

Appendix of HM3

A Comprehensive Related Work about HM3

A.1 Model Merging

Model merge refers to combining the parameters and features of multiple large pretrained models to generate a unified model that can perform better across multiple tasks. Existing model merging approaches rely on task vectors constructed from fine-tuned models and their common base model. These approaches typically perform parameter-level interpolation (e.g., Task Arithmetic [27], TIES merging [72], PCB Merging [19], CAT Merging [54] and Consensus Merging [65]) or apply parameter manipulation strategies such as drop and rescale in DARE [80]. Formally, given a base model with parameters Θ_{base} , and K task-specific models fine-tuned from it with parameters $\{\Theta_1, \Theta_2, \dots, \Theta_K\}$, the model merging process can be expressed as [14]:

$$\Theta_{\text{merge}} = \mathcal{G}(\Theta_{\text{base}}, \Theta_1, \Theta_2, \dots, \Theta_K) \quad (17)$$

where Θ_{merge} denotes the parameters of the merged model, and $\mathcal{G}(\cdot)$ represents the merging method.

These methods have demonstrated significant improvements in the performance of merged models. In particular, recent works have introduced evolutionary search algorithms such as evolutionary algorithms (EAs) to enhance model merging. For example, GENOME [81] employs one of classical EA named differential evolution to evolve new models within a shared architectural weight space through crossover–mutation–selection operations, and further performs ensemble inference. Similarly, Evolver [20] directly applies EA to the weight spaces of multiple fine-tuned models, mutating and crossing their parameter vectors to select higher-performing combinations—achieving parameter fusion without gradient-based fine-tuning. Both methods emphasize low-cost and high-efficiency strategies that yield competitive performance across different scenarios. However, they primarily focus on adjusting parameter configurations within a fixed architecture. In practice, models with diverse architectures may exhibit stronger representational capacities, and potentially extend performance beyond the limits of a single-structure model under multi-task scenarios. Motivated by this, Akiba et al. [1] recently explored the use of EA to search for optimal architectures in model merging. While promising, this approach faces scalability issues: as model size increases, the architecture search space becomes substantially more complex, often resulting in performance degradation. Furthermore, EA is population-based and requires expensive evaluations in each iteration. They are also typically designed for one-shot merging, which means that the search must be restarted from scratch for every new task. This leads to prohibitively high computational costs. In this work, we unify the strengths of both parameter-level and architecture-level merging by designing an efficient joint framework. Importantly, we train a reusable model that can generalize to new tasks without requiring full re-search from scratch.

Multi-Objective in Model Merging Existing methods typically rely on the model designer’s domain knowledge or intuitive understanding to manually determine these weights, resulting in a single trade-off solution for the merged model. However, task preferences may differ across users, or even for the same user at different times, thereby demanding merged models that reflect diverse preferences. Recent works [34, 33] have begun to explore the flexibility of multi-objective optimization in assigning weights across tasks. Nonetheless, these efforts remain in their early stages, and often fail to fully exploit the nature of multi-objective trade-offs, falling short of achieving high-quality Pareto-optimal merged models. In this paper, we design a multi-objective strategy to obtain approximate Pareto optimal parameters and architectures and meet different preferences.

A.2 Multi-objective Optimization

A.2.1 Definition

Generally, a multi-objective optimization problem can be formulated as:

$$\min f(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_K(\mathbf{x})) \quad \text{s.t.} \quad \mathbf{x} \in X, \quad (18)$$

where $\mathbf{x} = (x_1, x_2, \dots, x_d)$ is a decision vector, and $f(\cdot) : X \rightarrow Y$ represents k objective functions. Here, X denotes the decision space, and Y denotes the objective space. To compare the quality of solutions obtained by the multi-objective problem, the concept of Pareto dominance is introduced.

Pareto dominance Given two solutions \mathbf{x}_1 and \mathbf{x}_2 belonging to X , \mathbf{x}_1 is said to Pareto dominate \mathbf{x}_2 (denoted as $\mathbf{x}_1 \prec \mathbf{x}_2$) if and only if the following two conditions are satisfied:

1. For all objectives $i \in \{1, 2, \dots, K\}$, $f_i(\mathbf{x}_1) \leq f_i(\mathbf{x}_2)$, meaning that \mathbf{x}_1 is not worse than \mathbf{x}_2 in every objective.
2. There exists at least one objective $j \in \{1, 2, \dots, m\}$ such that $f_j(\mathbf{x}_1) < f_j(\mathbf{x}_2)$, indicating that \mathbf{x}_1 is strictly better than \mathbf{x}_2 in at least one objective.

A solution $\mathbf{x}^* \in X$ is considered Pareto optimal if no other solution $\mathbf{x} \in X$ Pareto dominates \mathbf{x}^* . The set of all Pareto optimal solutions is known as the Pareto set:

$$PS = \{\mathbf{x} \in X \mid \nexists \mathbf{x}' \in X, \mathbf{x}' \prec \mathbf{x}\} \quad (19)$$

The collection of objective vectors corresponding to the Pareto set is referred to as the Pareto front. The aim of multi-objective optimization is to approximate the Pareto set by identifying solutions that achieve both strong convergence and a diverse spread within the objective space.

In multi-objective optimization methods, since the true Pareto optimal solution set is unknown, we employ the commonly used metric called hypervolume (HV) [56] to comprehensively assess the diversity and convergence of the generated approximate Pareto optimal solution set. Let a point set $P \subset \mathbb{R}^d$ and a reference point $\mathbf{r} \in \mathbb{R}^d$, where $d = 3$ is the number of optimization objectives. The HV of the set P is computed as follows:

$$HV(P, \mathbf{r}) = \mathcal{L}_e \left(\bigcup_{\mathbf{p} \in P} \{\mathbf{q} \mid \mathbf{p} \preceq \mathbf{q} \preceq \mathbf{r}\} \right) \quad (20)$$

where $\mathcal{L}_e(\cdot)$ represents the Lebesgue measure of a set: $\mathcal{L}_e(\mathcal{S}) = \int_{\mathbf{s} \in \mathcal{S}} \mathbf{1}_{\mathcal{S}}(\mathbf{s}) d\mathbf{s}$. Here, $\mathbf{1}_{\mathcal{S}}$ is the characteristic function of the objective space \mathcal{S} . If $\mathbf{s} \in \mathcal{S}$, then $\mathbf{1}_{\mathcal{S}}(\mathbf{s}) = 1$; otherwise, $\mathbf{1}_{\mathcal{S}}(\mathbf{s}) = 0$. In the calculation of HV, the non-dominated solutions obtained by each algorithm are normalized using the same reference set, and the reference point is typically set at (1.1, 1.1). It is important to note that a larger HV indicates a better approximation of the Pareto optimal solution set and, consequently, better performance of the corresponding multi-objective optimization method.

As for multi-objective optimization in model merging, there are two early explorations. The first paper [34] introduced a novel method called model merging with amortized Pareto fronts, which approximated evaluation metrics using a quadratic surrogate model derived from a set of pre-selected scaling coefficients. However, while this approach primarily focuses on reducing computational complexity, it does not thoroughly explore how to accurately obtain the Pareto-optimal merged model. The second paper [33] employed parallel multi-objective Bayesian optimization to systematically explore the parameter space for optimal merging configurations. However, these works are only in the early stages of exploration. They merely use multi-objective optimization methods to facilitate model merging but do not fully consider the multi-objective and multi-task characteristics inherent in the models during the merging process.

B Detail of the proposed HM3

B.1 The Proof of Problem Transformation

To handle the K -dimensional vector-valued objective $\mathbf{f} = (f_1, \dots, f_K)$, we adopt a standard linear scalarization approach. Specifically, for a given task preference vector $\boldsymbol{\lambda} \in \Delta^K$ sampled from a Dirichlet distribution, the scalarized objective is defined as:

$$F_{\boldsymbol{\lambda}}(\boldsymbol{\Theta}, \alpha) := \sum_{k=1}^K \lambda_k f_k(\boldsymbol{\Theta}, \alpha). \quad (21)$$

We write $F := F_{\boldsymbol{\lambda}}$ for brevity. Solving the Stackelberg game for each $\boldsymbol{\lambda}$ produces a set of solutions that approximates the Pareto front.

Assumption 1 (Compactness and Continuity). *1. The parameter space $\mathcal{P} \subset \mathbb{R}^{d_{\theta}}$ is nonempty and compact. The architecture space \mathcal{M} is finite and contains at least one feasible base path α_{base} .*

2. For any $\Theta \in \mathcal{P}$, the follower's feasible set

$$\Omega(\Theta) := \{\alpha \in \mathcal{M} \mid |\alpha| \leq T_{\max}, \dim_{\text{out}}(m_t, l_t; \theta_{m_t, l_t}) = \dim_{\text{in}}(m_{t+1}, l_{t+1}; \theta_{m_{t+1}, l_{t+1}}), \forall t\} \quad (22)$$

is nonempty (since $\alpha_{\text{base}} \in \Omega(\Theta)$) and has a closed graph.

3. The scalarized utility $F(\Theta, \alpha)$ is jointly continuous in (Θ, α) .

Proof. (a) **Follower-level solution existence.** Since \mathcal{M} is finite, the constrained set $\Omega(\Theta) \subseteq \mathcal{M}$ is finite and nonempty for any fixed $\Theta \in \mathcal{P}$. Hence, the follower-level optimization

$$\alpha^*(\Theta) := \arg \max_{\alpha \in \Omega(\Theta)} F(\Theta, \alpha) \quad (23)$$

admits at least one solution. Furthermore, by Berge Maximum Theorem [6], the best-response mapping $\alpha^*(\Theta)$ is upper hemicontinuous with compact (finite) values due to the closed graph property and continuity of F .

(ii) **Continuity of leader's objective.** Define the upper-level objective:

$$\bar{F}(\Theta) := \max_{\alpha \in \Omega(\Theta)} F(\Theta, \alpha). \quad (24)$$

Because $\alpha^*(\Theta)$ is upper hemicontinuous and F is continuous, it follows from [2, Corollary 17.32] that \bar{F} is continuous on the compact domain \mathcal{P} . Therefore, by Weierstrass' Theorem, there exists a maximizer $\Theta^* \in \arg \max_{\Theta \in \mathcal{P}} \bar{F}(\Theta)$.

(iii) **Equilibrium construction.** Select any $\alpha^* \in \alpha^*(\Theta^*)$. Then the pair (Θ^*, α^*) satisfies the definition of a Stackelberg equilibrium [31, Def. 2.2], and achieves the same optimal value as the original joint objective. \square

Remark. Assumption (A2) is typically ensured in practice by guaranteeing at least one dimension-compatible base path (e.g., through 1x1 projections when needed). Each choice of λ induces a scalarized subproblem, and the collection of corresponding equilibria approximates the Pareto front.

B.2 The Proof of Wolpertinger Policy in Actor-Critic Framework

Let the discrete candidate action set generated by the Wolpertinger policy be denoted as $\mathcal{W}^* = \{A_1, A_2, \dots, A_M\}$, which contains the M nearest neighbors in Euclidean distance of the continuous proto-action $\hat{A} \in \mathbb{R}^d$. Among them, define A_{ℓ^*} to be the nearest discrete action to \hat{A} , i.e., the one selected by the simple projection method:

$$A_{\ell^*} = \arg \min_{A \in \mathcal{W}^*} \|A - \hat{A}\|_2. \quad (25)$$

Assume the action-value function $Q(\mathcal{S}, A)$ under fixed state \mathcal{S} satisfies the following statistical assumptions:

- For all $\ell \neq \ell^*$, the values $Q(\mathcal{S}, A_\ell)$ are i.i.d. samples from a uniform distribution:

$$Q(\mathcal{S}, A_\ell) \sim \mathcal{U}(Q(\mathcal{S}, A_{\ell^*}) - \xi, Q(\mathcal{S}, A_{\ell^*}) + \xi), \quad (26)$$

where $\xi > 0$ is a fixed constant.

- The value of the nearest action A_{ℓ^*} is set as the reference:

$$Q(\mathcal{S}, A_{\ell^*}) = q_0. \quad (27)$$

Let A_w^* be the action selected by the Wolpertinger strategy, i.e., the one in \mathcal{W}^* with the maximum Q-value:

$$A_w^* = \arg \max_{A \in \mathcal{W}^*} Q(\mathcal{S}, A). \quad (28)$$

Then, the expected Q-value of the selected action is:

$$\mathbb{E}[Q(\mathcal{S}, A_w^*)] = q_0 + \xi \left(1 - \frac{2(2M-1)}{M \cdot 2^M} \right). \quad (29)$$

Let X_1, X_2, \dots, X_{M-1} denote the i.i.d. uniform random variables representing the Q-values of the other $M - 1$ candidates:

$$X_i \sim \mathcal{U}(q_0 - \xi, q_0 + \xi), \quad i = 1, \dots, M - 1. \quad (30)$$

Let $X_{\max} = \max\{X_1, \dots, X_{M-1}\}$. Then the probability density function (PDF) of X_{\max} is:

$$f_{X_{\max}}(x) = (M - 1) \cdot \frac{1}{2\xi} \left(\frac{x - (q_0 - \xi)}{2\xi} \right)^{M-2}, \quad x \in [q_0 - \xi, q_0 + \xi]. \quad (31)$$

The expected maximum is:

$$\mathbb{E}[X_{\max}] = \int_{q_0 - \xi}^{q_0 + \xi} x \cdot f_{X_{\max}}(x) dx = q_0 + \xi \cdot \left(1 - \frac{2(2M - 1)}{M \cdot 2^M} \right). \quad (32)$$

Note that if $X_{\max} > q_0$, then the maximum selected action A_w^* will not be A_{ℓ^*} , but one of the other candidates. If all $X_i < q_0$, then A_{ℓ^*} is still chosen. Therefore, the expected Q-value of the Wolpertinger-selected action is:

$$\mathbb{E}[Q(\mathcal{S}, A_w^*)] = \mathbb{E}[\max\{q_0, X_1, \dots, X_{M-1}\}] = \mathbb{E}[\max\{q_0, X_{\max}\}]. \quad (33)$$

By integrating over the support and using order statistics of uniform distributions, the final result is:

$$\mathbb{E}[Q(\mathcal{S}, A_w^*)] = q_0 + \xi \left(1 - \frac{2(2M - 1)}{M \cdot 2^M} \right). \quad (34)$$

The proxy estimation error is bounded by:

$$|\tilde{Q}(\mathcal{S}, A) - Q(\mathcal{S}, A)| \leq \delta, \quad \forall A \in \mathcal{W}^*. \quad (35)$$

Then the expected Q-value of the Wolpertinger-selected action satisfies:

$$\mathbb{E}[Q(\mathcal{S}, A_w^*)] \geq Q(\mathcal{S}, A_s^*) + \xi \left(1 - \frac{2(2M - 1)}{M \cdot 2^M} \right) - \delta. \quad (36)$$

When $M > 1$ and $\delta < \xi \left(1 - \frac{2(2M-1)}{M \cdot 2^M} \right)$, we obtain:

$$\mathbb{E}[Q(\mathcal{S}, A_w^*)] > Q(\mathcal{S}, A_s^*), \quad (37)$$

which confirms the superiority of the Wolpertinger policy even in the presence of bounded proxy approximation error.

B.3 Algorithm Description

Based on the MDP, the execution of the RL is the following stages: **Stage 1: Input and initialization:** The algorithm begins by sampling preference vectors, each corresponding to a decomposed subproblem in the multi-objective framework. For each preference vector, an existing parameter-level merging method is applied to obtain an initial merged model parameter. Meanwhile, the parameters of the policy, value, and the MLP network for alignment are initialized. **Stage 2: Trajectory collection and MLP alignment:** For each preference vector, an inner loop is executed in RL. In each iteration, the parameter-level merged model and the fine-tuned models are evaluated to compute the reward and stored in the trajectory buffer. Then, at each step, the current trajectory-encoded state is used to generate a proto-action, which is discretized via the Wolpertinger policy to select the next action. The new state is converted, and the transition is recorded in the buffer. Once a complete path is collected, the algorithm performs MLP alignment. **Stage 3: Reward computation and network update:** After the alignment, the reward is computed and then is assigned to all time steps in the trajectory. Once the iteration number exceeds , the algorithm enters the network update phase. In this phase, a mini-batch is sampled from the buffer to compute GAE and target returns. The policy, value network, and MLP alignment network are updated. The entire process continues until Max_{iter} is reached, yielding the

Algorithm 1 HM3 in the architecture space

- 1: **Input:** A set of preference vectors $\{\lambda^1, \lambda^2, \dots, \lambda^N\}$ and their corresponding optimal merged models at the parameter space.
 - 2: **for** each preference vector λ^n in $\{\lambda^1, \lambda^2, \dots, \lambda^N\}$ **do**
 - 3: Input K fine-tuned models and the optimal merged model in the parameter space corresponding to λ_i .
 - 4: Initialize the parameters of policy network as μ_0 , of value network as ϕ_0 , of MLP network as MLP_{μ_0} .
 - 5: **for** each iteration $iter = 1, 2, \dots, \text{Max_iter}$ **do**
 - 6: Sample the current policy $\pi_{\mu_{iter}}(A_t|S_t)$ by interacting with the environment to generate a trajectory of length T as $\{S_t, A_t, R_t, S_{t+1}\}_{t=1}^T$.
 - 7: Obtain the state $S_t = (m_t, l_t)$.
 - 8: Select the action $A_t = (m_{t+1}, l_{t+1})$.
 - 9: Calculate the reward R_t based on A_t and the merged model in the t -th step.
 - 10: Compute the advantage function \hat{A}_t and \hat{G}_t .
 - 11: Update the policy network by maximizing $L^{CLIP}(\mu) = \mathbb{E}_t \left[\min \left(\rho_t(\mu) \hat{A}_t, \text{clip}(\rho_t(\mu), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right]$.
 - 12: Update the value network by minimizing $L^{VF}(\phi) = \mathbb{E}_t \left[\left(V_\phi(S_t) - \hat{G}_t \right)^2 \right]$.
 - 13: Compute the scaling matrix $W_{m,l}$, and update the MLP network parameters MLP_μ .
 - 14: **end for**
 - 15: **end for**
 - 16: **Output:** Optimal policy network parametered μ^* and the optimal inference path (i.e., the optimal sequence of actions) corresponding to the value network.
-

final policy network and the optimal inference paths. The well-trained networks can be reused for new tasks or model pools.

The overall algorithm of HM3 is summarized as Algorithm 1. It begins by taking in a collection of preference vectors and the associated best-merged models from the parameter space (Line 1). Each preference vector introduces the fine-tuned models and the corresponding best-merged model (Line 3). The initial parameters for the policy, value network, and MLP network are set up (Line 4). During each loop, the policy network interacts with the environment, creating a trajectory composed of state, action, and reward information (Line 6). The state at the current step and the chosen action are determined next (Lines 7-8). Subsequently, the reward for the current action is calculated based on the state within the merged model (Line 9). These rewards are then utilized to compute the advantage and target values (Line 10). The algorithm adjusts the policy network by enhancing the policy loss and refines the value network by minimizing the value loss (Lines 11-12). The process also involves calculating the scaling matrix and fine-tuning the MLP network through PPO (Line 13). Finally, it outputs the final set of optimized policy parameters and the sequence of actions that represent the optimal inference path tied to the value network (Line 16).

After obtaining the Pareto-optimal models, HM3 assumes users typically do not provide explicit preferences and supports two practical modes: (i) Offline preference sampling: We uniformly sample preference vectors to approximate the Pareto front. Users can later select a model matching their needs (no input required.) (ii) Optional user preference injection: If a user specifies a preference (e.g., prioritizing translation), we select the closest model on the front or conduct a targeted search, enabling both automated and interactive use.

B.4 Detail Analysis of Dimension Alignment via Statistical Matching

To accommodate distributional shifts across layers from different models, we introduce a feed-forward MLP network that generates a scaling matrix $W_{m,l}$. The input to the MLP consists of the layer index pair (m, l) and the current time step t , and its output is defined as:

$$W_{m,l} = \text{MLP}_\mu(m, l, t), \quad (38)$$

where MLP_μ is parameterized by μ and optimized via actor-critic method.

This design is motivated by the theory of statistical moment alignment. Suppose the hidden representations from the source and target layers follow Gaussian distributions:

$$z_{\text{src}} \sim \mathcal{N}(\mu_{\text{src}}, \Sigma_{\text{src}}), \quad z_{\text{tgt}} \sim \mathcal{N}(\mu_{\text{tgt}}, \Sigma_{\text{tgt}}). \quad (39)$$

The squared 2-Wasserstein distance between them is:

$$W_2^2 = \|\mu_{\text{src}} - \mu_{\text{tgt}}\|_2^2 + \text{Tr} \left(\Sigma_{\text{src}} + \Sigma_{\text{tgt}} - 2(\Sigma_{\text{tgt}}^{1/2} \Sigma_{\text{src}} \Sigma_{\text{tgt}}^{1/2})^{1/2} \right), \quad (40)$$

and the optimal affine transformation $T(z) = A_{\text{opt}}z + b$ is given by:

$$A_{\text{opt}} = \Sigma_{\text{tgt}}^{1/2} \Sigma_{\text{src}}^{-1/2}, \quad b = \mu_{\text{tgt}} - A_{\text{opt}}\mu_{\text{src}}. \quad (41)$$

This whitening followed by coloring transformation achieves exact alignment of first- and second-order statistics. To enable end-to-end learning, we approximate this process using the deep CORAL loss [53]:

$$\mathcal{L}_{\text{Deep-CORAL}} = \|C(H'_{\text{src}}) - C(H_{\text{tgt}})\|_F^2,$$

where $H'_{\text{src}} = W_{m,l}((H_{\text{src}} - \mu_{\text{src}}) \oslash \sigma_{\text{src}}) + b_{m,l}$, and $C(\cdot)$ denotes the empirical covariance matrix. If mean alignment is also desired, we add a mean alignment loss:

$$\mathcal{L}_{\mu} = \|\mu_{\text{src}} - \mu_{\text{tgt}}\|_2^2. \quad (42)$$

The final training objective for the MLP-based alignment layer becomes:

$$z'_{\text{src}} = W_{m,l} \cdot \frac{z_{\text{src}} - \mu_{\text{src}}}{\sqrt{\text{diag}(\Sigma_{\text{src}})}} + b_{m,l}, \quad \min_{W_{m,l}, b_{m,l}} \mathcal{L}_{\text{Deep-CORAL}} + \lambda \mathcal{L}_{\mu},$$

which allows the MLP to approximate the theoretically optimal mapping (A_{opt}, b) via gradient descent, thereby providing robust statistical alignment for seamless composition of heterogeneous transformer layers.

C Additional Results

C.1 Detail of Experimental Setup

The core objective of this study is to design and implement an efficient and multitask-adaptive model merging framework. To verify the generality and performance of the proposed merging method, we apply it to three popular series LLMs, namely the fine-tuned models based on Qwen-2.5-1.5B³ [59], Llama-2-7B⁴ [60] and Llama-2-13B⁵. Specifically, for Llama-2-7B [60], we include three fine-tuned models: Llama-7B-Chat⁶ for text translation [60], WizardMath-7B⁷ for mathematical reasoning [40], and CodeLlama-7B⁸ for code generation [47]. Similarly, for Qwen-2.5-1.5B [59], we include three fine-tuned models: Qwen-2.5-1.5B-Instruct⁹ for text translation [59], Qwen-2.5-Code-1.5B¹⁰ for code generation [26], and Qwen-2.5-Math-1.5B¹¹ for mathematical reasoning [76]. As for Llama-2-13B, we include three fine-tuned models: WizardLM-13B¹² for text translation [71], WizardMath-13B¹³ for mathematical reasoning [40], and WizardCoder-Python-13B¹⁴ for code generation [42].

- Llama-7B-Chat: <https://huggingface.co/meta-llama/Llama-2-7b-chat-hf>;

³<https://huggingface.co/Qwen/Qwen2.5-1.5B>

⁴<https://huggingface.co/meta-llama/Llama-2-7b-hf>

⁵<https://huggingface.co/meta-llama/Llama-2-13b-hf>

⁶<https://huggingface.co/meta-llama/Llama-2-7b-chat-hf>

⁷<https://huggingface.co/WizardLMTeam/WizardMath-7B>

⁸<https://huggingface.co/codellama/CodeLlama-7b-hf>

⁹<https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct>

¹⁰<https://huggingface.co/Qwen/Qwen2.5-Coder-1.5B>

¹¹<https://huggingface.co/Qwen/Qwen2.5-Math-1.5B>

¹²<https://huggingface.co/WizardLMTeam/WizardLM-13B-V1.2>

¹³<https://huggingface.co/vanillaOVO/WizardMath-13B-V1.0>

¹⁴<https://huggingface.co/WizardLMTeam/WizardCoder-Python-13B-V1.0>

- WizardMath-7B: <https://huggingface.co/WizardLMTeam/WizardMath-7B>;
- CodeLlama-7B: <https://huggingface.co/codellama/CodeLlama-7b-hf>;
- LLama-2-7B: <https://huggingface.co/meta-llama/Llama-2-7b-hf>.

By leveraging these fine-tuned LLMs, we can employ our proposed HM3 to merge LLMs excelled across multiple tasks. To evaluate the performance of the merged LLMs by HM3, we perform three tasks, including language translation, mathematical reasoning, and code generation. To achieve these evaluations quickly and efficiently, we employed two popular large model evaluation packages: Im-evaluation-harness [21] for text translation and mathematical reasoning tasks and big code-evaluation-harness for code generation tasks. These evaluation packages can be found in the following link:

- Im-evaluation-harness: <https://github.com/EleutherAI/Im-evaluation-harness>;
- bigcode-evaluation-harness: <https://github.com/bigcode-project/bigcode-evaluation-harness>.

To further demonstrate the effectiveness and superiority of our method compared to other model merging methods, we utilized the mergekit package [22] to merge models by using several merging methods, including Task Arithmetic, TIES, DARE-TIES, and EA [1]. The mergekit package can be found at the following link:

- mergekit: <https://github.com/arcee-ai/mergekit>

Additionally, for HM3, Max_{iter} is 1000 and the networks begin updating after $iter_0 = 200$; the clip range ϵ is set to 0.1, the discount factor γ is configured to 0.990, β_A is set to 0.95, and the coefficients c_1 and c_2 are set to 1.0 and 0.15, respectively. We split the dataset where 70% is used for RL inference evaluation, while the 30% is reserved for the evaluation of the obtained merged model. Then, we introduce the specific datasets for generative, text translation, math reasoning, and code generation tasks, as well as their metrics.

C.1.1 Generative Tasks

We use GLUE benchmark [63], a widely used benchmark for natural language understanding comprising nine sentence/sentence-pair classification tasks (i.e., CoLA, SST-2, MRPC, STS-B, QQP, MNLI, QNLI, RTE, WNLI). These tasks, drawn from established datasets, cover varied sizes, genres, and difficulties, offering a broad and challenging evaluation of language understanding.

C.1.2 Text Translation Tasks

To evaluate the multilingual translation capabilities of LLMs, we leveraged a set of translation tasks in the Im-evaluation-harness package, including WMT14¹⁵, WMT16¹⁶ [50], and IWSLT2017 [7], and Xnli [15]. These tasks evaluate the model’s translation accuracy and fluency across diverse language pairs. For the first translation tasks, we use the “chrF” metric, which measures translation quality based on character n-gram precision and recall. For the final task, we instead use accuracy as the evaluation metric.

C.1.3 Math Reasoning Task

In this paper, we use MathQA [3] and GSM8K [12] for evaluation of the math reasoning capability of the obtained models. MathQA [3] is a large-scale benchmark of roughly 37K English multiple-choice math word problems covering a broad range of mathematical topics. It also provides operation programs aligned with problems from the AQuA dataset. Model performance on MathQA is reported as accuracy. GSM8K [12] is a dataset meticulously designed for mathematical problem-solving tasks, comprising over 8,000 high-quality problems that span from basic arithmetic to complex algebra. The primary objective of this dataset is to evaluate the model’s reasoning and computational abilities when tackling structured mathematical problems. For evaluating the GSM8K dataset, we employ the “flexible match” metric, which allows for minor variations in the final answer.

¹⁵<https://www.statmt.org/wmt14/translation-task.html>

¹⁶<https://www.statmt.org/wmt16/translation-task.html>

Table 5: Comparison of merging methods for Llama-2-13B series LLMs on language tasks

Merging Methods	General	Translation	Math		Code		
	Glue	WMT&ISWT	Xnli	GSM8k	MathQA	HumanEval	MBPP
Fine-tuned Model - Chat	67.05	43.29	47.11	33.82	22.67	21.32	22.56
Fine-tuned Model - Math	45.08	27.02	41.23	55.74	31.33	19.21	20.83
Fine-tuned Model - Code	19.27	32.49	42.61	14.75	27.09	51.82	29.63
Task Arithmetic	35.24	29.16	37.33	27.74	21.47	28.52	25.80
Ties Merging	37.67	33.23	39.11	28.02	21.53	28.74	31.40
DARE + Ties Merging	48.17	33.56	40.40	38.83	23.81	30.18	24.43
Consensus Merging	45.15	32.54	43.03	36.24	24.21	38.82	25.03
PCB Merging	52.77	41.96	40.90	46.34	24.56	40.15	26.57
EA	15.92	25.06	28.23	17.36	19.29	13.31	14.48
HM3	53.20	41.12	45.92	46.82	29.13	43.93	33.29

C.1.4 Code Generation Task

In the code generation domain, we evaluate on MBPP and HumanEval. MBPP [5] contains 974 beginner-level tasks targeting short Python program synthesis from natural-language prompts; performance is measured with pass@1. HumanEval [9] is a benchmark dataset proposed by OpenAI, specifically designed to evaluate code generation capabilities. The dataset comprises 164 programming problems, where each problem requires the model to generate a Python function based on a natural language description. The evaluation metric of HumanEval is pass@100. The model is allowed to generate up to 100 code solutions for each problem. This metric assesses whether at least one of these generated solutions passes all test cases.

C.2 Detail of Main Results

Discussion An interesting observation from Table 1 and Table 2 is that the models produced by HM3 sometimes outperform or closely match task-specific fine-tuned models, which are typically considered performance upper bounds for their respective tasks. We attribute this to a key distinction: while fine-tuned models adapt only at the parameter level, HM3 performs both parameter-level and architecture-level merging. As discussed in the introduction, models with different architectures exhibit diverse representational capacities and task preferences, which can extend the performance boundary beyond that of a single fixed architecture. For example, if the original model has 36 layers and the merged model increases to 40 layers, then according to the scaling law, the merged model is expected to have a higher theoretical performance ceiling.

Merging LLAMA-2-13B model Table 5 presents the performance comparison of various merging methods on the LLaMA-15B series across general, translation, math, and code tasks. The proposed HM3 framework achieves consistent and significant improvements over existing classical and SOTA parameter-level, and architecture-level baselines. HM3 attains the highest average performance across all task categories, particularly excelling on GSM8K, Xnli, and HumanEval.

Merging ViT-B/32 model As shown in Table 3, the HM3 method consistently outperforms other approaches on ViT-B/32, achieving an average accuracy of 66.91%, which represents a significant improvement. Specifically, HM3 achieved 77.21% and 77.62% on the EuroSAT and SVHN datasets, respectively, and recorded a 68.21% accuracy on the GTSRB dataset, surpassing other methods. Although its performance on the DTD dataset is slightly lower compared to the other tasks, it still outperforms the Ties and Task Arithmetic methods.

Merging ViT-L/14 model Table 4 summarizes the performance of different merging methods on the ViT-L/14 model across various vision tasks. The results indicate that the HM3 method consistently achieved the best performance across most tasks, with particularly high accuracy on the SVHN and GTSRB datasets, reaching 90.48% and 83.43%, respectively. Notably, HM3 achieved an overall average accuracy of 80.30% across all datasets, significantly outperforming the other merging methods.

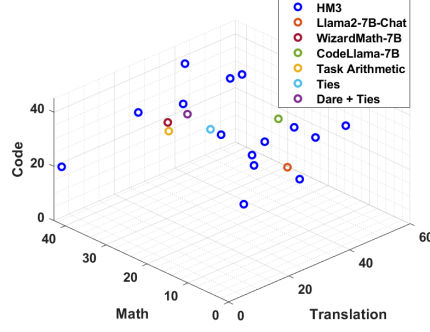


Figure 4: The illustration of different model merging methods in the text translation, math reasoning and code generation tasks.

C.3 Detail Analysis of Multi-Objective Model Merging

HM3 is capable of generating a set of approximately Pareto-optimal merged models, which enables adaptation to different user preferences. Unlike other merging methods that produce only a single solution, HM3 yields a diverse set of candidate models. To compare their quality fairly, we compute the Pareto dominance relations by pooling together all solutions from different methods. A solution x_a is said to be dominated by another solution x_b if x_b is no worse in all objectives and strictly better in at least one. The detailed computation procedure is provided in Appendix A.2. Based on this analysis, we evaluate the dominance relations across all solution sets, as shown in Figure 2. Yellow cells indicate that the solution on the vertical axis is Pareto dominated by the one on the horizontal axis, while blue cells denote no dominance. The results show that every competing method is dominated by at least one solution from the HM3 solution set (i.e., S1–S15). To compared with EA, we extend EA to a multi-objective version (MOEA) as a baseline. Since MOEAs also produce a set of solutions, we employ the hypervolume (HV) metric [25, 56], which measures both convergence and diversity. A higher HV value indicates a better overall solution set. HM3 achieves an HV of 1.6824, significantly outperforming the MOEA baseline, which obtains an HV of only 1.1329. This gap highlights the inefficiency of unguided EA-based methods in complex multi-objective spaces.

C.4 Effect of Different Number of Tasks

In this subsection, we demonstrate the effectiveness of HM3 across different numbers of tasks. In the main text, we illustrated the effectiveness of HM3 on three tasks, and the illustration of metrics for different merging methods is shown in Figure. 4. From it, our approach was capable of producing a set of Pareto-optimal merged models, along with their corresponding metrics, which provided valuable guidance for users to personalize their selection based on the specific needs of their tasks. In contrast, other methods were limited to generating only a single solution.

Here, we provide evidence of HM3’s effectiveness on two tasks: code generation and mathematical reasoning. The experimental results are presented in Figure 5, which clearly demonstrate the significant advantages of HM3. Specifically, HM3 is capable of generating a Pareto optimal set of solutions that excel not only in parameter optimization but also in architectural configurations. The blue circles in the figure represent HM3, showing that its solutions are well-distributed across the entire performance curve. Based on the available data, we used convex hull software and Gaussian process fitting to approximate the Pareto front. The results indicate that the solutions produced by HM3 closely approximate a comprehensive Pareto front, effectively capturing the optimal trade-offs under varying conditions. In contrast, other methods, such as Task Arithmetic, Ties, and DARE Ties, are restricted to generating a single optimal solution exclusively in the parameter space, as indicated by the red squares, yellow stars, and green diamonds, respectively. It is evident that the solution sets produced by these methods are confined to narrower regions of the performance spectrum, lacking the diversity and flexibility that HM3 provides. Moreover, these single solutions are noticeably farther from the approximated Pareto front. Consequently, HM3 not only demonstrates the ability to explore the parameter space but also effectively utilizes architectural optimizations to achieve a more comprehensive enhancement in performance.

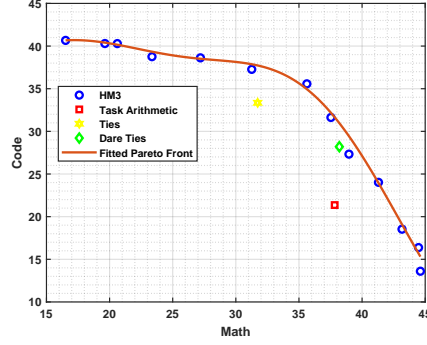


Figure 5: The illustration of different model merging methods in the math reasoning and code generation tasks.

Table 6: Performance of HM3 in different spaces

Instance	Single Objective			Multi-Objective
	Translation	Math	Code	HV
HM3 w.o. para. opt.	32.21	18.36	20.67	1.3506
HM3 w.o. archi. opt.	34.01	38.51	28.67	1.6387
HM3	44.68	45.62	43.62	1.8120

C.5 Detailed Analysis of Ablation Study

To assess the effectiveness of HM3 in jointly optimizing both the parameter and architecture spaces, we conduct two ablation experiments comparing the following three variants: (i) HM3, (ii) HM3 w.o. arch (without architecture-level optimization), and (iii) HM3 w.o. para (without parameter-level optimization). The results are summarized in Table 6. In the first experiment, we evaluate performance under a single-objective setting using a sampled preference vector. HM3 consistently outperforms the two ablated versions across all tasks, with especially notable gains in code generation. This suggests that the joint optimization of both spaces yields significant synergistic benefits, and that architecture-level adaptation plays a crucial role in tasks with more structural complexity. The second experiment examines the HV of these methods in a multi-objective setting. HM3 achieves the highest HV score, followed by HM3 w.o. arch, while HM3 w.o. para performs the worst. These results highlight the importance of parameter optimization in ensuring competitive solutions across objectives, while also demonstrating that architecture optimization further enhances the solution frontier. To this end, these findings confirm that both parameter space and architecture space are essential for achieving strong and flexible performance in model merging. HM3’s superiority in both single- and multi-objective settings underscores the effectiveness of its joint optimization design.

C.6 Computational Cost Analysis

Compared to the conventional pretraining and fine-tuning paradigm, HM3 requires significantly less computational power and time to achieve a high-performance model with a novel architecture. Moreover, it eliminates the need for high-quality data for pretraining or fine-tuning. Specifically, the network scale of policy and value networks with 200MB parameters in HM3 is much smaller than the LLM with 13.5GB parameters (i.e., Llama-2-7B) in the fine-tuning stage. In this manner, the computational power required for RL training is significantly lower than for fine-tuning. In this paper, we used A6000 or 3090 GPUs for RL training of HM3, whereas full fine-tuning typically requires A100-level GPU clusters. Additionally, the overall computation time of HM3 is significantly lower than that of fine-tuning. During RL training, only inference (i.e., forward propagation) is required, whereas full fine-tuning necessitates both forward and backward propagation of the LLM. According to time complexity analysis, the backward propagation is typically several times more consuming than the forward. Therefore, the time required for the full fine-tuning is several times greater than that for RL on the same dataset. Finally, we compare the performance of HM3 and the full fine-tuning at a similar time cost, and the results are provided in Table. 7. We can observe that HM3 still obtains

Table 7: Comparison between HM3 and full fine-tuning methods

Method		Translation	Math	Code
Fine-Tuning Method	LLAMA-2-7B-Chat	42.23	24.15	21.66
	WizardMath-7B	37.72	45.60	22.74
	CodeLLama-7B	36.12	18.37	43.11
Grid Search		29.31	21.21	23.12
Evolutionary Search		31.92	23.49	26.90
HM3		44.68	45.62	43.62

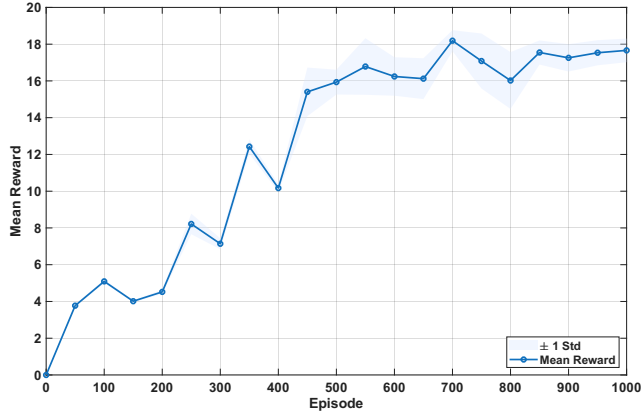


Figure 6: The convergence of RL in the HM3 at the architecture space.

the best performance compared with traditional fine-tuning methods at a similar time cost. Given a preference vector, we compared HM3 not only against traditional efficiency baselines such as Full Fine-tuning, but also introduced two additional search-based architecture merging baselines for a more comprehensive efficiency comparison: (i) Grid Search: Under the same maximum inference path length T_{\max} as HM3, this method exhaustively enumerates all possible path combinations until reaching approximately the same number of evaluations as HM3. Layer-wise composition is performed using the passthrough strategy implemented in Mergekit. (ii) Evolutionary Search: The individual encoding shares the same maximum path length as HM3. The search operators follow the standard Differential Evolution algorithm with default hyperparameters and population size. The number of evaluations is aligned with that of HM3. Layers are composed using Mergekit’s passthrough method. This approach differs from the EA baseline in the original manuscript, which restricts the search space by controlling model sequences via tokens—a design that significantly reduces complexity but at the cost of final merged model performance. The final performance comparison is summarized in the table. These results demonstrate that, under the same computational budget, HM3 achieves superior multi-task performance compared to other architecture merging methods based on search algorithms.

C.7 Effectiveness of RL

In this subsection, we delve into the convergence of HM3. Specifically, we randomly sample a preference vector and observe the obtained reward when merging models on text translation, mathematical reasoning, and code generation tasks. The experimental results are illustrated in Figure 6. As shown in Figure 6, the overall reward increases progressively as the number of training episodes increases. During the first 200 episodes, the growth in reward was relatively slow, which is attributed to PPO’s exploration phase, where HM3 had not yet accumulated sufficient experience and the policy network had not been trained. However, after episode 200, with the introduction of the experience replay mechanism, the reward begins to rise significantly, indicating that the algorithm is gradually learning from past experiences and improving its policy. As training continues, the reward shows a more stable upward trend and eventually converges to a value close to 18 around the 1000th episode. HM3 can effectively leverage past experiences to optimize its policy and achieve convergence.

D Discussions and Limitations

In this section, we discuss some concerns about this work from the multi-objective, architecture-level merging, and future work perspectives.

As for the multi-objective perspective, we set the number of objectives to three, namely translation, math, and code. These objectives are intentionally chosen for their conflicts and diversity, which help produce a well-separated and sparse Pareto front in multi-objective optimization. Adding more objectives that do not bring sufficiently greater diversity, such as a general language understanding objective, would shift the setting into the many-objective regime. That shift increases front dimensionality, densifies the set of solutions, and weakens dominance relations, all of which are known to degrade the effectiveness of standard multiobjective methods. For these reasons, we do not expand the objective set here. If expansion is necessary, pair it with objective selection and decorrelation, decomposition or reference-vector methods, indicator-based selection, and dimensionality reduction. Future work centers on many-objective algorithms and high-dimensional discrimination, theory for cross-architecture merging, and data or compute-efficient evaluation and transfer.

From the architecture-level merging perspective, HM3 is search-based and achieves high efficiency, though search itself can be unstable, sometimes falling into local optima or producing large variance. Compared with parameter-level methods, architecture-level merging typically incurs a higher time cost, a common limitation of current techniques. HM3 performs strongly when merging models that share the same base architecture but target different tasks, extending the performance boundary beyond parameter merging restricted to identical structures. This work is an initial step toward architecture-level merging, focusing on handling post-merge structural differences; the idea parallels SOLAR 10.7B, which concatenates the first twenty and last twenty layers of Mixtral-7B with continued pretraining, while we pursue similar architectural expansion via training-free model merging. When markedly heterogeneous architectures are placed in one candidate pool, for example, concatenating layers from Qwen and LLaMA, performance drops sharply because layer dimensions and distributions diverge; existing alignment only partially addresses low-order statistics, deeper semantic shifts remain, and interlayer semantics are not preserved. Without MLP-based alignment under cross-architecture settings, the merged model fails on target tasks. In summary, merging fully heterogeneous architectures is promising yet unsolved; our results underscore its necessity and suggest routes toward general architecture-level model merging across heterogeneous models.

In the future, we plan to focus on several theoretical directions for architecture-level model merging: (i) investigating nonlinear or piecewise mode connectivity under structural variations to reveal the reachability and transition paths in parameter space; (ii) systematically analyzing how mechanisms such as glue layers guarantee intermediate representation consistency, based on representation alignment and information bottleneck theories; and (iii) quantifying the effects of structural merging on generalization error and model expressiveness, as well as exploring how structural changes affect adaptability and transferability through task transfer and inductive bias analysis.