Cong, Gan Sun, Yulun Zhang, and Luc Van Gool. No one left behind: Real-world federated class-incremental learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems*, 2014.

Haiyang Guo, Fei Zhu, Wenzhuo Liu, Xu-Yao Zhang, and Cheng-Lin Liu. Pilora: Prototype guided incremental lora for federated class-incremental learning. In *Proceedings of the European Conference on Computer Vision*, 2024.

Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. Introducing eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. In *IEEE International Geoscience and Remote Sensing Symposium*, 2018.

Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, et al. The many faces of robustness: A critical analysis of out-of-distribution generalization. In *IEEE International Conference on Computer Vision*, 2021.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, 2019.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. Lorahub: Efficient cross-task generalization via dynamic lora composition. *arXiv preprint arXiv:2307.13269*, 2023.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *International Conference on Learning Representations*, 2023.

Xisen Jin, Xiang Ren, Daniel Preotiuc-Pietro, and Pengxiang Cheng. Dataless knowledge fusion by merging weights of language models. In *International Conference on Learning Representations*, 2023.

Dahuin Jung, Dongyoon Han, Jihwan Bang, and Hwanjun Song. Generating instance-level prompts for rehearsal-free continual learning. In *IEEE International Conference on Computer Vision*, 2023.

Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*, 2020.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 2017.

Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. In *Proceedings of the European Conference on Computer Vision*, 2020.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki M Asano. Vera: Vector-based random matrix adaptation. In *International Conference on Learning Representations*, 2024.

Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Master's thesis, University of Tront*, 2009.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Conference on Empirical Methods in Natural Language Processing*, 2021.

Qinbin Li, Yiqun Diao, Quan Chen, and Bingsheng He. Federated learning on non-iid data silos: An experimental study. In *IEEE International Conference on Data Engineering*, 2022.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *International Joint Conference on Natural Language Processing*, 2021.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 2022.

Mi Luo, Fei Chen, Dapeng Hu, Yifan Zhang, Jian Liang, and Jiashi Feng. No fear of heterogeneity: Classifier calibration for federated learning with non-iid data. *Advances in Neural Information Processing Systems*, 2021.

Michael S Matena and Colin A Raffel. Merging models with fisher-weighted averaging. *Advances in Neural Information Processing Systems*, 2022.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, 1989.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics*, 2017.

Martin Menabue, Emanuele Frascaroli, Matteo Boschini, Enver Sangineto, Lorenzo Bonicelli, Angelo Porrello, and Simone Calderara. Semantic residual prompts for continual learning. In *Proceedings of the European Conference on Computer Vision*, 2024.

Jaehoon Oh, Sungnyun Kim, Namgyu Ho, Jin-Hwa Kim, Hwanjun Song, and Se-Young Yun. Understanding cross-domain few-shot learning based on domain similarity and few-shot difficulty. *Advances in Neural Information Processing Systems*, 2022.

Guillermo Ortiz-Jimenez, Alessandro Favero, and Pascal Frossard. Task arithmetic in the tangent space: Improved editing of pre-trained models. *Advances in Neural Information Processing Systems*, 2024.

Angelo Porrello, Lorenzo Bonicelli, Pietro Buzzega, Monica Millunzi, Simone Calderara, and Rita Cucchiara. A second-order perspective on compositionality and incremental learning. *arXiv preprint arXiv:2405.16350*, 2024.

Adithya Renduchintala, Tugrul Konuk, and Oleksii Kuchaiev. Tied-lora: Enhancing parameter efficiency of lora with weight tying. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, 2024.

Tal Ridnik, Emanuel Ben-Baruch, Asaf Noy, and Lihi Zelnik-Manor. Imagenet-21k pretraining for the masses. In *Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 1995.

James Seale Smith, Leonid Karlinsky, Vyshnavi Gutta, Paola Cascante-Bonilla, Donghyun Kim, Assaf Arbelle, Rameswar Panda, Rogerio Feris, and Zsolt Kira. Coda-prompt: Continual decomposed attention-based prompting for rehearsal-free continual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2023.

Gido M Van de Ven, Tinne Tuytelaars, and Andreas S Tolias. Three types of incremental learning. *Nature Machine Intelligence*, 2022.

Zifeng Wang, Zizhao Zhang, Sayna Ebrahimi, Ruoxi Sun, Han Zhang, Chen-Yu Lee, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, et al. Dualprompt: Complementary prompting for rehearsal-free continual learning. In *Proceedings of the European Conference on Computer Vision*, 2022a.

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. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2022b.

Xun Wu, Shaohan Huang, and Furu Wei. Mixture of lora experts. In *International Conference on Learning Representations*, 2024.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. Ties-merging: Resolving interference when merging models. *Advances in Neural Information Processing Systems*, 2024.

Enneng Yang, Zhenyi Wang, Li Shen, Shiwei Liu, Guibing Guo, Xingwei Wang, and Dacheng Tao. Adamerging: Adaptive model merging for multi-task learning. In *International Conference on Learning Representations*, 2024.

Jaehong Yoon, Wonyong Jeong, Giwoong Lee, Eunho Yang, and Sung Ju Hwang. Federated continual learning with weighted inter-client transfer. In *International Conference on Machine Learning*, 2021.

Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Nghia Hoang, and Yasaman Khazaeni. Bayesian nonparametric federated learning of neural networks. In *International Conference on Machine Learning*, 2019.

Gengwei Zhang, Liyuan Wang, Guoliang Kang, Ling Chen, and Yunchao Wei. Slca: Slow learner with classifier alignment for continual learning on a pre-trained model. In *IEEE International Conference on Computer Vision*, 2023a.

Jie Zhang, Chen Chen, Weiming Zhuang, and Lingjuan Lyu. Target: Federated class-continual learning via exemplar-free distillation. In *IEEE International Conference on Computer Vision*, 2023b.

Jinghan Zhang, Junteng Liu, Junxian He, et al. Composing parameter-efficient modules with arithmetic operation. *Advances in Neural Information Processing Systems*, 2023c.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations*, 2023d.

Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

SUPPLEMENTARY MATERIAL OVERVIEW

This supplementary document provides detailed mathematical derivations and additional information to complement the main text. The contents are organized as follows:

- *Section A – Mathematical Derivations:* Presents the mathematical foundations and derivations related to the combination of different Parameter-Efficient Fine-Tuning methods with RegMean.
  - *Section A.1 – Combination of LoRA and RegMean:* Derives the optimization problem when integrating Low-Rank Adaptation (LoRA) with RegMean.
    - *Section A.1.1 – Solving for $\boldsymbol{A}$ by Fixing $\boldsymbol{B}$:* Details the solution for matrix $\boldsymbol{A}$ when matrix $\boldsymbol{B}$ is held constant.
    - *Section A.1.2 – Solving for $\boldsymbol{B}$ by Fixing $\boldsymbol{A}$:* Details the solution for matrix $\boldsymbol{B}$ when matrix $\boldsymbol{A}$ is held constant.
  - *Section A.2 – Combination of VeRA and RegMean:* Extends the derivations to the Vector-based Random matrix Adaptation (VeRA) method.
  - *Section A.3 – Combination of $(\texttt{IA})^3$ and RegMean:* Explores how the $(\texttt{IA})^3$ method can be combined with RegMean and provides the corresponding derivations.
  - *Section A.4 – RegMean Applied to Classification Heads:* Discusses the application of RegMean to classification heads in a class-incremental learning setting.
- *Section B – Details on Datasets and Metrics:* Offers comprehensive details about the datasets used and the evaluation metric employed.
- *Section C – Additional Results:* Presents supplementary experimental results and analyses.
  - *Section C.1 – Standard Deviations:* Reports the standard deviations for all evaluated approaches.
  - *Section C.2 – Additional Results on EuroSAT:* Provides extended results on the EuroSAT dataset.
- *Section D – Hyperparameters:* Details the hyperparameters used across different experiments.

A MATHEMATICAL DERIVATIONS

A.1 COMBINATION OF LORA AND REGMEAN

We consider a minimization problem over a single linear layer where $\boldsymbol{X}_i$ represents the input to the layer. The objective function is defined as:

$$\text{minimize } \Omega = \sum_{i=1}^{N} \|\boldsymbol{W}\boldsymbol{X}_i - \boldsymbol{W}_i\boldsymbol{X}_i\|_2^2. \tag{10}$$

Here, $\boldsymbol{W}$ and $\boldsymbol{W}_i$ are the weight matrices for the global model (server) and the local models (clients) respectively. In the context of a LoRA residual module, we replace $\boldsymbol{W}$ with $\boldsymbol{W}_0 + \boldsymbol{B}\boldsymbol{A}$, where $\boldsymbol{W}_0$ denotes the pre-trained weight matrix and $\boldsymbol{B}\boldsymbol{A}$ represents the learned low-rank difference. The objective function becomes:

$$\Omega = \sum_{i=1}^{N} \|(\boldsymbol{W}_0 + \boldsymbol{B}\boldsymbol{A})\boldsymbol{X}_i - (\boldsymbol{W}_0 + \boldsymbol{B}_i\boldsymbol{A}_i)\boldsymbol{X}_i\|_2^2.$$

Expanding the previous expression yields:

$$\Omega = \sum_{i=1}^{N} \|\boldsymbol{W}_{\theta_0}\boldsymbol{X}_i - \boldsymbol{W}_0\boldsymbol{X}_i + \boldsymbol{B}\boldsymbol{A}\boldsymbol{X}_i - \boldsymbol{B}_i\boldsymbol{A}_i\boldsymbol{X}_i\|_2^2.$$

Which simplifies the equation to:

$$\Omega = \sum_{i=1}^{N} \|\boldsymbol{B}\boldsymbol{A}\boldsymbol{X}_i - \boldsymbol{B}_i\boldsymbol{A}_i\boldsymbol{X}_i\|_2^2.$$

It is important to note that $\boldsymbol{B}$ and $\boldsymbol{A}$ refer to the merged LoRA matrices ($\boldsymbol{B}_M$ and $\boldsymbol{A}_M$ in the main paper). For simplicity, we omit the subscript $_M$. To minimize this objective function, we compute the partial derivatives with respect to the unknown matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, setting them equal to zero. This leads to the following system of equations:

$$\begin{cases} \dfrac{\partial \Omega}{\partial \boldsymbol{A}} = 0 \quad \Rightarrow \quad \displaystyle\sum_{i=1}^{N} 2\boldsymbol{B}^\top(\boldsymbol{B}\boldsymbol{A}\boldsymbol{X}_i - \boldsymbol{B}_i\boldsymbol{A}_i\boldsymbol{X}_i)\boldsymbol{X}_i^\top = 0 \\ \dfrac{\partial \Omega}{\partial \boldsymbol{B}} = 0 \quad \Rightarrow \quad \displaystyle\sum_{i=1}^{N} 2(\boldsymbol{B}\boldsymbol{A}\boldsymbol{X}_i - \boldsymbol{B}_i\boldsymbol{A}_i\boldsymbol{X}_i)\boldsymbol{X}_i^\top\boldsymbol{A}^\top = 0 \end{cases}$$

We can rearrange the equations as:

$$\begin{cases} \displaystyle\sum_{i=1}^{N} \boldsymbol{B}^\top\boldsymbol{B}\boldsymbol{A}\boldsymbol{X}_i\boldsymbol{X}_i^\top = \sum_{i=1}^{N} \boldsymbol{B}^\top\boldsymbol{B}_i\boldsymbol{A}_i\boldsymbol{X}_i\boldsymbol{X}_i^\top \\ \displaystyle\sum_{i=1}^{N} \boldsymbol{B}\boldsymbol{A}\boldsymbol{X}_i\boldsymbol{X}_i^\top\boldsymbol{A}^\top = \sum_{i=1}^{N} \boldsymbol{B}_i\boldsymbol{A}_i\boldsymbol{X}_i\boldsymbol{X}_i^\top\boldsymbol{A}^\top \end{cases}$$

Next, we multiply the second equation on the left by $\boldsymbol{B}^\top$, in order to align it with the structure of the first equation:

$$\sum_{i=1}^{N} \boldsymbol{B}^\top\boldsymbol{B}_i\boldsymbol{A}_i\boldsymbol{X}_i\boldsymbol{X}_i^\top\boldsymbol{A}^\top = \sum_{i=1}^{N} \boldsymbol{B}^\top\boldsymbol{B}_i\boldsymbol{A}_i\boldsymbol{X}_i\boldsymbol{X}_i^\top\boldsymbol{A}^\top.$$

By substituting the left-hand side of the first equation into the second, we obtain:

$$\sum_{i=1}^{N} \boldsymbol{B}^\top\boldsymbol{B}_i\boldsymbol{A}_i\boldsymbol{X}_i\boldsymbol{X}_i^\top\boldsymbol{A}^\top - \sum_{i=1}^{N} \boldsymbol{B}^\top\boldsymbol{B}_i\boldsymbol{A}_i\boldsymbol{X}_i\boldsymbol{X}_i^\top\boldsymbol{A}^\top = 0.$$

This simplifies to $0 = 0$, indicating that the system is indeterminate, with infinitely many possible solutions.

A.1.1 SOLVING FOR A BY FIXING B

Assuming the matrix $\boldsymbol{B}$ is fixed, we can solve for $\boldsymbol{A}$. The objective function Ω becomes:

$$\Omega_{\boldsymbol{A}} = \sum_{i=1}^{N} \|\boldsymbol{B}\boldsymbol{A}\boldsymbol{X}_i - \boldsymbol{B}\boldsymbol{A}_i\boldsymbol{X}_i\|_2^2.$$

To minimize $\Omega_{\boldsymbol{A}}$, we set its derivative with respect to $\boldsymbol{A}$ to zero:

$$\frac{\partial \Omega_{\boldsymbol{A}}}{\partial \boldsymbol{A}} = 0 \quad \Rightarrow \quad \sum_{i=1}^{N} 2\boldsymbol{B}^\top(\boldsymbol{B}\boldsymbol{A}\boldsymbol{X}_i - \boldsymbol{B}\boldsymbol{A}_i\boldsymbol{X}_i)\boldsymbol{X}_i^\top = 0,$$

which simplifies to:

$$\sum_{i=1}^{N} \boldsymbol{B}^\top\boldsymbol{B}(\boldsymbol{A}\boldsymbol{X}_i - \boldsymbol{A}_i\boldsymbol{X}_i)\boldsymbol{X}_i^\top = 0.$$

Since $\boldsymbol{B}\boldsymbol{A}$ is designed to be low-rank, the matrix $\boldsymbol{B}$ has more rows than columns. As a result, provided that no row or column of $\boldsymbol{B}$ is zero, the inverse $\left(\boldsymbol{B}^\top\boldsymbol{B}\right)^{-1}$ exists. By multiplying both sides of the equation from the left by $\left(\boldsymbol{B}^\top\boldsymbol{B}\right)^{-1}$, we obtain:

$$\sum_{i=1}^{N} (\boldsymbol{A}\boldsymbol{X}_i - \boldsymbol{A}_i\boldsymbol{X}_i)\boldsymbol{X}_i^\top = 0, \quad \Rightarrow \quad \boldsymbol{A}\sum_{i=1}^{N} \boldsymbol{X}_i\boldsymbol{X}_i^\top = \sum_{i=1}^{N} \boldsymbol{A}_i\boldsymbol{X}_i\boldsymbol{X}_i^\top.$$

Finally, solving for $\boldsymbol{A}$, we get:

$$\boldsymbol{A} = \left(\sum_{i=1}^{N} \boldsymbol{A}_i\boldsymbol{X}_i\boldsymbol{X}_i^\top\right)\left(\sum_{i=1}^{N} \boldsymbol{X}_i\boldsymbol{X}_i^\top\right)^{-1}.$$

A.1.2 SOLVING FOR B BY FIXING A

When the matrix $\boldsymbol{A}$ is fixed, we can solve for $\boldsymbol{B}$. The objective function Ω becomes:

$$\Omega_{\boldsymbol{B}} = \sum_{i=1}^{N} \|\boldsymbol{B}\boldsymbol{A}\boldsymbol{X}_i - \boldsymbol{B}_i\boldsymbol{A}\boldsymbol{X}_i\|_2^2.$$

To minimize $\Omega_{\boldsymbol{B}}$, we set its derivative with respect to $\boldsymbol{B}$ equal to zero:

$$\frac{\partial \Omega_{\boldsymbol{B}}}{\partial \boldsymbol{B}} = 0 \quad \Rightarrow \quad \sum_{i=1}^{N} 2(\boldsymbol{B}\boldsymbol{A}\boldsymbol{X}_i - \boldsymbol{B}_i\boldsymbol{A}\boldsymbol{X}_i)\boldsymbol{X}_i^\top\boldsymbol{A}^\top = 0.$$

Expanding the equation, we get:

$$\sum_{i=1}^{N} \boldsymbol{B}\boldsymbol{A}\boldsymbol{X}_i\boldsymbol{X}_i^\top\boldsymbol{A}^\top - \sum_{i=1}^{N} \boldsymbol{B}_i\boldsymbol{A}\boldsymbol{X}_i\boldsymbol{X}_i^\top\boldsymbol{A}^\top = 0,$$

which simplifies to:

$$\boldsymbol{B}\sum_{i=1}^{N} \boldsymbol{A}\boldsymbol{X}_i\boldsymbol{X}_i^\top\boldsymbol{A}^\top = \sum_{i=1}^{N} \boldsymbol{B}_i\boldsymbol{A}\boldsymbol{X}_i\boldsymbol{X}_i^\top\boldsymbol{A}^\top.$$

Finally, solving for $\boldsymbol{B}$, we obtain:

$$\boldsymbol{B} = \left(\sum_{i=1}^{N} \boldsymbol{B}_i\boldsymbol{A}\boldsymbol{X}_i\boldsymbol{X}_i^\top\right)\boldsymbol{A}^\top\left(\sum_{i=1}^{N} \boldsymbol{A}\boldsymbol{X}_i\boldsymbol{X}_i^\top\boldsymbol{A}^\top\right)^{-1}.$$

A.2 COMBINATION OF VERA AND REGMEAN

Following appendix A.1, we consider the same minimization problem as Equation 10. In the context of a VeRA residual module, we replace $\boldsymbol{W}$ with $\boldsymbol{W}_0 + \Lambda_b \boldsymbol{B} \Lambda_d \boldsymbol{A}$, where $\boldsymbol{W}_0$ denotes the pre-trained weight matrix and $\Lambda_b \boldsymbol{B} \Lambda_d \boldsymbol{A}$ represents the learned weight difference. More precisely, $\boldsymbol{B}$ and $\boldsymbol{A}$ are randomly initialized, while Λ_b and Λ_d are learned. The objective function becomes:

$$\Omega = \sum_{i=1}^{N} \|(\boldsymbol{W}_0 + \Lambda_b \boldsymbol{B} \Lambda_d \boldsymbol{A})\boldsymbol{X}_i - (\boldsymbol{W}_0 + \Lambda_{b,i} \boldsymbol{B} \Lambda_{d,i} \boldsymbol{A})\boldsymbol{X}_i\|_2^2.$$

While every component of LoRA is a full matrix, here we are considering Λ_b and Λ_d, which are diagonal matrices. In order to enforce this constraint into the previous formulation, we can rewrite them as column vectors λ^b and λ^d:

$$\Omega = \sum_{i=1}^{N} \|\left(\boldsymbol{W}_0 + \left(\left(\lambda^b \mathbf{1}_b\right) \odot \boldsymbol{B}\right) \left(\left(\lambda^d \mathbf{1}_d\right) \odot \boldsymbol{A}\right)\right) \boldsymbol{X}_i - (\boldsymbol{W}_0 + \left(\left(\lambda_i^b \mathbf{1}_b\right) \odot \boldsymbol{B}\right) \left(\left(\lambda_i^d \mathbf{1}_d\right) \odot \boldsymbol{A}\right))\boldsymbol{X}_i\|_2^2,$$

where $\mathbf{1}_b$ and $\mathbf{1}_d$ are two row vectors of ones with the same columns as $\boldsymbol{B}$ and $\boldsymbol{A}$, respectively. Please note that in our notation, indices of column vectors are denoted as superscripts, while those of row vectors are represented as subscripts. Expanding this objective, we can write:

$$\Omega = \sum_{i=1}^{N} \|\boldsymbol{W}_0 \boldsymbol{X}_i - \boldsymbol{W}_0 \boldsymbol{X}_i + \left(\left(\lambda^b \mathbf{1}_b\right) \odot \boldsymbol{B}\right) \left(\left(\lambda^d \mathbf{1}_d\right) \odot \boldsymbol{A}\right) \boldsymbol{X}_i - \left(\left(\lambda_i^b \mathbf{1}_d\right) \odot \boldsymbol{B}\right) \left(\left(\lambda_i^d \mathbf{1}_d\right) \odot \boldsymbol{A}\right) \boldsymbol{X}_i\|_2^2.$$

Which simplifies the equation to:

$$\Omega = \sum_{i=1}^{N} \|\left(\left(\lambda^b \mathbf{1}_b\right) \odot \boldsymbol{B}\right) \left(\left(\lambda^d \mathbf{1}_d\right) \odot \boldsymbol{A}\right) \boldsymbol{X}_i - \left(\left(\lambda_i^b \mathbf{1}_d\right) \odot \boldsymbol{B}\right) \left(\left(\lambda_i^d \mathbf{1}_d\right) \odot \boldsymbol{A}\right) \boldsymbol{X}_i\|_2^2.$$

If we compute the partial derivatives with respect to the unknown vectors λ^b and λ^d and set them equal to zero, we obtain an indeterminate system, as similarly discussed in appendix A.1. The detailed derivation is left to the reader. However, it is still possible to solve for either λ^b or λ^d when the other is known.

A.2.1 SOLVING FOR λ^d WITH FIXED λ^b

Assuming the vector λ^b is fixed, we can solve for λ^d. The objective function Ω becomes:

$$\Omega_{\lambda^d} = \sum_{i=1}^{N} \left\|\left(\left(\lambda^b \mathbf{1}_b\right) \odot \boldsymbol{B}\right) \left(\left(\lambda^d \mathbf{1}_d\right) \odot \boldsymbol{A}\right) \boldsymbol{X}_i - \left(\left(\lambda^b \mathbf{1}_b\right) \odot \boldsymbol{B}\right) \left(\left(\lambda_i^d \mathbf{1}_d\right) \odot \boldsymbol{A}\right) \boldsymbol{X}_i\right\|_2^2.$$

This can be rearranged as:

$$\Omega_{\lambda^d} = \sum_{i=1}^{N} \left\|\left(\left(\lambda^b \mathbf{1}_b\right) \odot \boldsymbol{B}\right) \left[\left(\left(\lambda^d \mathbf{1}_d\right) \odot \boldsymbol{A}\right) \boldsymbol{X}_i - \left(\left(\lambda_i^d \mathbf{1}_d\right) \odot \boldsymbol{A}\right) \boldsymbol{X}_i\right]\right\|_2^2.$$

Since $\left(\lambda^b \mathbf{1}_b\right) \odot \boldsymbol{B}$ is a multiplicative constant, it can be factored out and disregarded, leading to the following simplified objective:

$$\Omega_{\lambda^d} = \sum_{i=1}^{N} \left\|\left(\left(\lambda^d \mathbf{1}_d\right) \odot \boldsymbol{A}\right) \boldsymbol{X}_i - \left(\left(\lambda_i^d \mathbf{1}_d\right) \odot \boldsymbol{A}\right) \boldsymbol{X}_i\right\|_2^2.$$

To minimize Ω_{λ^d}, we take its derivative with respect to λ^d and set it to zero:

$$\frac{\partial \Omega_{\lambda^d}}{\partial \lambda^d} = 0 \quad \Rightarrow \quad 2\sum_{i=1}^{N} ((\lambda^d \mathbf{1}_d) \odot \boldsymbol{A})\boldsymbol{X}_i \boldsymbol{X}_i^\top \odot \boldsymbol{A}\mathbf{1}_d^\top - \sum_{i=1}^{N} ((\lambda_i^d \mathbf{1}_d) \odot \boldsymbol{A})\boldsymbol{X}_i \boldsymbol{X}_i^\top \odot \boldsymbol{A}\mathbf{1}_d^\top = 0.$$

This simplifies to:

$$((\lambda^d \mathbf{1}_d) \odot \boldsymbol{A}) \sum_{i=1}^{N} \boldsymbol{X}_i \boldsymbol{X}_i^\top = \sum_{i=1}^{N} ((\lambda_i^d \mathbf{1}_d) \odot \boldsymbol{A})\boldsymbol{X}_i \boldsymbol{X}_i^\top.$$

We then solve for λ^d as follows:

$$
\begin{aligned}
((\lambda^d \mathbf{1}_d) \odot \boldsymbol{A}) &= \left(\sum_{i=1}^{N} ((\lambda_i^d \mathbf{1}_d) \odot \boldsymbol{A}) \boldsymbol{X}_i \boldsymbol{X}_i^\top \right) \left(\sum_{i=1}^{N} \boldsymbol{X}_i \boldsymbol{X}_i^\top \right)^{-1}, \\
\lambda^d \mathbf{1}_d &= \left(\sum_{i=1}^{N} ((\lambda_i^d \mathbf{1}_d) \odot \boldsymbol{A}) \boldsymbol{X}_i \boldsymbol{X}_i^\top \right) \left(\sum_{i=1}^{N} \boldsymbol{X}_i \boldsymbol{X}_i^\top \right)^{-1} \odot \frac{1}{\boldsymbol{A}}, \\
\lambda^d \mathbf{1}_d \mathbf{1}_d^\top &= \left(\sum_{i=1}^{N} ((\lambda_i^d \mathbf{1}_d) \odot \boldsymbol{A}) \boldsymbol{X}_i \boldsymbol{X}_i^\top \right) \left(\sum_{i=1}^{N} \boldsymbol{X}_i \boldsymbol{X}_i^\top \right)^{-1} \odot \frac{1}{\boldsymbol{A}} \mathbf{1}_d^\top.
\end{aligned}
$$

Since $\mathbf{1}_d \mathbf{1}_d^\top = d$ is a scalar, we can finally find:

$$
\lambda^d = \left(\sum_{i=1}^{N} ((\lambda_i^d \mathbf{1}_d) \odot \boldsymbol{A}) \boldsymbol{X}_i \boldsymbol{X}_i^\top \right) \left(\sum_{i=1}^{N} \boldsymbol{X}_i \boldsymbol{X}_i^\top \right)^{-1} \odot \frac{1}{d\boldsymbol{A}} \mathbf{1}_d^\top.
$$

A.2.2 Solving for λ^b with fixed λ^d

When the vector λ^d is fixed, we solve for λ^b. The objective function Ω becomes:

$$
\Omega_{\lambda^b} = \sum_{i=1}^{N} \left\| \left((\lambda^b \mathbf{1}_b) \odot \boldsymbol{B} \right) \left((\lambda^d \mathbf{1}_d) \odot \boldsymbol{A} \right) \boldsymbol{X}_i - \left((\lambda_i^b \mathbf{1}_b) \odot \boldsymbol{B} \right) \left((\lambda^d \mathbf{1}_d) \odot \boldsymbol{A} \right) \boldsymbol{X}_i \right\|_2^2.
$$

To minimize Ω_{λ^b}, we compute its derivative with respect to λ^b and set it to zero:

$$
\begin{aligned}
\frac{\partial \Omega_{\lambda^b}}{\partial \lambda^b} = 0 \quad \Rightarrow \quad & 2 \sum_{i=1}^{N} \Big[((\lambda^b \mathbf{1}_b) \odot \boldsymbol{B})((\lambda^d \mathbf{1}_d) \odot \boldsymbol{A}) \boldsymbol{X}_i \boldsymbol{X}_i^\top - \\
& - ((\lambda_i^b \mathbf{1}_b) \odot \boldsymbol{B})((\lambda^d \mathbf{1}_d) \odot \boldsymbol{A}) \boldsymbol{X}_i \boldsymbol{X}_i^\top \Big] ((\lambda^d \mathbf{1}_d) \odot \boldsymbol{A})^\top \odot \boldsymbol{B} \mathbf{1}_b^\top = 0.
\end{aligned}
$$

This simplifies to:

$$
((\lambda^b \mathbf{1}_b) \odot \boldsymbol{B}) \sum_{i=1}^{N} \mathcal{A} \boldsymbol{X}_i \boldsymbol{X}_i^\top \mathcal{A}^\top = \sum_{i=1}^{N} ((\lambda_i^b \mathbf{1}_b) \odot \boldsymbol{B}) \mathcal{A} \boldsymbol{X}_i \boldsymbol{X}_i^\top \mathcal{A}^\top,
$$

where we use $\mathcal{A}$ to denote $((\lambda^d \mathbf{1}_d) \odot \boldsymbol{A})$ for simplicity. Solving for λ^b, we get:

$$
\begin{aligned}
((\lambda^b \mathbf{1}_b) \odot \boldsymbol{B}) &= \left(\sum_{i=1}^{N} ((\lambda_i^b \mathbf{1}_b) \odot \boldsymbol{B}) \mathcal{A} \boldsymbol{X}_i \boldsymbol{X}_i^\top \mathcal{A}^\top \right) \left(\sum_{i=1}^{N} \mathcal{A} \boldsymbol{X}_i \boldsymbol{X}_i^\top \mathcal{A}^\top \right)^{-1}, \\
\lambda^b \mathbf{1}_b &= \left(\sum_{i=1}^{N} ((\lambda_i^b \mathbf{1}_b) \odot \boldsymbol{B}) \mathcal{A} \boldsymbol{X}_i \boldsymbol{X}_i^\top \mathcal{A}^\top \right) \left(\sum_{i=1}^{N} \mathcal{A} \boldsymbol{X}_i \boldsymbol{X}_i^\top \mathcal{A}^\top \right)^{-1} \odot \frac{1}{\boldsymbol{B}}, \\
\lambda^b \mathbf{1}_b \mathbf{1}_d^\top &= \left(\sum_{i=1}^{N} ((\lambda_i^b \mathbf{1}_b) \odot \boldsymbol{B}) \mathcal{A} \boldsymbol{X}_i \boldsymbol{X}_i^\top \mathcal{A}^\top \right) \left(\sum_{i=1}^{N} \mathcal{A} \boldsymbol{X}_i \boldsymbol{X}_i^\top \mathcal{A}^\top \right)^{-1} \odot \frac{1}{\boldsymbol{B}} \mathbf{1}_d^\top.
\end{aligned}
$$

Finally, since $\mathbf{1}_d \mathbf{1}_d^\top = d$ is a scalar, we can solve for λ^b:

$$
\lambda^b = \left(\sum_{i=1}^{N} ((\lambda_i^b \mathbf{1}_b) \odot \boldsymbol{B}) \mathcal{A} \boldsymbol{X}_i \boldsymbol{X}_i^\top \mathcal{A}^\top \right) \left(\sum_{i=1}^{N} \mathcal{A} \boldsymbol{X}_i \boldsymbol{X}_i^\top \mathcal{A}^\top \right)^{-1} \odot \frac{1}{d\boldsymbol{B}} \mathbf{1}_d^\top,
$$

where $\mathcal{A} = ((\lambda^d \mathbf{1}_d) \odot \boldsymbol{A})$.

A.3 COMBINATION OF (IA)^3 AND REGMEAN

(IA)^3 (Liu et al., 2022) adapts the pretrained model by learning multiplicative weight vectors ℓ^k that modulate the intermediate activations — where k is the dimension of the considered activation. Specifically, for each attention layer, these weight vectors are applied to the input keys, values, and output. Notably, it can be demonstrated that, for a single linear layer, (IA)^3 can be expressed as a residual module (Porrello et al., 2024):

$$\ell^k \odot (\boldsymbol{W}_0 \boldsymbol{X}) = \left[\boldsymbol{W}_0 + ((\ell^k - \mathbf{1}_k^\top)\mathbf{1}_h) \odot \boldsymbol{W}_0\right] \boldsymbol{X},$$

where $\mathbf{1}_o$ and $\mathbf{1}_h$ represent row vectors of ones, respectively corresponding to the dimensions of the output and input of the considered layer. For the sake of simplicity, we deviate from the original formulation, which initializes ℓ^k as a column vector of ones. Instead, we begin with a column vector of zeroes at the beginning of training. This adjustment does not affect the outcome, as we simultaneously modify our formulation as follows:

$$\ell^k \odot (\boldsymbol{W}_0 \boldsymbol{X}) = \left[\boldsymbol{W}_0 + (\ell^k \mathbf{1}_h) \odot \boldsymbol{W}_0\right] \boldsymbol{X},$$

Then, following appendix A.1, we consider the same minimization problem as Equation 10. This time, we replace $\boldsymbol{W}$ with $\boldsymbol{W}_0 + (\ell^k \mathbf{1}_h) \odot \boldsymbol{W}_0$, where $\boldsymbol{W}_0$ denotes the pre-trained weight matrix and $(\ell^k \mathbf{1}_h) \odot \boldsymbol{W}_0$ represents the learned weight difference and ℓ^k is zero initialized. The objective function becomes:

$$\Omega_{\texttt{(IA)}^3} = \sum_{i=1}^{N} \|(\boldsymbol{W}_0 + (\ell^k \mathbf{1}_h) \odot \boldsymbol{W}_0)\boldsymbol{X}_i - (\boldsymbol{W}_0 + (\ell_i^k \mathbf{1}_h) \odot \boldsymbol{W}_0)\boldsymbol{X}_i\|_2^2.$$

Following appendix A.1, we can rearrange $\Omega_{\texttt{(IA)}^3}$ as:

$$\Omega_{\texttt{(IA)}^3} = \sum_{i=1}^{N} \|((\ell^k \mathbf{1}_h) \odot \boldsymbol{W}_0)\boldsymbol{X}_i - ((\ell_i^k \mathbf{1}_h) \odot \boldsymbol{W}_0)\boldsymbol{X}_i\|_2^2.$$

To minimize $\Omega_{\texttt{(IA)}^3}$, we take its derivative with respect to ℓ^k and set it to zero:

$$\frac{\partial \Omega_{\texttt{(IA)}^3}}{\partial \ell^k} = 0 \quad \Rightarrow \quad 2\sum_{i=1}^{N} \left[((\ell^k \mathbf{1}_h) \odot \boldsymbol{W}_0)\boldsymbol{X}_i\boldsymbol{X}_i^\top - ((\ell_i^k \mathbf{1}_h) \odot \boldsymbol{W}_0)\boldsymbol{X}_i\boldsymbol{X}_i^\top\right] \odot \boldsymbol{W}_0 \mathbf{1}_h^\top = 0.$$

This simplifies to:

$$((\ell^k \mathbf{1}_h) \odot \boldsymbol{W}_0) \sum_{i=1}^{N} \boldsymbol{X}_i\boldsymbol{X}_i^\top = \sum_{i=1}^{N} ((\ell_i^k \mathbf{1}_h) \odot \boldsymbol{W}_0)\boldsymbol{X}_i\boldsymbol{X}_i^\top.$$

Solving for ℓ^k, we get:

$$((\ell^k \mathbf{1}_h) \odot \boldsymbol{W}_0) = \left(\sum_{i=1}^{N} ((\ell_i^k \mathbf{1}_h) \odot \boldsymbol{W}_0)\boldsymbol{X}_i\boldsymbol{X}_i^\top\right) \left(\sum_{i=1}^{N} \boldsymbol{X}_i\boldsymbol{X}_i^\top\right)^{-1}.$$

$$\ell^k \mathbf{1}_h = \left(\sum_{i=1}^{N} ((\ell_i^k \mathbf{1}_h) \odot \boldsymbol{W}_0)\boldsymbol{X}_i\boldsymbol{X}_i^\top\right) \left(\sum_{i=1}^{N} \boldsymbol{X}_i\boldsymbol{X}_i^\top\right)^{-1} \odot \frac{1}{\boldsymbol{W}_0}.$$

$$\ell^k \mathbf{1}_h \mathbf{1}_h^\top = \left(\sum_{i=1}^{N} ((\ell_i^k \mathbf{1}_h) \odot \boldsymbol{W}_0)\boldsymbol{X}_i\boldsymbol{X}_i^\top\right) \left(\sum_{i=1}^{N} \boldsymbol{X}_i\boldsymbol{X}_i^\top\right)^{-1} \odot \frac{1}{\boldsymbol{W}_0} \mathbf{1}_h^\top.$$

Finally, since $\mathbf{1}_h \mathbf{1}_h^\top = h$ is a scalar, we can solve for ℓ^k:

$$\ell^k = \left(\sum_{i=1}^{N} ((\ell_i^k \mathbf{1}_h) \odot \boldsymbol{W}_0)\boldsymbol{X}_i\boldsymbol{X}_i^\top\right) \left(\sum_{i=1}^{N} \boldsymbol{X}_i\boldsymbol{X}_i^\top\right)^{-1} \odot \frac{1}{h\boldsymbol{W}_0} \mathbf{1}_h^\top.$$

A.4 RegMean Applied to Classification Heads

While each client adapts the pretrained model to the specific task using residual parameter-efficient modules, the classification head must be re-initialized and trained from scratch. Specifically, following the approach of Caccia et al. (2022), we initialize a distinct classification head for each task. This practice has recently emerged as the *de facto* standard in Class-Incremental Learning, consistently providing a performance boost across various methodologies.

In the context of Class-Incremental Learning, all task-specific classification heads must eventually be merged into a unified classifier. To address this, we aim to minimize the RegMean problem, as defined in Equation 10, across the classifiers of different tasks. Rather than considering a separate set of parameters for each client, we now introduce a weight matrix for each task:

$$\text{minimize } \Omega_{\text{cls}} = \sum_{t=1}^{T} \|\boldsymbol{W}\boldsymbol{X}_t - \boldsymbol{W}_t\boldsymbol{X}_t\|_2^2.$$

Notably, in this case, $\boldsymbol{W}$ and $\boldsymbol{W}_t$ differ in dimensionality, as the classification heads $\{\boldsymbol{W}_1, \boldsymbol{W}_2, \ldots, \boldsymbol{W}_T\}$ are trained independently. Specifically, while $\boldsymbol{W}$ and each $\boldsymbol{W}_t$ share the same number of columns (input dimension), they differ in the number of rows (output dimension). The total number of rows of $\boldsymbol{W}$ corresponds to the sum of the rows of all $\boldsymbol{W}_t$, which is equivalent to the total number of classes $\mathcal{C}$; instead, each $\boldsymbol{W}_t$ has $\mathbb{c}$ rows. This structure enables us to decompose Ω_{cls} into T separate minimization problems:

$$\text{minimize} \sum_{t=1}^{T} \Omega_{\text{cls}}^t, \quad \text{where} \quad \Omega_{\text{cls}}^t = \|\boldsymbol{W}_{\mathbb{c}t:\mathbb{c}(t+1)}\boldsymbol{X}_t - \boldsymbol{W}_t\boldsymbol{X}_t\|_2^2.$$

Since the tasks are class-disjoint, each $\{\Omega_{\text{cls}}^1, \Omega_{\text{cls}}^2, \ldots, \Omega_{\text{cls}}^T\}$ operates on distinct rows of $\boldsymbol{W}$. Consequently, the optimal solution for each Ω_{cls}^t is to assign $\boldsymbol{W}_{\mathbb{c}_t:\mathbb{c}(t+1)} = \boldsymbol{W}_t$ for each t.

B Details on Datasets and Metrics

B.1 CIFAR-100

For the incremental split of CIFAR-100, we follow the standard partitioning into 10 tasks, each consisting of 10 classes, as commonly employed in numerous Continual Learning studies (Wang et al., 2022b;a; Smith et al., 2023). For data augmentation on the training set, we apply bicubic interpolation to resize each image from 32×32 to 224×224, followed by random horizontal flipping and normalization. For the test set, each image is first resized to 256×256 using bicubic interpolation, followed by a center crop to 224×224, and a final normalization step.

B.2 ImageNet-R

For the incremental split of ImageNet-R, we employ the widely used partitioning into 10 tasks, each comprising 20 classes, as is common in various Continual Learning studies (Wang et al., 2022a; Smith et al., 2023; Zhang et al., 2023a). For data augmentation on the training set, we process each 224×224 image with a random horizontal flip, followed by normalization. In the test set, we first resize each image to 256×256 using bicubic interpolation, apply a center crop to 224×224, and then perform normalization.

B.3 EuroSAT

For the incremental split of EuroSAT, we follow a partitioning scheme of 5 tasks, each consisting of 2 classes, consistent with prior Continual Learning studies (Jung et al., 2023; Menabue et al., 2024). For data augmentation in the training set, each 64×64 image is resized to 224×224 using bicubic interpolation. The same interpolation process is applied to the test set.

B.4 Final Average Accuracy (FAA)

We evaluate the performance of all methods using the Final Average Accuracy (FAA), a widely adopted metric in the FCIL literature.

FAA quantifies the mean accuracy across all tasks at the conclusion of the incremental training process. Formally, let a^t represent the accuracy on the tth task after completing the incremental training. The FAA is defined as:

$$\text{FAA} = \frac{1}{T}\sum_{t=1}^{T} a^t, \tag{11}$$

where T represents the total number of tasks.

C Additional Results

C.1 Standard Deviations

The standard deviations for all evaluated approaches, based on three runs, are reported for the in-domain datasets in Table A and for the out-of-domain dataset in Table B.

Table A: *In-domain standard deviations.* Standard deviations for all evaluated approaches.

	CIFAR-100			ImageNet-R		
Joint	±0.66			±1.08		
Partition	$\beta = 0.5$	$\beta = 0.1$	$\beta = 0.05$	$\beta = 0.5$	$\beta = 0.1$	$\beta = 0.05$
EWC (Kirkpatrick et al., 2017)	±0.95	±0.57	±0.83	±0.89	±1.09	±0.83
LwF (Li & Hoiem, 2017)	±0.90	±0.89	±0.53	±0.81	±0.80	±0.98
DER++ (Buzzega et al., 2020)	±0.61	±0.75	±1.41	±0.47	±0.58	±1.38
L2P (Wang et al., 2022b)	±1.24	±1.37	±0.82	±1.10	±0.66	±1.15
CODA-P (Smith et al., 2023)	±0.82	±0.58	±0.63	±0.50	±1.53	±0.91
FisherAvg (Matena & Raffel, 2022)	±1.28	±0.67	±0.58	±1.09	±1.15	±0.03
RegMean (Jin et al., 2023)	±0.28	±0.36	±0.78	±1.41	±0.67	±0.56
CCVR (Luo et al., 2021)	±0.86	±0.49	±0.55	±0.39	±0.66	±0.72
TARGET (Zhang et al., 2023b)	±0.54	±0.60	±0.47	±0.23	±1.33	±0.94
PILoRA (Guo et al., 2024)	±0.86	±0.48	±0.80	±1.57	±1.14	±1.56
LoRM (ours)	±0.73	±0.62	±0.59	±1.36	±0.70	±1.46

C.2 Additional Results on EuroSAT

In Table C

D Hyperparameters

All method-specific parameters not explicitly mentioned follow the settings described in their original papers.

D.1 Acronyms

List of all the acronyms used:

- lr: learning rate,
- lr_{pr}: learning rate for prototypes,
- λ_{KL}: Knowledge Distillation Loss multiplier,
- r: rank for low rank matrices,
- n_p: number of prompts,
- g_{ep}: epochs of training and generation for the generator network,
- γ: decay factor for off-diagonal elements of the Gram matrices.

Table B: *Out of domain standard deviations.* Standard deviations for all evaluated approaches.

EuroSAT			
Joint		±0.66	
Partition	$\beta = 1.0$	$\beta = 0.5$	$\beta = 0.2$
EWC	±1.17	±0.74	±0.83
LwF	±0.67	±0.33	±0.87
DER++	±0.94	±0.82	±0.85
L2P	±1.69	±1.18	±1.36
CODA-P	±0.57	±1.28	±0.73
FisherAvg	±1.38	±0.77	±0.62
RegMean	±0.17	±0.78	±0.98
CCVR	±0.92	±0.87	±1.56
TARGET	±0.73	±0.82	±0.86
PILoRA	±1.01	±0.87	±0.45
LoRM (ours)	±1.04	±0.73	±0.81

Table C: *Additional EuroSAT comparison.* Results in terms of FAA(%).

Joint		93.72	
Partition	$\beta = 1.0$	$\beta = 0.5$	$\beta = 0.2$
EWC	55.88	46.23	44.16
LwF	23.91	21.31	21.64
CODA-P	62.96	59.20	62.23
FisherAvg	48.83	43.19	42.48
RegMean	27.12	30.80	32.31

D.2 CIFAR-100

D.2.1 Distribution β = 0.5

Joint:	*lr*: 1e-3;
EWC:	*lr*: 3e-06;
LwF:	*lr*: 3e-06;
DER++:	*lr*: 3e-06; α:34; β:13
L2P:	*lr*: 3e-02;
CODA-P:	*lr*: 1e-03;
FisherAvg:	*lr*: 3e-06;
RegMean:	*lr*: 3e-06; γ: 0.5
CCVR:	*lr*: 3e-06;
TARGET:	*lr*: 3e-04; λ_{KL}: 25; g_{ep}: 30
PILoRA:	*lr*: 2e-02; lr_{pr}: 1e-4;
LoRM:	*lr*: 1e-03; *r*: 16;

D.2.2 DISTRIBUTION β = 0.1

Joint:	*lr*: 1e-03;
EWC:	*lr*: 3e-06;
LwF:	*lr*: 3e-06;
DER++:	*lr*: 1e-4; α: 0.3; β: 0.8
L2P:	*lr*: 3e-02;
CODA-P:	*lr*: 1e-03;
FisherAvg:	*lr*: 3e-06;
RegMean:	*lr*: 3e-06; γ: 0.5
CCVR:	*lr*: 3e-06;
TARGET:	*lr*: 3e-04; λ_{KL}: 25; g_{ep}: 30
PILoRA:	*lr*: 2e-02; lr_{pr}: 1e-4;
LoRM:	*lr*: 3e-04; *r*: 16;

D.2.3 DISTRIBUTION β = 0.05

Joint:	*lr*: 1e-03;
EWC:	*lr*: 3e-06;
LwF:	*lr*: 1e-05;
DER++:	*lr*: 3e-06; α: 0.3; β: 0.8
L2P:	*lr*: 3e-02;
CODA-P:	*lr*: 1e-03; n_p: 100
FisherAvg:	*lr*: 3e-06;
RegMean:	*lr*: 3e-06; γ: 0.5
CCVR:	*lr*: 3e-06;
TARGET:	*lr*: 3e-04; λ_{KL}: 25; g_{ep}: 30
PILoRA:	*lr*: 2e-02; lr_{pr}: 1e-4;
LoRM:	*lr*: 3e-04; *r*: 32;

D.3 IMAGENET-R

D.3.1 DISTRIBUTION β = 0.5

Joint:	*lr*: 1e-03;
EWC:	*lr*: 1e-05;
LwF:	*lr*: 1e-05;
DER++:	*lr*: 1e-05; α: 0.3; β: 0.8
L2P:	*lr*: 3e-04;
CODA-P:	*lr*: 3e-04; n_p: 200
FisherAvg:	*lr*: 1e-05;
RegMean:	*lr*: 1e-05; γ: 0.5
CCVR:	*lr*: 3e-06;
TARGET:	*lr*: 3e-04; λ_{KL}: 25; g_{ep}: 30
PILoRA:	*lr*: 2e-02; lr_{pr}: 1e-4;
LoRM:	*lr*: 1e-03; *r*: 16

D.3.2 DISTRIBUTION β = 0.1

Joint:	*lr*: 1e-03;
EWC:	*lr*: 1e-05;
LwF:	*lr*: 1e-05;
DER++:	*lr*: 1e-05; α: 0.3; β: 0.8
L2P:	*lr*: 3e-04;
CODA-P:	*lr*: 3e-04; n_p: 200
FisherAvg:	*lr*: 1e-05;
RegMean:	*lr*: 1e-05; γ: 0.5
CCVR:	*lr*: 1e-04;
TARGET:	*lr*: 3e-04; λ_{KL}: 25; g_{ep}: 30
PILoRA:	*lr*: 2e-02; lr_{pr}: 1e-4;
LoRM:	*lr*: 1e-03; *r*: 32

D.3.3 DISTRIBUTION β = 0.05

Joint:	*lr*: 1e-03;
EWC:	*lr*: 1e-05;
LwF:	*lr*: 1e-05;
DER++:	*lr*: 1e-05; α: 0.3; β: 0.8
L2P:	*lr*: 3e-04;
CODA-P:	*lr*: 3e-04; n_p: 200
FisherAvg:	*lr*: 1e-05;
RegMean:	*lr*: 3e-06; γ: 0.5
CCVR:	*lr*: 1e-05;
TARGET:	*lr*: 3e-04; λ_{KL}: 25; g_{ep}: 30
PILoRA:	*lr*: 2e-02; lr_{pr}: 1e-4;
LoRM:	*lr*: 1e-03; *r*: 16

D.4 EUROSAT

D.4.1 DISTRIBUTION β = 1.0

Joint:	*lr*: 1e-03;
EWC:	*lr*: 1e-05;
LwF:	*lr*: 1e-05;
DER++:	*lr*: 1e-04; α: 0.3; β: 0.8
L2P:	*lr*: 3e-04;
CODA-P:	*lr*: 3e-04; n_p: 10
FisherAvg:	*lr*: 1e-05;
RegMean:	*lr*: 3e-06; γ: 0.5
CCVR:	*lr*: 1e-05;
TARGET:	*lr*: 3e-04; λ_{KL}: 25; g_{ep}: 30
PILoRA:	*lr*: 2e-02; lr_{pr}: 1e-4;
LoRM:	*lr*: 1e-03; *r*: 16

D.4.2 DISTRIBUTION β = 0.5

Joint:	*lr*: 1e-03;
EWC:	*lr*: 1e-05;
LwF:	*lr*: 1e-05;
DER++:	*lr*: 3e-06; α: 0.3; β: 0.8
L2P:	*lr*: 3e-04;
CODA-P:	*lr*: 3e-04; n_p: 10
FisherAvg:	*lr*: 1e-05;
RegMean:	*lr*: 1e-05; γ: 0.5
CCVR:	*lr*: 1e-05;
TARGET:	*lr*: 3e-04; λ_{KL}: 25; g_{ep}: 30
PILoRA:	*lr*: 2e-02; lr_{pr}: 1e-4;
LoRM:	*lr*: 1e-03; *r*: 16

D.4.3 DISTRIBUTION β = 0.2

Joint:	*lr*: 1e-03;
EWC:	*lr*: 1e-05;
LwF:	*lr*: 1e-05;
DER++:	*lr*: 1e-05; α: 0.3; β: 0.8
L2P:	*lr*: 3e-04;
CODA-P:	*lr*: 3e-04; n_p: 10
FisherAvg:	*lr*: 1e-05;
RegMean:	*lr*: 1e-05; γ: 0.5
CCVR:	*lr*: 1e-05;
TARGET:	*lr*: 3e-04; λ_{KL}:25; g_{ep}: 30
PILoRA:	*lr*: 2e-02; lr_{pr}: 1e-4;
LoRM:	*lr*: 1e-03; *r*: 16