535510 RL Team Project: MMCR: Sequential Layer-wise Model Merging via Reinforcement Learning

張周芳　賴子永　林致葆
Chou Fang, Chang　Zi-Yong, Lai　Chih-Pao, Lin
Institute of Electrical and Computer Engineering
National Yang Ming Chiao Tung University
{fangfangirl037.ee14, ee314511062.ee14, elimath9104.ee14}@nycu.edu.tw

Project Page: https://fangfangirl.github.io/RL_final/

1 Project Overview

1.1 Profile

Track: 1. Algorithm, 3. Application

Abstract and project goal: Model merging aims to combine multiple task-specific models into a single unified model without retraining. Existing methods such as Task Arithmetic, TIES, and AdaMerging usually merge models through fixed rules or coefficient optimization, but they do not explicitly model the layer-wise merging process as a sequential decision problem. In this project, we study MMCR, a reinforcement-learning-based framework for sequential, layer-wise task-vector model merging. At each step, the policy observes the current layer progress, task-vector geometry, and previous coefficient history, then outputs a coefficient vector for merging the current layer. Our goal is to investigate whether an RL policy can learn adaptive layer-wise merging strategies that better preserve the abilities of multiple source models.

TL;DR: RL-based sequential layer-wise task-vector model merging with adaptive coefficient learning.

1.2 Motivation

Why is the problem interesting?

Model merging is an important technique for integrating multiple task-specific models without retraining, and is widely used in multi-task learning, continual learning, and efficient model deployment. Given several models fine-tuned from the same pretrained backbone, the goal is to construct a single merged model that preserves the abilities of all source models. This is attractive because it reduces storage and inference cost while avoiding additional full-model retraining.

However, model merging is challenging because different task-specific models may encode knowledge in different or even conflicting parameter directions. A single global merging coefficient may be too coarse to handle these conflicts, since different layers can have different levels of task interference. Therefore, an adaptive layer-wise merging strategy is needed. This motivates us to formulate model merging as a sequential decision-making problem, where an RL agent decides how to combine source task vectors layer by layer.

Critical Challenges

- **Challenge 1: Layer-wise task-vector conflict.**
  Source models are fine-tuned on different tasks, so their task vectors may have different magnitudes and directions at each layer. Some layers may contain compatible updates, while other layers may contain conflicting updates. Fixed merging rules may fail to adapt to such layer-dependent differences.
- **Challenge 2: Sequential dependency across layer-wise decisions.**
  Layer-wise merging decisions are not independent. The coefficient choice made in an early layer changes the partially merged model and may influence how later layers should be merged. Therefore, one-shot merging or independent layer-wise estimation may not fully capture the trajectory-level effect of merging decisions.
- **Challenge 3: Reward evaluation and credit assignment.**
  The quality of a merging decision is usually reflected only after evaluating the merged model on multiple tasks. Evaluating the model after every layer can be computationally expensive, while evaluating only at the end makes it harder to assign credit to each layer-wise decision. This creates a trade-off between reward informativeness and computational efficiency.

Justify why the problem remains open or unsolved.

Existing model merging methods can be broadly categorized into the following groups, and each category has its own limitations:

- **Optimization-based coefficient learning methods:** Methods such as AdaMerging [Yang et al., 2023] learn merging coefficients by optimizing an objective on validation data. These methods can achieve strong performance because the coefficients are adapted to the target tasks. However, they usually optimize coefficients in a non-sequential manner and do not explicitly formulate the layer-wise merging process as a trajectory of decisions.
- **Heuristic task-vector merging methods:** Approaches such as Task Arithmetic [Ilharco et al., 2022], TIES [Yadav et al., 2023], and DARE [Yu et al., 2024] merge models using predefined rules over task vectors. These methods are simple and efficient, but their merging strategies are mostly fixed. As a result, they may not adapt well to different task combinations or layer-specific task-vector conflicts.
- **Training-free coefficient estimation methods:** Methods such as NAN [Si et al., 2025] estimate merging coefficients without iterative policy learning. Although such methods are efficient, they are typically based on one-shot estimation and do not explicitly model the sequential dependency among layer-wise merging decisions.

Overall, existing methods still leave a gap between adaptive coefficient optimization and sequential layer-wise decision making. Optimization-based methods can learn useful coefficients, while heuristic and training-free methods are efficient, but most of them do not explicitly model model merging as a trajectory where earlier layer-wise decisions influence later ones. This motivates us to explore reinforcement learning as a mechanism for adaptive sequential model merging.

State-of-the-art methods.

AdaMerging [Yang et al., 2023], TIES [Yadav et al., 2023], DARE [Yu et al., 2024], and NAN [Si et al., 2025] are considered representative baselines in model merging research.

2 Problem Formulation

We study the problem of multi-task model merging, where the goal is to combine several task-specific models into a single unified model without retraining the full network. Given multiple models fine-tuned from the same pretrained backbone, we aim to construct a merged model that preserves the abilities of all source models.

Rather than treating model merging as a one-shot static operation, we formulate task-vector model merging as a layer-wise sequential decision-making problem. At each step, an

RL agent decides how much each source task vector should contribute to the current layer. The environment then updates the current layer of the merged model, evaluates the merged model when reward evaluation is triggered, and moves to the next layer. This formulation allows the agent to learn adaptive layer-wise merging coefficients instead of using a single global coefficient for the whole model.

2.1 Multi-Model Task-Vector Merging

Given K source models fine-tuned on different tasks or datasets:

$$\{M^{(1)}, M^{(2)}, \ldots, M^{(K)}\},$$

and a shared pretrained model $M^{(0)}$, all models are assumed to have compatible architectures with L mergeable layers.

We denote:

- $k \in \{1, \ldots, K\}$: index of the source models;
- $l \in \{1, \ldots, L\}$: index of the mergeable layers;
- $\theta_l^{(0)}$: parameters of the l-th layer of the pretrained model;
- $\theta_l^{(k)}$: parameters of the l-th layer of the k-th fine-tuned source model;
- $\Delta_l^{(k)}$: task vector of source model k at layer l;
- $\theta_l^{(m,t)}$: parameters of the l-th layer of the merged model after step t;
- Θ_t: the whole merged model after the first t layer-wise decisions.

The task vector of source model k at layer l is defined with respect to the pretrained model:

$$\Delta_l^{(k)} = \theta_l^{(k)} - \theta_l^{(0)}.$$

The merged parameter of layer l is constructed by adding a weighted combination of source task vectors to the pretrained parameter:

$$\theta_l^{(m)} = \theta_l^{(0)} + \sum_{k=1}^{K} \lambda_{l,k} \Delta_l^{(k)},$$

where $\lambda_{l,k}$ is the merging coefficient for source model k at layer l.

Before the first layer-wise decision, the coefficient table is initialized by a fixed value λ_{init}. This gives the initial merged model Θ_0. During an episode, the policy gradually overwrites the initialized coefficients layer by layer.

This formulation preserves the pretrained model as the common reference point and lets the policy decide how much task-specific update should be added to each layer. Compared with using a single global coefficient for the whole model, layer-wise task-vector merging allows different layers to use different combinations of source models.

2.2 Sequential Layer-wise Merging

We formulate the merging process as a sequential decision process with horizon L, where each episode corresponds to constructing one complete merged model. At step $t \in \{1, \ldots, L\}$, the environment focuses on the t-th mergeable layer.

The policy observes the current state s_t and samples a raw action from a Gaussian distribution. This raw action is then transformed into a valid coefficient vector:

$$a_t = \lambda_t = [\lambda_{t,1}, \lambda_{t,2}, \ldots, \lambda_{t,K}] \in \mathbb{R}^K.$$

Each coefficient $\lambda_{t,k}$ controls how much the task vector of source model k contributes to the current layer. The environment then updates only the current layer:

$$\theta_t^{(m,t)} = \theta_t^{(0)} + \sum_{k=1}^{K} \lambda_{t,k} \Delta_t^{(k)}.$$

All other layers remain unchanged during this step:

$$\theta_l^{(m,t)} = \theta_l^{(m,t-1)}, \quad l \neq t.$$

Thus, the whole merged model evolves from Θ_{t-1} to Θ_t after applying action a_t. After updating layer t, the environment records the coefficient vector λ_t, advances to layer $t+1$, and constructs the next state s_{t+1}.

The full trajectory is:

$$\tau = (s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_L, a_L, r_L),$$

where $a_t = \lambda_t$ is the coefficient action for layer t, and r_t is the reward returned after applying this layer-wise decision.

2.3 MDP Formulation

We formulate the layer-wise merging process as a Markov Decision Process:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma),$$

where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, P is the transition function, r is the reward function, and γ is the discount factor.

2.3.1 State

The state s_t is observed before updating layer t. It provides the policy with information about the current merging progress, the geometry of the current layer's task vectors, and the coefficient history of the current trajectory. In our final setting, we use a full-coefficient state representation:

$$s_t = [p_t,\ g_t,\ \mathrm{vec}(\Lambda_t),\ m_t].$$

Here, p_t is the layer progress:

$$p_t = \frac{t}{L}.$$

It tells the policy which part of the layer-wise merging trajectory it is currently in.

The term g_t describes the task-vector geometry of the current layer. For each source model, we extract two features from its task vector at layer t: the relative update magnitude and the update direction similarity. Thus, the geometry feature is:

$$g_t = [n_{t,1}, \ldots, n_{t,K}, d_{t,1}, \ldots, d_{t,K}].$$

The first part, $n_{t,k}$, measures how large the task vector of source model k is at the current layer compared with the average task-vector magnitude at the same layer:

$$n_{t,k} = \frac{\|\Delta_t^{(k)}\|}{\frac{1}{K}\sum_{j=1}^{K} \|\Delta_t^{(j)}\|}.$$

If $n_{t,k} > 1$, source model k changes this layer more than the average source model. If $n_{t,k} < 1$, its update at this layer is smaller than average. This feature tells the policy how strong each source model's parameter update is at the current layer.

The second part, $d_{t,k}$, measures whether the update direction of source model k is consistent with the average update direction of all source models. Let the average task vector at layer t be:

$$\bar{\Delta}_t = \frac{1}{K}\sum_{j=1}^{K} \Delta_t^{(j)}.$$

Then the direction feature is defined as:

$$d_{t,k} = \frac{\langle \Delta_t^{(k)}, \bar{\Delta}_t \rangle}{\|\Delta_t^{(k)}\| \|\bar{\Delta}_t\|}.$$

This is the cosine similarity between the task vector of source model k and the average task vector. A value close to 1 means that the source model's update direction is similar to the common update direction. A smaller value means that this source model may be less consistent with the others at this layer.

In summary, g_t tells the policy two things about the current layer: which source models have larger parameter updates, and which source models have update directions that are more consistent with the others.

The matrix

$$\Lambda_t \in \mathbb{R}^{L\times K}$$

stores the coefficient vector for every layer in the current trajectory before applying action a_t. If a layer has already been visited, its row stores the coefficient vector chosen by the policy. If a layer has not yet been visited, its row remains at the initial coefficient value.

The filled mask

$$m_t \in \{0,1\}^L$$

indicates which layers have already been updated:

$$m_{t,l} = \begin{cases} 1, & l < t, \\ 0, & l \geq t. \end{cases}$$

This mask helps the policy distinguish between coefficients that were actually selected by the policy and coefficients that are still only initialization values.

Therefore, for K source models and L mergeable layers, the state dimension is:

$$1 + 2K + LK + L.$$

2.3.2 Action

The policy is a stochastic Gaussian policy. Given the current state s_t, the policy outputs the mean and standard deviation of a Gaussian distribution:

$$\mu_t, \sigma_t = \pi_\phi(s_t),$$

where $\mu_t, \sigma_t \in \mathbb{R}^K$ and K is the number of source models.

A raw action is sampled from this distribution:

$$u_t \sim \mathcal{N}(\mu_t, \mathrm{diag}(\sigma_t^2)).$$

The raw action is then transformed into a valid coefficient vector:

$$a_t = \lambda_t = T_{\mathrm{pos}}(u_t),$$

where $T_{\mathrm{pos}}(\cdot)$ denotes the positive coefficient transformation. This ensures that the final merging coefficients are non-negative.

Each element $\lambda_{t,k}$ determines how much the task vector of source model k contributes to the current layer t:

$$\theta_t^{(m,t)} = \theta_t^{(0)} + \sum_{k=1}^{K} \lambda_{t,k} \Delta_t^{(k)}.$$

2.3.3 Transition

The transition is deterministic. Given the current state s_t and action $a_t = \lambda_t$, the environment updates only the current layer:

$$\theta_t^{(m,t)} = \theta_t^{(0)} + \sum_{k=1}^{K} \lambda_{t,k} \Delta_t^{(k)}.$$

All other layers remain unchanged:

$$\theta_l^{(m,t)} = \theta_l^{(m,t-1)}, \quad l \neq t.$$

The coefficient table is updated by writing the current coefficient vector into the row corresponding to layer t:

$$(\Lambda_{t+1})_{t,:} \leftarrow \lambda_t,$$

while all other rows remain unchanged:

$$(\Lambda_{t+1})_{l,:} = (\Lambda_t)_{l,:}, \quad l \neq t.$$

The filled mask is also updated so that the current layer is marked as visited:

$$m_{t+1,l} = \begin{cases} 1, & l \leq t, \\ 0, & l > t. \end{cases}$$

The layer index advances from t to $t+1$. The next state s_{t+1} is then constructed from the next layer's geometry, the updated coefficient table Λ_{t+1}, and the updated mask m_{t+1}:

$$s_{t+1} = P(s_t, a_t).$$

Therefore, the state-action transition can be summarized as:

$$s_t \to u_t \to \lambda_t \to \Theta_t \to r_t \to s_{t+1}.$$

The policy first samples a raw action u_t, the environment transforms it into the coefficient vector λ_t, the coefficient vector updates the merged model, the reward is computed from the updated model when evaluation is triggered, and the next state records the updated coefficient history.

2.3.4 Reward Function

We use an entropy-based reward to evaluate the current merged model during the layer-wise merging process. The reward is computed after applying the action, so it evaluates the updated merged model Θ_t.

Let B_k denote the reward batch for task k, and let h_k be the task-specific head for that task. Given the current merged encoder f_{Θ_t}, the logits for an input $x \in B_k$ are:

$$z_{t,k}(x) = h_k(f_{\Theta_t}(x)).$$

The task score is defined as the negative prediction entropy:

$$q_{t,k} = -\frac{1}{|B_k|} \sum_{x \in B_k} H\left(\text{softmax}\left(z_{t,k}(x)\right)\right).$$

A higher $q_{t,k}$ indicates lower entropy and therefore more confident predictions on task k. To aggregate the scores across all tasks, we define the objective:

$$J_t = \frac{1}{K} \sum_{k=1}^{K} q_{t,k} - \alpha \cdot \text{Std}(q_{t,1}, \ldots, q_{t,K}),$$

where α penalizes imbalance across tasks.

Reward evaluation is performed at selected intermediate steps and at the terminal step. Let $e_t \in \{0, 1\}$ indicate whether reward evaluation is performed at step t:

$$e_t = \begin{cases} 1, & t \in \mathcal{E} \text{ or } t = L, \\ 0, & \text{otherwise}, \end{cases}$$

where $\mathcal{E}$ is the set of intermediate reward evaluation steps.

If $e_t = 1$, the intermediate reward is defined as the improvement of the objective:

$$r_t^{\text{step}} = \eta_{\text{step}}(J_t - J_{\text{prev}}),$$

where J_{prev} is the most recent previously evaluated objective. After the reward is computed, we update $J_{\text{prev}} \leftarrow J_t$. If $e_t = 0$, then:

$$r_t^{\text{step}} = 0.$$

At the final layer, we additionally use a terminal reward:

$$r_L^{\text{term}} = \eta_{\text{term}} J_L.$$

Thus, the reward at step t is:

$$r_t = r_t^{\text{step}} + \mathbb{I}[t = L] r_L^{\text{term}}.$$

This reward design provides both intermediate feedback and a final objective. The intermediate reward encourages the policy to make layer-wise decisions that improve the current merged model, while the terminal reward encourages the final merged model to have confident predictions across all tasks.

2.4 Learning Objective

We learn a stochastic Gaussian policy $\pi_\phi(u_t \mid s_t)$. At each step, the policy samples a raw action u_t, which is transformed into the coefficient action $a_t = \lambda_t$ before being applied to the current layer. The objective is to maximize the expected cumulative return over the full merging trajectory:

$$\max_\phi \mathbb{E}_{\tau \sim \pi_\phi} \left[\sum_{t=1}^{L} \gamma^{t-1} r_t \right].$$

The trajectory is:

$$\tau = (s_1, u_1, \lambda_1, r_1, \ldots, s_L, u_L, \lambda_L, r_L).$$

Here, u_t is the sampled raw action used for policy optimization, while λ_t is the transformed coefficient vector applied to the current layer. In this formulation, model merging becomes a trajectory-level optimization problem. The policy learns how to assign source task-vector coefficients across layers so that the final merged model obtains a higher entropy-based objective than fixed one-shot merging rules.

3 Empirical Evaluation

3.1 Performance Metrics

We plan to evaluate the proposed method based on the following metrics:

- **Task performance:** For vision tasks, we report classification accuracy on CIFAR-100 and ImageNet-style benchmarks. For language models, we evaluate generation quality using standard instruction-following benchmarks (e.g., accuracy or task-specific scores). This metric reflects the overall effectiveness of the merged model.
- **Efficiency:** We measure the computational cost of the merging process, including the number of optimization steps and wall clock time. This metric reflects the practicality of the proposed method compared to existing approaches.

3.2 Baseline Methods

We compare our method with the following representative model merging approaches:

- **Task Arithmetic [Ilharco et al., 2022]:** A simple data-free method that merges models by linearly combining task vectors derived from fine-tuned models.
- **TIES [Yadav et al., 2023]:** A data-free merging method that resolves parameter conflicts using sign agreement and sparsification techniques.

- **DARE [Yu et al., 2024]:** A recent method that improves model merging by reweighting parameters and removing redundant updates.
- **AdaMerging [Yang et al., 2023]:** A data-based approach that learns optimal merging coefficients using validation data.
- **NAN [Si et al., 2025]:** A training-free method that estimates merging coefficients in a one-shot manner without iterative optimization.

These methods represent both data-free and data-based approaches and are considered strong baselines in the model merging literature.

3.3 Benchmark Tasks or Datasets

We evaluate our method on model merging tasks within individual domains, focusing on merging multiple models trained on different tasks.

- **Vision benchmarks:** We follow standard vision model merging settings using Vision Transformers (ViT) [Dosovitskiy et al., 2020]. We merge multiple models (e.g., 2 to 8 models) fine-tuned on different classification tasks, including SUN397 [Xiao et al., 2010], Stanford Cars [Krause et al., 2013], RESISC45 [Cheng et al., 2017], EuroSAT [Helber et al., 2019], SVHN [Netzer et al., 2011], GTSRB [Stallkamp et al., 2011], MNIST [LeCun, 1998], and DTD [Cimpoi et al., 2014]. The merged model is evaluated on all tasks using classification accuracy.
- **Merging settings:** We focus on multi-model merging within the same domain, where multiple task-specific models are combined into a single unified model. We vary the number of source models (e.g., merging 2, 4, or 8 models) to evaluate scalability.

4 Method

4.1 Overview

The MDP formulation in Section 2 defines model merging as a sequential layer-wise decision problem. This section describes the proposed optimization framework for learning the layer-wise merging policy. Figure 1 presents the overall pipeline of MMCR, including task-vector construction, TIES preprocessing, environment setup, sequential layer-wise merging, policy optimization, and final evaluation.

Our method consists of four components. First, we construct task vectors from fine-tuned source models with respect to a shared pretrained model. Second, we apply TIES-based preprocessing to reduce parameter-level conflicts among task vectors. Third, we build a computationally efficient reward estimation scheme by sampling a small reward subset from each task dataset. Finally, we optimize a stochastic Gaussian policy using a GRPO-style objective with a group leave-one-out advantage estimator.

During policy training, each episode constructs one complete merged model by visiting mergeable layers sequentially. At each layer, the policy samples a coefficient action, the environment updates the corresponding layer of the merged model, and the current model is evaluated at selected reward steps using the task-specific reward subsets. This allows the agent to receive intermediate feedback without requiring full-dataset evaluation after every layer-wise decision.

4.2 TIES Preprocessing for Task Vectors

Before policy optimization, we apply TIES preprocessing to the task vectors in order to reduce parameter-level interference among source models. Although the general formulation in Section 2 defines merging over task vectors $\Delta^{(k)}$, our final method applies the RL policy to preprocessed task vectors.

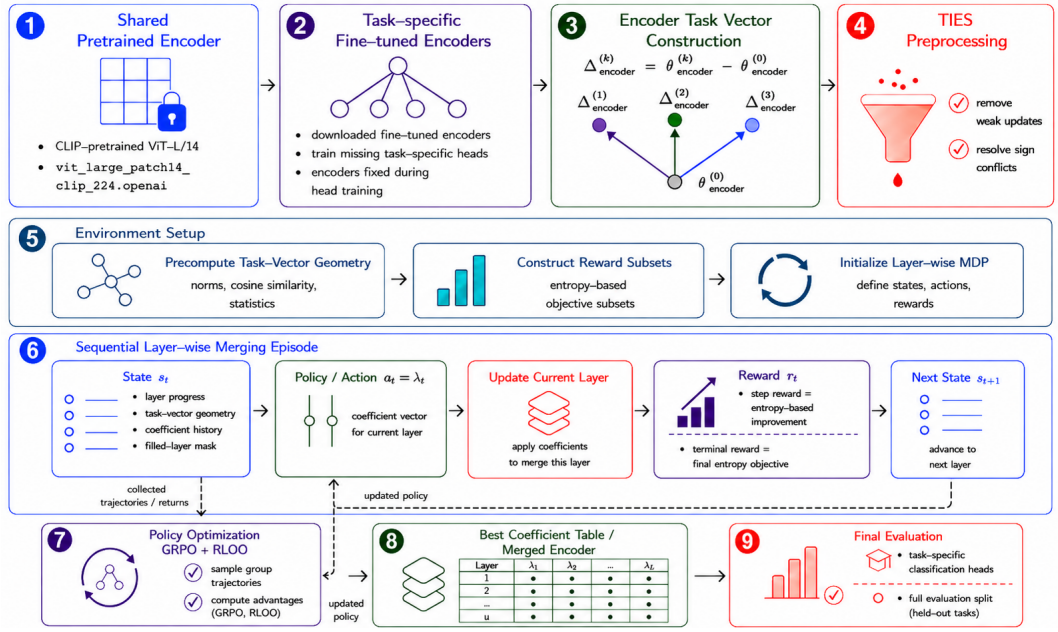

Figure 1: Overview of the proposed MMCR framework. Starting from a shared pretrained encoder and multiple task-specific fine-tuned encoders, we first construct encoder task vectors and apply TIES preprocessing. We then build the RL environment by precomputing task-vector geometry, constructing reward subsets, and initializing a layer-wise MDP. During each merging episode, the policy sequentially selects layer-wise coefficients, the environment updates the current layer, computes entropy-based rewards, and transitions to the next state. Finally, GRPO with RLOO is used to optimize the policy, and the learned coefficient table is used to construct the final merged encoder for evaluation.

Given a source model $M^{(k)}$ and the shared pretrained model $M^{(0)}$, the original task vector is:

$$\Delta^{(k)} = \theta^{(k)} - \theta^{(0)}.$$

TIES preprocessing first removes small-magnitude updates from each task vector. These small updates may contain noise or weak task-specific information, and removing them reduces unnecessary interference during merging. Then, for each parameter position, we determine the dominant update direction across source task vectors and retain only updates whose signs are consistent with this direction. Updates with conflicting signs are discarded.

After preprocessing, we obtain cleaned task vectors:

$$\{\tilde{\Delta}^{(1)}, \tilde{\Delta}^{(2)}, \ldots, \tilde{\Delta}^{(K)}\}.$$

The layer-wise merging operation used in our final method is therefore:

$$\theta_l^{(m)} = \theta_l^{(0)} + \sum_{k=1}^{K} \lambda_{l,k} \tilde{\Delta}_l^{(k)}.$$

This preprocessing step reduces destructive interference before reinforcement learning. As a result, the policy learns layer-wise coefficients over a more stable set of task vectors instead of directly operating on all raw task-vector updates.

4.3 Reward Subset Construction

A major computational challenge in reinforcement-learning-based model merging is reward evaluation. Since each episode contains many layer-wise decisions, evaluating the current

merged model on the full validation set at every reward step would be prohibitively expensive.

To address this issue, we construct a small reward subset for each task before policy training. For task k, we sample n examples from its dataset to form:

$$B_k = \{x_{k,1}, x_{k,2}, \ldots, x_{k,n}\}.$$

This subset corresponds to the reward batch B_k used in the entropy-based reward function in Section 2. The complete reward subset collection is:

$$\mathcal{B}_{\text{reward}} = \{B_1, B_2, \ldots, B_K\}.$$

During RL training, reward computation is performed only on $\mathcal{B}_{\text{reward}}$ rather than on the full datasets. This provides an efficient approximation of the multi-task reward objective while keeping the evaluation cost manageable.

Importantly, the reward subsets are used only for policy optimization. Final model evaluation is performed separately using the full evaluation protocol described in Section 5.

4.4 Environment Implementation

We implement the MDP as a layer-wise merging environment. The environment maintains the current merged model, the coefficient table, the filled-layer mask, and the reward evaluation state.

At the beginning of each episode, the coefficient table is initialized with a fixed value. The environment then visits mergeable layers in a fixed order. At step t, the policy provides a coefficient vector for the current layer. The environment writes this vector into the coefficient table and updates only the corresponding layer of the merged model:

$$\theta_t^{(m,t)} = \theta_t^{(0)} + \sum_{k=1}^{K} \lambda_{t,k} \tilde{\Delta}_t^{(k)}.$$

The filled-layer mask is updated after each step to indicate which layers have already been modified by the policy. This information is included in the state representation, allowing the policy to condition its decisions on the current progress of the merging trajectory.

To reduce environment overhead, we precompute layer-wise task-vector geometry before training. These geometry features include relative update magnitudes and direction similarities for each layer. Because they depend only on the source task vectors, they remain fixed throughout policy optimization.

The environment also manages reward evaluation. Instead of using full validation sets, it stores the reward subsets $\mathcal{B}_{\text{reward}}$ constructed for all tasks. When reward evaluation is triggered, the environment evaluates the current merged model on these reward subsets and computes the entropy-based objective.

4.5 Policy Parameterization

We use a stochastic Gaussian policy to generate layer-wise coefficient actions. Given the current state s_t, the policy outputs the mean and standard deviation of a diagonal Gaussian distribution:

$$\mu_t, \sigma_t = \pi_\phi(s_t),$$

where $\mu_t, \sigma_t \in \mathbb{R}^K$.

A raw action is sampled as:

$$u_t \sim \mathcal{N}(\mu_t, \text{diag}(\sigma_t^2)).$$

The sampled raw action is transformed into a valid coefficient vector:

$$\lambda_t = T_{\text{pos}}(u_t),$$

where $T_{\text{pos}}(\cdot)$ denotes a positive coefficient transformation. This constrains the final coefficients to be non-negative, preventing the policy from reversing source task-vector directions.

4.6 Reward Scheduling

We use the entropy-based reward defined in Section 2. However, the timing of reward evaluation is an important design choice.

A terminal-only reward evaluates the merged model only after all layers have been updated. Although computationally simple, this produces a sparse learning signal and makes credit assignment difficult, since many layer-wise decisions contribute jointly to the final result.

Therefore, our final method uses intermediate reward evaluation. The environment evaluates the current merged model every fixed number of layers and always at the terminal step. At each evaluated step, the reward is computed as the improvement of the entropy-based objective:

$$r_t^{\text{step}} = \eta_{\text{step}}(J_t - J_{\text{prev}}),$$

where J_{prev} is the most recent previously evaluated objective.

The objective J_t is estimated using the reward subsets $\mathcal{B}_{\text{reward}}$. This design provides denser feedback than terminal-only reward while avoiding the cost of full-dataset evaluation at every layer.

4.7 GRPO with Group Leave-One-Out Advantage

We optimize the policy using a GRPO-style objective. At each training iteration, we sample a group of G trajectories from the current policy:

$$\{\tau_1, \tau_2, \ldots, \tau_G\}.$$

Each trajectory corresponds to one complete layer-wise merging process. Let R_i denote the cumulative return of trajectory τ_i.

Instead of training a separate value function, we estimate the advantage using a group leave-one-out baseline. For trajectory i, the baseline is computed as the average return of the other trajectories in the same group:

$$b_i = \frac{1}{G-1} \sum_{j \neq i} R_j.$$

The advantage is then computed by comparing the trajectory return with the leave-one-out baseline and normalizing it by the standard deviation of returns within the same group:

$$A_i = \frac{R_i - b_i}{\text{Std}(R_1, \ldots, R_G) + \epsilon}.$$

This estimator compares each sampled trajectory against alternative trajectories generated under the same policy. A trajectory with a return higher than the group baseline receives a positive advantage, while a lower-performing trajectory receives a negative advantage. The group-level standard-deviation normalization further stabilizes the reward scale across different trajectory groups.

The policy is updated using a clipped objective:

$$\mathcal{L}_{\text{GRPO}} = -\frac{1}{G} \sum_{i=1}^{G} \sum_{t=1}^{L} \min\left(\rho_{i,t} A_i,\ \text{clip}(\rho_{i,t}, 1-\epsilon_{\text{clip}}, 1+\epsilon_{\text{clip}}) A_i\right),$$

where

$$\rho_{i,t} = \frac{\pi_\phi(u_{i,t} \mid s_{i,t})}{\pi_{\phi_{\text{old}}}(u_{i,t} \mid s_{i,t})}.$$

The clipping term stabilizes policy updates, while the group leave-one-out baseline removes the need for an additional critic network. The standard-deviation normalization reduces sensitivity to the absolute scale of trajectory returns and makes the policy update more stable.

4.8 Training Procedure

Algorithmically, the training process proceeds as follows. We first compute task vectors from all source models and apply TIES preprocessing. We then construct task-specific reward subsets and initialize the layer-wise merging environment.

For each training iteration, the current policy samples a group of complete merging trajectories. Each trajectory sequentially updates the model layer by layer using sampled coefficient actions. When reward evaluation is triggered, the current merged model is evaluated on the reward subsets using the entropy-based objective. After all trajectories in the group are collected, we compute their returns, estimate group leave-one-out advantages, and update the policy using the clipped GRPO objective.

The best coefficient table is selected according to the entropy-based objective observed during training. The corresponding merged model is then constructed and evaluated using the full evaluation protocol.

5 Experimental Results

5.1 Experimental Setup

5.1.1 Benchmark Tasks

We evaluate our method on vision model merging tasks using eight image classification datasets: SUN397, Stanford Cars, RESISC45, EuroSAT, SVHN, GTSRB, MNIST, and DTD. All source encoders share the same architecture and are derived from the same pretrained reference encoder. Specifically, we use `vit_large_patch14_clip_224.openai` as the pretrained reference encoder, which corresponds to a CLIP-pretrained ViT-L/14 image encoder with an input resolution of 224.

For each downstream task, we use an available task-specific fine-tuned encoder checkpoint. These task-specific encoders are not further fine-tuned in our experiments. Since the downloaded task-specific encoders do not include classification heads, we attach and train a separate task-specific classification head for each dataset while keeping the corresponding encoder fixed.

During model merging, MMCR merges only the encoder parameters through task-vector-based merging. The task vector for each source encoder is computed with respect to the shared pretrained reference encoder. The goal is to obtain a single merged encoder that preserves performance across all tasks.

Final evaluation is conducted on the full evaluation split of each task using the corresponding task-specific classification head.

5.1.2 Evaluation Metrics

We evaluate each method from two perspectives: merged-model performance and optimization cost.

For merged-model performance, we report task-wise classification accuracy and average accuracy across all tasks:

$$\mathrm{AvgAcc} = \frac{1}{K} \sum_{k=1}^{K} \mathrm{Acc}_k.$$

For optimization cost, we report wall-clock runtime of the merging process. Runtime measures the time required to obtain the final merged model, including coefficient optimization or policy optimization, but excluding the one-time cost of training the source models.

In this work, runtime is treated as a primary comparison metric rather than an auxiliary statistic. This is because the practicality of model merging depends not only on final accuracy, but also on how efficiently the merged model can be obtained. Therefore, we report

both final accuracy and runtime to analyze the trade-off between merged-model quality and practical optimization cost.

5.1.3 Reward Subsets and Data Budget Alignment

During policy training, our RL-based method uses task-specific reward subsets for entropy-based reward computation. For each task, we sample a fixed number of examples to construct the reward batch used by the RL environment. These reward subsets are used only for policy optimization, while final evaluation is performed on the full evaluation split.

For methods that require data during coefficient optimization or reward computation, we align the number of evaluated samples whenever possible. Specifically, our RL-based method computes entropy rewards using a fixed reward subset of 64 samples per task. PPO-based variants and other data-dependent baselines are configured to access the same number of samples per task when estimating their optimization objective. When mini-batch evaluation is used, we keep the total number of evaluated samples equivalent, for example using one batch of 64 samples or two batches of 32 samples. This prevents a method from obtaining an advantage simply by using more reward or validation samples during optimization.

5.1.4 Hardware and Runtime Protocol

All experiments are conducted on a workstation equipped with two NVIDIA RTX 3090 GPUs. For runtime comparison, we separate the analysis into two settings.

First, we compare AdaMerging and MMCR under a single-GPU setting. This setting controls the hardware budget and provides a fair comparison of optimization cost under the same number of GPUs.

Second, we evaluate MMCR under a two-GPU setting to analyze the parallelization benefit of the proposed trajectory-based formulation. This distinction is important because MMCR samples multiple trajectories during policy optimization. These trajectories are independent before the policy update, so their evaluations can be naturally distributed across multiple GPUs.

AdaMerging is executed as a gradient-based coefficient optimization baseline. In contrast, MMCR uses a GRPO-based policy to sample and evaluate multiple layer-wise merging trajectories. Therefore, the multi-GPU setting is used not as a strictly fair comparison against AdaMerging, but as an analysis of the practical parallelization capability of MMCR.

Our final setting uses TIES-preprocessed task vectors, positive coefficient transformation, intermediate entropy reward, and GRPO with return leave-one-out advantage estimation.

5.2 Baseline Model Merging Results

We first evaluate representative model merging baselines on the eight vision classification tasks. The compared methods include Task Arithmetic, TIES, DARE-TA, NAN-TA, and AdaMerging. We also report the performance of the pretrained model and the original fine-tuned source models as references.

Table 1: Baseline model merging results on eight vision classification tasks.

Method	SUN397	Cars	RESISC45	EuroSAT	SVHN	GTSRB	MNIST	DTD	Avg.
Pretrained model θ_{pre}	68.24	77.94	71.33	62.72	58.45	50.55	76.36	55.37	65.12
Fine-tuned source models θ_{task}	**81.33**	**92.74**	**97.10**	**99.68**	**97.19**	**97.26**	**99.10**	**84.06**	**93.06**
AdaMerging	78.70	87.66	90.62	88.78	89.08	89.79	92.61	73.67	86.36
TIES	76.11	83.93	83.65	80.66	84.70	89.14	89.29	65.53	81.63
Task Arithmetic (0.3)	75.15	82.91	80.25	79.74	76.12	82.89	87.26	65.00	78.67
NAN-TA	77.57	83.86	80.78	73.41	65.28	69.54	68.55	67.98	73.37
DARE-TA	75.01	82.65	80.02	79.07	75.86	82.41	86.93	64.36	78.29

As shown in Table 1, the original fine-tuned source models achieve the highest average accuracy of 93.06%, while the pretrained model obtains only 65.12%. This gap indicates that task-specific fine-tuning introduces substantial task knowledge into each source model. Therefore, model merging needs to preserve these task-specific updates while compressing multiple models into a single shared model.

Among the baseline merging methods, AdaMerging achieves the strongest average accuracy of 85.81%. This makes AdaMerging the strongest baseline in our comparison and provides a challenging reference for evaluating our RL-based method. Among heuristic task-vector methods, TIES performs best with an average accuracy of 81.63%, outperforming Task Arithmetic, DARE-TA, and NAN-TA. This suggests that reducing sign conflicts and removing weak updates are important for stabilizing task-vector merging.

5.3 Main Results of MMCR

We evaluate our final RL-based method, denoted as MMCR. MMCR combines GRPO, return leave-one-out advantage estimation, TIES-preprocessed task vectors, and intermediate entropy reward. The goal is to determine whether a sequential policy can improve over fixed or one-shot merging strategies by assigning adaptive coefficients to different layers.

For AdaMerging, we report two variants. AdaMerging-300 Epochs uses the same optimization budget as MMCR, while AdaMerging-500 Epochs uses a longer optimization budget. This distinction is important because AdaMerging is sensitive to the number of optimization epochs. Therefore, AdaMerging-300 Epochs is used for budget-matched comparison, while AdaMerging-500 Epochs is used as a stronger long-run baseline.

Table 2: Main results of MMCR compared with model merging baselines. AdaMerging-300 Epochs uses the same optimization budget as MMCR, while AdaMerging-500 Epochs uses a longer optimization budget.

Method	SUN397	Cars	RESISC45	EuroSAT	SVHN	GTSRB	MNIST	DTD	Avg.
TIES	76.11	83.93	83.65	80.66	84.70	89.14	89.29	65.53	81.63
AdaMerging-300 Epochs	78.20	85.56	87.67	83.50	68.45	86.81	61.74	70.96	77.86
AdaMerging-500 Epochs	78.71	87.58	90.21	88.14	88.13	89.37	91.45	72.87	85.81
AdaMerging-600 Epochs	78.70	87.66	90.62	88.78	89.08	89.79	92.61	73.67	86.36
MMCR (ours)	77.57	89.11	89.06	85.72	90.65	89.96	92.54	76.64	**86.41**

As shown in Table 2, MMCR achieves an average accuracy of 86.41%. Under the same optimization budget, MMCR substantially outperforms AdaMerging-300, improving the average accuracy from 77.86% to 86.41%. AdaMerging-300 Epochs also performs worse than TIES, suggesting that AdaMerging does not converge well within the same optimization budget used by MMCR.

Compared with the strongest AdaMerging-600 Epochs baseline, MMCR achieves a slightly higher average accuracy, improving from 86.36% to 86.41%. Although the margin is small, AdaMerging-600 Epochs requires a longer optimization budget and longer runtime. This suggests that MMCR can reach competitive accuracy more efficiently through sequential layer-wise policy learning.

Compared with TIES, MMCR improves the average accuracy by 4.78 percentage points. This suggests that TIES preprocessing alone is not sufficient; learning adaptive layer-wise coefficients can further improve the merged model. MMCR improves performance especially on Stanford Cars, SVHN, MNIST, and DTD.

Compared with AdaMerging-500 Epochs, MMCR performs better on Stanford Cars, SVHN, GTSRB, MNIST, and DTD, but performs worse on SUN397, RESISC45, and EuroSAT. This suggests that MMCR improves the overall multi-task average, although its gains are not uniform across all datasets. Overall, these results show that sequential layer-wise coefficient learning can improve the balance across tasks and achieve competitive performance against optimization-based merging baselines.

5.4 Runtime and Parallelization Analysis

In addition to final accuracy, we analyze the runtime of AdaMerging and MMCR. Since runtime can be affected by both optimization budget and hardware allocation, we report AdaMerging with multiple optimization budgets and MMCR under both single-GPU and two-GPU settings. AdaMerging-300 Epochs uses the same optimization budget as MMCR, while AdaMerging-500 Epochs and AdaMerging-600 Epochs use longer optimization budgets and achieve stronger accuracy.

Table 3: Runtime comparison of AdaMerging and MMCR under different optimization budgets and GPU settings.

Method	GPUs Used	Runtime	Time (min)	Avg. Acc.
TIES	No (CPU)	3m 14s	3.23	81.63
AdaMerging-300 Epochs	1 GPU	3h 05m 00s	185.00	77.86
AdaMerging-500 Epochs	1 GPU	5h 56m 52s	356.87	85.81
AdaMerging-600 Epochs	1 GPU	6h 58m 54s	418.90	86.36
MMCR (ours)	1 GPU	4h 05m 10s	245.17	**86.41**
MMCR (ours)	2 GPUs	2h 35m 58s	**155.97**	**86.41**

Table 3 reports both controlled single-GPU runtime and practical multi-GPU runtime. Under the same 300-iteration optimization budget, AdaMerging-300 Epochs takes 185.00 minutes but only achieves 77.86% average accuracy. This is lower than TIES, suggesting that AdaMerging does not converge well within this budget.

As the optimization budget increases, AdaMerging improves substantially. AdaMerging-500 Epochs reaches 85.81% average accuracy with 356.87 minutes of runtime, and AdaMerging-600 Epochs further improves to 86.36% with 418.90 minutes of runtime. This shows that AdaMerging can approach MMCR when given a longer optimization budget, but the improvement comes with a clear runtime cost.

In comparison, MMCR achieves 86.41% average accuracy in 245.17 minutes under the single-GPU setting. Therefore, under single-GPU execution, MMCR achieves slightly higher accuracy than AdaMerging-600 Epochs while requiring less runtime. Although the accuracy gap is small, only 0.05 percentage points, MMCR uses 173.73 fewer minutes than AdaMerging-600.

The two-GPU setting further shows the parallelization advantage of MMCR. With two GPUs, MMCR reduces the runtime from 245.17 minutes to 155.97 minutes while maintaining the same final accuracy. This corresponds to about a 1.57× wall-clock speedup:

$$\frac{245.17}{155.97} \approx 1.57.$$

This speedup is possible because MMCR samples multiple trajectories during policy optimization. Before the policy update, these trajectories are independent, so their evaluations can be distributed across multiple GPUs.

In contrast, AdaMerging optimizes a single set of merging coefficients through gradient-based updates, and its most direct implementation is less naturally parallel at the trajectory level. Therefore, MMCR is competitive with AdaMerging under single-GPU execution and benefits naturally from multi-GPU trajectory parallelism.

Overall, the runtime analysis shows that AdaMerging can improve with a longer optimization budget, but this comes with higher runtime. AdaMerging-600 Epochs nearly matches MMCR, but MMCR still achieves slightly higher accuracy with much shorter single-GPU runtime. When two GPUs are available, MMCR further reduces wall-clock time through parallel trajectory evaluation.

5.5 Ablation Studies

To better understand the contribution of each design choice in MMCR, we conduct ablation studies on four components: task-vector preprocessing, policy optimization algorithm, reward evaluation frequency, and policy transferability. These studies are designed to answer four questions.

First, we examine whether TIES preprocessing provides a cleaner task-vector space for RL-based coefficient learning. Second, we compare different policy optimization algorithms to determine whether GRPO-style group-relative optimization is more suitable than PPO-style optimization for the layer-wise merging environment. Third, we study how reward evaluation frequency affects the trade-off between reward density, final accuracy, and runtime. Finally,

we evaluate whether the learned policy can transfer across different task groups, and whether transferability depends on task similarity and task-group complexity.

5.5.1 Effect of TIES Preprocessing

We first analyze whether TIES preprocessing is necessary for RL-based model merging. Without TIES preprocessing, the policy directly operates on raw task vectors. However, raw task vectors may contain small noisy updates and sign conflicts across tasks, making the coefficient search space more difficult to optimize. With TIES preprocessing, weak updates are removed and conflicting update directions are reduced before RL-based coefficient learning.

Table 4: Effect of TIES preprocessing on MMCR.

Setting	SUN397	Cars	RESISC45	EuroSAT	SVHN	GTSRB	MNIST	DTD	Avg.
MMCR w/o TIES	77.26	88.82	87.11	80.17	88.21	88.23	84.78	74.89	83.69
MMCR w/ TIES	77.57	89.12	89.06	85.73	90.65	89.96	92.54	76.65	**86.41**
Gain	+0.31	+0.30	+1.95	+5.56	+2.44	+1.73	+7.76	+1.76	+2.72

As shown in Table 4, TIES preprocessing improves the average accuracy from 83.69% to 86.41%, yielding a gain of 2.72 percentage points. The improvement is consistent across all eight tasks, suggesting that TIES preprocessing provides a cleaner and more stable task-vector space for RL-based coefficient learning.

The largest improvements are observed on MNIST and EuroSAT, where TIES improves accuracy by 7.76 and 5.56 percentage points, respectively. This indicates that some tasks are more sensitive to noisy or conflicting task-vector updates. By removing weak updates and reducing sign conflicts, TIES helps the policy search over more reliable update directions. These results confirm that TIES preprocessing is an important component of MMCR.

5.5.2 Effect of RL Optimization Algorithms

We compare different policy optimization variants for learning the layer-wise merging policy. All variants use the same model merging environment, reward function, TIES-preprocessed task vectors, policy network, and evaluation protocol. The main differences lie in how the policy update is computed, including advantage estimation, advantage normalization, clipping strategy, sampling strategy, and policy-ratio computation. This ablation helps identify which reinforcement learning objective is more suitable for the MMCR model merging environment.

Table 5: Effect of policy optimization algorithms on MMCR.

Method	SUN397	Cars	RESISC45	EuroSAT	SVHN	GTSRB	MNIST	DTD	Avg.	Runtime	Time (min)
PPO	78.43	85.28	83.25	76.90	75.03	78.58	78.93	69.10	78.19	24m 30s	24.50
Dr. GRPO-style	78.26	88.73	88.33	83.56	77.86	87.13	82.75	75.21	82.73	2h 28m 28s	148.47
DAPO-lite	77.47	89.24	88.08	84.62	89.41	90.73	93.88	75.00	86.05	6h 33m 56s	393.93
GSPO-like	77.29	89.30	89.43	87.63	88.33	90.48	92.31	75.80	86.32	2h 31m 11s	151.18
GRPO + RLOO	77.57	89.11	89.06	85.72	90.65	89.96	92.54	76.64	**86.41**	2h 35m 58s	155.97

In the standard GRPO + RLOO setting, each policy update samples a group of trajectories. In our experiments, the group size is $G = 8$, so each update compares eight complete layer-wise merging trajectories. Each trajectory corresponds to a complete sequence of coefficient decisions over all layers and receives a trajectory return after evaluating the merged model. For trajectory i, the leave-one-out baseline is computed from the average return of the other trajectories in the same group:

$$b_i = \frac{1}{G-1} \sum_{j \neq i} R_j.$$

The advantage is then normalized by the group-level standard deviation:

$$A_i = \frac{R_i - b_i}{\mathrm{Std}(R_1, \ldots, R_G) + \epsilon}.$$

This provides a relative learning signal without requiring an additional value network.

As shown in Table 5, GRPO + RLOO achieves the best average accuracy of 86.41%. GSPO-like achieves a very close result of 86.32%, which is only 0.09 percentage points lower. DAPO-lite also reaches a competitive average accuracy of 86.05%, but it requires much longer runtime. In contrast, Dr. GRPO-style drops to 82.73%, and PPO obtains the lowest average accuracy of 78.19%.

PPO is included as a standard value-based policy optimization baseline. Although PPO finishes in only 24.50 minutes, it does not converge to a stable merging policy under the same experimental configuration. Its final average accuracy is only 78.19%, which is the lowest among all tested methods. This suggests that PPO is less suitable for the MMCR setting. A possible reason is that PPO relies on a learned value function to estimate the advantage. However, in model merging, each trajectory is a complete sequence of layer-wise coefficient decisions, and the reward is obtained only after evaluating the merged model. The reward is sparse, expensive, and highly dependent on the interaction among all layer decisions. Under this setting, the value function may not provide a stable learning signal.

DAPO-lite differs from GRPO mainly in the clipping rule and sampling strategy. It uses asymmetric clipping and dynamic resampling when the reward diversity within a group is too low. This may avoid low-diversity groups with weak learning signals, but it also increases the number of trajectory evaluations. As a result, DAPO-lite takes 393.93 minutes, which is much longer than GRPO + RLOO. However, the additional runtime does not improve final accuracy, suggesting that dynamic resampling introduces too much overhead for this task.

GSPO-like differs from GRPO in the policy-ratio computation. Standard GRPO computes the probability ratio at each layer-wise action step. In contrast, GSPO-like computes a trajectory-level ratio by summing the log probabilities over the full sequence of layer-wise actions. This makes the update depend on the quality of the complete coefficient sequence rather than each individual layer decision. This design is well aligned with model merging because the reward is evaluated on the merged model as a whole. In our experiments, GSPO-like is slightly faster and highly competitive, but its final average accuracy is still slightly lower than GRPO + RLOO.

Dr. GRPO-style differs from standard GRPO mainly in advantage normalization. Since our model merging environment does not involve variable-length responses, the length-normalization issue discussed in Dr. GRPO is not relevant here. Therefore, our Dr. GRPO-style variant mainly removes the group standard-deviation normalization from the RLOO advantage:

$$A_i = R_i - b_i.$$

Although this variant achieves a shorter runtime of 148.47 minutes, its average accuracy drops to 82.73%. This suggests that standard-deviation normalization is important for stabilizing reward scales across trajectory groups in our model merging environment.

Overall, GRPO + RLOO provides the best balance between final accuracy and runtime. GSPO-like is slightly faster and nearly matches GRPO, but does not surpass it. DAPO-lite introduces substantial sampling overhead without improving accuracy. Dr. GRPO-style is faster than GRPO + RLOO but loses too much performance. PPO is the fastest in wall-clock time, but it fails to converge under the same experimental setting and obtains the lowest accuracy. Therefore, we use GRPO + RLOO as the final policy optimization algorithm in MMCR.

5.5.3 Effect of Reward Evaluation Frequency

We study the effect of reward evaluation frequency in GRPO-based model merging. In our environment, `reward_eval_interval` controls how often the current partially merged model is evaluated during a trajectory. A larger interval provides sparser feedback and lower evaluation cost, while a smaller interval provides denser feedback but requires more frequent reward computation. We also include a terminal-only setting, where the policy receives reward only after all layers have been merged.

We compare three reward settings: terminal-only reward, reward evaluation interval 13, and reward evaluation interval 6. The terminal-only setting provides the sparsest reward

signal and the lowest reward-evaluation cost. The interval of 13 corresponds to less frequent intermediate reward evaluation, while the interval of 6 provides denser reward feedback.

Table 6: Effect of reward evaluation frequency.

Reward Setting	SUN397	Cars	RESISC45	EuroSAT	SVHN	GTSRB	MNIST	DTD	Avg.	Runtime	Time (min)
Terminal-only	77.97	89.22	89.30	85.32	88.04	89.54	92.62	77.82	86.23	50m 24s	50.40
Interval 13	77.57	89.11	89.06	85.72	90.65	89.96	92.54	76.64	86.41	2h 35m 58s	155.97
Interval 6	77.21	89.38	90.13	87.20	88.43	90.15	93.23	76.86	**86.57**	4h 07m 22s	247.37

As shown in Table 6, terminal-only reward already achieves a strong average accuracy of 86.23% with the shortest runtime of 50.40 minutes. This suggests that the final merged-model reward provides a useful learning signal even without intermediate reward evaluation.

Adding intermediate reward further improves accuracy. Interval 13 improves the average accuracy to 86.41%, and interval 6 further improves it to 86.57%. The improvement from terminal-only reward to interval 13 is 0.18 percentage points, while the improvement from terminal-only reward to interval 6 is 0.34 percentage points. This indicates that denser intermediate reward feedback can help the policy learn slightly better layer-wise coefficient assignments.

However, the accuracy improvement comes with a substantial runtime cost. Compared with terminal-only reward, interval 13 increases runtime from 50.40 minutes to 155.97 minutes, while interval 6 further increases runtime to 247.37 minutes. Therefore, although interval 6 achieves the highest average accuracy, it also requires the longest runtime among the compared settings. Terminal-only reward is the most efficient, while interval 13 provides a middle-ground trade-off between accuracy and runtime.

This ablation shows that reward evaluation frequency is an important design choice in RL-based model merging. Denser intermediate reward can improve final accuracy, but the gain is relatively small compared with the additional evaluation cost. In our final MMCR setting, we use interval 13 because it provides a better balance between accuracy and runtime than interval 6.

5.5.4 Effect of Policy Transferability

We further evaluate whether the learned GRPO policy can transfer across different task groups. This ablation examines whether MMCR learns a reusable layer-wise merging strategy, rather than only optimizing for the task group used during policy training.

In each transfer experiment, the policy is first trained on a source task group and then directly applied to a target task group without additional policy optimization. We consider three transfer scenarios. The first scenario is two-task related transfer, where the source and target task groups have similar task properties. The second scenario is three-task related transfer, where both the source and target groups contain three tasks. The third scenario is two-task unrelated transfer, where the source and target groups have different task properties.

For each scenario, we compare three methods on the target group. **TIES baseline** represents static task-vector merging without policy learning. **Transferred policy** uses the policy trained on the source group and directly applies it to the target group. **Target policy** trains the policy directly on the target group and serves as a target-specific reference.

For this ablation, we report both average accuracy and retention. Retention measures how much of the corresponding fine-tuned performance is preserved after merging:

$$\text{Retention} = \frac{\text{AvgAcc}_{\text{merged}}}{\text{AvgAcc}_{\text{fine-tuned}}} \times 100\%.$$

Two-task related transfer. The first scenario transfers the policy from EuroSAT/SVHN to RESISC45/MNIST. This setting is considered related because EuroSAT and RESISC45 are both remote-sensing or scene-oriented datasets, while SVHN and MNIST are both digit-recognition datasets.

Before transfer, the policy trained on EuroSAT/SVHN achieves an average accuracy of 95.23% on its source group, showing that the learned policy works well on the tasks used

Table 7: Two-task related transfer from EuroSAT/SVHN to RESISC45/MNIST.

Method	Policy Training Group	Avg. Acc.	Retention
TIES baseline	–	95.20	97.05%
Transferred policy	EuroSAT/SVHN	97.33	99.22%
Target policy	RESISC45/MNIST	**97.54**	**99.43%**

for policy training. As shown in Table 7, the transferred policy achieves 97.33% average accuracy on RESISC45/MNIST. This improves over the TIES baseline by 2.13 percentage points. More importantly, the transferred policy is only 0.21 percentage points lower than the target policy trained directly on RESISC45/MNIST. This result suggests that the policy trained on EuroSAT/SVHN can transfer effectively to the related RESISC45/MNIST task group.

Three-task related transfer. The second scenario increases the number of tasks from two to three. The policy is trained on EuroSAT/SVHN/DTD and transferred to RESISC45/MNIST/GTSRB. This setting tests whether policy transfer remains effective when the merging problem becomes more complex and involves more task vectors.

Before transfer, the policy trained on EuroSAT/SVHN/DTD achieves an average accuracy of 89.99% on its source group. The corresponding fine-tuned source models achieve an average accuracy of 93.64%, so the source policy retains 96.10% of the fine-tuned performance. This shows that the policy is effective on the source group, although the three-task source setting is more challenging than the two-task setting.

Table 8: Three-task related transfer from EuroSAT/SVHN/DTD to RESISC45/MNIST/GTSRB.

Method	Policy Training Group	Avg. Acc.	Retention
TIES baseline	–	93.20	95.28%
Transferred policy	EuroSAT/SVHN/DTD	91.05	93.08%
Target policy	RESISC45/MNIST/GTSRB	**96.26**	**98.40%**

As shown in Table 8, the transferred policy achieves 91.05% average accuracy on RESISC45/MNIST/GTSRB. This is lower than the TIES baseline, which achieves 93.20%, and also lower than the target policy, which achieves 96.26%. The gap between the transferred policy and the target policy is 5.21 percentage points.

This result suggests that policy transfer becomes more difficult when the number of merged tasks increases from two to three. Although the source and target groups are still related, the policy trained on EuroSAT/SVHN/DTD does not transfer as effectively to RESISC45/MNIST/GTSRB. One possible reason is that three-task merging introduces more complex interactions among task vectors, so the layer-wise coefficient strategy learned from the source group may not directly match the target group. Therefore, this result indicates that policy transfer is promising in simpler related settings, but may require additional adaptation when the target group contains more tasks.

Two-task unrelated transfer. The third scenario evaluates a more challenging transfer setting. The policy is trained on EuroSAT/RESISC45 and transferred to MNIST/SVHN. In this case, the source group mainly contains remote-sensing or scene-oriented tasks, while the target group contains digit-recognition tasks. This setting tests whether policy transfer becomes weaker when the source and target task groups have different task properties.

Before transfer, the policy trained on EuroSAT/RESISC45 achieves an average accuracy of 96.20% on its source group. The corresponding fine-tuned source models achieve an average accuracy of 98.39%, so the source policy retains 97.77% of the fine-tuned performance. This shows that the policy works well on its training task group before being transferred.

As shown in Table 9, the transferred policy achieves 94.02% average accuracy on MNIST/SVHN. This is lower than the TIES baseline, which achieves 96.97%, and also lower than

Table 9: Two-task unrelated transfer from EuroSAT/RESISC45 to MNIST/SVHN.

Method	Policy Training Group	Avg. Acc.	Retention
TIES baseline	–	96.97	98.80%
Transferred policy	EuroSAT/RESISC45	94.02	95.80%
Target policy	MNIST/SVHN	**97.06**	**98.90%**

the target policy, which achieves 97.06%. The gap between the transferred policy and the target policy is 3.04 percentage points.

This result suggests that policy transfer becomes weaker when the source and target task groups have different task properties. Although the policy trained on EuroSAT/RESISC45 performs well on its own source group, the learned layer-wise coefficient strategy does not directly transfer to MNIST/SVHN. A possible reason is that remote-sensing or scene-oriented datasets and digit-recognition datasets rely on different visual features, so their useful layer-wise merging patterns may not be aligned. Therefore, this result indicates that MMCR policy transfer is more reliable when the source and target task groups are related, but may require target-specific optimization or adaptation when the domains are substantially different.

6 Conclusion

In this project, we proposed MMCR, a reinforcement-learning-based framework for sequential layer-wise model merging. Different from fixed task-vector merging methods and one-shot coefficient optimization methods, MMCR formulates model merging as a sequential decision-making problem. At each layer, the policy observes the current merging progress, task-vector geometry, and coefficient history, and then selects adaptive coefficients for combining TIES-preprocessed task vectors.

Our experimental results show that MMCR is effective on eight vision classification model merging benchmarks. MMCR achieves an average accuracy of 86.41%, outperforming TIES, AdaMerging-300, and the longer-running AdaMerging-500 baseline. This suggests that learning adaptive layer-wise coefficients can better handle layer-dependent task-vector conflicts than fixed or one-shot merging strategies.

The runtime analysis also shows that MMCR has practical advantages under multi-GPU execution. Although MMCR requires policy optimization and is therefore more expensive than simple data-free merging methods, its trajectory-based formulation allows multiple merging trajectories to be evaluated in parallel. As a result, MMCR reduces the runtime from 341.03 minutes on one GPU to 155.97 minutes on two GPUs while maintaining the same final accuracy.

The ablation studies further confirm the importance of several design choices in MMCR. TIES preprocessing improves the stability of the task-vector space, GRPO with return leave-one-out advantage estimation provides a strong policy optimization signal, and intermediate entropy reward offers a useful trade-off between reward density and computational cost. In addition, the transfer experiments show that the learned policy can transfer well between related two-task groups, but transfer becomes less reliable when the number of tasks increases or when the source and target task groups are less related.

These findings show that reinforcement learning provides a promising way to learn adaptive layer-wise merging strategies. While MMCR improves average performance and benefits from trajectory-level parallelization, its transferability still depends on task similarity and task-group complexity. Future work may focus on improving policy generalization, reducing reward evaluation cost, and extending the framework to larger model families such as language models or multimodal models.

References

Enneng Yang, Zhenyi Wang, Li Shen, Shiwei Liu, Guibing Guo, Xingwei Wang, and Dacheng Tao. Adamerging: Adaptive model merging for multi-task learning. *arXiv preprint arXiv:2310.02575*, 2023.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*, 2022.

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*, 36:7093–7115, 2023.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *Forty-first International Conference on Machine Learning*, 2024.

Chongjie Si, Kangtao Lv, Jingjing Jiang, Yadao Wang, Yongwei Wang, Xiaokang Yang, Wenbo Su, Bo Zheng, and Wei Shen. Nan: A training-free solution to coefficient estimation in model merging. *arXiv preprint arXiv:2505.16148*, 2025.

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. *arXiv preprint arXiv:2010.11929*, 2020.

Jianxiong Xiao, James Hays, Krista A Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 3485–3492. IEEE, 2010.

Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 554–561, 2013.

Gong Cheng, Junwei Han, and Xiaoqiang Lu. Remote sensing image scene classification: Benchmark and state of the art. *Proceedings of the IEEE*, 105(10):1865–1883, 2017.

Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7):2217–2226, 2019.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 4. Granada, 2011.

Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In *The 2011 international joint conference on neural networks*, pages 1453–1460. IEEE, 2011.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3606–3613, 2014.