LEVERAGING INSTRUCTION TUNING AND MERGING FOR REASONING MODEL ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Reasoning language models (RLMs) demonstrate impressive performance by leveraging test-time compute in the form of reasoning tokens. However, this behavior makes adapting RLMs to new domains challenging and expensive. The reason is that further training can disturb the learned behavior and degrade model performance. This makes it difficult to leverage supervised fine-tuning data with human-written solutions: although it contains high-quality annotations, it lacks reasoning tokens. In this work, we show how, despite this challenge, such data can be used efficiently for RLM adaptation. For this, we first use standard instruction tuning. Next, we leverage model merging to combine the instruction-tuned model with the original RLM, picking the merging ratio such that the resulting model's reasoning behavior on the target domain is recovered. We evaluate our method across four RLMs on coding and text summarization tasks, where it improves target-task performance by up to 11.0% while preserving reasoning behavior and limiting the out-of-distribution score degradation to on average 0.7%. Importantly, our adaptations are efficient and economical, costing less than USD $\$10$ per model.

1 INTRODUCTION

Reasoning language models (RLMs) have defined the frontier of language model capabilities by demonstrating impressive results on tasks such as mathematics and programming (OpenAI et al., 2024b; Yang et al., 2025). RLMs are trained to produce reasoning traces that explore potential solutions or perform logical reasoning before producing a final answer (Shao et al., 2024). The dominant training recipes for RLMs are based on reinforcement learning with verifiable rewards (RLVR) (DeepSeek-AI, 2025; Team Olmo, 2025). While this approach is unquestionably powerful, it is highly expensive: Training requires repeated model rollouts for each training step, with long rollouts requiring thousands of forward passes of the model (Huang & Wan, 2026).

For non-reasoning models, adaptation to new domains and knowledge is typically achieved through instruction fine-tuning (IFT): The model is trained directly on task descriptions paired with high-quality task solutions, collected through scraping or human annotation (Zhang et al., 2025; Lambert et al., 2024; He et al., 2024). Large amounts of training data are available for IFT, which makes the approach cheap, effective, and widely applicable (Wei et al., 2022; Chung et al., 2022).

However, the task solutions in IFT training data, unlike RLM outputs, lack reasoning traces, which creates a distributional mismatch when training RLMs using IFT. This mismatch can severely degrade general RLM performance (Lobo et al., 2025).

This work: Leveraging IFT for training RLMs In this work, we show that this mismatch can be mitigated effectively with a two-step procedure. First, we perform standard IFT on the input-output pairs without reasoning traces. Second, we merge the resulting IFT checkpoint with the original reasoning model. The merge ratio is selected as the largest coefficient that preserves target-task reasoning on a small calibration set. In contrast to prior proposed methods, this procedure does not require a reliable verifier or data beyond the task-specific training set and is highly cost-effective.

We evaluate our approach on four open RLMs spanning different model families, sizes, and training paradigms: OpenThinker 7B (Guha et al., 2025), Qwen R1-D 7B (DeepSeek-AI, 2025), Olmo3 7B (Team Olmo, 2025), and Apriel 15B (Radhakrishna et al., 2025). We evaluate our method on

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Rust coding and text summarization and measure out-of-distribution performance on three unrelated benchmarks. On average, our method results in out-of-distribution performance loss of 0.7%, while resulting in target-task gains between 1.0% and 11.0%. End-to-end adaptation costs between USD \$1.07 and \$9.17, reducing cost by 63% to 99% relative to our evaluated RL baselines.

Key contributions Our key contributions are: (i) We identify and study a practical adaptation setting for RLMs where only input-output supervision is available, without verified reasoning traces, (ii) we propose a lightweight IFT-and-merge method that adapts an RLM while selecting the merge ratio using only reasoning behavior on target-task calibration data, and (iii) in our extensive evaluation, we show that our method effectively balances target-task performance with reasoning and out-of-distribution behavior. We will publicly release our code implementation and datasets.

2 Background

In this section, we introduce (reasoning) language models, instruction fine-tuning, and model merging.

Language models An autoregressive language model (LM) is a statistical model of language with parameters θ. We can sample from a model M_θ given context $\mathbf{c}$ to obtain an *answer* $\hat{\mathbf{y}} \sim M_\theta(\mathbf{c})$. Pre-trained large language models (LLMs) are LMs with billions of parameters, trained on trillions of tokens of training data (OpenAI et al., 2024a; DeepSeek-AI, 2025; Yang et al., 2025). Through such pre-training, the models acquire a variety of general skills, in particular language understanding.

RLMs Reasoning language models (RLMs) are LLMs trained to generate a *reasoning trace* $\mathbf{r}$ before producing the answer $\hat{\mathbf{y}}$. The reasoning trace can serve as internal scratch-pad, not scored for the final answer, and is separated from the answer by an end-of-reasoning token. RLMs are either trained via reinforcement learning with verifiable rewards (RLVR) (OpenAI et al., 2024b; DeepSeek-AI, 2025; Yang et al., 2025) or through distillation of reasoning traces from a stronger model (Guha et al., 2025; DeepSeek-AI, 2025). Empirically, this results in strong capabilities for logic-based tasks such as mathematics and coding (Shao et al., 2024; DeepSeek-AI, 2025).

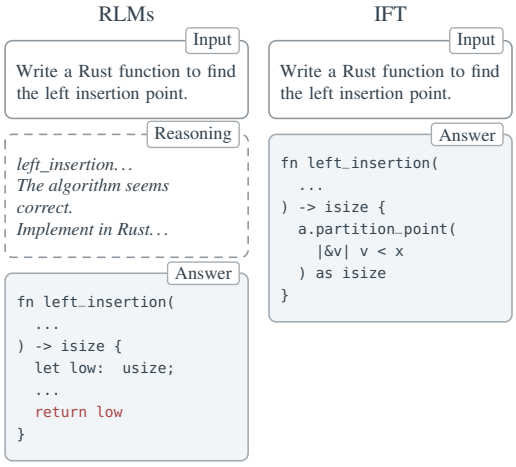


Figure 1: RLMs reason before answering; IFT targets start directly with the answer.

Supervised and instruction fine-tuning In practice, models can be adapted to specialized tasks by training them on novel knowledge (Farn et al., 2025; Lambert et al., 2024). Such adaptation is commonly achieved through supervised fine-tuning (SFT) (Stiennon et al., 2022; He et al., 2024). SFT updates a model's parameters using a labeled dataset $\mathcal{D} = \{(\mathbf{x}, \mathbf{y})\}$ of input-output pairs, and trains the model to predict $\mathbf{y}$ for input $\mathbf{x}$. Instruction fine-tuning (IFT) is the special case of SFT where $\mathbf{x}$ is an instruction or task description and $\mathbf{y}$ is an answer or solution to the task (Muennighoff et al., 2024; Chung et al., 2022). Large datasets of high-quality input-output pairs for a variety of target domains exist (Lambert et al., 2024; Lozhkov et al., 2024) or can be synthesized (Honovich et al., 2022; Wang et al., 2023), which allows model users to adapt models to a desired domain.

Model merging Model merging combines two sets of model weights, θ_1 and θ_2, into weights θ_α using linear interpolation with a *merging ratio* α.

$$\theta_\alpha = \theta_1 + \alpha\Delta, \quad \alpha \in [0, 1], \quad \Delta = \theta_2 - \theta_1.$$

Model merging can be used to combine model capabilities (Matena & Raffel, 2022; Yadav et al., 2023) and recover forgotten behavior (Alexandrov et al., 2024; Farn et al., 2025). Prior work shows that merging can produce capable models along various merging ratios of related low-loss training checkpoints (Ilharco et al., 2023; Frankle et al., 2020; Alexandrov et al., 2024; Yang et al., 2024b). Non-linear merging variants exist as well (Yadav et al., 2023; Shoemake, 1985).

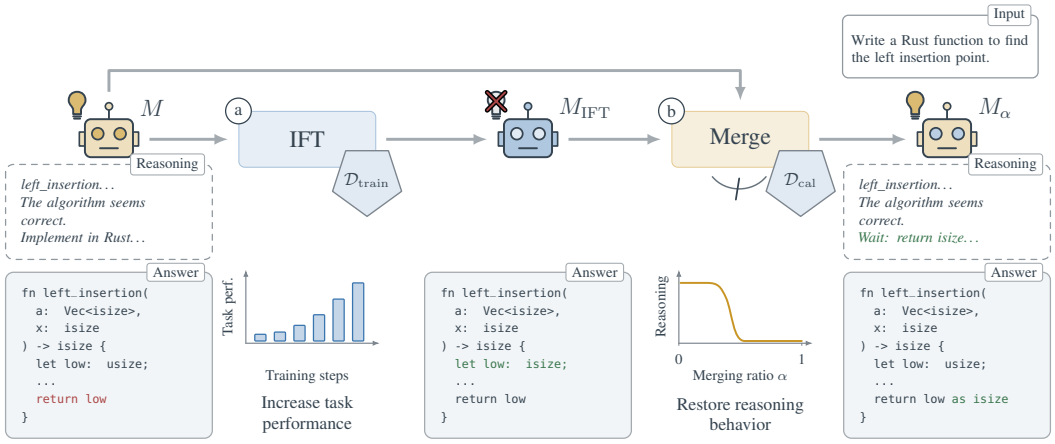


Figure 2: We propose a lightweight two-step pipeline for RLM adaptation to a target task, here Rust coding. It consists of (a) training on IFT data and then (b) merging the resulting model M_{IFT} with the original model M, picking the maximal merging ratio that preserves reasoning behavior. It balances capabilities obtained through IFT training with observed reasoning behavior.

3 Training RLMs via IFT and Model Merging

In this section, we outline the key challenges and describe our core pipeline for adapting RLMs. The key insight behind our method is that traditional IFT can still be used for RLMs when carefully combined with a model-merging technique that preserves the model's reasoning behavior.

Main challenges The main challenge for adapting RLMs lies in the learned reasoning behavior. Training the model without reasoning traces as in IFT creates a conflict with the learned output distribution: the model is trained to exhibit reasoning during rollouts, but such reasoning traces are absent from the training data available during IFT as shown in Figure 1. This poses a challenge because training RLMs without reasoning traces can lead to performance degradation and catastrophic forgetting (Twist et al., 2026). Meanwhile, continuing RLVR on the target domain is expensive (Ma et al., 2026; Shao et al., 2024), and works predominantly in domains where reliable verifiers are available (Sprague et al., 2025; Amodei et al., 2016). Our work addresses these challenges by training RLMs on IFT data without requiring reasoning traces. As such, our method can leverage widely available IFT datasets to teach models novel knowledge while preserving learned reasoning behavior.

Motivation Our key insight is twofold: First, IFT is a proven and tested method for adapting standard LMs (Muennighoff et al., 2024; He et al., 2024; Stiennon et al., 2022). Second, reasoning behavior is learned late in the model training process (Team Olmo, 2025; Yang et al., 2025). Its absence correlates in general with capability degradation (Lobo et al., 2025) and might serve as a canary for general model forgetting of learned skills. Model merging is a lightweight intervention that mitigates model forgetting (Alexandrov et al., 2024) and at the same time can recover reasoning behavior when adequately scaled (Zhang et al., 2024).

Overview Our core method for RLM training consists of a lightweight two-step pipeline, visualized in Figure 2: We first perform standard IFT on RLM M using dataset $\mathcal{D}_{\text{train}}$, which produces a fine-tuned model M_{IFT} with higher task accuracy but potentially compromised reasoning behavior and other forgotten capabilities. In the shown example, Apriel correctly inserts missing type casts in Rust code after fine-tuning, but does not emit any reasoning about the code. We then merge M and M_{IFT}, picking the maximal merge ratio α that restores model reasoning on calibration dataset $\mathcal{D}_{\text{cal}}$. The merge produces M_α, a reasoning model with maintained or improved task capability. In the example, the merged Apriel then reasons about the typing mismatch and required cast and correctly inserts it in the resulting code. Complete outputs are presented in §B. By design, our method requires no reliable verifiers, expensive model rollouts, or stronger RLMs.

Datasets We use $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{cal}}$, and $\mathcal{D}_{\text{test}}$ for fine-tuning, merge-ratio selection, and final evaluation, respectively. Each task example is an input-output pair (i, o), where i is the task input and o is the target output. For a reasoning model, we write a sampled response as $(r, \hat{o}) \sim M(i)$, where r is the reasoning trace and $\hat{o}$ is the final answer.

Fine-tuning We first fine-tune the RLM M on $\mathcal{D}_{\text{train}}$ using standard IFT (Ouyang et al., 2022). Since our dataset $\mathcal{D}_{\text{train}}$ contains only (i, o) pairs and does not contain the reasoning trace r, we serialize each training target as (ε, o): the empty reasoning trace ε followed by the target output. As a result, our pipeline obtains model M_{IFT} with weights θ_{IFT}.

Because the fine-tuning targets contain an empty reasoning trace, this step can reduce the model's reasoning behavior: M_{IFT} may directly output the final answer on examples where the original reasoning model would have produced reasoning tokens.

Merging To recover reasoning behavior, we perform a model merge between the original reasoning model M and M_{IFT}. We select the merging ratio α by searching over model reasoning on a target-task calibration set $\mathcal{D}_{\text{cal}}$. Since M is an RLM in our setting, it has a full calibration reasoning rate, i.e., $\rho(M, \mathcal{D}_{\text{cal}}) = 1$. For any model M' and dataset $\mathcal{D}$, we define the reasoning rate $\rho(M', \mathcal{D})$ as the fraction of examples for which the model produces a non-empty reasoning trace.

$$\rho(M', \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(i,o)\in\mathcal{D}} \mathbf{1}\{r_i \neq \varepsilon\}, \qquad (r_i, \hat{o}_i) \sim M'(i). \tag{1}$$

We design a search procedure to select the model closest to the IFT checkpoint that preserves target-task performance gains while recovering the model's target-task reasoning. Let ρ_{min} be the minimum acceptable reasoning rate; in our experiments, $\rho_{\text{min}} = 1.0$. We are then looking for the largest merge ratio whose target-task calibration reasoning rate remains acceptable, i.e.,

$$\alpha^\star = \max_{\alpha\in\mathcal{A}_K} \{\alpha : \rho_{\text{min}} \leq \rho(M_\alpha, \mathcal{D}_{\text{cal}})\}. \tag{2}$$

This objective favors the model closest to M_{IFT} among those that still preserve enough reasoning behavior.

Algorithm 1 Our Adaptation Algorithm

Input: RLM M; $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{cal}}$; ρ_{min}; K
Output: Adapted RLM $M_{\alpha^\star}$

Step 1: Standard IFT
$M_{\text{IFT}} \leftarrow \text{IFT}(M, \mathcal{D}_{\text{train}})$
Step 2: Merging
$\mathcal{A}_K \leftarrow \{j/K \mid j \in \{0, \ldots, K\}\}$
$\alpha^\star \leftarrow 0$
for $\alpha \in \mathcal{A}_K$ **do**
  $M_\alpha \leftarrow \text{MERGE}(M, M_{\text{IFT}}, \alpha)$
  $\rho_\alpha \leftarrow \rho(M_\alpha, \mathcal{D}_{\text{cal}})$
  if $\rho_\alpha \geq \rho_{\text{min}}$ **then**
    $\alpha^\star \leftarrow \alpha$
return $M_{\alpha^\star}$

We describe the algorithm in Algorithm 1. After IFT, we evaluate a small uniform grid of merge ratios. For a grid resolution K, we use $\mathcal{A}_K = \{0, 1/K, 2/K, \ldots, 1\}$. The point $\alpha = 0$ is the original model M and is guaranteed to satisfy $\rho(M, \mathcal{D}_{\text{cal}}) = 1$, while $\alpha = 1$ is the standard IFT model M_{IFT}. We evaluate all points on $\mathcal{D}_{\text{cal}}$, and select the largest α whose reasoning rate is at least ρ_{min}.

Optimizations We apply two optimizations to speed up this process: running calibration only on the first few tokens and using binary search in place of exhaustive grid search.

First, we observe that, when M_{IFT} does not reason, it emits the end-of-reasoning token immediately at the beginning of its output and does not fall back into reasoning afterwards. We leverage this to speed up the merge-ratio selection: for each calibration input, we generate only the first few tokens after the reasoning start token and check whether the model begins a non-empty reasoning trace. This short-prefix evaluation is enough to reliably determine $\rho(M_\alpha, \mathcal{D}_{\text{cal}})$ for the grid search.

Second, we reduce the computational effort for the search in Algorithm 1. We observe that the reasoning rate decreases monotonically as the merge ratio increases. This allows us to employ binary search instead of grid search in compute-constrained settings. We use binary search to identify the largest merge ratio at which reasoning on the calibration dataset is at least the minimum threshold.

4 Experimental Evaluation

In this section, we evaluate the trade-off achieved by our method between target-task performance and general reasoning capabilities across four reasoning models and two datasets.

4.1 Experimental Setup

Models We compare four recent RLMs: OpenThinker 7B (Guha et al., 2025), Qwen R1-D 7B (DeepSeek-AI, 2025), Olmo3 7B (Team Olmo, 2025), and Apriel 15B (Radhakrishna et al., 2025). These models cover diverse training approaches and model families: OpenThinker and Qwen R1-D are post-trained variants of Qwen2.5 7B (Yang et al., 2024a), obtaining reasoning capabilities via distillation on reasoning traces generated by DeepSeek R1 (DeepSeek-AI, 2025). Apriel and Olmo3 are based on pre-trained base models and post-trained via IFT, DPO (only Olmo3) and RLVR.

Training datasets We train all RLMs on two tasks: Rust coding and text summarization. While Rust coding requires logical reasoning and knowledge of the Rust language, text summarization requires deep textual understanding. Additionally, text summarization represents a domain without reliable verifiers, for which standard RL may be subject to reward hacking (Amodei et al., 2016).

We curate two datasets as $\mathcal{D}_{\text{train}}$. (1) *Rust coding*: A set of synthetic single-function coding tasks in the Rust language (Oxen.ai Team, 2024b), filtered to test-passing instances, resulting in 6,761 training samples. We reserve 750 samples for $\mathcal{D}_{\text{cal}}$. (2) *Text summarization*: The 10,000-sample Reddit TLDR split of Stiennon et al. (2022), a collection of user-written Reddit posts with self-written summaries. We reserve 400 samples for $\mathcal{D}_{\text{cal}}$. Additional details on dataset construction in §A.1.

Metrics For Rust coding, we use the Rust-translated MBPP (Cassano et al., 2022) and measure pass@1 (Chen et al., 2021), reported as *MBRP*. For text summarization, we report the average of fluency, consistency, relevance, and coherence on the CNN/Daily Mail split of the SummEval dataset (Fabbri et al., 2021) as *SummEval*. Precise score definitions are in §A.2. The training and test datasets are intentionally drawn from slightly different distributions while retaining similar task formats to measure generalization beyond training settings.

We further evaluate out-of-distribution model forgetting and general reasoning capabilities via independent benchmarks, denoted as *OOD*. OOD is the equal-weight average of MATH500 (Lightman et al., 2023), IF-Bench (Pyatkin et al., 2025), and GPQA Diamond (Rein et al., 2023). Separately, we report the rate of non-empty reasoning traces on the evaluation dataset (*Reasoning*).

Methods Unless otherwise mentioned, our method (+Merge) is implemented with linear merging, binary search, up to eight search steps, with $\rho_{\text{min}} = 1$. We report the unadapted RLM (*Unadapted*), instruction fine-tuning (*IFT*), and two additional baselines: First, On-Policy Distillation (+OPD) (Agarwal et al., 2024; Lu, 2025) uses an unrelated instruction fine-tuning dataset (Lambert et al., 2024) to distill unadapted RLM behavior into the student model for a small number of steps. Second, we report RL-based methods (*RL*). For Rust, we use the test suites of the training dataset as a verifiable reward for RLVR (Shao et al., 2024). For text summarization, we implement RLAIF (Lee et al., 2024), using each unadapted model as an output judge for generated outputs.

Protocol For all methods, we tune their hyperparameters in a grid sweep against $\mathcal{D}_{\text{cal}}$. We tune the learning rate, batch size, and number of epochs for each combination of model, method, and dataset, and report the hyperparameters in §A.3. Each setting uses two training seeds, with 10 samples for Rust, two for summarization, and one per OOD task.

For scores, we **boldface** the best method per model and underline insignificantly different methods, determined via a one-sided paired Wilcoxon signed-rank permutation test with Holm correction ($p_{\text{adj}} \geq 0.05$). We **boldface** the lowest cost. In all plots, we draw bands indicating the minimum and maximum average scores obtained by each training run.

4.2 Main Results

Overall, we find that our method increases target task performance, maintains OOD performance and reasoning behavior, and is significantly cheaper than comparable baselines.

Target-task improvements Our method +Merge improves the target score in all eight model–task settings, averaging $+5.56\%$ on Rust and $+2.56\%$ on text summarization. It maintains the performance gains of IFT, on which it is based, with -0.55% and -0.60% on average, respectively, and performs better for 2 of 4 models on Rust, and better or tied for 2 models in text summarization.

Table 1: Across four models, our method (+MERGE) improves both target tasks in all eight settings and preserves mean Reasoning at 99.7% and 99.7% on Rust (MBRP) and text summarization (SE). Mean OOD changes are +0.60% and −1.94%, respectively. Costs are lower than RL and OPD.

		Rust coding				Text summarization			
Model	Method	MBRP	OOD	Reasoning	Cost	SE	OOD	Reasoning	Cost
OpenThinker 7B	Unadapted	56.9	45.4	**100.0%**	$0.00	83.8	46.9	99.7%	$0.00
	RL	58.6	**47.3**	**100.0%**	$22.63	30.9	47.6	**100.0%**	$50.61
	IFT	57.3	32.9	0.0%	$2.55	**91.6**	20.0	0.0%	$0.81
	+ OPD	55.6	45.2	**100.0%**	$3.32	74.8	**47.8**	11.8%	$1.65
	+MERGE	**60.7**	46.0	98.6%	**$2.84**	88.2	45.0	**100.0%**	**$1.08**
Qwen R1-D 7B	Unadapted	30.6	51.0	**100.0%**	$0.00	78.3	51.9	**100.0%**	$0.00
	RL	31.8	**51.3**	**100.0%**	$18.85	40.8	**52.3**	**100.0%**	$10.56
	IFT	**50.1**	48.6	63.5%	$3.85	81.3	46.0	0.8%	$3.67
	+ OPD	44.1	50.0	**100.0%**	$5.00	**83.6**	51.8	**100.0%**	$4.38
	+MERGE	41.6	50.8	**100.0%**	**$4.09**	79.3	49.9	99.9%	**$3.86**
Olmo3 7B	Unadapted	46.2	59.9	**100.0%**	$0.00	88.9	**61.6**	**100.0%**	$0.00
	RL	46.3	**63.1**	**100.0%**	$37.41	88.8	61.4	**100.0%**	$98.77
	IFT	**50.0**	59.4	**100.0%**	$1.74	**92.4**	58.6	**100.0%**	$0.93
	+ OPD	45.8	60.3	**100.0%**	$2.82	90.9	60.4	**100.0%**	$1.18
	+MERGE	**50.0**	59.4	**100.0%**	**$1.88**	**92.4**	58.6	**100.0%**	**$1.07**
Apriel 15B	Unadapted	62.8	56.9	**100.0%**	$0.00	95.6	58.4	**100.0%**	$0.00
	RL	64.9	58.3	**100.0%**	$25.70	81.8	**58.9**	**100.0%**	$26.53
	IFT	63.6	56.5	33.1%	$8.77	94.0	43.4	35.2%	$1.85
	+ OPD	63.4	56.8	**100.0%**	$11.34	**97.2**	53.9	**100.0%**	$3.60
	+MERGE	**66.5**	**59.4**	**100.0%**	**$9.17**	97.0	57.7	98.9%	**$2.25**

Meanwhile, +OPD reduces the Rust score relative to IFT for all four models. In §A.5, we find that IFT in text summarization mostly changes consistency but degrades relevance, which is recovered by +MERGE and +OPD. For Rust, RL improves target task performance consistently by 1.26%. The average decrease of RL on text summarization is −26.1%. For all models, text scores decline relative to the unadapted model, most significantly OpenThinker and Qwen R1-D. We link this to the poor supervision signal in RLAIF: Both affected models often failed to prefer human-written summaries to their own or even to produce parseable grading scores during rollouts.

Preservation of OOD and Reasoning Direct IFT lowers OOD performance in all eight settings by an average of 3.94% on Rust and 12.72% on text summarization. Meanwhile, mean reasoning rates drop to 49.2% and 34.0%. Merging recovers the unadapted models' mean OOD scores, inducing changes of only +0.60% and −1.94% on average. At the same time, our method restores reasoning rates to 99.7% on both tasks. The majority of OOD losses occur from reasoning-heavy MATH500, we provide additional details in §A.5. RL preserves reasoning completely and obtains the highest mean OOD scores. This comes at the price of significantly higher cost.

Runtime and cost RL is the most expensive evaluated method in all eight settings. Even though we apply only 100 training steps, whereas typical setups use several thousand (DeepSeek-AI, 2025; Team Olmo, 2025), it averages 7.9 h for Rust coding. For RLAIF in text summarization, the average is 14.1 h, as additional model inference is required for model scoring. At a representative rental price of USD $3.31 per H200 GPU per hour (Silicon Data, 2026), this results in an average RL training costs of USD $26.15 and $46.62 for Rust coding and text summarization, respectively.

IFT takes 76.6 min for Rust coding and 32.9 min for text summarization on average. Because our method +MERGE employs only lightweight operations, it takes only 4.9 min on average for Rust and 4.5 min for text summarization on top of IFT time. In cost, this adds an average of only USD $0.27 and $0.25 to the corresponding IFT costs of USD $4.23 and $1.82, respectively.

In contrast, +OPD requires additional training steps, adding USD $1.39 and $0.89 for Rust coding and text summarization, respectively, 5.2 and 3.6 times more than our merging. In practice, this cost is paid several times, as the setup requires an additional hyperparameter search for learning rate and number of steps to conduct for OPD. This search is included in the costs of our merge ratio search, but not in the OPD costs.

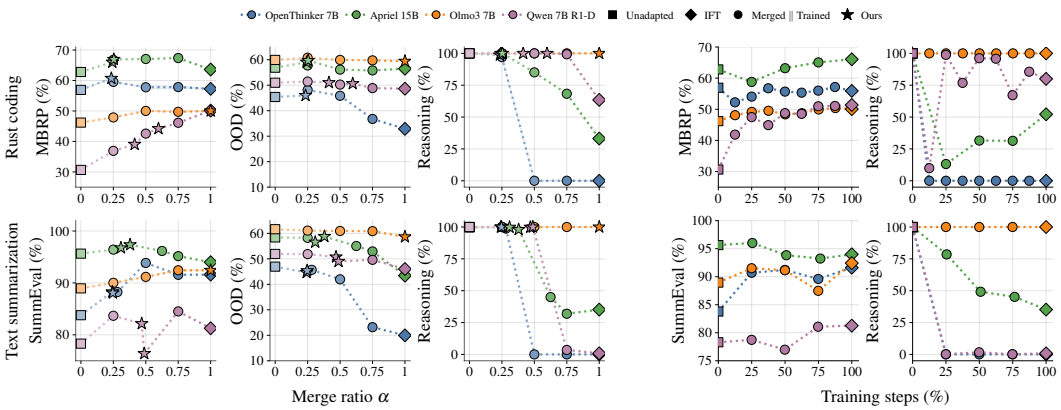


Figure 3: (Left) Across merge ratios, a 10-point reasoning decrease is associated with a 0.94-point OOD decrease. Ratios selected by +MERGE retain 98.6 to 100% reasoning. (Right) During IFT training, target-task performance increases slowly, while reasoning behavior quickly collapses.

Table 2: Across the models that exhibited reasoning loss, aborting before reasoning loss (+ABORT) falls short of the IFT target score by 7.14% on Rust and 11.32% on text summarization. Forcing M_{IFT} to emit a non-empty trace (+FORCE) lowers OOD performance relative to the unadapted model by 1.05% and 9.89%, respectively.

	OpenThinker				Apriel				Qwen R1-D			
Method	Rust	OOD	Text	OOD	Rust	OOD	Text	OOD	Rust	OOD	Text	OOD
Unadapted	56.9	45.4	83.8	**46.9**	62.8	56.9	95.6	**58.4**	30.6	51.0	78.3	**51.9**
IFT	57.3	32.9	**91.6**	20.0	63.6	56.5	94.0	43.4	**50.1**	48.6	**81.3**	46.0
+ Abort	57.4	**47.8**	83.4	46.9	63.2	58.9	72.2	56.6	28.9	**52.2**	77.3	51.3
+ Force	43.2	45.1	52.7	30.4	**66.5**	55.4	92.6	55.4	49.5	49.5	39.7	41.8
+MERGE	**60.7**	46.0	88.2	45.0	66.5	**59.4**	**97.0**	57.7	41.6	50.8	79.3	49.9

Other methods Prior works have suggested training RLMs using self-distillation (SDFT) (Shenfeld et al., 2026) or reverse loss on reasoning traces from golden answers (VeriFree) (Zhou et al., 2025). They require substantially more resources than our approach. For example, SDFT reports an expected cost of 4 times the cost of IFT. In our experiments, it used over 20 times our GPU resources: 4.5 h on 4 H200 GPUs. We estimate VeriFree would require around 70 h on 4 H200 GPUs. We report our attempts to reproduce both methods in §A.7.

4.3 ABLATION AND ADDITIONAL ANALYSIS

Merging ratio choice Our method is based purely on reasoning behavior, yet it preserves OOD performance. We investigate the relationship of reasoning behavior with model performance across a five-ratio uniform grid. As shown in the left panels of Figure 3, across the 40 available points, a 10-point decrease in reasoning rate is associated with a 0.94-point decrease in OOD performance within model–task trajectories. Further, reasoning rate typically degrades quickly after a model-specific threshold. Even though we choose a maximal merge ratio that preserves reasoning, the ratio selected by our method retains reasoning rates between 98.6% and 100%.

Meanwhile, selected ratios fall within one percent point of the best evaluated target score in 5 of 8 model–task settings, with a median gap of 0.60%. Overall, we find that our method selects high-reasoning points that provide a reliable trade off between generalization and target task learning. We analyse the change in reasoning length in §A.6 and find no clear relationship with model performance, in contrast to reasoning behavior.

Behavior changes during training An alternative to merging could be to abort training when a change in reasoning behavior is observed. We implement an early stopping control (+ABORT): every 5 training steps, we stop when the reasoning rate on $\mathcal{D}_{\text{cal}}$ falls below 90% and keep the last

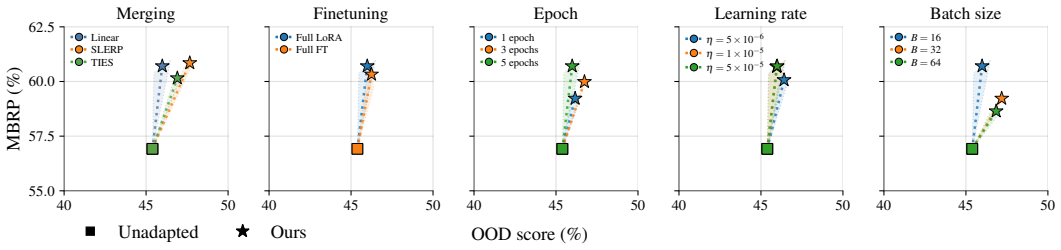


Figure 4: OpenThinker ablations on Rust coding, from left to right: merging techniques, LoRA vs. full fine-tuning, training epochs, learning rate η, and batch size B. Across the seven hyperparameter configurations, the trend of preserved OOD performance and improved target performance holds.

checkpoint before this threshold. Since Olmo3 does not change reasoning behavior, we exclude it from this comparison. Five of the remaining six settings cross this threshold within the first quarter of training (Figure 3); the plotted checkpoints are too sparse to show the first few steps. Consequently, early stopping underperforms full IFT in 5 of 6 settings, with mean target-score deficits of 7.14% on Rust and 11.32% on summarization (Table 2).

Causality of reasoning for performance Finally, we investigate whether the presence of reasoning, without any model training, recovers OOD performance. We force the IFT model to reason at inference time (+*Force*) by preventing it from closing the reasoning trace without producing any content. As shown in Table 2, this intervention exceeds the unmodified IFT target score in only 1 of 6 settings and remains below the OOD score of the untrained RLM in all six. Therefore, forcing a non-empty trace is insufficient to recover performance under our intervention.

Method variants For OpenThinker on Rust, we compare our linear merging (Linear), Spherical Linear Interpolation (SLERP) (Shoemake, 1985; Goddard et al., 2025), and TIES (Yadav et al., 2023), shown in the leftmost panel of Figure 4. Target and OOD performance gains are $+3.79\%$ and $+0.60\%$ for Linear, $+3.93\%$ and $+2.28\%$ for SLERP, and $+3.23\%$ and $+1.52\%$ for TIES, respectively. While SLERP slightly outperforms linear merging, the results demonstrate that the overall trend is independent of merging technique.

In the second panel of Figure 4, we ablate the fine-tuning method, using full fine-tuning (Full FT) of all model weights. Before merging, Full FT and LoRA reduce OOD performance by -34.11% and -12.51%, respectively. Merging recovers these to $+0.86\%$ and $+0.60\%$, respectively. Again, the fine-tuning technique does not influence the trend.

Hyperparameter Ablation We ablate fine-tuning hyperparameters for the IFT stage in the three rightmost panels of Figure 4. Across seven configurations varying the number of epochs, learning rate, and batch size, our merging improves target task performance on average 2.87%. The remarkably stable performance may be caused by the adaptive reasoning ratio choice: Stronger nominal training settings generally select smaller merge ratios, α and $(\text{epochs} \times \eta \times 16/B)$ have Spearman $\rho = -0.955$.

5 Related Work

We list prior work on adaptation of RLMs and model merging in general.

Training reasoning models with verifiers The standard technique for training and adapting reasoning models uses RLVR (DeepSeek-AI, 2025; Team Olmo, 2025; Stojanovski et al., 2025). This works well for tasks that permit such verifiers, but leaves out many domains, such as text summarization. Moreover, while this technique is well explored for bootstrapping reasoning model performance (DeepSeek-AI, 2025; Team, 2025; Team Olmo, 2025), continuous adaptation of RLMs to new tasks, as done in our setting, remains underexplored.

Training reasoning models without verifiers Verifier-free or verifier-light methods try to enable training reasoning models without reasoning traces by optimizing the likelihood of known answers directly (Zhou et al., 2025) or training teacher models to provide feedback and justifications for

reference answers (Shenfeld et al., 2026). These methods are close in motivation to our setting, but they are designed to elicit reasoning from models not previously trained for reasoning tasks.

Lu (2025) uses on-policy distillation from the base RLM on an unrelated instruction dataset to recover reasoning behavior after IFT. As we show in our experiments, our method is less expensive, does not introduce another hyperparameter for tuning, and does not depend on an external dataset.

Reasoning distillation and synthetic traces Assuming the availability of a stronger RLM on the desired task, reasoning distillation on synthetic traces has been proposed. In reasoning distillation, for each task $\mathbf{x}$ in the dataset, a reasoning trace $\mathbf{r}$ and answer $\hat{\mathbf{y}}$ are obtained by sampling a stronger RLM (DeepSeek-AI, 2025; Guha et al., 2025; Yang et al., 2025). Supervised fine-tuning is then performed on the new dataset with pairs $(\mathbf{x}, (\mathbf{r}, \hat{\mathbf{y}}))$. However, this approach requires the availability of a stronger RLM on the desired task. In this work, we explore a setting where such a model cannot be obtained, since a stronger model is not available for any frontier training setups.

Model merging Model merging combines multiple checkpoints into a single model. Prior methods include simple weight averaging (Goddard et al., 2025), task arithmetic (Ilharco et al., 2023), and sparse or sign-based variants such as TIES (Yadav et al., 2023). Merging is usually performed to combine task-specific skills or mitigate forgetting (Alexandrov et al., 2024; Yang et al., 2024b; Matena & Raffel, 2022). Our use is narrower: we merge an IFT checkpoint back with its own original reasoning checkpoint to trade off target-task adaptation against preservation of reasoning behavior. Recent work on tunable reasoning through model merging suggests that interpolation can control the degree of reasoning behavior in language models (Lan et al., 2025).

6 Discussion and Future Work

Inconsistent reasoning loss In our experiments, we observe that OpenThinker, Qwen R1-D and Apriel lose their reasoning behavior in all settings, while Olmo3 never loses its reasoning behavior. We investigate this further and find that in one hyperparameter sweep setting, at learning rate $\eta = 10^{-3}$ and 1 epoch, we observe reasoning loss for Olmo3 too, resulting in around 86% reasoning. We find that for this setting, the norm of the weight update is 167.29, more than 10 times the norms of our main experiments, 14.80 and 14.82. However, in our validation, this setting underperforms the hyperparameters chosen for our main experiments. This indicates interesting future work on exploring the relationship between task adaptation, weight updates, and reasoning loss more closely.

Lightweight adaptation on reasoning domains Our method provides a lightweight way to adapt reasoning models to new tasks by leveraging instruction-completion pairs, following the same paradigm as in the pre-RLM era. However, more knowledge may provide great benefit on domains where the quality of reasoning itself is relevant to performance improvements, such as mathematics. Indeed, in preliminary experiments, we applied our method to mathematical proofs, and did not observe significant target-task improvements. We believe this remains interesting future work.

Composability and continual learning Our results demonstrate how to adapt RLMs to a single task using IFT data while preventing changes in reasoning behavior or the loss of out-of-distribution knowledge. An interesting direction for future work is to explore the degree of composability enabled by this finding, as it potentially enables new approaches to continual learning.

7 Conclusion

We studied how to adapt RLMs with standard input-output pairs. We introduce a simple two-step pipeline to adapt RLMs using IFT data: first fine-tune the RLM with standard IFT, then merge the fine-tuned checkpoint with the untuned model. The merge ratio is selected using the reasoning rate on a target-task calibration set. Across the eight evaluated model–task settings, this procedure improves the target score by an average of 5.56% and 2.56% on Rust coding and text summarization, respectively. Meanwhile, it preserves out-of-distribution performance, with average changes of only $+0.60\%$ and -1.94%. Our ablations show the same qualitative recovery across seven hyperparameter configurations, full fine-tuning, and three merging algorithms. Our method operates without a verifier, expensive rollouts, or stronger teacher. More importantly, it is cheaper than the evaluated OPD and RL baselines by on average 22.3% and 84.6%, respectively.

AI USE STATEMENT

In this work, we used generative AI tools for feedback on research methodology or experiments, method implementation, and support for qualitative and thematic data analysis. We did not use generative AI tools for result interpretation, the design of research methodology or experiments, or the proposal or refinement of hypotheses; the formulation or proof of mathematical claims is not applicable to this work. The dataset by Oxen.ai Team (2024a) that we used was generated synthetically using generative AI by a third party. Additionally, we used generative AI tools for the creation or modification of scientific figures, the execution and monitoring of training runs, the discovery of related research, and brainstorming. We have reviewed all AI-assisted work. LLM-generated code and training configurations were reviewed by the first author, created figures were cross-checked against raw data, and qualitative data analysis was conducted manually based only on pointers generated by generative AI. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We specify the proposed method in §3 and Algorithm 1. The models, datasets, baselines, evaluation metrics, statistical tests, sampling protocol, and hyperparameter-selection procedure are described in §4. The appendix documents the Rust dataset filtering and splits (§A.1), scoring procedure and judge validation (§A.2), complete training and inference hyperparameters (§A.3), prompts (§C), detailed SummEval and per-benchmark results (§A.5), and hardware and runtime measurements (§A.8). The anonymous supplementary material includes the code and processed data needed to reproduce the experiments and generate the reported tables and figures.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. *arXiv*, 2306.13649, 2024. URL https://arxiv.org/abs/2306.13649.

Anton Alexandrov, Veselin Raychev, Mark Niklas Müller, Ce Zhang, Martin T. Vechev, and Kristina Toutanova. Mitigating catastrophic forgetting in language transfer via model merging. In *Findings of EMNLP*, 2024.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv*, 1606.06565, 2016. URL https://arxiv.org/abs/1606.06565.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, and others. MultiPL-E: A scalable and extensible approach to benchmarking neural code generation. *arXiv*, 2208.08227, 2022. URL https://arxiv.org/abs/2208.08227.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and others. Evaluating large language models trained on code. *arXiv*, 2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, and others. Scaling instruction-finetuned language models. *arXiv*, 2210.11416, 2022. URL https://arxiv.org/abs/2210.11416.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv*, 2501.12948, 2025. URL https://arxiv.org/abs/2501.12948.

Alexander R. Fabbri, Wojciech Kryściński, Bryan McCann, Caiming Xiong, Richard Socher, and Dragomir Radev. SummEval: Re-evaluating summarization evaluation. *arXiv*, 2007.12626, 2021. URL https://arxiv.org/abs/2007.12626.

Hua Farn, Hsuan Su, Shachi H Kumar, Saurav Sahay, Shang-Tse Chen, and Hung yi Lee. Safeguard fine-tuned LLMs through pre- and post-tuning model merging. *arXiv*, 2412.19512, 2025. URL https://arxiv.org/abs/2412.19512.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. *arXiv*, 1912.05671, 2020. URL https://arxiv.org/abs/1912.05671.

Charles Goddard, Shamane Siriwardhana, Malikeh Ehghaghi, Luke Meyers, Vlad Karpukhin, Brian Benedict, Mark McQuade, and Jacob Solawetz. Arcee's MergeKit: A toolkit for merging large language models. *arXiv*, 2403.13257, 2025. URL https://arxiv.org/abs/2403.13257.

Google DeepMind. Gemini Pro, 2025. URL https://deepmind.google/technologies/gemini/pro/.

Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, and others. OpenThoughts: Data recipes for reasoning models. *arXiv*, 2506.04178, 2025. URL https://arxiv.org/abs/2506.04178.

Jingxuan He, Mark Vero, Gabriela Krasnopolska, and Martin Vechev. Instruction tuning for secure code generation. *arXiv*, 2402.09497, 2024. URL https://arxiv.org/abs/2402.09497.

Or Honovich, Thomas Scialom, Omer Levy, and Timo Schick. Unnatural instructions: Tuning language models with (almost) no human labor. *arXiv*, 2212.09689, 2022. URL https://arxiv.org/abs/2212.09689.

Baizhou Huang and Xiaojun Wan. Pros: Towards compute-efficient rlvr via rollout prefix reuse. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 45546–45562, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/4badb55ba9c18bccb9d4146be328a948-Paper-Conference.pdf.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. *arXiv*, 2212.04089, 2023. URL https://arxiv.org/abs/2212.04089.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, and others. Tülu 3: Pushing frontiers in open language model post-training. *arXiv*, 2411.15124, 2024. URL https://arxiv.org/abs/2411.15124.

Xiaochong Lan, Yu Zheng, Shiteng Cao, and Yong Li. The thinking spectrum: An empirical study of tunable reasoning in LLMs through model merging. *arXiv*, 2509.22034, 2025. URL https://arxiv.org/abs/2509.22034.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, and others. RLAIF vs. RLHF: Scaling reinforcement learning from human feedback with AI feedback. *arXiv*, 2309.00267, 2024. URL https://arxiv.org/abs/2309.00267.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv*, 2305.20050, 2023. URL https://arxiv.org/abs/2305.20050.

Elita Lobo, Chirag Agarwal, and Himabindu Lakkaraju. On the impact of fine-tuning on chain-of-thought reasoning. *arXiv*, 2411.15382, 2025. URL https://arxiv.org/abs/2411.15382.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, and others. StarCoder 2 and The Stack v2: The next generation. *arXiv*, 2402.19173, 2024. URL https://doi.org/10.48550/arXiv.2402.19173.

Kevin Lu. On-policy distillation. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20251026. https://thinkingmachines.ai/blog/on-policy-distillation.

Lu Ma, Hao Liang, Meiyi Qiang, Lexiang Tang, Xiaochen Ma, Zhen Hao Wong, Junbo Niu, Chengyu Shen, Runming He, Yanhao Li, and others. Learning what reinforcement learning can't: Interleaved online fine-tuning for hardest questions. In *ICLR*, 2026. URL https://arxiv.org/abs/2506.07527.

Michael S Matena and Colin Raffel. Merging models with Fisher-weighted averaging. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *NeurIPS*, volume 35, pp. 17703–17716. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/70c26937fbf3d4600b69a129031b66ec-Paper-Conference.pdf.

Niklas Muennighoff, Qian Liu, Armel Randy Zebaze, Qinkai Zheng, Binyuan Hui, Terry Yue Zhuo, Swayam Singh, Xiangru Tang, Leandro von Werra, and Shayne Longpre. OctoPack: Instruction tuning code large language models. In *ICLR*, 2024. URL https://openreview.net/forum?id=mw1PWNSWZP.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, and others. GPT-4 technical report. *arXiv*, 2303.08774, 2024a. URL https://arxiv.org/abs/2303.08774.

OpenAI, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, and others. OpenAI o1 system card. *arXiv*, 2412.16720, 2024b. URL https://arxiv.org/abs/2412.16720.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and others. Training language models to follow instructions with human feedback. *arXiv*, 2203.02155, 2022. URL https://arxiv.org/abs/2203.02155.

Oxen.ai Team. Rust verified code dataset. https://www.oxen.ai/ox/Rust, 2024a. Accessed: 2025-01-01.

Oxen.ai Team. Training a Rust 1.5B coder LM with reinforcement learning (GRPO). https://ghost.oxen.ai/training-a-rust-1-5b-coder-lm-with-reinforcement-learning-grpo/, 2024b. Technical report.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following. In *NeurIPS*, volume 38, 2025.

Shruthan Radhakrishna, Soham Parikh, Gopal Sarda, Anil Turkkan, Quaizar Vohra, Raymond Li, Dhruv Jhamb, Kelechi Ogueji, Aanjaneya Shukla, Oluwanifemi Bamgbose, and others. Apriel-Nemotron-15B-Thinker. *arXiv*, 2508.10948, 2025. URL https://arxiv.org/abs/2508.10948.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level Google-proof Q&A benchmark. *arXiv*, 2311.12022, 2023. URL https://arxiv.org/abs/2311.12022.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and others. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv*, 2402.03300, 2024. URL https://arxiv.org/abs/2402.03300.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. *arXiv*, 2601.19897, 2026. URL https://arxiv.org/abs/2601.19897.

Ken Shoemake. Animating rotation with quaternion curves. In *SIGGRAPH*, SIGGRAPH '85, pp. 245–254, New York, NY, USA, 1985. Association for Computing Machinery. ISBN 0897911660. doi: 10.1145/325334.325242. URL https://doi.org/10.1145/325334.325242.

Silicon Data. H200 rental price index, 2026. URL https://web.archive.org/web/20260924105358/https://www.silicondata.com/products/silicon-index/h200. Archived on September 24, 2026.

Zayne Sprague, Fangcong Yin, Juan Diego Rodriguez, Dongwei Jiang, Manya Wadhwa, Prasann Singhal, Xinyu Zhao, Xi Ye, Kyle Mahowald, and Greg Durrett. To CoT or not to CoT? chain-of-thought helps mainly on math and symbolic reasoning. *arXiv*, 2409.12183, 2025. URL https://arxiv.org/abs/2409.12183.

Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. Learning to summarize from human feedback. *arXiv*, 2009.01325, 2022. URL https://arxiv.org/abs/2009.01325.

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning Gym: Reasoning environments for reinforcement learning with verifiable rewards. *arXiv*, 2505.24760, 2025. URL https://arxiv.org/abs/2505.24760.

Qwen Team. QwQ-32B: Embracing the power of reinforcement learning, March 2025. URL https://qwenlm.github.io/blog/qwq-32b/.

Team Olmo. Olmo 3. *arXiv*, 2512.13961, 2025. URL https://arxiv.org/abs/2512.13961.

Lukas Twist, Helen Yannakoudakis, and Jie M. Zhang. Reasoning-trace collapse: Evaluating the loss of explicit reasoning during fine-tuning. *arXiv*, 2605.21127, 2026. URL https://arxiv.org/abs/2605.21127.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-Instruct: Aligning language models with self-generated instructions. *arXiv*, 2212.10560, 2023. URL https://arxiv.org/abs/2212.10560.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners. *arXiv*, 2109.01652, 2022. URL https://arxiv.org/abs/2109.01652.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. TIES-merging: Resolving interference when merging models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=xtaX3WyCj1.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and others. Qwen2.5 technical report. *arXiv*, 2412.15115, 2024a. URL https://doi.org/10.48550/arXiv.2412.15115.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and others. Qwen3 technical report. *arXiv*, 2505.09388, 2025. URL https://arxiv.org/abs/2505.09388.

Enneng Yang, Li Shen, Guibing Guo, Xingwei Wang, Xiaochun Cao, Jie Zhang, and Dacheng Tao. Model merging in LLMs, MLLMs, and beyond: Methods, theories, applications and opportunities. *arXiv*, 2408.07666, 2024b. URL https://arxiv.org/abs/2408.07666.

Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, and others. Instruction tuning for large language models: A survey. *arXiv*, 2308.10792, 2025. URL https://arxiv.org/abs/2308.10792.

Yiming Zhang, Baoyi He, Shengyu Zhang, Yuhao Fu, Qi Zhou, Zhijie Sang, Zijin Hong, Kejing Yang, Wenjun Wang, Jianbo Yuan, and others. Unconstrained model merging for enhanced LLM reasoning. *arXiv*, 2410.13699, 2024. URL https://arxiv.org/abs/2410.13699.

Xiangxin Zhou, Zichen Liu, Anya Sims, Haonan Wang, Tianyu Pang, Chongxuan Li, Liang Wang, Min Lin, and Chao Du. Reinforcing general reasoning without verifiers. *arXiv*, 2505.21493, 2025. URL https://arxiv.org/abs/2505.21493.

A Experimental Details and Ablations

In this section, we provide additional details about the implementation, hyperparameters, and datasets.

A.1 Dataset Curation

Rust coding In our study, the Rust corpus from Oxen.ai Team (2024a;b) is used for supervised fine-tuning and in-distribution evaluation. Each sample in the corpus consists of four fields: a task identifier, a natural-language Rust prompt that describes the target function, a Rust code implementation that solves the task, and a list of executable test cases. Consequently, the dataset provides complete function-level programming tasks paired with verification harnesses, enabling evaluation of generated code correctness.

Before using the dataset, we performed verification and filtering to improve data quality and ensure experimental reproducibility. For each code-test pair, we compiled the provided Rust implementation and executed its associated test suite ten times in our local environment. We retained only samples for which the implementation successfully compiled and passed the complete test suite in all ten runs. This repeated-execution protocol filters out examples whose correctness is unstable across executions, including cases affected by nondeterministic behavior or code-test pairs that only pass stochastically. We did not manually repair such cases; any sample that failed compilation or testing in at least one run was excluded from the final dataset.

Starting from the original 7,554 Rust tasks, this filtering process retained 7,511 valid samples. We then split the filtered dataset into 6,761 examples for supervised fine-tuning and 750 examples for validation. Each entry in the final split retains the same four-field structure: task identifier (`task_id`), Rust prompt (`rust_prompt`), Rust implementation (`rust_code`), and the complete list of test cases (`rust_test_list`).

We confirm that the obtained datasets are disjoint from the MBRP dataset by checking for closest matches using cosine similarity.

Text summarization We sample 10,000 and 400 instances from the TL;DR set of Stiennon et al. (2022) for our training and calibration set respectively. The core of the 10,000 samples is formed by 9,113 samples with at least 20 summary words and 150 post words, and 887 additional random samples to fill to 10,000.

A.2 Details of Task Scoring

Rust coding For Rust coding, we use the test suite provided by the MBPP translation as scorer. The model output receives a score of 1 when all test cases pass, and 0 otherwise.

Text summarization For text summarization, we report SummEval, the average of the four metrics fluency, consistency, relevance, and coherence on the CNN/Daily Mail split of the SummEval dataset (Fabbri et al., 2021). Each metric is assigned a score between 1 and 5 by Gemini 3.1 Pro Preview (Google DeepMind, 2025) based on clearly outlined criteria, shown in §C. We multiply these scores by 20 and report them as percentages, so a rating of 5 corresponds to 100%.

To confirm the validity of this LLM-as-a-judge setup, we compare the Gemini 3.1 Pro Preview ratings with human annotations provided by Fabbri et al. (2021) and obtain a mean Spearman rank correlation coefficient of $\bar{r}_s = 0.65$, $p < 0.001$ across the four metrics, indicating a consistently high level of agreement.

A.3 Hyperparameters

We run each training twice with seed 42 and 43.

When using LoRA, we train the query, key, value, and output projections in self-attention, as well as the gate, up, and down projections in the feed-forward network. We tune the IFT hyperparameters separately for each model and dataset combination using a validation set as training target. Table 4 reports the resulting configurations together with the baseline and ablation hyperparameters. The

Table 3: SummEval correlations for Gemini 3.1 Pro Preview.

Metric	Spearman ρ	p-value
Fluency	0.6280	1.69×10^{-182}
Relevance	0.5917	1.66×10^{-159}
Coherence	0.7058	3.30×10^{-253}
Consistency	0.6793	2.89×10^{-227}

Table 4: Hyperparameters used for the main experiments, baselines, and ablations. Batch size denotes the effective training batch size; r denotes LoRA rank. For RL (RLVR for Rust, RLAIF for text summarization), training length lists the validation-selected checkpoint steps for runs 1 / 2, from a budget of 100 updates per run.

Dataset	Model	Method	Training length	Learning rate	Batch size	Extra
Rust	OpenThinker	IFT	5 epochs	5×10^{-5}	16	$r = 32$
Rust	Olmo3	IFT	3 epochs	5×10^{-5}	16	$r = 32$
Rust	Apriel	IFT	5 epochs	1×10^{-4}	64	$r = 32$
Rust	Qwen R1-D	IFT	5 epochs	2×10^{-4}	16	$r = 64$
Text sum.	OpenThinker	IFT	1 epoch	5×10^{-5}	16	$r = 32$
Text sum.	Olmo3	IFT	1 epoch	1×10^{-5}	16	$r = 32$
Text sum.	Apriel	IFT	1 epoch	1×10^{-4}	64	$r = 32$
Text sum.	Qwen R1-D	IFT	3 epochs	2×10^{-4}	32	$r = 64$
All	All	+FORCE	IFT config			force first 10 tokens
Rust	OpenThinker	RL	75 / 25 steps	2×10^{-5}	64	$r = 32$
Rust	Olmo3	RL	100 / 25 steps	2×10^{-5}	64	$r = 32$
Rust	Apriel	RL	50 / 75 steps	2×10^{-5}	64	$r = 32$
Rust	Qwen R1-D	RL	100 / 100 steps	2×10^{-5}	64	$r = 32$
Text sum.	OpenThinker	RL	100 / 100 steps	1×10^{-5}	64	$r = 32$
Text sum.	Olmo3	RL	100 / 100 steps	1×10^{-5}	64	$r = 32$
Text sum.	Apriel	RL	100 / 100 steps	1×10^{-5}	64	$r = 32$
Text sum.	Qwen R1-D	RL	100 / 100 steps	1×10^{-5}	64	$r = 32$
Rust	OpenThinker	+OPD	25 steps	2×10^{-4}	8	$r = 64$
Rust	Olmo3	+OPD	25 steps	2×10^{-4}	8	$r = 32$
Rust	Apriel	+OPD	50 steps	4×10^{-4}	4	$r = 32$
Rust	Qwen R1-D	+OPD	25 steps	2×10^{-4}	8	$r = 64$
Text sum.	OpenThinker	+OPD	25 steps	2×10^{-4}	8	$r = 64$
Text sum.	Olmo3	+OPD	5 steps	1×10^{-5}	8	$r = 32$
Text sum.	Apriel	+OPD	50 steps	4×10^{-4}	4	$r = 32$
Text sum.	Qwen R1-D	+OPD	25 steps	2×10^{-4}	8	$r = 64$
Rust	OpenThinker	Full FT	5 epochs	5×10^{-5}	16	all weights

+OPD rows only report the training parameters of the subsequent OPD training; the preceding IFT uses the corresponding IFT row.

During merge search, we sample 20 tokens plus the length of the longest end-of-reasoning marker to determine whether a model reasons. To further speed up the merging search, calibration evaluation is performed using vLLM in eager loading mode (skipping model graphs) with a batch size of 128. All merging runs are performed on a single GPU.

During inference, the maximum token limit is 16,384 tokens for all benchmarks. We consistently use temperature 0.6 and top-p 0.95. For OpenThinker, we use the default repetition penalty of 1.05.

A.4 RL SETUP

RLVR We use the same 6,761 Rust training problems used for IFT, each containing a problem statement, reference solution, and tests. RL discards the reference solution and validates outputs using the reference tests. We filter out prompts longer than 2,048 tokens. Validation requires complete Rust code. We compile the result with `rustc` and then execute the reference tests. The reward is 1 only if every test passes and 0 otherwise. Compilation errors, timeouts, crashes, and incomplete answers receive a reward of 0.

We train each RLM using GRPO and LoRA adapters with rank 32, $\alpha = 64$, and dropout 0. For each model, we run 100 updates with two seeds and a learning rate of 2×10^{-5}. Each update samples 64 completions: eight solutions for each of eight problems. For rollouts, the maximum model output is 8,192 tokens, the temperature is 0.8, and top-p is 1.

During training, we evaluate eight completions for each of 64 fixed problems from the separate 750-problem calibration set at steps 0, 25, 50, 75, and 100. We select the checkpoint with the highest mean test-pass rate, breaking ties in favor of the earliest checkpoint.

RLAIF For text summarization, we implement Reinforcement Learning from AI Feedback (RLAIF) using the frozen untrained RLM as an automated judge. We use the 10,000 training examples from the filtered Reddit TLDR split (Stiennon et al., 2022).

Each RLM is trained using GRPO (Shao et al., 2024) with the DAPO loss objective and group reward scaling. As in IFT, we train LoRA adapters with rank $r = 32$, $\alpha = 64$, and dropout 0, applied to all attention projections (`q_proj`, `k_proj`, `v_proj`, `o_proj`) and feed-forward layers (`gate_proj`, `up_proj`, `down_proj`). Training runs for 100 update steps across two random seeds (42 and 43) with a constant learning rate of 1×10^{-5} (0 warmup steps) using the AdamW optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$, weight decay 0). The KL divergence coefficient is set to $\beta = 0.05$. Each update step uses an effective batch size of 64 completions, sampling 8 completions for each of 8 prompts. Rollouts generate up to 8,192 tokens with temperature 0.8 and top-p 1.0. Training uses bfloat16, masks truncated completions from the loss, and applies token-level truncated importance-sampling corrections to vLLM rollouts.

During rollouts, the policy receives the standard summarization prompt (Figure 11) and is instructed to place its complete summary after the `<SUMMARY>` tag. Candidate summaries are extracted from the completions; if extraction fails or the output summary is empty, a reward of 0 is assigned without querying the judge. Across all four models and both random seeds, the step 100 checkpoint is used for evaluation.

To avoid reliance on stronger LLM judges, which are unavailable in our setting, each RLM acts as its own judge using its frozen base weights without adapter weights. The judge evaluates candidate summaries across the four independent SummEval dimensions: relevance, factual consistency, fluency, and coherence. Each dimension is queried in an independent context using the corresponding rubric prompt (Figures 12–15). For OpenThinker, the model-native OpenThoughts system prompt (**??**) is included to elicit systematic long thinking; the remaining models rely on their default native chat templates.

Judge generation uses temperature 0.6, top-p 0.95, unrestricted top-k, repetition penalty 1.0, and a maximum completion length of 16,384 tokens to permit extensive reasoning traces by the judge before outputting its final score. If a judge generation is truncated or fails to parse a valid numeric score, the same prompt is tried again with the next seed, for at most ten total attempts, starting at seed 0. Scores are assigned as for the main evaluation. The unnormalized quality score $\bar{S}$ is the arithmetic mean of the four dimensions:

$$\bar{S} = \frac{1}{4}\left(S_{\text{relevance}} + S_{\text{consistency}} + S_{\text{fluency}} + S_{\text{coherence}}\right) \in [1, 5]. \tag{3}$$

The final scalar reward normalized to $[0, 1]$ passed to GRPO is:

$$R = \frac{\bar{S} - 1.0}{4.0}. \tag{4}$$

A.5 Detailed Scores

SummEval We provide the individual SummEval dimensions in Table 5. They highlight a trade-off between consistency and relevance that is not visible in the average. For the OpenThinker, Qwen R1-D, and Apriel, all models whose reasoning behavior changes after IFT, direct IFT increases consistency but reduces relevance. Merging recovers relevance to near or above the base score, except for Qwen R1-D (94.7, 77.0, and 99.5, respectively), while keeping consistency above the corresponding base score (63.1, 56.1, and 89.4). Thus, the aggregate summarization gains after merging primarily combine recovered relevance with retained consistency gains. For RLAIF, all four SummEval dimensions decline substantially for OpenThinker, Qwen R1-D, and Apriel. Olmo3 is the exception, with its summarization dimensions remaining close to the unadapted model.

OOD We provide the individual OOD results in Table 5. The per-benchmark results show that forgetting is concentrated in particular model–benchmark combinations. For OpenThinker, direct IFT reduces MATH500 from 82.8 to 46.4 after Rust training and from 84.8 to 14.7 after summarization training; merging restores the scores to 83.7 and 83.3, respectively. Similarly, for Apriel trained on summarization, IFT reduces MATH500 from 96.0 to 83.3 and GPQA Diamond from 57.6 to 28.3, while merging restores them to 95.6 and 56.1. By contrast, Olmo3 changes less across the individual OOD benchmarks, and merging leaves its IFT checkpoint unchanged because its reasoning behavior is retained. RLAIF's individual OOD scores remain comparatively close to those of the unadapted models. In particular, Olmo3 remains close to the unadapted model across all three OOD benchmarks.

A.6 Reasoning Length and Performance

We additionally investigate whether producing a longer reasoning trace is associated with better performance. We count tokens before the model's native end-of-reasoning marker using its native tokenizer, excluding the reasoning markers themselves. Empty traces have length zero. We first average over examples and evaluation seeds, and then equally weight available training runs at each merge ratio or training checkpoint. Figure 5 shows the resulting trajectories.

Mean trace length confounds two distinct quantities: how often the model reasons and how long it reasons when it does. This distinction matters because the loss of reasoning behavior is already associated with capability loss in our experiments. To focus on length rather than presence, we restrict the analysis to points with at least 90% target-task reasoning and divide mean length by the non-empty-trace rate. This gives the mean length conditional on a non-empty trace. The retained points still span substantial ranges: 1,321–3,799 tokens for the Rust merge trajectories, 311–3,799 for Rust training, 276–1,923 for text merging, and 355–1,923 for text training.

Table 6 reports exploratory Spearman correlations pooled across models. We find no evidence that longer traces improve target-task performance: the four correlations are small ($\rho = -0.17$–0.07) and none is significant (Holm-adjusted $p = 1.00$ for all four). For example, Olmo3 on text summarization, mean non-empty reasoning length falls from 1,923 to 1,137 tokens between $\alpha = 0$ and $\alpha = 0.75$, while SummEval increases from 89.0 to 92.4 and reasoning remains at 100%. For Qwen R1-D on Rust, length falls from 2,144 to approximately 1,321 tokens by $\alpha = 0.60$, while MBRP rises from 30.6 to 44.2 and reasoning remains above 99%.

Some correlations with OOD are positive, especially for the text trajectories. These do not establish a benefit from longer reasoning: merge ratio and training progress jointly change trace length, task adaptation, and forgetting, so length is not independently manipulated. Moreover, neither text correlation remains significant after Holm correction across the eight exploratory tests in Table 6. We therefore interpret the results as evidence that preserving the *presence* of reasoning is informative, whereas the available experiments provide no evidence that making an already non-empty trace longer improves target-task performance.

A.7 Further Baseline Experiments

We attempted to reproduce two popular related methods that are also applicable to our setting of training RLMs using only standard IFT data.

Table 5: Detailed out-of-domain and SummEval results. We observe that GPQA Diamond and IF-Bench scores are stable for most models, in contrast to reasoning-heavy MATH500. In text summarization, the strongest improved metric by IFT is consistency, while relevance decreases.

		Rust coding			Text summarization			SummEval			
Model	Method	Math500	IFBench	GPQA D.	Math500	IFBench	GPQA D.	Fluency	Relevance	Coherence	Consistency
OpenThinker	Unadapted	82.8	17.0	36.4	84.8	18.7	37.4	99.3	89.2	**97.2**	49.4
	RL	**86.4**	19.0	36.6	84.9	17.5	**40.4**	33.7	31.8	33.2	24.8
	IFT	46.4	**21.7**	30.6	14.7	16.2	29.0	**99.5**	83.3	96.8	**86.9**
	+OPD	81.3	18.3	35.9	83.3	**21.3**	38.9	80.8	79.2	77.1	61.9
	+MERGE	83.7	19.7	34.6	83.3	16.8	34.8	99.5	**94.7**	95.3	63.1
	+Abort	84.7	20.3	**38.4**	**86.9**	18.8	35.1	97.8	89.7	94.3	51.9
	+Force	82.4	15.7	37.4	44.6	11.5	35.1	59.6	46.8	55.8	48.7
Qwen R1-D	Unadapted	**92.8**	12.7	47.5	**93.8**	14.0	48.0	99.3	84.8	83.7	45.4
	RL	92.6	13.7	47.7	92.3	13.0	**51.5**	48.7	42.5	43.1	28.8
	IFT	89.3	14.7	41.9	88.2	8.7	41.2	97.0	67.7	89.5	**70.8**
	+OPD	90.8	12.2	47.0	92.5	13.8	49.0	**99.5**	**88.7**	**91.0**	55.2
	+MERGE	92.6	**16.2**	43.7	91.6	12.8	45.2	98.0	77.0	86.0	56.1
	+Abort	92.6	13.7	**50.3**	92.5	**15.0**	46.5	92.2	69.7	81.9	65.4
	+Force	90.1	13.8	44.7	90.4	9.2	25.8	56.0	29.5	42.3	31.1
Olmo3	Unadapted	94.0	40.3	45.5	94.6	**43.3**	47.0	**99.9**	95.5	95.0	65.4
	RL	94.2	**44.2**	**51.0**	94.2	40.2	**49.7**	99.7	95.4	95.7	64.4
	IFT	**95.2**	35.8	47.2	94.2	36.5	45.2	99.5	96.1	96.4	**77.7**
	+OPD	93.5	38.7	48.7	**95.4**	37.5	48.2	99.8	**96.6**	**96.8**	70.4
	+MERGE	**95.2**	35.8	47.2	94.2	36.5	45.2	99.5	96.1	96.4	**77.7**
	+Abort	**95.2**	35.8	47.2	94.2	36.5	45.2	99.5	96.1	96.4	**77.7**
	+Force	**95.2**	35.8	47.2	94.2	36.5	45.2	99.5	96.1	96.4	**77.7**
Apriel	Unadapted	95.6	22.0	53.0	**96.0**	21.7	**57.6**	**100.0**	99.2	**99.4**	84.0
	RL	**96.6**	24.0	54.3	95.6	**24.2**	57.1	85.3	85.0	84.5	72.6
	IFT	94.6	22.8	52.0	83.3	18.7	28.3	99.6	83.6	97.7	**95.2**
	+OPD	94.7	21.0	54.8	94.6	16.7	50.5	98.7	99.5	98.9	91.9
	+MERGE	95.3	24.2	**58.6**	95.6	21.3	56.1	100.0	**99.5**	99.3	89.4
	+Abort	95.6	**24.5**	56.6	95.6	21.3	52.8	72.8	72.8	71.9	71.2
	+Force	93.3	20.7	52.3	94.9	20.7	50.8	95.0	93.8	91.1	90.6

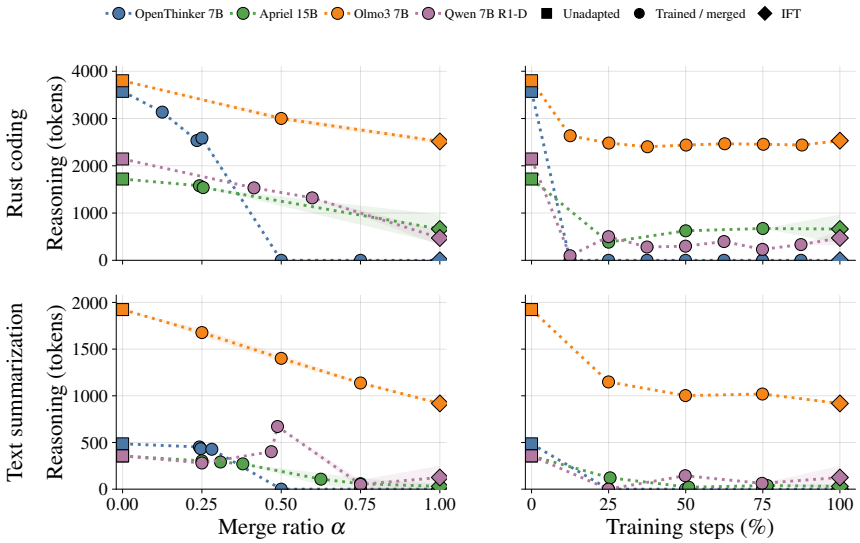


Figure 5: Mean target-task reasoning length across merge ratios (left) and IFT training progress (right) indicate that all models experience a slight decrease in reasoning length. We can not determine a relationship to task performance beyond reasoning presence or absence.

Table 6: Spearman correlations (ρ) between mean reasoning length conditional on a non-empty trace and performance on points with at least 90% target-task reasoning. Holm-adjusted p-values across all eight tests are not significant.

			Target score		OOD	
Task	Trajectory	n	ρ	p	ρ	p
Rust	Merge ratios	13	-0.011	1.000	-0.110	1.000
	Training checkpoints	15	0.071	1.000	0.521	0.324
Text sum.	Merge ratios	17	-0.167	1.000	0.343	1.000
	Training checkpoints	8	-0.024	1.000	0.881	0.031

Table 7: Runtime breakdown for our method in minutes. Cal. reports separately measured calibration evaluation; Load and Other report model loading and residual overhead, respectively.

Task	Model	IFT	Merge	Cal.	Load	Other	Total
Rust	OpenThinker 7B	46.1	1.32	1.36	1.51	0.43	51.6
Rust	Qwen R1-D 7B	69.8	0.03	2.38	1.33	0.30	74.1
Rust	Olmo3 7B	31.6	0.35	0.24	1.20	0.35	34.1
Rust	Apriel 15B	159.0	0.06	4.83	1.52	0.41	166.2
Text sum.	OpenThinker 7B	14.7	0.87	1.05	1.81	0.38	19.6
Text sum.	Qwen R1-D 7B	66.5	0.02	1.71	1.28	0.22	69.9
Text sum.	Olmo3 7B	16.9	0.38	0.19	1.28	0.33	19.4
Text sum.	Apriel 15B	33.5	1.70	2.12	1.36	0.65	40.8
Average	Rust	76.6	0.44	2.20	1.39	0.37	81.5
Average	Text sum.	32.9	0.74	1.26	1.43	0.39	37.4
Average	All	54.8	0.59	1.73	1.41	0.38	59.4

The first method, VeriFree (Zhou et al., 2025), requires substantial resources. Based on our initial experiments, we estimated that a single training run would require 4 H200 GPUs for $70\,\mathrm{h}$. Given this cost, which applies to each step of a preliminary hyperparameter search for fair comparison, we consider a comparison infeasible.

Second, we made extensive attempts to reproduce SDFT (Shenfeld et al., 2026) on Olmo3 for Rust coding. We performed 26 runs with the following IFT settings: epochs $\in \{1, 2, 3, 5\}$, learning rate $\eta \in \{1 \times 10^{-4}, 5 \times 10^{-5}, 1 \times 10^{-5}, 5 \times 10^{-6}\}$, batch size $B \in \{16, 32, 64\}$, and EMA coefficient $\alpha \in \{0.01, 0.02, 0.05\}$. In order to fit into the GPU memory, matching our RL setup, we set the maximum token length for rollouts to $4,096$ and discarded training samples exceeding this budget, leaving roughly one-third of the examples. The best-performing run used two epochs, a learning rate of 5×10^{-6}, a batch size of 64, and EMA $\alpha = 0.05$, and completed in $4.5\,\mathrm{h}$ using four H200 GPUs. The best performance we could achieve with SDFT was 41.4%, a decrease from the untuned performance of 46.2%, while performance on OOD also slightly decreased from 95.5% to 93.1%. As such, despite our extensive reproduction efforts, we did not include the above related methods in the direct comparison in our experiments.

A.8 Runtime Details

We report the runtime breakdown for the total runtime (Total) of our method in Table 7, generated from the same per-run runtimes as Table 1. The total runtime consists of the IFT training (IFT), model merge operations (Merge), token sampling on the calibration dataset (Cal.), model merge loading to prepare sampling (Load), and auxiliary overhead (Other).

B Case Study

In this section, we provide detailed outputs for an MBRP example where the model is tasked with producing code that finds the correct insertion point in a sorted array. The unadapted model Apriel,

shown in Figure 6, reasons correctly and produces sensible code but fails to recognize a type mismatch. It initializes the running variable `low` as `0` and `high` from the length method `a.len()`. The length method returns the type `usize` (unsigned integer), so `high` is inferred to have type `usize`. Meanwhile, numeric literals in Rust do not have a fixed type and can be unsigned or signed depending on context. Since `low` interacts with `high`, it is also inferred to have type `usize` and thus mismatches the target return type `isize`.

The IFT model, shown in Figure 7, gives a correct, direct answer without reasoning by immediately casting the output of `a.len()` to `isize`, causing both `low` and `high` to be inferred to have the correct type.

Finally, the merged model, shown in Figure 8, reasons like the unadapted model but recognizes the potential type conflict early (as highlighted in the reasoning trace in blue). As a result, it produces code similar to that of the unadapted model with a crucial type cast to `isize` for the return value (highlighted in red).

Note that Apriel does not generate explicit thinking markers, but only `[BEGIN FINAL RESPONSE]` to mark the final answer. We insert thinking tokens (`[BEGIN THOUGHT]` and `[END THOUGHT]`) for clarity.

C PROMPTS

We use one task prompt for Rust code generation, one task prompt for text summarization, and four judge prompts for text summarization evaluation and RLAIF training. Figure 9 shows the Rust prompt used for both IFT training and MBRP evaluation. Figure 11 shows the text summarization prompt used for both IFT training and RLAIF rollouts on the TLDR split, as well as evaluation on the CNN/Daily Mail split. For the LLM-based text summarization metrics, as well as the reward computation in RLAIF training, the models are evaluated separately for each criterion: Figure 12 for relevance, Figure 13 for fluency, Figure 14 for factual consistency, and Figure 15 for coherence.

User

```
Write a function to locate the left insertion point for a specified value in sorted order. The signature of the
   function is: 'fn left_insertion(a: Vec<isize>, x: isize) -> isize'
```

Assistant

```
[BEGIN THOUGHT]
Okay, I need to write a Rust function called left_insertion that takes a vector of isize and a value x, and returns the left
   insertion point. The left insertion point is the index where the element x should be inserted to maintain the sorted order, such
   that all elements before are less than x.

First, I remember that in Python, the bisect module has bisect_left which does exactly this. So the approach here is similar. The
   vector is sorted, so we can perform a binary search.

Let me think about the steps. The function needs to find the first position where a[i] >= x. If all elements are smaller, then the
   insertion point is the length of the vector.

So the binary search will have low and high pointers. Initialize low to 0 and high to a.len(). Then, while low is less than high,
   find the mid point. If a[mid] is less than x, then the insertion point must be to the right of mid, so set low to mid + 1.
   Otherwise, set high to mid. When the loop ends, low should be the insertion point.
Wait, let me test this logic with an example. Suppose the vector is [1,3,5,7], and x is 4. The mid points would be calculated, and
   the loop would adjust low and high until it finds the correct position. Let's see:
Initial low=0, high=4. Mid is 2. a[2] is 5, which is greater than 4. So high becomes 2. Now low is 0, high 2. Mid is 1. a[1] is 3,
   which is less than 4. So low becomes 2. Now low equals high, loop ends. Return 2, which is correct.
Another example: x is 0. The insertion point is 0. The loop would check mid 2 (5), which is greater, so high becomes 2. Then mid 1
   (3), high becomes 1. Then mid 0 (1), high becomes 0. Loop ends, return 0.
Another case: x is 8. The insertion point is 4. The loop would proceed with low starting at 0, high 4. Mid 2 (5 <8), low becomes 3.
   Mid (3+4)/2=3.5, which is 3. a[3] is7 <8, so low becomes4. Now low equals high, return 4.

So the algorithm seems correct.

Now, implementing this in Rust. The function signature is given as fn left_insertion(a: Vec<isize>, x: isize) -> isize.

So the code would be:

Initialize low as 0, high as a.len(). Then loop while low < high. Compute mid as (low + high) / 2. But in Rust, integer division is
   done with /, and for even and odd numbers, it truncates towards zero. So (low + high) / 2 is correct.

Compare a[mid] with x. If a[mid] < x, then set low = mid +1. Else, set high = mid. At the end, return low.

But wait, what if the vector is empty? Then a.len() is 0, and the loop doesn't run, returns 0, which is correct.

So the code would look like:

[... copy of code below ...]

Yes, that should work. Let me check the examples again. For the vector [1,3,5,7], x=4, the code returns 2. For x=0, returns 0. For x
   =8, returns 4. All correct.

So this should be the correct implementation.
[END THOUGHT]
[BEGIN FINAL RESPONSE]
```rust
fn left_insertion(a: Vec<isize>, x: isize) -> isize {
   let mut low = 0;
   let mut high = a.len();
   while low < high {
      let mid = (low + high) / 2;
      if a[mid] < x {
         low = mid + 1;
      } else {
         high = mid;
      }
   }
   low
}
```
[END FINAL RESPONSE]
```

Compiler

```
error[E0308]: mismatched types
  --> /var/folders/tc/ybwwqpzd0774whvhgf_b077m0000gn/T/apriel_diag_292_t0fq4h2a/case.rs:12:5
   |
 1 | fn left_insertion(a: Vec<isize>, x: isize) -> isize {
   | ----- expected `isize` because of return type
...
12 | low
   | ^^^ expected `isize`, found `usize`
```

Figure 6: The non-tuned Apriel model produces a non-empty reasoning trace but fails to produce compiling Rust code because it does not recognize that the implicit type of variable `low` is `usize`, which conflicts with the target return type `isize`.

User

Write a function to locate the left insertion point for a specified value in sorted order. The signature of the function is: 'fn left_insertion(a: Vec<isize>, x: isize) -> isize'

Assistant

[BEGIN THOUGHT]
[END THOUGHT]
[BEGIN FINAL RESPONSE]

```rust
fn left_insertion(a: Vec<isize>, x: isize) -> isize {
   let mut low = 0;
   let mut high = a.len() as isize;

   while low < high {
      let mid = low + (high - low) / 2;
      if a[mid as usize] < x {
         low = mid + 1;
      } else {
         high = mid;
      }
   }

   low
}
```

[END FINAL RESPONSE]

Figure 7: The IFT model emits an empty reasoning trace and then a direct correct solution by immediately casting the running variables to the target type `isize` (highlighted in red).

User

Write a function to locate the left insertion point for a specified value in sorted order. The signature of the function is: 'fn left_insertion(a: Vec<isize>, x: isize) -> isize'

Assistant

```
[BEGIN THOUGHT]
Okay, I need to write a Rust function called left_insertion that finds the left insertion point for a value x in a sorted vector a.
   The function should return the index where x should be inserted to maintain the sorted order. Let me think about how to approach
   this.

First, I remember that the left insertion point is the first position where the element is greater than or equal to x. So, for
   example, if the vector is [1,3,5,7] and x is 4, the left insertion point would be 2 because 5 is the first element greater than 4.
    [...]

So the binary search approach seems to work. Now, how to implement this in Rust.

The function signature is given as fn left_insertion(a: Vec<isize>, x: isize) -> isize. So the parameters are a vector of isize and x
    is isize. The return is isize, which is the index.

Implementing the binary search:

Initialize low as 0, high as a.len() as isize. Wait, but in Rust, the vector's len() is usize.  So when we do operations, we need to
   handle the types.  Let's see.  Let's use usize for low and high.  Because the insertion point can't be larger than the vector's
   length, which is a usize.

So, low starts at 0, high starts at a.len(). Then, while low < high, compute mid. Mid can be (low + high) / 2. But in Rust, adding
   two usize can cause overflow, but since the vector's length is a valid usize, and low and high are within that, it's safe here.

Wait, but in the case where the vector is empty, a.len() is 0, so high is 0, and the loop doesn't run. Then return 0, which is
   correct.

So, code steps:

[... copy of code below ...]

Wait, but in Rust, the vector is passed as a Vec<isize>. So when accessing a[mid], if mid is within the vector's bounds, it's okay.
   But in the case where the vector is empty, the loop doesn't run, and low is 0, which is correct.

Testing the code with the examples:
Example 1: a = vec![1,3,5,7], x=4. The code should return 2. Let's see:
low=0, high=4. mid=2. a[2] is 5 >=4, so high=2. Now low=0, high=2. mid=1. a[1] is 3 <4, so low=2. Now low=2, high=2. Loop ends.
   Return 2. Correct.

Another example: x=0. The code returns 0. Correct.
Another example: x=8. The code returns 4. Correct.
What about when the element is present in the vector? For example, a = vec![2,4,4,4,6], x=4. The left insertion point is 1. Let's see
   :
low=0, high=5. mid=2. a[2] is 4 >=4, so high=2. Now low=0, high=2. mid=1. a[1] is 4 >=4, so high=1. Now low=0, high=1. mid=0. a[0] is
    2 <4, so low=1. Now low=1, high=1. Return 1. Correct.

So the code seems to handle all cases correctly.

Therefore, the implementation should be as written.

[END THOUGHT]
[BEGIN FINAL RESPONSE]
```rust
fn left_insertion(a: Vec<isize>, x: isize) -> isize {
   let mut low = 0;
   let mut high = a.len();
   while low < high {
      let mid = (low + high) / 2;
      if a[mid] < x {
         low = mid + 1;
      } else {
         high = mid;
      }
   }
   low as isize
}
```
[END FINAL RESPONSE]
```

Figure 8: The model merged with our method produces a non-empty reasoning trace and correctly recognizes the implicit types of `low` and `high` and the potential conflict with the function signature (highlighted in blue). It thus inserts a crucial cast at the return position (highlighted in red).

User

````
You are a helpful coding assistant producing high-quality Rust code.
Strictly follow the instruction below to complete the function specified in the
instruction.
Your response should include all dependencies, headers and function declaration
to be directly usable (even for the ones seen in the given part).
You should NOT call or test the function in your response.
Output your complete implementation in a single code block wrapped in triple
backticks with `rust` specified, like this:
```rust
// your function here
```
Instruction:
{{RUST_PROMPT}}
````

Figure 9: Prompt used for both training and evaluation in the Rust code generation task. The task instruction varies per example and is inserted as `{{RUST_PROMPT}}`.

User

```
{{PROBLEM}}
Please reason step by step, and put your final answer within \boxed{}.
```

Figure 10: Prompt used for MATH-500 evaluation. The problem is inserted as `{{PROBLEM}}`.

User

```
You are an expert in writing summarization.
Your task is to read the following Article and write a summary about it.

Output your complete summary after the <SUMMARY> tag, like this:
<SUMMARY>
// your summary here

Article:
{{ARTICLE}}
```

Figure 11: Prompt used for both training and evaluation in the text summarization task. The article is inserted as `{{ARTICLE}}`.

User

```
You are a helpful assistant in evaluating the quality of a summary.
You will be given a news article and a summary written for that article.
Your task is to evaluate the *relevance* of the summary.

Definition of Relevance:
Relevance measures how well the summary captures the important information from
the article. A relevant summary includes the most important main idea of the
article and avoids redundancies and minor, trivial, or unrelated details.

Evaluation Criteria (Relevance: 1-100)

80-100 -- The summary captures the key idea of the article accurately and
   completely. It focuses on the essential information and avoids any
   redundancies and unnecessary or irrelevant content.

40-79 -- The summary includes some important information but only partially
   capture the main idea. It misses key points or includes some redundancies
   or minor details.

1-39 -- The summary fails to capture the main idea of the article and completely
   deviate from the main idea. It may focus on unimportant details or
   irrelevant content or omit critical points.

Evaluation Steps:
1. Read and understand the article.
2. Identify the article's main idea and secondary details.
3. Read the summary and judge if it captures the main idea of the article.
4. Identify if there are any secondary details or redundancy in the summary.
5. Assign a score from 1 to 100 based on the Evaluation Criteria above.

Output your detailed thought process and formal justification based on the
Evaluation Criteria, and finally output your final score in the format shown
below:

<think>
// your thought process and justification
</think>

<Final score>
Relevance: // your final score

Article:
{{ARTICLE}}

Summary:
{{SUMMARY}}
```

Figure 12: Prompt used to evaluate summary relevance with Gemini 3.1 Pro Preview and by the frozen base judges during RLAIF training. The article and generated summary are inserted as `{{ARTICLE}}` and `{{SUMMARY}}`.

User

```
You are a helpful assistant in evaluating the quality of a summary.
You will be given a news article and a summary written for that article.
Your task is to evaluate the *fluency* of the summary.

Definition of Fluency:
Fluency measures how easy the summary is to read.
All sentences in a fluent summary need to be readable and natural.
Minor grammatical, formatting, capitalization, or tokenization issues should
NOT be heavily penalized as long as they do not make the text difficult to read
or understand.

Evaluation Criteria (Fluency: 1-5)

5 -- The summary is easy to read and all the sentences are understandable and
   natural. Minor issues such as awkward wording, missing capitalization, or
   tokenization artifacts are acceptable if they do not hinder readability.

3 -- The summary is readable but some of sentences include awkward phrasing,
   inconsistent grammar, or formatting problems that reduce readability.

1 -- The summary is difficult to read. It contains frequent grammatical errors,
   broken or incomplete sentences, or severe formatting problems that
   significantly hinder understanding.

Evaluation Steps:
1. Read and understand the article.
2. Read the summary and judge whether the text is easy to read and whether the
  individual sentences are natural.
3. Assign a score from 1 to 5 based on the Evaluation Criteria above.

Only output the score. Do not include any additional explanations or text.
Use the following format:
Fluency: <score>

Article:
{{ARTICLE}}

Summary:
{{SUMMARY}}
```

Figure 13: Prompt used to evaluate summary fluency with Gemini 3.1 Pro Preview and by the frozen base judges during RLAIF training.

User

```
You are a helpful assistant in evaluating the quality of a summary.
You will be given a news article and a summary written for that article.
Your task is to evaluate the *consistency* of the summary.

Definition of Consistency:
Consistency measures how factually aligned the summary is with the article.
A consistent summary should not introduce information that contradicts the
article or contain hallucinated statements not supported by the source article.

Evaluation Criteria (Consistency: 1-5)

5 -- All statements in the summary are fully supported by the article. No
   contradictions, distortions, or hallucinated details appear.

3 -- The summary is mostly consistent but includes minor inaccuracies, unclear
   references, or slightly misleading phrasing. These issues do not
   significantly alter the meaning.

1 -- The summary contains factual errors or statements that contradict the
   article or introduce unsupported information.

Evaluation Steps:
1. Read and understand the article.
2. Read the summary and check whether each fact is supported by the article.
3. Assign a score from 1 to 5 based on the Evaluation Criteria above.

Only output the score. Do not include any additional explanations or text.
Use the following format:
Consistency: <score>

Article:
{{ARTICLE}}

Summary:
{{SUMMARY}}
```

Figure 14: Prompt used to evaluate summary consistency with Gemini 3.1 Pro Preview and by the frozen base judges during RLAIF training.

User

```
You are a helpful assistant in evaluating the quality of a summary.
You will be given a news article and a summary written for that article.
Your task is to evaluate the *coherence* of the summary.

Definition of Coherence:
Coherence measures how well the ideas in the summary fit together.
A coherent summary presents information in a logical order, with smooth
transitions between sentences. It should read as a connected, well-structured
whole.

Evaluation Criteria (Coherence: 1-5)

5 -- The summary is clearly organized and easy to follow. Sentences flow
   naturally, and ideas progress in a logical order.

3 -- The summary is somewhat coherent but has noticeable issues in flow or
   structure. Some sentences feel disconnected or out of place, yet the overall
   meaning is still clear.

1 -- The summary is hard to follow. The sentences are out of order and loosely
   connected, which makes the summary feel fragmented.

Evaluation Steps:
1. Read and understand the article.
2. Read the summary and assess whether the ideas are presented in a clear and
  logical order.
3. Assign a score from 1 to 5 based on the Evaluation Criteria above.

Only output the score. Do not include any additional explanations or text.
Use the following format:
Coherence: <score>

Article:
{{ARTICLE}}

Summary:
{{SUMMARY}}
```

Figure 15: Prompt used to evaluate summary coherence with Gemini 3.1 Pro Preview and by the frozen base judges during RLAIF training.