Behavior Knowledge Merge in Reinforced Agentic Models

Anonymous ACL submission

Abstract

Reinforcement learning (RL) is central to post-training, particularly for agentic models that require specialized reasoning behaviors. In this setting, model merging offers a practical mechanism for integrating multiple RL-trained agents from different tasks into a single generalist model. However, existing merging methods are designed for supervised fine-tuning (SFT), and they are suboptimal to preserve task-specific capabilities on RL-trained agentic models. The root is a task-vector mismatch between RL and SFT: on-policy RL induces task vectors that are highly sparse and heterogeneous, whereas SFT-style merging implicitly assumes dense and globally comparable task vectors. When standard global averaging is applied under this mismatch, RL's non-overlapping task vectors that encode critical task-specific behaviors are reduced and parameter updates are diluted. To address this issue, we propose **R**einforced **A**gent **M**erging (**RAM**), a distribution-aware merging framework explicitly designed for RL-trained agentic models. RAM disentangles shared and task-specific unique parameter updates, averaging shared components while selectively preserving and rescaling unique ones to counteract parameter update dilution. Experiments across multiple agent domains and model architectures demonstrate that RAM consistently retains specialized agent capabilities and outperforms existing baselines, achieving state-of-the-art performance. Code is available[1].

1 Introduction

Post-training has become a cornerstone for aligning large language models (LLMs) to diverse domains (Wei et al., 2021; Ouyang et al., 2022; Jaech et al., 2024). While specializing a model for a single task is effective, real-world applications typically require a single model to possess multi-task capabilities. Traditionally, these capabilities are obtained by mixing offline datasets and performing joint training. However, training such a generalist model from scratch is computationally expensive, and maintaining separate checkpoints for each task is storage-inefficient. Consequently, model merging, which combines multiple task-specific models fine-tuned from the same base model into a unified model, has emerged as a widely adopted solution (Ilharco et al., 2023; Yadav et al., 2023; Matena and Raffel, 2022; Yu et al., 2024). It offers significant advantages, including data privacy preservation, the elimination of additional training costs, and minimal sacrificed performance.

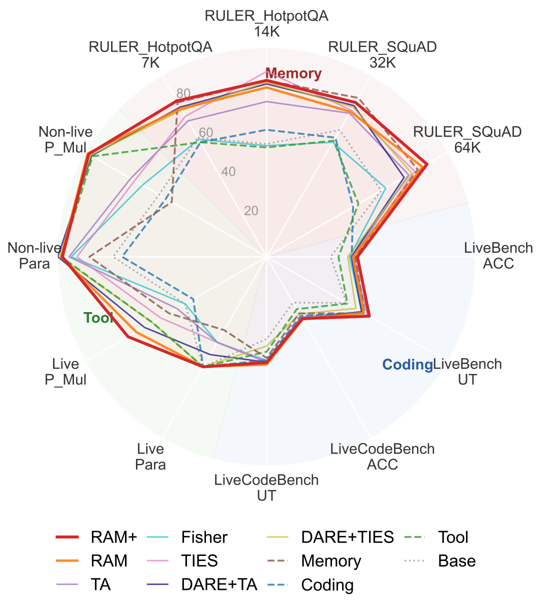

Figure 1: Performance comparison of RAM/RAM+ and baselines on 12 tasks across three agent domains. Our method achieves the best average performance and secures SOTA results on 9 out of 12 tasks, surpassing even the original specialized agents (Coding, Memory, Tool).

Recently, post-training has shifted from supervised fine-tuning (SFT) to reinforcement learning (RL) (Team et al., 2025a; Guo et al., 2025), particularly for agentic models with strong reasoning

[1] Code is available at RAM.

capabilities. This shift fundamentally changes the role of model merging. In the SFT scenarios, multi-task capabilities can still be easily obtained by performing joint training. In contrast, joint multi-task training under on-policy RL is impractical in real-world systems, as it requires parallel task-specific environments and reward models to ensure on-policy training. Consequently, model merging becomes merely convenient solution in the RL setting. A representative example from large-scale industrial agent systems is UI-TARS2 (Wang et al., 2025a), which trains specialized vertical agents in isolated environments via RL, and subsequently merges them into a unified generalist agent. This paradigm reflects a practical compromise: specialization through RL, followed by post-hoc integration through model merging.

However, directly applying existing model merging methods to RL-trained agents leads to performance degradation. Most prior approaches, including Task Arithmetic (Ilharco et al., 2023), TIES-Merging (Yadav et al., 2023), and DARE (Yu et al., 2024), are developed under the assumption of SFT parameter updates, or task vectors, and are therefore mismatched to the RL setting. Unlike SFT, which typically induces dense and redundant parameter updates (Chu et al., 2025; Shenfeld et al., 2025), on-policy RL produces highly sparse and often disjoint task vector distributions, shaped by task-specific reward signals and RL objectives that target narrow behaviors. When such sparse updates are globally averaged during merging, task-specific unique parameter updates are divided by the number of models, resulting in signal dilution, which degrades task-specific behavior knowledge.

To address this mismatch, we propose **R**einforced **A**gent **M**erging (**RAM**), a distribution-aware merging framework designed specifically for RL-trained agents. RAM explicitly disentangles shared and task-specific unique regions of task vectors obtained via RL processes, averaging shared regions to preserve common capabilities while selectively preserving and rescaling unique regions to prevent signal dilution. By maintaining RL task vectors during merging, RAM enables specialized behavior knowledge from each model to coexist within the merged unified model. Our contributions for reinforced agent merging are:

- We identify the mismatch between merging RL-trained agentic models and existing merging method for SFT-trained models, which is signal dilution in heterogeneous distribution of sparse parameter updates.
- We propose a distribution-aware merging method that treats shared and task-unique parameter updates differently, averaging the former while preserving and rescaling the latter based on distribution, avoiding signal dilution and compensate for performance degradation.
- Extensive experiments demonstrate that RAM not only outperforms existing merging methods across diverse architectures and domains but also unlocks synergistic potential among agents. Unified RAM model achieves performance superior to that of individual specialized agents on their domain tasks.

2 Related Works

Post-training Agents with RL RL has recently emerged as a pivotal paradigm for enhancing the reasoning capabilities of Large Language Models (LLMs) (Jaech et al., 2024; Shao et al., 2024; Team et al., 2025a). Several general-purpose algorithms, such as PPO (Schulman et al., 2017), GRPO (Guo et al., 2025), and DAPO (Yu et al., 2025b), have been developed to support this direction. Beyond general reasoning, RL is extensively applied to specialize agents for diverse domains. In coding, methods like CURE (Wang et al., 2025b), SWE-RL (Wei et al., 2025b), and GLM-4.5 (Zeng et al., 2025) have demonstrated significant success. For memory extension, MemAgent (Yu et al., 2025a) and various cache-based approaches (Shi et al., 2025) optimize long-context handling. Furthermore, RL has been instrumental in developing tool-integrated reasoning agents, such as AutoTIR (Wei et al., 2025a) and ToolRL (Qian et al., 2025), as well as search-augmented agents (Jin et al., 2025b; Team et al., 2025b; Sun et al., 2025) and computer-use agents (Wang et al., 2025a; Ye et al., 2025; Wanyan et al., 2025).

Model Merging for LLMs Model merging has demonstrated superiority in multi-task learning by synthesizing different task-specific models into a single entity without additional training (Ilharco et al., 2023; Jin et al., 2025a; Matena and Raffel, 2022). Techniques such as Task Arithmetic (Ilharco et al., 2023), TIES-Merging (Yadav et al., 2023), and DARE (Yu et al., 2024) have been successfully validated for merging SFT models across multiple tasks. Beyond multi-task integration, merging

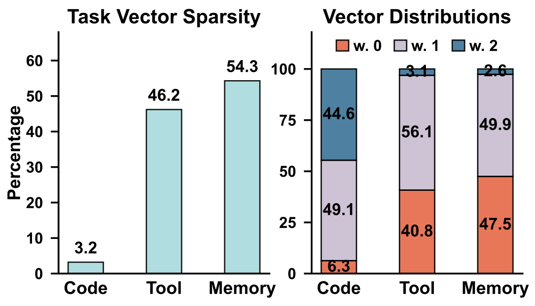

Figure 2: **Left**: Sparsity of task vectors varies between agent models. **Right**: Non-zero elements distributions of task vector varies on the number of overlaps with other task vectors.

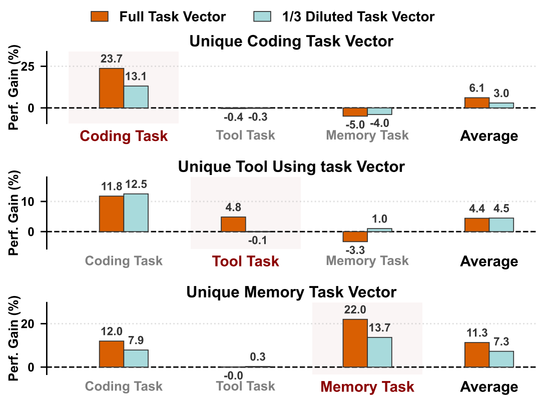

Figure 3: The performance gain (%) of merging unique regions of reinforced task vectors across domains.

has also been employed to mitigate model collapse (IMM (Yuan et al., 2025b)) and enhance reasoning efficiency (Wu et al., 2025). While recent works like UI-TARS2 (Wang et al., 2025a) attempt to merge RL-trained agentic models, they rely on simple weight interpolation, which remains suboptimal for this regime. Distinguished from prior studies, our work is the first to systematically characterize the unique behaviors of task vectors induced by RL and to design a tailored merging strategy, that aligns their specific heterogeneity.

3 Reinforced Task Vector Behaviors

In this section, we characterize the properties of task vectors derived from reinforcement learning and explain how these properties render existing model merging methods suboptimal.

3.1 Preliminaries and Settings

Task Vectors Task vector is the set of parameter updates for a specific task. Let $\theta_{\text{pre}} \in \mathbb{R}^d$ denote the parameters of a pre-trained base model. We consider N different tasks, where each task $t \in \{1, \ldots, N\}$ yields a fine-tuned model θ_t. The task vector for task t is defined as $\boldsymbol{\tau}_t = \theta_t - \theta_{\text{pre}}$. In this study, we specifically examine task vectors induced by RL fine-tuning, referring to them as **Reinforced Task Vectors**. The objective of model merging is to synthesize a single merged task vector $\boldsymbol{\tau}_{\text{merged}}$ to construct a final model $\theta_{\text{merged}} = \theta_{\text{pre}} + \boldsymbol{\tau}_{\text{merged}}$.

Vector Sparsity We define the task vector sparsity of a model θ_t relative to the base model as $\text{sparsity}(\theta_t, \theta_{\text{pre}}) := 1 - \|\theta_t - \theta_{\text{pre}}\|_0 / d$, where $\|\cdot\|_0$ represents the number of non-zero elements and d is the total parameter dimension. Following standard practice[2] (Mukherjee et al., 2025; Paszke et al., 2019), we consider two parameters equal (implying a zero element in the task vector) if their absolute difference is $\leq 10^{-5}$.

Reinforced Agentic Models To investigate these properties, we select three representative agentic models trained via RL to specialize in coding, tool-use, and long-context memory as follows. **CURE** (Wang et al., 2025b): A coding agent that co-evolves with a unit tester to enhance code generation capabilities. **ToolRL** (Qian et al., 2025): A reasoning agent optimized for general-purpose tool selection and application. **MemAgent** (Yu et al., 2025a): An agent optimized for long-context tasks, with a workflow for extended memory retention. All reinforced agents are initialized from the same base model, Qwen2.5-7B-Instruct (Yang et al., 2024). To evaluate their specialized agentic capabilities and reinforced task vector behaviors, we employ benchmarks across the coding, tool-use, and memory domains; detailed evaluation settings are provided in Section 5.1.

3.2 Heterogeneity in Reinforced Task Vectors

In this section, we have the two key observations by analyzing the distribution of reinforced task vectors in parameter space: 1) Reinforced task vectors are sparsely distributed in parameter space, with notable heterogeneous sparsity and distributions. 2) Heterogeneity causes shared and unique regions of reinforced task vectors. unique regions are critical for improving corresponding agentic domain performances, and exert little negative interference on other domains' performances.

[2]PyTorch uses 10^{-5} as the default tolerance for gradient checking (refer to PyTorch Documentation).

Heterogeneity in Sparsity and Distribution Recent work (Mukherjee et al., 2025; Yuan et al., 2025a) highlights the inherent sparsity of RL updates: unlike SFT, which updates global parameters, RL tends to fine-tune specific sub-networks. Our analysis confirms this and further unveils heterogeneous patterns in both sparsity and spatial distribution of them.

First, the percentage of non-zero elements, or the sparsity levels of reinforced task vectors vary drastically. As shown in Figure 2 (left), the coding agent exhibits extreme sparsity, modifying only 3.2% of parameters. In contrast, agents optimized for tool use and long-context memory induce significantly denser updates, affecting 46.2% and 54.3% of the parameter space, respectively.

Second, these non-zero elements are distributed across disparate regions, creating different spatial overlap patterns. We categorize parameters (task vector elements) based on whether they are updated by a single agent (unique) or multiple agents (shared). Figure 2 (right) reveals that the coding, tool-use, and memory agents concentrate different fractions of their updates in unique, non-overlapping regions (6.3%, 40.8%, and 47.5% of their respective non-zero elements). Analysis in Appendix C.1 confirms that this heterogeneity generalizes to other reinforced agents with different model architectures and specialized domains.

The Role of Heterogeneity We further investigate the impact of these heterogeneous, unique regions on both in-domain and out-of-domain agentic tasks. In our experiments, we isolate the unique component of a single task vector, merge it into the base model, and evaluate performance across all domains. Results in Figure 3 demonstrate that unique regions cause almost no negative interference on out-of-domain tasks (occasionally yielding improvements) while driving significant gains on the corresponding in-domain tasks. When we intentionally dilute the magnitude of these unique vectors to $1/N(N = 3)$ of their original value, in-domain performance drops sharply, verifying the positive contribution of these unique components.

Signal Dilution Previous heterogeneity analysis uncovers the root cause of performance degradation when using previous SOTA merging methods on RL models. Existing methods typically employ element-wise averaging or variants (e.g., $\boldsymbol{\tau}_{\text{merged}} = \frac{1}{N} \sum \boldsymbol{\tau}_i$). While averaging is beneficial for shared regions of task vectors by balancing multi-task performance, it is detrimental to task-specific, unique regions in RL scenarios. For a unique parameter updates for task t, averaging with $N - 1$ zero-valued updates effectively scales its magnitude by $1/N$. This operation dilutes the learned signal without providing any balancing benefit, as these regions hardly interfere with out-of-domain tasks. We term this phenomenon *Signal Dilution*. Figure 3 illustrates that this dilution (simulated with $N = 3$) causes significant performance regression. The prevalence of unique task vector components in RL agents therefore necessitates a merging strategy capable of disentangling these regions to prevent signal dilution. We provide a detailed analysis of signal dilution in each existing merging strategy in Appendix D.

4 Merging Reinforced Agentic Models

Our analysis in Section 3.2 demonstrates that reinforced task vectors are inherently sparse and heterogeneous. While task vectors or parameters updated by multiple agents (shared regions) benefit from averaging to stabilize the consensus direction, parameter updated by a single agent (unique regions) suffer from *Signal Dilution* when standard averaging is applied. To address this, we propose **R**einforced **A**gent **M**erging (**RAM**) illustrated in Figure 4, a method that explicitly disentangles these regions based on distribution statistics of task vectors. RAM applies selective merging strategies: it averages shared parameters to absorb unified multi-task capabilities while preserving the full magnitude of unique parameters to prevent signal dilution. Additionally, we introduce a distribution-aware rescaling mechanism to further amplify unique task capabilities.

4.1 Probing Vector Distribution

First, we probe the active updated parameters for each reinforced task vector $\boldsymbol{\tau}_t$. Using the threshold established in Section 3.2 (e.g., $\epsilon = 10^{-5}$), we compute a binary mask $\mathbf{m}_t \in \{0, 1\}^d$ for task t:

$$m_{t,i} = \mathbb{I}(|\tau_{t,i}| > \epsilon), \tag{1}$$

where i indexes the parameter dimensions and $\mathbb{I}(\cdot)$ is the indicator function. We then define the overlap count vector $\mathbf{c} = \sum_{t=1}^{N} \mathbf{m}_t$, where $c_i \in \{0, \ldots, N\}$ represents the number of agents that actively update the i-th parameter.

For any given task t, the set of total updated parameters is partitioned into two disjoint subsets:

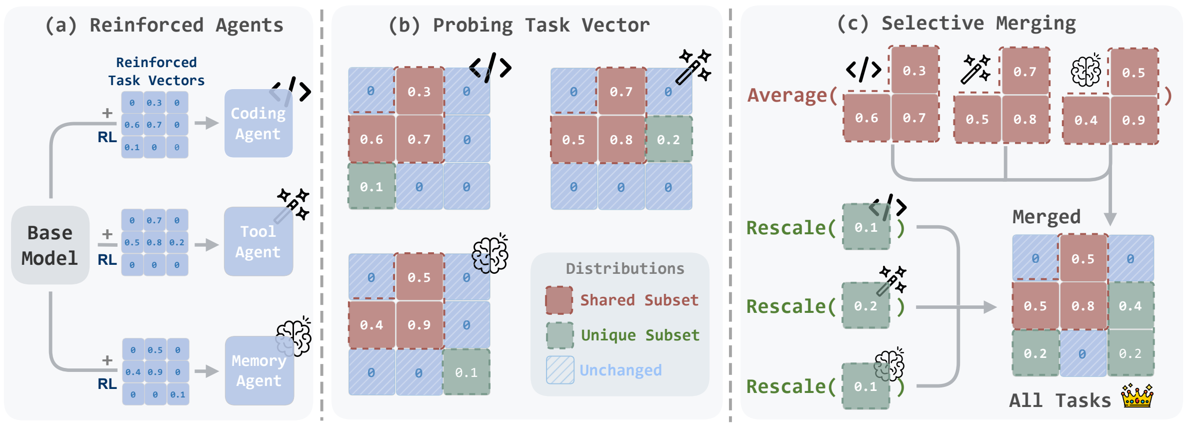

Figure 4: **Method Overview.** **(a)** A base model is trained via RL to different agents, we track the distributions of obtained reinforced task vectors. **(b)** Probing the distribution of task vectors to shared, unique, unchanged sets. **(c)** selective merging task vectors by averaging shared set and rescaling unique set to the base model.

the **Shared** subset (where $c_i \geq 2$) and the **Unique** subset (where $c_i = 1$). To quantify the structural distribution of the agent's updates, we define the *Overlap-Unique Ratio* ρ_t:

$$\rho_t = \frac{\sum_{i:c_i \geq 2} m_{t,i}}{\sum_{i:c_i=1} m_{t,i}}. \tag{2}$$

Here, the numerator represents the count of shared parameters, and the denominator represents the count of unique parameters. Since the sum of these two components constitutes the total updated parameter volume, a higher ρ_t indicates that model learns task t largely within the shared subspace.

4.2 Rescaling Unique Regions

Reinforced task vectors with high Overlap-Unique Ratios (ρ_t) are likely to suffer greater degradation of task capabilities when their substantial shared regions are averaged with other vectors. Therefore, we proportionally rescale the unique regions of such parts of task vectors to compensate for the performance loss incurred in the shared regions. To achieve this, we calculate a task-specific scaling factor λ_t derived from a functional equivalence hypothesis as follows.

Let Δf_t denote the functional gain, or task performance gain, induced by the task vector $\boldsymbol{\tau}_t$. We decompose the total gain into shared and unique components based on the overlap statistics defined in Section 4.1:

$$\Delta f_t = \mathcal{C}_{\text{shared}} + \mathcal{C}_{\text{unique}}, \tag{3}$$

where the gains are modeled as the task vector element $\tau_{t,k}$ weighted by local sensitivity g_i, which indicates the contribution coefficient mapping the task vector element to performance gain. The performance gains are therefore represented as: $\mathcal{C}_{\text{shared}} = \sum_{i:c_i \geq 2} g_i \tau_{t,i} m_{t,i}, \mathcal{C}_{\text{unique}} = \sum_{i:c_i=1} g_i \tau_{t,i} m_{t,i}$. In the merged task vectors, elements in the shared regions are averaged across vectors, and the corresponding task performances are degraded. We model this as a contraction of the effective signal by a coefficient $1 - r \in (0, 1)$. To counteract this, we rescale the magnitudes of parameters in unique regions and have a new functional gain expression:

$$\Delta \hat{f}_t = (1 - r)\mathcal{C}_{\text{shared}} + \lambda_t \mathcal{C}_{\text{unique}}. \tag{4}$$

We hypothesize that this rescaling operation achieves the *functional equivalence* to have the same performance gain for the task vector τ_t on task t: $\Delta \hat{f}_t \approx \Delta f_t$. Under the simplifying assumption that parameter importance is isotropic ($g_i \tau_{t,i} \approx$ const on average), the ratio of functional contributions $\mathcal{C}_{\text{shared}} / \mathcal{C}_{\text{unique}}$ reduces to the ratio of parameter counts defined in Eq. 2. Then by solving *functional equivalence* as our objective, the required amplification satisfies:

$$\lambda_t - 1 = r \frac{\mathcal{C}_{\text{shared}}}{\mathcal{C}_{\text{unique}}} \approx r \frac{\sum_{i:c_i \geq 2} m_{t,i}}{\sum_{i:c_i=1} m_{t,i}} = r \rho_t, \tag{5}$$

which is approximately proportional to the Overlap-Unique Ratio ρ_t. This suggests that tasks with higher overlap require stronger compensation in their unique parameter subspace to counteract the degradation induced by averaging. However, directly instantiating this proportional relationship may lead to numerical instability when ρ_t is large. To balance signal compensation with stability, we employ a clipped linear scaling rule:

$$\lambda_t = 1 + r \cdot \text{clip}\left(\rho_t, 0, \alpha\right), \tag{6}$$

where r controls the overall amplification strength and α serves as a stable bound. This design preserves the monotonic growth implied by the hypothesis while preventing excessive amplification in high-overlap scenarios.

4.3 Selective Merging

Finally, we construct the merged task vector $\boldsymbol{\tau}_{\text{merged}}$ element-wise. Unlike existing merging methods, which effectively divide unique parameters by N (causing signal dilution), our strategy differentiates between shared and unique regions. For each parameter index i, let $\mathcal{T}_i = \{t \mid m_{t,i} = 1\}$ denote the set of indices of active tasks for that parameter. Note that the cardinality $|\mathcal{T}_i|$ corresponds to the overlap count c_i defined in Section 4.1. The merged element $\tau_{\text{merged},i}$ is computed as:

$$\tau_{\text{merged},i} = \begin{cases} 0 & \text{if } |\mathcal{T}_i| = 0, \\ \lambda_t \cdot \tau_{t,i} & \text{if } \mathcal{T}_i = \{t\}, \\ \frac{1}{|\mathcal{T}_i|} \sum_{t \in \mathcal{T}_i} \tau_{t,i} & \text{if } |\mathcal{T}_i| \geq 2. \end{cases} \quad (7)$$

This selective strategy ensures that: 1) *Shared Knowledge* ($|\mathcal{T}_i| \geq 2$) is averaged to balance multi-task capabilities. 2) *Task-Specific Knowledge* ($|\mathcal{T}_i| = 1$) is completely preserved and amplified by λ_t to compensate for the contraction of the effective signal in shared regions, explicitly targeting functional equivalence. 3) *No Knowledge* ($|\mathcal{T}_i| = 0$) is set to zero to filter out insignificant parameter fluctuations and ensure that the base model's general capabilities remain undisturbed.

5 Experiments

5.1 Setup

Baselines We categorize our baselines into two groups: original specialized agent models and established model merging techniques. **(i) Original Agent Models:** We utilize **Qwen2.5-7B-Instruct** (Yang et al., 2024) as the shared base model. The task-specific reinforced agents include: **CURE** (Wang et al., 2025b) for coding, **ToolRL** (Qian et al., 2025) for tool-use, and **MemAgent** (Yu et al., 2025a) for long-context memory. **(ii) Merging Methods:** We compare our approach against prominent merging strategies: **Task Arithmetic** (Ilharco et al., 2023), which linearly combines task vectors; **Fisher Merging** (Matena and Raffel, 2022), which weighs parameters based on Fisher information; **TIES-Merging** (Yadav et al., 2023), which mitigates parameter interference through trimming and sign consensus; and **DARE** (Yu et al., 2024), which randomly drops and rescales the parameters. Following standard practice, we combine DARE with Task Arithmetic and TIES for evaluation. We introduce more details about baselines in Appendix D.

Evaluations We evaluate the models across three critical agentic domains: coding, tool-use, and long-context memory. For **Coding**, we measure generated code pass accuracy (ACC) and unit test pass accuracy (UT) on the LiveBench (White et al., 2025) and LiveCodeBench (Jain et al., 2024) benchmarks. For **Tool Use**, we utilize the Berkeley Function Call Leaderboard (BFCL) (Patil et al., 2025), specifically reporting results on the Live/Non-Live Parallel (Para) and Parallel Multiple (P_Mul) subsets. For **Long-Context Memory**, following (Yu et al., 2025a), we employ the RULER benchmark (Hsieh et al., 2024) to assess performance on long-context tasks, including RULER (Hsieh et al., 2024) HotpotQA and SQuAD. Additional datasets and detailed evaluation criteria are provided in Appendix B.

Implementations For hyperparameters, we set $r = 0.1$ and $\alpha = 2.0$. We denote our method without task-specific rescaling as **RAM** and with rescaling as **RAM+**. RAM is the special case of RAM+ when $r = 0$. We provide the details of pseudocode and agents for merging in Appendix A.

5.2 RAM is a Better Fit for Reinforced Models

Table 1 demonstrates that both RAM and RAM+ consistently outperform all baselines, establishing a new SOTA. Specifically, RAM achieves an average score of 64.82 across all tasks, surpassing the strongest baseline DARE (63.33). Building on this foundation, RAM+ further pushes the boundary to 66.55 after rescaling unique regions, inducing extra improvements where the merged generalist exceeds the capabilities of specialized task agents on most of evaluations. For instance, in the **Coding** domain, RAM+ surpasses the specialist Coding agent on LiveBench and LiveCodeBench, suggesting that reasoning signals from other tasks enhance coding precision. This superiority extends to **Tool Use**, where RAM+ significantly outperforms the Tool agent in complex parallel scenarios (Live P_Mul: 70.83 vs. 58.33), and to **Long-Context Memory**, where it achieves global optimal performance on SQuAD 64k (82.03), beating the dedicated Memory agent (77.34).

Model	**Coding**				**Tool Using**				**Memory**				**Avg**
	LiveBench		LiveCodeBench		Live		Non-Live		HotpotQA		SQuAD		
	ACC	UT	ACC	UT	Para	P_Mul	Para	P_Mul	7K	14K	32K	64K	
						Base and Task Models							
Base	28.35	40.87	23.43	36.42	**56.25**	41.67	68.00	55.00	60.94	50.00	64.84	58.59	48.70
CURE (Coding)	37.70	49.27	30.23	45.76	**56.25**	37.50	64.00	51.50	58.59	56.25	60.94	44.22	49.35
ToolRL (Tool)	31.84	41.36	26.76	42.05	**56.25**	58.33	91.00	89.00	58.59	48.44	59.38	46.95	54.16
MemAgent (Memory)	39.25	50.12	28.92	44.80	37.50	50.00	78.50	48.50	78.91	78.12	81.25	77.34	57.77
						Merged Models							
TA	38.09	51.62	31.95	46.69	43.75	45.83	87.50	69.50	69.53	68.75	73.44	72.66	58.28
Fisher	36.72	48.73	30.87	45.89	43.75	41.67	86.5	63.5	60.06	49.22	58.59	60.94	52.20
TIES	39.25	49.88	30.63	46.32	43.75	54.17	84.00	67.50	71.88	**82.03**	75.00	75.78	60.02
DARE+TA	37.50	48.60	31.95	46.69	50.00	62.50	**92.50**	89.50	76.56	76.56	77.34	70.31	63.33
DARE+TIES	35.93	45.66	29.26	39.53	**56.25**	58.33	91.50	90.00	75.00	77.34	76.56	74.22	62.47
RAM	38.28	49.71	**31.96**	**47.72**	**56.25**	66.67	91.00	**91.50**	75.78	75.00	74.22	79.69	64.82
RAM+	**40.23**	**52.57**	31.60	46.84	**56.25**	**70.83**	90.50	91.00	**79.69**	78.13	**78.91**	**82.03**	**66.55**

Table 1: **Main results of agent merging.** We evaluate the capabilities across three domains: Coding (LiveBench, LiveCodeBench), Tool Use (Live, Non-Live), and Memory (HotpotQA, SQuAD). **Bold** and underlined values denote the best and second-best performance among *merged models*, respectively. Cells highlighted in red indicate the best performance across *all evaluated models*, including the specialized Task Models.

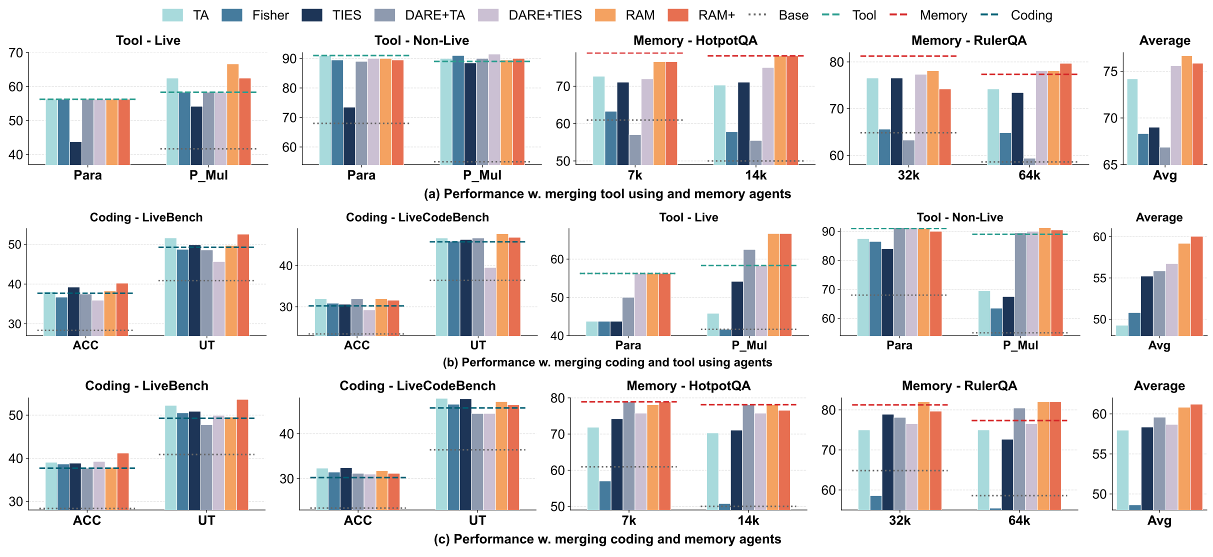

Figure 5: The performances of merging two agents across domains.

5.3 Extending Model Combinations

To evaluate the effectiveness of RAM beyond tri-agent merging, we extend pairwise agent merging experiments with more model combinations across Tool+Memory, Coding+Tool, and Coding+Memory scenarios, as illustrated in Figure 5 (details are provided in Appendix C). Across all three combinations, RAM/RAM+ consistently achieves the highest average performance, demonstrating superior robustness on various combinations compared to baselines. Specifically, in the Coding+Tool setting, RAM+ attains an average score of 60.04, significantly outperforming the strongest baseline DARE+TIES (56.74) and effectively bridging the capability gap that traditional methods like Task Arithmetic and TIES fail to address due to signal dilution. Similarly, in the Tool+Memory and Coding+Memory scenarios, RAM+ maintains dominant performance with average scores of 75.86 and 61.21 respectively, confirming that RAM can be successful in multiple agent combinations.

5.4 Ablation Study

We ablate our method to RAM ($r = 0$) and RAM+, and investigate the sensitivity of our proposed method to the scaling factor r. Table 2 presents the ablation results with r varying from 0.00 to 0.20. Note that when $r = 0$, the method represents RAM. As observed, the model's overall per-

r	Code				Tool				Memory				Avg
	LiveBench		LiveCodeBench		Live		Non-Live		HotpotQA		RulerQA		
	ACC	UT	ACC	UT	Para	P_Mul	Para	P_Mul	7k	14k	32k	64k	
0.00	38.28	49.71	31.96	**47.72**	56.25	66.67	91.00	**91.50**	75.78	75.00	74.22	79.69	64.82
0.05	37.70	50.49	30.63	45.59	**62.50**	62.50	**92.00**	**91.50**	75.78	**79.69**	75.00	**82.03**	65.45
0.10	**40.23**	**52.57**	31.60	46.84	56.25	**70.83**	90.50	91.00	**79.69**	78.13	78.91	**82.03**	**66.55**
0.15	38.67	49.85	31.41	46.53	**62.50**	66.67	89.50	**91.50**	78.12	**79.69**	79.69	75.78	65.83
0.20	39.45	50.36	**32.58**	47.54	56.25	62.50	91.00	90.50	78.12	78.12	**80.47**	77.34	65.35

Table 2: **Ablation Study. Bold** and underlined values denote the best and second-best performance.

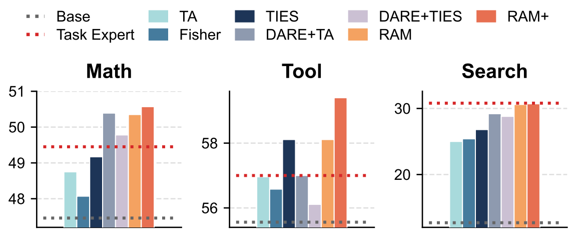

Figure 6: Merging results for RL agents trained from Llama3.2-3B-Instruction base model.

formance (Avg) exhibits a trend of initially increasing and then decreasing. The performance peaks at $r = 0.10$, achieving the highest average score of 66.55. Specifically, setting $r = 0.10$ (RAM+) yields the best or second-best results across the majority of metrics, particularly showing significant gains in LiveBench (Coding) and HotpotQA (Memory) compared to RAM ($r = 0.00$). However, further increasing r beyond 0.10 leads to diminishing returns, with the average score dropping to 65.35 at $r = 0.20$. This suggests that while a moderate scaling factor effectively enhances task-specific capabilities, an excessively large r may disrupt the general knowledge of the merged model.

5.5 Extending Architecture and Domains

Besides the agents trained from Qwen architecture, we extend the experiment to Llama architecture and additional domains. We choose Llama3.2-3B (Grattafiori et al., 2024) as the base model, and choose models trained from it via RL: search agent ZeroSearch (Sun et al., 2025), math reasoning agent (Zhao et al., 2025), and tool-using agent ToolRL (Qian et al., 2025). The evaluation details are provided in Appendix B.4. Figure 6 illustrates that, consistent with our findings on the Qwen, RAM and RAM+ demonstrate superior performance across multiple agentic domains, consistently outperforming baselines. Notably, they achieve positive synergy in both Math and Tool domains, where the merged generalist surpasses the performance of the original specialized agents. For instance, in the Tool domain, RAM+ exhibits a significant margin over the specialist, suggesting that reasoning capabilities from Math and Search agents synergize to enhance tool-use. In the Search domain, RAM/RAM+ successfully retain original capability, whereas baselines show notable regression. These results confirm that the heterogeneity of reinforced task vectors is a general property, and RAM effectively addresses this by preserving task-specific specialized knowledge independent of model scale and architecture.

Additional Experiments Besides the above experiments, we further provide the instruction following evaluation to assess the forgetting after merging in Appendix C.2; Merging efficiency comparison in Appendix C.3; Evaluation on the additional tasks in Appendix C.5; Additional rescaling strategy performance in Appendix E.

6 Conclusion

In this work, we address the critical challenge of merging agents fine-tuned via RL, identifying a fundamental mismatch between standard merging techniques designed for dense SFT updates and the sparse, heterogeneous nature of on-policy RL task vectors. We demonstrate that treating global task vectors equally in previous methods in this setting leads to signal dilution of task-specific capabilities. To bridge this gap, we propose Reinforced Agent Merging (RAM), a framework that explicitly disentangles shared and unique parameter update regions and applies a distribution-based rescaling strategy to preserve specialized behaviors. Extensive evaluations across multiple agentic domains and model architectures show that RAM significantly outperforms existing baselines, achieving SOTA results and surpassing the original specialists as a unified generalist on most tasks.

Limitations

While Reinforced Agent Merging (RAM) effectively mitigates signal dilution for RL-trained agents, our current study has certain boundaries. First, our experiments focus on merging a common number of agents; as N scales significantly, the probability of parameter collision in the shared subspace increases, potentially requiring more complex conflict resolution strategies beyond simple averaging. Second, the derivation of our rescaling factor relies on an isotropic assumption of parameter importance, which, while empirically robust, does not explicitly account for element-wise curvature information that could offer finer-grained control at a higher computational cost. Third, although we identified a default hyperparameter configuration that generalizes well across Qwen and Llama architectures, optimal performance on agents trained with fundamentally different RL algorithms or modalities may require task-specific tuning. Finally, our evaluation is primarily conducted on 3B and 7B parameter models; verifying whether the sparsity hypothesis and RAM's efficacy persist in massive-scale models (70B+) remains an open question for future research.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. 2025. SFT memorizes, RL generalizes: A comparative study of foundation model post-training. In *Forty-second International Conference on Machine Learning*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. The language model evaluation harness.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*.

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

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025a. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan O Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025b. Search-r1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Second Conference on Language Modeling*.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, and 1 others. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, and 1 others. 2022. Competition-level code generation with alphacode. *Science*.

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

Sagnik Mukherjee, Lifan Yuan, Dilek Hakkani-Tur, and Hao Peng. 2025. Reinforcement learning finetunes small subnetworks in large language models. *arXiv preprint arXiv:2505.11711*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, and 1 others. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*.

Guilherme Penedo, Anton Lozhkov, Hynek Kydlíček, Loubna Ben Allal, Edward Beeching, Agustín Piqueres Lajarín, Quentin Gallouédec, Nathan Habib, Lewis Tunstall, and Leandro von Werra. 2025. Codeforces. https://huggingface.co/datasets/open-r1/codeforces.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru WANG, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025. ToolRL: Reward is all tool learning needs. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Idan Shenfeld, Jyothish Pari, and Pulkit Agrawal. 2025. RL's razor: Why on-policy reinforcement learning forgets less. In *AI That Keeps Up: NeurIPS 2025 Workshop on Continual and Compatible Foundation Model Updates*.

Dachuan Shi, Yonggan Fu, Xiangchi Yuan, Zhongzhi Yu, Haoran You, Sixu Li, Xin Dong, Jan Kautz, Pavlo Molchanov, and Yingyan Celine Lin. 2025. Lacache: Ladder-shaped kv caching for efficient long-context modeling of large language models. In *Forty-second International Conference on Machine Learning*.

Hao Sun, Zile Qiao, Jiayan Guo, Xuanbo Fan, Yingyan Hou, Yong Jiang, Pengjun Xie, Fei Huang, and Yan Zhang. 2025. Zerosearch: Incentivize the search capability of llms without searching. *arXiv preprint arXiv:2505.04588*.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, and 1 others. 2025a. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*.

Tongyi DeepResearch Team, Baixuan Li, Bo Zhang, Dingchu Zhang, Fei Huang, Guangyu Li, Guoxin Chen, Huifeng Yin, Jialong Wu, Jingren Zhou, and 1 others. 2025b. Tongyi deepresearch technical report. *arXiv preprint arXiv:2510.24701*.

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, and 1 others. 2025a. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*.

Yinjie Wang, Ling Yang, Ye Tian, Ke Shen, and Mengdi Wang. 2025b. CURE: Co-evolving coders and unit testers via reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Yuyang Wanyan, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Jiabo Ye, Yutong Kou, Ming Yan, Fei Huang, Xiaoshan Yang, and 1 others. 2025. Look before you leap: A gui-critic-r1 model for pre-operative error diagnosis in gui automation. *arXiv preprint arXiv:2506.04614*.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.

Yifan Wei, Xiaoyan Yu, Yixuan Weng, Tengfei Pan, Angsheng Li, and Li Du. 2025a. Autotir: Autonomous tools integrated reasoning via reinforcement learning. *arXiv preprint arXiv:2507.21836*.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I Wang. 2025b. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution. *arXiv preprint arXiv:2502.18449*.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh-Agrawal, Sandeep Singh Sandha, Siddartha Venkat Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. 2025. Livebench: A challenging, contamination-limited LLM benchmark. In *The Thirteenth International Conference on Learning Representations*.

Taiqiang Wu, Runming Yang, Tao Liu, Jiahao Wang, and Ngai Wong. 2025. Revisiting model interpolation for efficient reasoning. *arXiv preprint arXiv:2510.10977*.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. 2023. Resolving interference when merging models. *arXiv preprint arXiv:2306.01708*, 1.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 40 others. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, and 1 others. 2025. Mobile-agent-v3: Fundamental agents for gui automation. *arXiv preprint arXiv:2508.15144*.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and 1 others. 2025a. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent. *arXiv preprint arXiv:2507.02259*.

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

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, YuYue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, LingJun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, and 17 others. 2025b. DAPO: An open-source LLM reinforcement learning system at scale. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Xiangchi Yuan, Xiang Chen, Tong Yu, Dachuan Shi, Can Jin, Wenke Lee, and Saayan Mitra. 2025a. Mitigating forgetting between supervised and reinforcement learning yields stronger reasoners. *arXiv preprint arXiv:2510.04454*.

Xiangchi Yuan, Chunhui Zhang, Zheyuan Liu, Dachuan Shi, Leyan Pan, Soroush Vosoughi, and Wenke Lee. 2025b. Superficial self-improved reasoners benefit from model merging. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, and 1 others. 2025. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*.

Xuandong Zhao, Zhewei Kang, Aosong Feng, Sergey Levine, and Dawn Song. 2025. Learning to reason without external rewards. *arXiv preprint arXiv:2505.19590*.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*.

A Implementation Details

A.1 Pseudocode

Here we provide the Pseudocode of RAM in Algorithm 1.

Algorithm 1 Reinforced Agent Merging (RAM)

```
Require: Task vectors $\{\boldsymbol{\tau}_t\}_{t=1}^N$; threshold $\epsilon$; rescale strength $r$; clip bound $\alpha$
Ensure: Merged task vector $\boldsymbol{\tau}_{\text{merged}}$
 1: // Stage 1: Probing Vector Distribution (Sec. 4.1)
 2: for $t = 1$ to $N$ do
 3:     $\mathbf{m}_t \leftarrow \mathbb{I}(|\boldsymbol{\tau}_t| > \epsilon)$    // Compute binary mask (Eq. 1)
 4: end for
 5: $\mathbf{c} \leftarrow \sum_{t=1}^N \mathbf{m}_t$    // Compute overlap count vector
 6: // Stage 2: Rescaling Unique Regions (Sec. 4.2)
 7: for $t = 1$ to $N$ do
 8:     $N_{\text{shared}} \leftarrow \sum_{i:c_i \geq 2} m_{t,i}$
 9:     $N_{\text{unique}} \leftarrow \sum_{i:c_i = 1} m_{t,i}$
10:     $\rho_t \leftarrow N_{\text{shared}} / N_{\text{unique}}$    // Ratio (Eq. 2)
11:     $\lambda_t \leftarrow 1 + r \cdot \text{clip}(\rho_t, 0, \alpha)$    // Task-specific scaling
12: end for
13: // Stage 3: Selective Merging (Sec. 4.3)
14: for each parameter index $i$ do
15:     $\mathcal{T}_i \leftarrow \{t \mid m_{t,i} = 1\}$    // Set of active tasks
16:     if $|\mathcal{T}_i| = 0$ then
17:         $\tau_{\text{merged},i} \leftarrow 0$
18:     else if $|\mathcal{T}_i| = 1$ then
19:         Let $t$ be the unique element in $\mathcal{T}_i$
20:         $\tau_{\text{merged},i} \leftarrow \lambda_t \cdot \tau_{t,i}$    // Preserve & amplify unique
21:     else
22:         $\tau_{\text{merged},i} \leftarrow \frac{1}{|\mathcal{T}_i|} \sum_{t \in \mathcal{T}_i} \tau_{t,i}$    // Average shared
23:     end if
24: end for
25: return $\boldsymbol{\tau}_{\text{merged}}$
```

A.2 Details of Reinforced Task Agents

In this section, we provide the detailed specifications and sources for the reinforced agentic models and base models used in our experiments. All models are publicly available on Hugging Face.

Qwen2.5-7B-Instruction Series: We utilize the following agents initialized from the Qwen2.5-7B-Instruct:

Base Model (Qwen2.5-7B-Instruct) (Yang et al., 2024)
🤗 `Qwen2.5-7B-Instruct`

Coding Agent (CURE) (Wang et al., 2025b)
🤗 `ReasonFlux-Coder-7B`

Tool Agent (ToolRL) (Qian et al., 2025)
🤗 `Qwen2.5-7B-Instruct-ToolRL-grpo-cold`

Memory Agent (MemAgent) (Yu et al., 2025a)
🤗 `RL-MemoryAgent-7B`

Search Agent (ZeroSearch) (Sun et al., 2025)
🤗 `ZeroSearch_google_V2_Qwen2.5_7B_Instruct`

AutoTIR Agent (Wei et al., 2025a)
🤗 `AutoTIR-Qwen2.5-7B-Instruct`

Llama-3.2-3B-Instruction Series To verify generalization across architectures, we utilize the following agents based on Llama-3.2-3B-Instruct:

Base Model (Llama-3.2-3B-Instruct) (Grattafiori et al., 2024)
🤗 `Llama-3.2-3B-Instruct`

Math Agent (GRPO-Math) (Guo et al., 2025)
🤗 `Llama-3.2-3B-Instruct-GRPO-MATH-1EPOCH`

Tool Agent (ToolRL) (Qian et al., 2025)
🤗 `ToolRL-Llama3.2-3B`

Search Agent (ZeroSearch) (Sun et al., 2025)
🤗 `ZeroSearch_google_V2_Llama_3.2_3B_Instruct`

B Evaluation Details

B.1 Coding Evaluation

Following the evaluation setting established in Wang et al. (2025b), we conduct a comprehensive evaluation of coding capabilities across five widely adopted coding benchmarks.

Datasets. We utilize **LiveBench** (White et al., 2025) (standard test set), **MBPP** (Austin et al., 2021) (standard test set), and **LiveCodeBench** (Jain et al., 2024) (Version 2, 511 problems). For competition-level tasks, we include **CodeContests** (Li et al., 2022), filtering for tasks with difficulty ≤ 2 and utilizing a held-out split of 200 examples. Additionally, we use **CodeForces** (Penedo et al., 2025), comprising 500 randomly sampled examples distinct from CodeContests.

Evaluation Protocol. To ensure consistency, all datasets are standardized to the stdio format. Functional inputs from LiveBench, LiveCodeBench, and MBPP are converted by placing variables on separate lines and flattening lists. For verification, we use official ground-truth solutions for CodeContests and MBPP. For the remaining datasets (CodeForces, LiveCodeBench, LiveBench), we utilize high-quality reference solutions generated by QwQ-32B (via Best-of-3 sampling). We employ vLLM (Kwon et al., 2023) for generation. Following standard practices, sampling parameters are set to temperature $T = 1.0$, top-$p = 0.95$. We report performance using pass accuracy, including code pass (ACC) and unit test pass (UT) (Pass@1) and Best-of-N (BoN, N=4) metrics.

B.2 Long-Context Memory Evaluation

To rigorously assess the long-context memory capabilities of our agents, we adopt the evaluation protocol from the RULER benchmark (Hsieh et al., 2024), strictly following the data synthesis configurations established in Yu et al. (2025a). We specifically select RULER-HotpotQA and RULER-SQuAD as our primary benchmarks to evaluate multi-hop reasoning and precise fact retrieval.

Datasets. We utilize the following two tasks adapted for long-context evaluation:

- **RULER-HotpotQA:** This task serves as a robust testbed for *multi-hop reasoning*. In this setup, multiple "golden paragraphs" containing necessary evidence are embedded within a vast amount of distractor content (the haystack). The model must effectively identify and synthesize these scattered pieces of evidence from its memory to correctly answer complex questions.

- **RULER-SQuAD:** Adapted from the SQuAD dataset, this task evaluates precise reading comprehension. Ground-truth passages are inserted into long distractor texts, requiring the model to maintain high fidelity to specific facts over extended sequences. This tests the agent's ability to accurately recall specific instructions or details without hallucination.

Evaluation Protocol. Consistent with Yu et al. (2025a), we synthesize test samples with varying context lengths to stress-test memory capacity with different context lengths (8K-128K for SQuAD and 7K-896K for HotPotQA). The primary evaluation metric is the Substring Exact Match (sub_em) of the generated answers. High accuracy in this setting demonstrates that the merged agent successfully retains critical task-specific memory capabilities and can effectively filter out noise (distractors) inherent in long-context processing.

B.3 Tool Use Evaluation

To comprehensively evaluate the tool-use (function calling) capabilities of our agents, we employ the Berkeley Function Calling Leaderboard (BFCL) (Patil et al., 2025), widely recognized as the standard benchmark for assessing LLM agentic behaviors. We specifically use the *Live* and *Non-Live* datasets to measure performance across both real-world and synthetic scenarios.

Datasets. We utilize the following two subsets to assess distinct dimensions of function calling:

- **Non-Live Dataset (Synthetic & Curated):** Derived from BFCL V1, this subset consists of expert-curated synthetic tasks designed to test fundamental logic across various languages (Python, Java, JavaScript) and SQL. It evaluates the model's adherence to precise instructions in controlled environments. The tasks of Non-Live datasets include: `Multiple`, `Parallel`, `Relevance`, `Simple`, `Parallel_multiple` and `Irrelevance`.

- **Live Dataset (Real-World & Crowd-sourced):** Introduced in BFCL V2, this subset comprises user-contributed examples from real-world agent interactions. Unlike the Non-Live set, these samples are diverse and noisy, involving complex APIs with nested parameters. This benchmark specifically challenges the model's robustness in handling ambiguous queries and detecting function irrelevance, including seven tasks: `Multiple`, `Irrelevance`, `Simple_java`, `Simple_javascript`, `Parallel_multiple`, `Parallel` and `Simple_python`.

Evaluation Protocol. To ensure robust evaluation, we utilize the Abstract Syntax Tree (AST) matching method provided by the BFCL framework. Unlike simple string matching, AST evaluation parses generated function calls into syntax trees to structurally verify argument permutations and formatting variations while enforcing strict type correctness. We report accuracy for both Live and Non-Live splits, with a particular focus on the challenging *Parallel* and *Parallel Multiple* categories to demonstrate advanced planning capabilities.

B.4 Evaluation Details for Llama-based Agents

We evaluate agents trained from LLama3.2-3B-Instruction on three domains: math, search, and tool-use. For the math domain, we evaluate the model on GSM8K (Cobbe et al., 2021) and MATH500 (Hendrycks et al., 2021) datasets, provided by LM-Evaluation-Harness (Gao et al., 2024). For the search domain, we follow the evaluation setting provided in ZeroSearch (Sun et al., 2025) and evaluate the agent on NQ (Kwiatkowski et al., 2019) and 2WikiMultiHopQA (Ho et al., 2020). For tool-use, the setting is the same as Section 5.1. We take the average score across tasks for each domain for evaluation.

C Additional Experiments

C.1 Additional Reinforced Task Vector Analysis

To further verify heterogeneity in reinforced task vectors introduced by Section 3.2, we extend the number, domains, and architecture of the reinforced agents.

First, we include extra agents, web search agent ZeroSearch (Sun et al., 2025) and tool-integrated reasoning agent AutoTIR (Wei et al., 2025a), which are both RL-trained from Qwen2.5-Instruct-7B. Figure 7 shows that when including five agents specialized in multiple domains together, the heterogeneity in sparsity and distribution remains significant. Specifically, the sparsity of task vectors spans a wide spectrum, ranging from merely 3.2% for the Code agent to 54.3% for the Memory agent. The overlap analysis further reveals distinct behaviors: the Code agent is highly entangled with others, with 43.7% of its changed parameters shared among 3 or 4 other agents (w. 3-4). In contrast, agents like Tool and Memory maintain higher independence, with unique parameter ratios (w. 0) of 26.1% and 21.9%, respectively.

Second, we extend the experiment to Llama architecture and additional domains. We choose Llama3.2-3B-Instruction (Grattafiori et al., 2024) as the base model, and choose models trained from it via RL: search agent ZeroSearch (Sun et al., 2025), math reasoning agent (Zhao et al., 2025), and tool-using agent ToolRL (Qian et al., 2025). Figure 8 demonstrates that similar heterogeneity in task vectors persists across different model architectures. As shown in the left panel, the sparsity of task vectors varies significantly, ranging from 17.0% for the Math agent to 56.8% for the Tool agent. The overlap distribution (right panel) further highlights this diversity: the Tool agent modifies a large proportion of unique parameters (54.0%), whereas the Math agent shares the majority of its updates with other tasks, with only 24.2% of its modified parameters being unique. This confirms that the diverse characteristics of reinforced task vectors are consistent across different base models and task domains.

C.2 Instruction Following Evaluation

A primary concern in model merging, particularly when combining agents fine-tuned via RL on disparate domains, is the potential degradation of the base model's general instruction following capabilities (i.e., catastrophic forgetting). To rigorously evaluate whether RAM compromises the model's ability to follow general instructions while pursuing task specialization, we conducted evaluations on the **IFEval** (Instruction Following Evaluation) benchmark (Zhou et al., 2023) provided by LM-Evaluation-Harness (Gao et al., 2024). We report results across four metrics: Instruction Accuracy and Prompt Accuracy, under both Loose and Strict evaluation criteria. The results are presented in Table 3. We evaluated two sets of models:

- **Qwen2.5-7B-Instruct:** The primary setting used in the main paper, trained via RL to obtain the Coding, Tool, and Memory agents.
- **Llama-3.2-3B-Instruction:** To test the generalization of our method on a different architecture and scale, we use the fine-tuned Math, Tool, and Search agents based on Llama-3.2-3B-Instruction.

On the Qwen-based agents, RAM not only retains the general capabilities of the Base model but explicitly outperforms it across most metrics (e.g., +1.44 in Loose Instruction Accuracy and +1.56 in Strict Instruction Accuracy). This suggests that the specialized reasoning circuits preserved by RAM's disjoint merging strategy can positively transfer to general instruction following. RAM+ shows a slight trade-off, generally maintaining parity with the base model on instruction-level metrics while incurring minor regressions in prompt-level accuracy. Merging on smaller Llama-based models is inherently more challenging due to limited parameter redundancy. While all merging methods exhibit some regression compared to the Base model, RAM demonstrates superior stability with less forgetting on instruction following. Notably, baseline methods like TIES and DARE+TIES suffer from severe performance collapse, dropping over 10 percentage points (e.g., -11.15 in Loose Instruction Accuracy). Even in strict evaluation, TIES fails to maintain robustness. In contrast, RAM avoids this collapse, showing significantly smaller regressions (approx. 2-3) and proving it is a much safer merging strategy for smaller architectures compared to aggressive trimming methods.

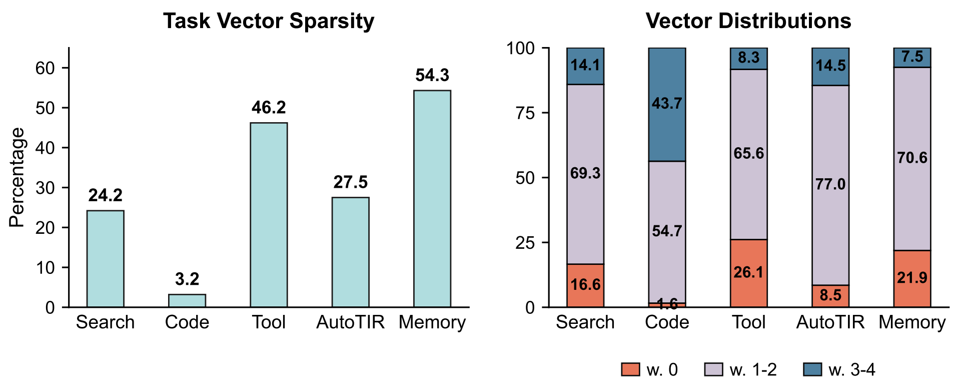

Figure 7: Additional distribution analysis for sparse task vectors.

	Qwen2.5-7B-Instruct				Llama-3.2-3B-Instruction			
	Instruction Acc (%)		Prompt Acc (%)		Instruction Acc (%)		Prompt Acc (%)	
Model	**Loose**	**Strict**	**Loose**	**Strict**	**Loose**	**Strict**	**Loose**	**Strict**
				Base and Task Experts				
Base Model	69.18	64.39	58.96	53.23	**69.90**	**63.31**	**59.15**	**51.20**
Code/Math	73.14 (+3.96)	68.11 (+3.72)	62.48 (+3.52)	58.75 (+5.52)	68.59 (-1.31)	61.99 (-1.32)	57.49 (-1.66)	48.43 (-2.77)
Tool	71.82 (+2.64)	66.07 (+1.68)	61.55 (+2.59)	54.34 (+1.11)	68.11 (-1.79)	61.87 (-1.44)	57.12 (-2.03)	49.35 (-1.85)
Memory/Search	69.78 (+0.60)	66.07 (+1.68)	58.41 (-0.55)	53.60 (+0.37)	67.63 (-2.27)	60.67 (-2.64)	55.45 (-3.70)	46.58 (-4.62)
				Merged Models				
Task Arithmetic (TA)	70.74 (+1.56)	65.71 (+1.32)	58.90 (-0.06)	53.23 (0.00)	68.82 (-1.08)	61.87 (-1.44)	57.12 (-2.03)	49.72 (-1.48)
Fisher	**72.06** (**+2.88**)	**66.91** (**+2.52**)	**61.18** (**+2.22**)	**55.08** (**+1.85**)	67.87 (-2.03)	61.75 (-1.56)	55.82 (-3.33)	49.17 (-2.03)
TIES	71.34 (+2.16)	67.39 (+3.00)	59.52 (+0.56)	55.64 (+2.41)	58.75 (-11.15)	58.87 (-4.44)	45.84 (-13.31)	46.21 (-4.99)
DARE+TA	70.38 (+1.20)	65.95 (+1.56)	58.41 (-0.55)	53.05 (-0.18)	59.11 (-10.79)	54.20 (-9.11)	46.21 (-12.94)	41.04 (-10.16)
DARE+TIES	69.42 (+0.24)	65.11 (+0.72)	57.67 (-1.29)	52.31 (-0.92)	58.63 (-11.27)	53.48 (-9.83)	45.66 (-13.49)	39.56 (-11.64)
RAM (Ours)	70.62 (+1.44)	65.95 (+1.56)	59.70 (+0.74)	53.23 (0.00)	67.39 (-2.51)	61.39 (-1.92)	55.45 (-3.70)	47.87 (-3.33)
RAM+ (Ours)	69.18 (0.00)	64.75 (+0.36)	57.86 (-1.10)	51.94 (-1.29)	66.19 (-3.71)	61.27 (-2.04)	53.97 (-5.18)	47.50 (-3.70)

Table 3: Evaluation of General Capabilities on IFEval benchmark. We report **Instruction Accuracy** and **Prompt Accuracy** under both **Loose** and **Strict** criteria. Values in parentheses denote the absolute change relative to the Base Model. Green values indicate improvement, while orange values indicate regression. RAM demonstrates superior robustness compared to TIES/DARE, especially on the smaller Llama-3.2-3B-Instruction model.

C.3 Merging Efficiency

Beyond merged performance, computational efficiency is another important factor for practical model merging. Figure 9 illustrates the comparison between merging time (in seconds) and the average score across benchmarks. Current baselines exhibit a clear dichotomy: methods like TA and Fisher are computationally efficient (<110s) but suffer from suboptimal performance (around 58.2), while complex methods like TIES and DARE variants achieve better scores at the cost of significant computational overhead (>400s). In contrast, our proposed methods occupy the Pareto frontier of the efficiency-performance landscape. Specifically, RAM achieves a remarkable score of 64.82 in just 75.4 seconds, surpassing DARE+TA in performance while offering a 5.5× speedup. Even our more intensive variant, RAM+, establishes a new SOTA performance (66.55) while remaining significantly faster than both TIES and DARE. These results demonstrate that RAM effectively identifies critical parameters for merging processing without the extensive computational redundancy found in previous SOTAs.

C.4 Detailed Numerical Results for Pairwise Agent Merging

In Section 5.2, we visualize the performance of merging two agents using bar charts to highlight overall trends and comparative advantages. Here, we provide the corresponding detailed numerical results for all pairwise agent combinations, including **Coding+Tool**, **Tool+Memory**, and **Coding+Memory**, as shown in Tables 4, 5, and 6, respectively. These tables serve as a precise quantitative complement to the bar chart visualizations, enabling a fine-grained inspection of per-domain and per-metric behaviors.

	Code				Tool				
Model	LiveBench		LiveCodeBench		Live		Non-Live		**Avg**
	ACC	UT	ACC	UT	Para	P_Mul	Para	P_Mul	
Base and Task Models									
Base	28.35	40.87	23.43	36.42	56.25	41.67	68.00	55.00	43.75
CURE (Coding)	37.70	49.27	30.23	45.76	56.25	37.50	64.00	51.50	46.53
ToolRL (Tool)	31.84	41.36	26.76	42.05	56.25	58.33	91.00	89.00	54.57
Merged Models									
TA	37.11	49.09	29.55	45.83	50.00	37.50	86.00	59.00	49.26
Fisher	36.72	50.29	30.68	**48.18**	50.00	37.50	88.50	64.50	50.80
TIES	35.74	46.50	31.21	44.48	56.25	54.17	83.00	90.50	55.23
DARE+TA	35.74	47.40	30.04	44.62	56.25	58.33	90.00	84.50	55.86
DARE+TIES	35.54	45.79	31.21	44.48	56.25	**66.67**	88.50	85.50	56.74
RAM	**39.45**	**51.42**	31.21	46.51	56.25	**66.67**	**91.00**	**91.00**	59.19
RAM+	**39.45**	51.10	**32.53**	47.55	**62.50**	**66.67**	90.50	90.00	**60.04**

Table 4: **Detailed results of model merging with code and tool using agents. Bold** and underlined values denote the best and second-best performance among *merged models*, respectively. Cells highlighted in red indicate the best performance across *all evaluated models*, including the Task Models.

	Tool				Memory				
Model	Live		Non-Live		HotpotQA		RulerQA		**Avg**
	Para	P_Mul	Para	P_Mul	7k	14k	32k	64k	
Base and Task Models									
Base	56.25	41.67	68.00	55.00	60.94	50.00	64.84	58.59	56.91
ToolRL (Tool)	56.25	58.33	91.00	89.00	58.59	48.44	59.38	46.95	63.49
MemAgent (Memory)	37.50	50.00	78.50	48.50	78.91	78.12	81.25	77.34	66.27
Merged Models									
TA	56.25	62.50	**91.00**	90.00	72.66	70.31	76.56	74.22	74.19
Fisher	56.25	58.33	89.50	91.00	63.28	57.81	65.62	64.84	68.33
TIES	43.75	54.17	73.50	88.50	71.09	71.09	76.56	73.44	69.01
DARE+TA	**62.50**	58.33	89.00	90.00	57.03	55.47	63.28	59.38	66.87
DARE+TIES	**62.50**	58.33	90.00	**91.50**	**77.34**	71.97	75.00	78.12	75.60
RAM	56.25	**66.67**	90.00	89.50	76.56	78.12	**78.12**	78.12	**76.67**
RAM+	56.25	62.50	89.50	90.00	76.56	**78.13**	74.22	**79.69**	75.86

Table 5: **Detailed results of model merging with memory and tool using agents. Bold** and underlined values denote the best and second-best performance among *merged models*, respectively. Cells highlighted in red indicate the best performance across *all evaluated models*, including the Task Models.

	Code				Memory				
Model	LiveBench		LiveCodeBench		HotpotQA		RulerQA		**Avg**
	ACC	UT	ACC	UT	7k	14k	32k	64k	
Base and Task Models									
Base	28.35	40.87	23.43	36.42	60.94	50.00	64.84	58.59	45.43
CURE (Coding)	37.70	49.27	30.23	45.76	58.59	56.25	60.94	44.22	47.87
MemAgent (Memory)	39.25	50.12	28.92	44.8	78.91	78.12	81.25	77.34	59.84
Merged Models									
TA	39.06	52.25	32.34	**47.89**	71.88	70.31	75.00	75.00	57.97
Fisher	38.67	50.53	31.46	46.58	57.03	50.78	58.59	55.47	48.64
TIES	38.87	50.88	**32.42**	47.80	74.22	71.09	78.91	72.66	58.36
DARE+TA	37.69	47.77	31.16	44.51	**78.91**	**78.12**	78.12	80.47	59.59
DARE+TIES	39.25	49.95	31.02	44.52	75.78	75.78	76.56	76.56	58.68
RAM	37.89	49.56	31.75	47.17	78.12	**78.12**	**82.03**	**82.03**	60.83
RAM+	**41.21**	**53.64**	31.16	46.46	**78.91**	76.56	79.69	**82.03**	**61.21**

Table 6: **Detailed results of model merging with coding and memory agents. Bold** and underlined values denote the best and second-best performance among *merged models*, respectively. Cells highlighted in red indicate the best performance across *all evaluated models*, including the Task Models.

Model	Code				Tool				Memory				Avg
	LiveBench		LiveCodeBench		Live		Non-Live		HotpotQA		RulerQA		
	ACC	UT	ACC	UT	Para	P_Mul	Para	P_Mul	7k	14k	32k	64k	
RAM	38.28	49.71	**31.96**	**47.72**	**56.25**	66.67	**91.00**	**91.50**	75.78	75.00	74.22	79.69	64.82
RAM+	**40.23**	**52.57**	31.60	46.84	**56.25**	**70.83**	90.50	91.00	**79.69**	**78.13**	**78.91**	**82.03**	**66.55**
RAM+s	39.84	51.83	30.82	46.59	**56.25**	**70.83**	90.00	**91.50**	77.34	76.56	76.56	77.34	65.46

Table 7: The comparison results of a soft-saturation rescaling transformation.

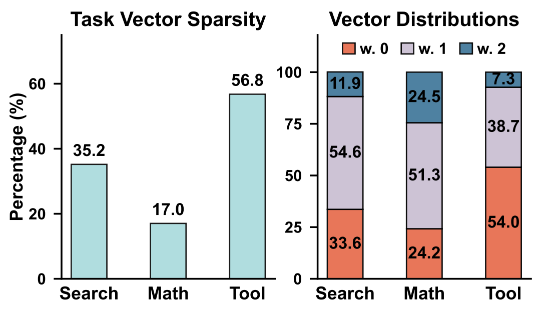

Figure 8: The task vector analysis for agents trained from Llama3.2-3B-Instruction via RL. **Left**: Sparsity of task vectors varies between tool-using agent, web search agent, and the math reasoning agent models. **Right**: The number of overlaps with other task vectors.

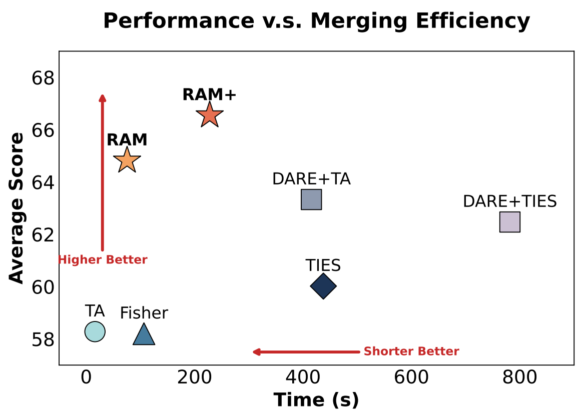

Figure 9: RAM/RAM+ demonstrates a superior trade-off between merging time and average score compared to the baseline method.

Coding + Tool. As reported in Table 6, RAM and RAM+ consistently achieve the highest average performance among all merged models. In particular, RAM+ attains an average score of 60.04, significantly outperforming the strongest baseline DARE+TIES (56.74). Notably, RAM/RAM+ improve both coding accuracy (LiveBench / LiveCodeBench) and complex tool-use metrics (Parallel and Parallel-Multiple), indicating that preserving task-unique reinforced updates enables the merged model to simultaneously retain algorithmic reasoning and structured tool invocation capabilities. In contrast, baseline methods exhibit a clear trade-off, improving one domain at the expense of the other due to signal dilution in sparse RL task vectors.

Tool + Memory. Table 5 presents the detailed results for merging Tool and Memory agents. RAM achieves the highest overall average score (76.67), while RAM+ remains highly competitive (75.86), both surpassing all baselines. RAM-based models show strong performance on long-context memory tasks (HotpotQA and RulerQA) without degrading tool-use accuracy, particularly on challenging Parallel and Parallel-Multiple subsets. These results demonstrate that RAM effectively isolates memory-specific reinforced updates from tool-specific ones, preventing destructive interference that commonly occurs in averaging-based merging methods.

Coding + Memory. As shown in Table 6, merging Coding and Memory agents further stresses the heterogeneity of reinforced task vectors, as these two domains exhibit minimal parameter overlap. Despite this challenge, RAM+ achieves the best average performance (61.21), outperforming all baselines and even exceeding the original Coding agent on LiveBench ACC/UT. This highlights the effectiveness of overlap-aware rescaling in compensating for performance loss in shared subspaces, while fully preserving unique reasoning and memory patterns critical for each task.

Across all pairwise combinations, the numerical results reported here are fully consistent with the trends observed in the bar charts in the main text. Specifically, RAM and RAM+ not only deliver the highest average scores but also exhibit superior robustness across heterogeneous domains and metrics. These findings further confirm that the advantages of RAM are not limited to tri-agent merging, but generalize naturally to arbitrary agent combinations, reinforcing its suitability as a unified merging framework for reinforced agentic models.

C.5 Additional Results in Merging Three Agents

In Section 5.2, we present the agent performance in three domains. Here, we further provide experiment results on additional tasks and settings to comprehensively verify the advantages of RAM.

Overall Analysis. Tables 8, 9, and 10 report comprehensive results of merging three reinforced agents across coding, tool-use, and long-context memory domains, substantially extending the representative results in Section 5.2. These results consistently corroborate the advantages of RAM and RAM+ under a wide range of evaluation metrics and task granularities.

Coding Domain. As shown in Table 8, RAM-based methods achieve the strongest overall performance among merged models across LiveBench, LiveCodeBench, MBPP, and CodeContests. Notably, RAM attains the highest average score (49.64), outperforming all baseline merging strategies, while RAM+ further improves performance on challenging subsets such as LiveBench ACC/UT and MBPP ACC/UT. Importantly, RAM and RAM+ frequently match or exceed the original Coding agent on multiple metrics (highlighted in red), indicating that preserving and selectively amplifying task-unique reinforced updates effectively avoids signal dilution that hampers prior methods. In contrast, methods relying on global averaging or random dropping (TA, Fisher, DARE) exhibit inconsistent gains and fail to simultaneously retain high unit-test robustness and generalization accuracy.

Tool-Use Domain. Table 9 presents detailed tool-use evaluations over both Live and Non-Live settings. Across diverse function-calling scenarios—including parallel, parallel-multiple, irrelevance detection, and language-specific tasks—RAM achieves the best average score (77.51), while RAM+ remains a close second (77.45), both surpassing all existing merging baselines. Crucially, RAM-based models consistently excel in structurally complex settings such as Live Parallel_multiple and Non-Live Parallel, where signal dilution in sparse RL task vectors is most detrimental. These results demonstrate that RAM effectively preserves tool-specific reasoning circuits while still benefiting from shared knowledge introduced by other agents.

Long-Context Memory Domain. As reported in Table 10, RAM and RAM+ show clear dominance on long-context memory benchmarks across all context lengths. RAM+ achieves the highest overall average score (77.57), outperforming both merged baselines and, in several settings, the specialized MemAgent itself. In particular, RAM+ consistently delivers state-of-the-art performance on long-context HotpotQA (112K–896K) and Ruler-SQuAD (16K–64K), confirming that overlap-aware rescaling effectively compensates for performance degradation introduced by averaging shared subspaces. By contrast, Fisher and Task Arithmetic suffer from substantial performance drops at long context lengths, reflecting their inability to preserve sparse, task-specific memory-related updates.

Across all three domains, the additional results reinforce three central conclusions. First, reinforced task vectors exhibit strong heterogeneity, making uniform merging strategies fundamentally suboptimal. Second, preserving and rescaling task-unique parameters is essential for maintaining expert-level performance after merging. Third, RAM and RAM+ consistently outperform prior SOTA merging methods not only in average performance but also in robustness across metrics, datasets, and context scales. Together, these findings provide strong empirical evidence that RAM is a principled and effective solution for merging multiple RL-trained agents into a unified generalist model.

D Baseline Details

D.1 Baselines with Signal Dilution

Here we provide a detailed introduction of baselines and explain signal dilution happend in them. In summary, despite their distinct algorithmic designs, Task Arithmetic, TIES-Merging, and DARE all inevitably succumb to Signal Dilution in the RL setting due to a shared inability to distinguish between *shared consensus* and *unique specialization*. Specifically, Task Arithmetic is forced to apply a conservative global scaling ($\lambda \approx 1/N$) to prevent magnitude explosion in shared subspaces, collaterally suppressing unique task vectors that require no scaling. Similarly, TIES-Merging fails to strictly isolate disjoint parameters due to "noise drift" in inactive RL models, which erroneously inflates the normalization denominator with meaningless artifacts. Even DARE, despite introducing a rescaling

Model	LiveBench				LiveCodeBench				MBPP				CodeContests				Avg
	ACC	UT	BoN ACC	BoN UT	ACC	UT	BoN ACC	BoN UT	ACC	UT	BoN ACC	BoN UT	ACC	UT	BoN ACC	BoN UT	
Base and Task Models																	
Base	28.35	40.87	30.07	48.56	23.43	36.42	26.67	42.26	62.72	68.50	70.91	76.84	18.20	28.35	23.01	32.06	41.08
CURE	37.70	49.27	46.01	**58.96**	30.23	45.76	38.36	**55.60**	70.36	76.38	**79.64**	**85.78**	**26.05**	36.81	**32.21**	**45.61**	**50.92**
ToolRL	31.84	41.36	32.81	43.78	26.76	42.05	30.14	46.05	66.97	72.31	76.02	81.29	22.18	32.94	26.78	37.71	44.44
MemAgent	39.25	50.12	39.84	48.48	28.92	44.80	32.16	48.63	67.63	73.99	71.04	77.84	21.86	31.68	24.69	36.09	46.06
Merged Models																	
TA	38.09	51.62	43.75	52.97	31.95	46.69	35.61	46.44	69.68	75.71	74.21	80.39	25.73	36.70	28.45	38.70	48.54
Fisher	36.72	48.73	45.31	57.79	30.87	45.89	35.81	53.26	68.67	74.92	75.57	83.23	24.26	35.87	27.20	39.38	48.97
TIES	39.25	49.88	44.53	53.08	30.63	46.32	36.59	54.30	67.30	73.35	74.66	82.34	**26.05**	**37.00**	**30.50**	42.05	49.24
DARE+TA	37.50	48.60	46.10	56.41	31.94	46.69	37.18	53.44	69.68	75.71	78.73	83.83	23.74	32.82	28.45	38.02	49.30
DARE+TIES	35.93	45.66	**46.87**	57.88	29.26	39.53	**38.55**	51.76	70.70	76.34	**80.64**	**86.22**	21.13	27.62	27.20	35.93	48.20
RAM	38.28	49.71	42.97	**57.98**	**31.96**	**47.72**	37.45	**54.36**	69.46	75.56	76.47	81.89	24.06	35.07	26.78	**44.51**	**49.64**
RAM+	**40.23**	**52.57**	46.10	57.20	31.60	46.84	34.64	51.56	**70.93**	**76.98**	77.38	84.13	22.80	33.21	23.01	35.46	49.04

Table 8: **Additional results of model merging on coding domains. Bold** and underlined values denote the best and second-best performance among *merged models*, respectively. Cells highlighted in red indicate the best performance across *all evaluated models*, including the Task Models.

Models	**Live**						**Non-live**								**Avg**
	Multiple	Parallel	Relevance	Simple	Parallel_multiple	Irrelevance	Multiple	Irrelevance	S_java	S_javascript	Parallel_multiple	Parallel	S_python		
Base and Task Models															
Base	58.59	56.25	87.50	68.99	41.67	68.21	77.50	77.50	54.00	62.00	55.00	68.00	87.25		66.34
CURE	59.54	56.25	81.25	69.77	37.50	68.33	76.50	77.92	58.00	60.00	51.50	64.00	91.50		65.54
ToolRL	76.83	56.25	93.75	79.07	58.33	71.95	94.00	82.92	62.00	62.00	89.00	91.00	93.50		77.74
MemAgent	73.98	37.50	93.75	69.77	50.00	56.11	82.50	71.67	66.00	66.00	48.50	78.50	90.25		68.04
Merged Models															
TA	**76.16**	43.75	**93.75**	75.19	45.83	**68.55**	91.00	79.17	60.00	62.00	69.50	87.50	**94.75**		72.86
Fisher	74.55	43.75	**93.75**	74.03	41.67	70.02	91.50	81.25	58.00	58.00	63.50	86.50	94.50		71.62
TIES	75.50	43.75	87.50	75.19	54.17	63.46	**92.50**	76.25	**65.00**	66.00	67.50	84.00	94.50		72.71
DARE+TA	75.50	50.00	**93.75**	**76.36**	62.50	63.57	92.00	74.17	**65.00**	66.00	89.50	**92.50**	93.75		76.50
DARE+TIES	75.31	**56.25**	**93.75**	**76.36**	58.33	63.80	92.00	**76.67**	**65.00**	**68.00**	90.00	91.50	93.25		76.94
RAM	75.50	**56.25**	**93.75**	75.58	**66.67**	67.99	92.00	**76.67**	**65.00**	64.00	**91.50**	91.00	91.75		**77.51**
RAM+	75.12	**56.25**	**93.75**	75.58	**66.67**	67.87	92.00	**76.67**	**65.00**	66.00	90.50	90.00	91.50		77.45

Table 9: **Additional results of model merging on tool using. Bold** and underlined values denote the best and second-best performance among *merged models*, respectively. Cells highlighted in red indicate the best performance across *all evaluated models*, including the Task Models.

Model	**Ruler_SQuAD**					**Ruler_HotpotQA**								**Avg**
	8K	16K	32K	64K	128K	7K	14K	28K	56K	112K	224K	448K	896K	
Base and Task Models														
Base	65.63	62.50	64.84	58.59	56.25	60.94	50.00	51.56	48.44	42.19	36.72	27.34	25.78	50.06
CURE	46.80	61.72	60.94	44.22	61.72	58.59	56.25	46.88	47.66	40.63	35.93	42.97	35.16	49.19
ToolRL	62.50	57.81	59.38	46.95	67.18	58.59	48.44	51.56	45.31	42.97	42.19	35.16	35.94	50.31
MemAgent	83.59	78.12	81.25	77.34	81.25	78.91	78.12	81.25	77.34	79.69	72.66	76.56	72.66	78.36
Merged Models														
TA	72.66	70.31	73.44	72.66	72.66	69.53	68.75	68.75	67.19	63.28	64.84	57.03	48.44	66.89
Fisher	60.16	67.97	58.59	60.94	67.97	60.06	49.22	49.22	49.22	39.06	32.81	35.94	35.94	51.32
TIES	71.88	75.00	75.00	75.78	73.44	71.88	82.03	71.97	68.75	67.97	70.31	62.50	64.06	71.58
DARE+TA	71.88	77.34	77.34	70.31	**80.47**	76.56	76.56	72.65	71.88	70.31	73.44	71.88	70.31	73.92
DARE+TIES	75.78	78.13	76.56	74.22	78.91	75.00	77.34	75.00	70.36	**70.43**	76.56	**74.44**	**74.22**	75.15
RAM	**79.68**	82.03	74.22	79.69	76.56	75.78	75.00	**78.91**	**75.78**	70.31	75.78	71.09	**74.22**	76.08
RAM+	77.34	**82.81**	**78.91**	**82.03**	79.68	**79.69**	**78.13**	**78.91**	75.56	69.53	**78.13**	73.43	**74.22**	**77.57**

Table 10: **Additional results of model merging on memory for long-context tasks. Bold** and underlined values denote the best and second-best performance among *merged models*, respectively. Cells highlighted in red indicate the best performance across *all evaluated models*, including the Task Models.

mechanism, only compensates for *internal* dropout rather than *external* merging, thereby remaining dependent on the global scaling of Task Arithmetic to function. Consequently, all three paradigms effectively drive the magnitude of task-specific updates towards $\frac{1}{N}\boldsymbol{\tau}$, underscoring the necessity of a topology-aware merging framework.

Fisher Fisher merging (Matena and Raffel, 2022) improves upon uniform averaging by weighing parameters according to their diagonal Fisher information F, which approximates the posterior precision (or local curvature) of the model. Formally, the merged update is computed as $\boldsymbol{\tau}_{\text{merged}} = \frac{\sum_i F_i \boldsymbol{\tau}_i}{\sum_i F_i}$. However, this method remains susceptible to signal dilution in the sparse RL setting. Consider a "unique" parameter updated solely by task t (where $\boldsymbol{\tau}_t \neq 0$ and $\boldsymbol{\tau}_{i \neq t} = 0$). Crucially, the inactive models ($i \neq t$) still contribute to the normalization term in the denominator, as their Fisher values F_i—representing the confidence of the pre-trained base model—are typically non-zero. Consequently, the effective scaling factor for the unique signal becomes $\frac{F_t}{F_t + \sum_{i \neq t} F_i}$. Since the denominator accumulates the "inertia" (precision) of all inactive models, the task-specific update $\boldsymbol{\tau}_t$ is inevitably scaled down, mirroring the dilution effect observed in uniform averaging as the number of tasks N increases.

Task Arithmetic. Task Arithmetic (Ilharco et al., 2023) constructs a multi-task model by linearly combining task vectors, typically expressed as $\boldsymbol{\tau}_{\text{merged}} = \sum_i \lambda \boldsymbol{\tau}_i$. While effective for disjoint tasks in SFT, it faces a critical trade-off in the RL setting due to the heterogeneity of parameter overlap. To prevent catastrophic magnitude shifts in *shared* subspaces (where multiple agents push parameters in similar directions), the scaling factor λ must be conservative (typically $\lambda \approx 1/N$) to maintain the merged weights within a valid optimization landscape. However, this global scaling creates a structural conflict: *unique* task vectors, which reside in non-overlapping subspaces and do not suffer from additive interference, are subjected to the same aggressive downscaling. Consequently, the update for a task-specific parameter is reduced to $\frac{1}{N}\boldsymbol{\tau}_t$, effectively suppressing the critical, idiosyncratic behaviors required for expert-level performance in domains like tool-use or memory.

TIES-Merging. TIES-Merging (Yadav et al., 2023) attempts to mitigate interference by keeping only the top-$k\%$ magnitude parameters (Trimming) and calculating a disjoint mean. However, its reliance on a *uniform* trimming rate k creates a critical vulnerability when handling the **heterogeneous sparsity** of RL task vectors. As shown in Figure 2, RL agents exhibit vastly different update densities (e.g., Code: $\sim 3\%$, Memory: $\sim 54\%$). Applying a fixed k (e.g., 20%) inevitably leads to a dilemma: it either over-trims dense vectors (losing info) or, more disastrously, under-trims sparse vectors. For a sparse agent like the Code model, a standard k retains a large volume of noise parameters alongside the true signal. These noise parameters, mistakenly treated as valid updates, overlap with the active parameters of other tasks, inflating the normalization factor in the disjoint mean calculation ($\tau_m = \frac{1}{|A_p|}\sum \tau_t$). Consequently, the critical, unique signal from one task is averaged with noise from others, resulting in signal dilution.

DARE. DARE (Yu et al., 2024) randomly drops parameters and enlarges others to reduce redundancy, theoretically approximating the original task vector's expectation. However, DARE fails to address signal dilution for two reasons. First, its factor $(1/(1-p))$ is designed solely to compensate for the internal dropout rate p, not for the external averaging with other models. When combined with Task Arithmetic (DARE+TA or DARE+TIES), the global merging scale λ must still be kept small (e.g., $1/N$) to stabilize shared parameters, inevitably downscaling the unique, sparsity-preserved updates. Second, DARE assumes that parameter updates are highly redundant (a property of SFT), whereas RL updates are inherently sparse and functionally essential. Randomly dropping parameters in an already sparse RL vector risks severing critical reasoning circuits, which cannot be recovered simply by rescaling the remaining weights.

D.2 Hyperparameters Search

Table 11 shows the searched ranges of model merging methods' hyperparameters. We search for the best performance for evaluation and comparison.

E Rescaling Variant

To bridge the theoretical requirement with numerical stability mentioned in Section 4.2, besides clipping, we further explore a soft-saturation transformation $\phi(x) = \frac{x}{1+x}$ to map the unbounded ratio ρ_t to the bounded interval $[0, 1)$. Our operational

Table 11: Searched ranges of hyperparameters of model merging baselines.

Model Merging Methods	Search Ranges of Hyperparameters
Task Arithmetic	scaling term to merge model parameters: [0.1, 0.3, 0.5, 0.7, 0.9, 1.0]
Fisher	scaling term to merge model parameters: [0.1, 0.3, 0.5, 0.7, 0.9, 1.0], number of examples to compute Fisher information matrix: [256, 512, 1024, 2048]
TIES	scaling term to merge model parameters: [0.1, 0.3, 0.5, 0.7, 0.9, 1.0], ratio to retain parameters with largest-magnitude values: [0.1, 0.2, 0.3]
DARE	search the drop rate p in [0.1, 0.2, ... , 0.9]

scaling rule is thus defined as:

$$\lambda_t = 1 + r \cdot \left(\frac{\rho_t}{1 + \rho_t} \right). \tag{8}$$

Here, the hyperparameter r absorbs the dilution factor $(1 - \beta)$.

It is worth noting that this normalized formulation is mathematically equivalent to the ratio of shared parameters to the *total* active parameters. By substituting the definition of ρ_t from Eq. 2, we obtain:

$$\frac{\rho_t}{1 + \rho_t} = \frac{\sum_{i:c_i \geq 2} m_{t,i}}{||\mathbf{m}_t||_0}. \tag{9}$$

Here, the numerator sums the shared parameters, while the denominator $||\mathbf{m}_t||_0$ represents the total count of active parameters (satisfying $||\mathbf{m}_t||_0 = \sum_{i:c_i \geq 2} m_{t,i} + \sum_{i:c_i = 1} m_{t,i}$). This transformation provides a dual advantage: it retains the monotonicity derived from the conservation principle (higher overlap yields higher compensation) while enforcing a strict upper bound to ensure optimization robustness. Table 7 presents the comparative performance of the soft-saturation rescaling strategy, denoted as **RAM+s**. Empirically, RAM+s achieves an average score of 65.46, consistently outperforming the non-rescaled baseline RAM (64.82) across all three domains. This reinforces our core hypothesis that compensating for signal dilution in unique parameter subspaces is essential for recovering expert capabilities, regardless of the specific scaling function used. However, RAM+s slightly underperforms compared to the clipped linear variant (RAM+, 66.55). While the soft-saturation transformation $\phi(x) = \frac{x}{1+x}$ offers a theoretically elegant, strictly bounded mapping, it appears to dampen the scaling factor more aggressively than the linear approach. This conservatism limits performance in tasks requiring robust signal preservation, such as Long-Context Memory, where RAM+s scores 77.34 on RulerQA (64k) compared to RAM+'s 82.03. Conversely, in the Tool domain, RAM+s remains highly effective, matching RAM+ with a score of 70.83 on Live Parallel-Multiple tasks. These findings suggest that while the soft-saturation rule provides a stable alternative, the clipped linear rule offers a superior trade-off between signal amplification and numerical stability.