Leveraging Instruction Tuning and Merging for Reasoning Model Adaptation

Yu-Du Feng [*1] **Niels Mündler-Sasahara** [*1] **Mark Vero** [1] **Martin Vechev** [1]

Abstract

Reasoning language models (RLMs) have demonstrated impressive performance in domains such as mathematics and coding. These domains permit reliable verification of model outputs, which is important for enabling the reinforcement learning that drives RLM performance gains. However, training RLMs on domains that lack reliable verifiers remains challenging. Meanwhile, for both verifiable and unverifiable domains, large amounts of unused supervised fine-tuning data with human-written solutions exist. In this work, we show that these data can be used efficiently to further improve RLM performance. For this, we first use classic instruction tuning, supervised fine-tuning without reasoning traces, on the RLM. Next, we merge our instruction-tuned model with the original reasoning model, recovering its reasoning behavior on the target domain. Our extensive evaluation demonstrates that our technique improves RLM performance in both verifiable and hard-to-verify domains, including coding and text summarization, while preserving RLM capabilities across other domains. Importantly, our method is highly cost-effective, enabling such improvements for less than USD $3.

1. Introduction

Reasoning language models (RLMs) have changed the frontier of language model capabilities by demonstrating impressive results on tasks such as mathematics and programming (OpenAI et al., 2024a; DeepSeek-AI, 2025; Yang et al., 2025). RLMs are trained to produce reasoning traces that explore potential solutions or perform logical reasoning before producing a final answer (OpenAI et al., 2024a; DeepSeek-AI, 2025). The dominant training recipes for RLMs are based on reinforcement learning with verifiable rewards (RLVR) (DeepSeek-AI, 2025; Guha et al., 2025; Radhakrishna et al., 2025; Olmo et al., 2025). However, RLVR requires automatic verifiers to determine whether the proposed answer is objectively correct. This leaves performance improvements on a much wider class of tasks unresolved, in particular domains that do not permit reliable verifiers, such as text summarization or coding without comprehensive unit tests.

In such domains, large amounts of training data are available: task descriptions paired with high-quality task solutions (Lambert et al., 2024; Stiennon et al., 2022; Oxen.ai Team, 2024a). However, these data lack reasoning traces and are thus unlike RLM output. The most direct approach to leveraging these data, instruction fine-tuning (IFT), trains models to produce the solution directly from the task description. This approach is cheap and widely applicable (Wei et al., 2022; Chung et al., 2022; Hu et al., 2022), but it creates a distributional mismatch for RLMs, which expect reasoning traces before producing task solutions. This mismatch degrades RLM performance by effectively training it not to produce reasoning traces (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, ignoring reasoning traces. Second, we linearly merge the resulting IFT checkpoint with the original reasoning model. The merge ratio is selected using a small target-task calibration set as the largest coefficient that preserves target-task reasoning. This procedure does not require a verifier or a reward model.

We evaluate our approach on four open RLMs, OpenThinker 7B (Guha et al., 2025), Apriel Nemotron 15B Thinker (Radhakrishna et al., 2025), Olmo3 7B Think (Olmo et al., 2025), and DeepSeek R1 Qwen 7B Distilled (DeepSeek-AI, 2025), across Rust coding and text summarization. We use MATH500 as a held-out dataset to measure the preservation of general reasoning capability. Across settings, standard IFT often collapses reasoning behavior and can lead to substantially reduced MATH500 performance. Our merging technique recovers most or all of the lost reasoning capability while retaining significant parts of the target-task gain

[*]Equal contribution [1]Department of Computer Science, ETH Zurich, Zurich, Switzerland. Correspondence to: Yu-Du Feng <yufeng1@student.ethz.ch>, Niels Mündler-Sasahara <niels.muendler@inf.ethz.ch>.

Accepted to *Foundations of Deep Generative Models: Understanding Memorization, Generalization, and Reasoning (FoGen)*, , an ICML 2026 workshop (non-archival). Copyright 2026 by the author(s).

from IFT. In addition, this method is highly cost-effective, allowing model adaptation in under one hour for less than USD \$3, consistently less than comparable baselines that achieve similar or worse performance.

Our contributions Our key contributions are: [1]

- We identify and study a practical adaptation setting for RLMs where only input-output supervision is available, without verified reasoning traces.
- 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.
- We evaluate the method across four RLMs and two target tasks, showing that it preserves general reasoning capabilities while retaining target-task improvements and is more cost-effective than competitive baselines.

2. Background

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

Language models An autoregressive language model (LM), parameterized by θ, models the probability of the next token c_T in a sequence $\mathbf{c}$ conditioned on input context $\mathbf{c}_{<T}$. This is achieved by factorizing the joint probability into a product of conditional probabilities for each token:

$$p_\theta(c_T \mid \mathbf{c}_{<T}) = \prod_{t=1}^{T} p_\theta(c_t \mid \mathbf{c}_{<t})$$

where $\mathbf{c}_{<t} = (c_1, \ldots, c_{t-1})$ represents preceding tokens.

During inference, we split the context $\mathbf{c}$ into a pair $(\mathbf{x}, \hat{\mathbf{y}})$ of user-provided input $\mathbf{x}$ and model-generated output $\hat{\mathbf{y}}$. The first inference step samples $\hat{y}_0$ from $p_\theta(\hat{y}_0|\mathbf{x})$, and later steps obtain $\hat{y}_i$ from $p_\theta(\hat{y}_i|\mathbf{x} + \hat{\mathbf{y}}_{<i})$. We refer to $\mathbf{x}$ as the *prompt* and $\hat{\mathbf{y}}$ as the *answer*. Pre-trained large language models (LLMs) are LMs with billions of parameters, trained on trillions of tokens of training data (OpenAI et al., 2024b; DeepSeek-AI, 2025; Yang et al., 2025). Through such pre-training, the models acquire a variety of general skills, in particular language understanding.

RLMs and RLVR Reasoning Language Models (RLMs) are LLMs trained on challenging tasks using reinforcement learning with verifiable rewards (RLVR) (OpenAI et al., 2024a; DeepSeek-AI, 2025; Yang et al., 2025). In RLVR, model answers $\hat{\mathbf{y}} = \mathbf{r} + \mathbf{a}$ are split into a *reasoning trace* $\mathbf{r}$ and a final answer $\mathbf{a}$. Crucially, the final answer permits reliable scoring for correctness, such as numerical results or executable code. The LLM is then trained to prefer reasoning traces that lead to correct final answers (DeepSeek-AI, 2025). Empirically, this results in strong solutions for challenging math and code problems (Shao et al., 2024; DeepSeek-AI, 2025), which is attributed to the process of reasoning, as non-reasoning correlates with performance loss (Yang et al., 2025).

However, RLVR has three key limitations. First, RLVR only works if the model has a non-zero solve rate on the dataset in question (Prakash & Buvanesh, 2025). Second, RLVR is an expensive process since it requires many reasoning rollouts during training (Shao et al., 2024; DeepSeek-AI, 2025; Olmo et al., 2025). Third, this approach requires a reliable verifier. If the verifier can be influenced by spurious correlations, the reasoning model can learn to exploit these correlations and produce undesired outputs (MacDiarmid et al., 2025; DeepSeek-AI, 2025; Amodei et al., 2016). For many relevant tasks, such as text summarization, a reliable verifier is not available.

Supervised and instruction fine-tuning In practice, it is often necessary to adapt the model for specialized tasks (Farn et al., 2025; Lambert et al., 2024). Adapting LLMs is commonly achieved through Supervised Fine-Tuning (SFT). SFT updates a model's parameters using a labeled dataset $\mathcal{D} = \{(\mathbf{x}, \mathbf{y})\}$ of input-output pairs. Instruction Fine-Tuning (IFT) is the special case of SFT where $\mathbf{x}$ is an instruction or task description and $\mathbf{y}$ is a direct answer. For an RLM, IFT sets $\mathbf{r} = \varepsilon$, i.e., it sets the reasoning trace as empty. The training objective is to minimize the cross-entropy loss, shown below.

$$\mathcal{L}_{\text{IFT}}(\theta, \mathcal{D}) = -\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}}\left[\sum_{t=1}^{|\mathbf{y}|} \log p_\theta(y_t \mid \mathbf{x} + \mathbf{y}_{<t})\right]. \tag{1}$$

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

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

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 connects this idea to task arithmetic and task arithmetic-like combinations, and shows that interpolation can produce usable points along low-loss trajectories between related checkpoints (Ilharco et al., 2023; Frankle et al., 2020; Alexandrov et al., 2024; Yang et al., 2024b). Non-linear interpolation variants exist as well (Yadav et al., 2023; Shoemake, 1985).

[1] We release our code implementation and datasets at https://github.com/eth-sri/rlm-training-merging

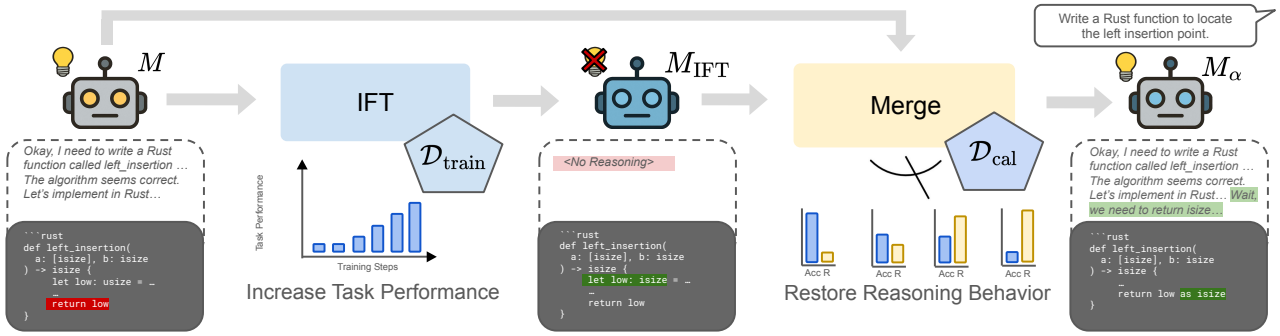

Figure 1. Our core method for RLM training consists of a lightweight two-step pipeline: We first perform standard IFT on $\mathcal{D}_{\text{train}}$, which produces a fine-tuned model M_{IFT} with higher task accuracy but possibly compromised reasoning behavior and forgetting. In the shown example, Apriel 15B 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}, using a calibration dataset D_{cal} to find the maximal merge ratio α that restores model reasoning. This merging produces M_α, a reasoning model with maintained or improved task capability. In the example, the merged Apriel 15B then reasons about the typing mismatch and required cast and correctly inserts it in the resulting code. Complete outputs are presented in §C.

Main challenges The key challenge for training RLMs is that RLMs are trained to produce reasoning traces. To avoid disturbing this behavior, training data for RLMs typically provides reasoning traces, often leveraging a stronger RLM (Guha et al., 2025; DeepSeek-AI, 2025; Deng et al., 2025). When neither a verifier nor a stronger RLM is available for the given task, we cannot easily obtain relevant reasoning traces. This poses a challenge because training RLMs without reasoning traces can lead to performance degradation, as they lose reasoning capabilities (Twist et al., 2026).

Our work resolves these challenges by presenting a method for training RLMs without requiring reasoning traces. As such, our method is able to leverage widely available IFT datasets while preserving reasoning traces and associated model performance. Due to the design of this method, it neither requires robust verifiers nor stronger RLMs.

3. Training RLMs via IFT and Model Merging

In this section, we describe our core pipeline that adapts RLMs using IFT on reasoning-free training data and merging to recover reasoning behavior.

Overview Our method is a two-step pipeline, visualized in Figure 1: Given a base RLM M, we first perform IFT on a task-specific training set $\mathcal{D}_{\text{train}}$ that contains no reasoning traces. This produces a fine-tuned model M_{IFT} with potentially compromised reasoning behavior. We then linearly merge M_{IFT} with the untuned model M using coefficient α, resulting in a merged model M_α. The merge ratio α is selected on a held-out target-task calibration set $\mathcal{D}_{\text{cal}}$ such that M_α remains close to M_{IFT} while preserving the reasoning behavior of M on the target task, as measured by non-empty reasoning traces.

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 base model M on $\mathcal{D}_{\text{train}}$ using standard IFT with the loss described in Equation (1). 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. We set the target answer to $\mathbf{y}(o) = \text{serial}(\varepsilon, o)$, rendering the answer with model-native thinking trace and final answer formatting. 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: after IFT, M_{IFT} may directly output the final answer on examples where the original reasoning model would have produced a non-empty trace.

Merging To recover reasoning behavior, we perform a model *merge*, an interpolation between the original reasoning model M and M_{IFT}. This choice is motivated by the use of merging to mitigate forgetting (Alexandrov et al., 2024).

The merging has an interpolation ratio α, which controls the strength of individual weights in the merged model. We select α using a search over model reasoning on a target-task calibration set $\mathcal{D}_{\text{cal}}$. Since M is an RLM in our setting, it has 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.

$$\rho(M', \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(i,o)\in\mathcal{D}} \mathbf{1}\{r_i \neq \varepsilon\}, \qquad (r_i, \hat{o}_i) \sim M'(i). \tag{2}$$

Algorithm 1 RLM Training Pipeline

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

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

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 $\rho_{\min}$ be the minimum acceptable reasoning rate; in our experiments, $\rho_{\min} = 0.9$. We are then looking for the largest merge ratio whose target-task calibration reasoning rate remains acceptable, i.e.,

$$\alpha^\star = \max_{\alpha \in \mathcal{A}} \{\alpha : \rho_{\min} \leq \rho(M_\alpha, \mathcal{D}_{\text{cal}})\}. \tag{3}$$

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

We describe the entire algorithm in Algorithm 1. After IFT, we evaluate a small uniform grid of merge ratio rather than performing an adaptive search. 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 and is guaranteed to satisfy $\rho(M, \mathcal{D}_{\text{cal}}) = 1$, while $\alpha = 1$ is the standard IFT model. After evaluating all grid points on $\mathcal{D}_{\text{cal}}$, we select the largest α whose reasoning rate is at least $\rho_{\min}$.

Optimizations We apply two optimizations to speed up this process: running calibration only on the first few tokens, and employing binary search in Algorithm 1.

To speed up calibration, we first observe that in practice, when a fine-tuned model no longer reasons, it emits the end-of-reasoning token immediately after the start-of-reasoning token (since this is the format the model was trained on). We therefore do not need to sample full responses during 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 estimate $\rho(M_\alpha, \mathcal{D}_{\text{cal}})$ 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 employ binary search to determine whether reasoning on the calibration dataset exceeds the minimum reasoning threshold. Since we also observe that model reasoning degrades rapidly around the critical merging ratio, we abort the search as soon as we observe a reasoning rate that is less than 100% and greater than or equal to the minimum threshold $\rho_{\min}$.

4. Experimental Evaluation

In this section, we demonstrate that our method reliably obtains a strong trade-off between model performance on the target task and general reasoning capabilities across four different reasoning models and two datasets.

4.1. Experimental Setup

Models We compare a diverse set of four recent RLMs: OpenThinker 7B (Guha et al., 2025), Apriel Nemotron 15B Thinker (Radhakrishna et al., 2025), Olmo3 7B Think (Olmo et al., 2025), and DeepSeek R1 Qwen 7B Distilled (Qwen 7B R1-D) (DeepSeek-AI, 2025). OpenThinker and Qwen 7B R1-D are based on Qwen2.5 7B (Yang et al., 2024a), distilled on reasoning traces by DeepSeek R1 (DeepSeek-AI, 2025). Apriel 15B is post-trained through CPT, IFT, and GRPO from a base Apriel 15B model. Olmo3 was trained from scratch with a fully open pipeline, including IFT, DPO, and RLVR.

Methods We first evaluate the unadapted RLM as *None*. We compare it to the IFT variant of the model, fine-tuned using LoRA as described in the first step of our training pipeline in §3. The use of LoRA reduces the memory footprint and training cost compared with full fine-tuning (Vaswani et al., 2023). Finally, we evaluate *Ours*, the merged version of the model at the optimal merge ratio determined by the binary search described above, using up to eight search steps and linear merging. We ablate over the use of LoRA and the choice of linear merginging in §4.3.

Baselines We compare our proposed method with two baselines. The first baseline uses On-Policy Distillation (*IFT+OPD*) (Agarwal et al., 2024) with the untuned RLM as the teacher model and the IFT model as the student model. As proposed by (Lu, 2025), we use an unrelated instruction fine-tuning dataset (Lambert et al., 2024) to train the student model for a small number of steps and recover its reasoning capabilities.

The second baseline technique, *IFT+KL*, uses a KL term on the thinking-trace section of the training output. Writing $\mathbf{c}_t = \mathbf{x} + \mathbf{y}_{<t}$, and assuming that tokens i to $i + k$ cover the thinking-trace-related parts of the training data, the loss

is updated to minimize the divergence from the untrained model over the thinking trace. An auxiliary hyperparameter λ controls the weight of this term.

$$\mathcal{L}_{\text{IFT+KL}}(\theta, \mathcal{D}) = \mathcal{L}_{\text{IFT}}(\theta, \mathcal{D}) + \lambda \tau^2 \, \mathbb{E}_{\substack{(\mathbf{x},\mathbf{y})\sim\mathcal{D} \\ t\sim\mathcal{U}\{i,\ldots,i+k\}}} \left[D_{\text{KL}}\big(p_{\theta_0}^{(\tau)}(\cdot \mid \mathbf{c}_t) \,\big\|\, p_{\theta}^{(\tau)}(\cdot \mid \mathbf{c}_t)\big) \right]$$

Tasks We train the RLMs on two distinct tasks: Rust coding and text summarization. We reserve a third task of mathematical reasoning to monitor model forgetting.

Rust coding: Implement a Rust function that solves a problem described in natural language.

Text summarization: Produce a concise, relevant, cohesive, and consistent summary of a long-form natural language text.

Mathematical reasoning: Derive a numerical answer to a high-school level mathematical question.

The tasks differ substantially in complexity and difficulty. In addition, neither training task permits reliable verifiers in general. For Rust, verification requires the existence of high-quality test suites (Liu et al., 2023; Zhao et al., 2026).

Metrics For Rust coding, we choose a Rust translation of the MBPP dataset (Cassano et al., 2022) and measure the percentage of solutions that implement a function described in natural language correctly, as measured by a set of unit tests (Chen et al., 2021).

For text summarization, we report SummEval, the average of the four metrics fluency, consistency, relevance, and coherence on the CNN split of the SummEval dataset (Fabbri et al., 2021). Each metric is assigned a score between 1 and 5 by Gemini 3 Pro (Google DeepMind, 2025) based on clearly outlined criteria, shown in §B. To confirm the validity of this LLM-as-a-judge setup, we compare the Gemini 3 Pro ratings with human annotations provided by Fabbri et al. (2021) and establish Spearman correlations of over 60% for each metric. We provide the details on this validation in §A.

For mathematical reasoning, we measure the performance of the models on MATH500 (Lightman et al., 2023). Note that we do not train on this task and use it to assess model forgetting and the loss of general reasoning capabilities.

Separately, we report *Reasoning*, the rate of non-empty reasoning traces on the evaluation dataset. Our calibration leverages this signal on the respective validation datasets.

Training datasets We devise two datasets for training. To measure generalization, we draw the training and test datasets from different distributions while retaining similar task formats. (1) *Rust coding*: We train the model on a set of synthetic single-function coding tasks in the Rust language (Oxen.ai Team, 2024b). We filter the dataset to the subset of code samples that pass the corresponding test suite consistently, resulting in 6761 high-quality training samples. We confirm that the obtained datasets are disjoint from the MBPP dataset by checking for closest matches using cosine similarity. (2) *Text summarization*: We use the Reddit TLDR split of Fabbri et al. (2021), which is intentionally different from the CNN task split that we use for evaluation.

Hyperparameters Unless otherwise indicated, we tune the learning rate, batch size, and number of epochs for each combination of model and dataset, and report the hyperparameters in §A. 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.

All models are trained and merged twice with different seeds, and we report averaged results from 10 evaluation runs for Rust, text summarization, and MATH500. In all plots we draw bands indicating the run-to-run variance, spanning the respective minimum and maximum average scores obtained by each training run.

4.2. Main Results

Rust coding We first train the models on Rust coding and evaluate their performance on MBPP-Rust and MATH500. As can be seen in Table 1, standard IFT has a strong impact on out-of-domain behavior: On MATH500 OpenThinker 7B drops from 79% to 35.9%. For Apriel 15B, target-task performance degrades as the model stops reasoning on the target task. All baselines recover reasoning. The recovered reasoning leads to improved performance compared with both the untuned model and the IFT model, with our merging method resulting in the highest average increase of 7.0 percentage points, compared to only 3.8 due to IFT.

Text summarization In text summarization, standard IFT improves SummEval by 0.16 points on average across all models, but target-task reasoning decreases by 71.4 percentage points on average. MATH500 performance similarly drops by 23.9 percentage points on average. Our method restores target-task reasoning and MATH500 almost completely while retaining 95.5% of the IFT SummEval gain. While KL and OPD also restore reasoning and MATH500, they worsen target-task performance for OpenThinker 7B by half a point, and recover less performance on Apriel 15B.

Runtime and cost A key benefit of our method is its low cost. Our adaptation method requires 52 min on average to complete on a single NVIDIA H200 GPU. At a representative rental price of USD 3.39 per hour, this corresponds to less than USD $3 on average. As shown in Table 1, our end-to-end runtime is lower than OPD and KL by an average of 22.8% and 33.0%, respectively, across the non-synthetic comparisons. For Olmo3 7B, no additional training or merg-

Table 1. Target-task score, MATH500 score, target-task reasoning rate (Reasoning), and measured training cost for each model and approach (Cost). Base rows report the original target-task and MATH500 scores; adapted rows report point changes from the corresponding base model (Δ). Our merging recovers reasoning and MATH500 performance while preserving the score increase from IFT at consistently low cost. We highlight MATH500 or reasoning loss due to IFT in red and highlight the rows of our merging method in blue. Costs are computed from the measured runtime at USD $3.39 per H200 GPU-hour.

		Rust coding				Text summarization			
Model	Method	Score / Δ	MATH500 / Δ	Reasoning	Cost	Score / Δ	MATH500 / Δ	Reasoning	Cost
OpenThinker 7B	None	53.36	78.7	100.0%	$0.00	4.30	78.7	99.7%	$0.00
	IFT	+3.01	−42.8	0.0%	$2.59	+0.32	−77.0	0.0%	$0.83
	+OPD	+0.71	−4.0	**100.0%**	$3.05	−0.43	**+4.5**	**100.0%**	$1.44
	+KL	**+7.03**	**+1.3**	**100.0%**	$4.72	−0.60	−18.5	**100.0%**	$2.67
	+Merge	+6.28	+0.0	97.4%	**$2.64**	**+0.16**	−0.9	98.8%	**$0.88**
Apriel 15B	None	63.79	95.2	100.0%	$0.00	4.72	95.2	100.0%	$0.00
	IFT	−5.78	+0.3	25.4%	$6.04	+0.03	−15.8	13.1%	$1.89
	+OPD	−0.40	−1.7	**100.0%**	$7.83	+0.13	−0.5	**100.0%**	$3.73
	+KL	+1.68	−1.2	**100.0%**	$11.90	+0.09	−0.5	**100.0%**	$2.98
	+Merge	**+3.26**	**+0.2**	**100.0%**	**$6.37**	**+0.14**	**+0.3**	82.8%	**$2.22**
Olmo 7B	None	44.18	91.1	100.0%	$0.00	4.46	91.1	100.0%	$0.00
	IFT	+5.79	−0.7	100.0%	$1.77	+0.15	−0.7	100.0%	$0.96
	+OPD	**+5.79**	**−0.7**	**100.0%**	**$1.77**	**+0.15**	**−0.7**	**100.0%**	**$0.96**
	+KL	+1.82	−6.3	**100.0%**	$3.20	**+0.15**	**−0.7**	**100.0%**	**$0.96**
	+Merge	**+5.79**	**−0.7**	**100.0%**	**$1.77**	**+0.15**	**−0.7**	**100.0%**	**$0.96**
Qwen 7B R1-D	None	29.44	88.6	100.0%	$0.00	3.95	88.6	100.0%	$0.00
	IFT	+12.55	−0.3	85.1%	$4.95	+0.11	−1.9	1.0%	$3.76
	+OPD	+12.36	**−0.4**	**100.0%**	$5.69	**+0.21**	**+0.4**	**100.0%**	$4.48
	+KL	+12.32	−0.5	87.0%	$5.77	+0.10	−0.2	**100.0%**	**$3.52**
	+Merge	**+12.76**	−0.7	92.5%	**$5.04**	+0.16	−0.1	99.8%	$3.85

ing is required, so we consider the cost beyond IFT to be 0 for all methods.

Note that OPD and KL are heavily advantaged in this comparison, since it includes the hyperparameter search for α in our merging technique, but omits the hyperparameter search for OPD and KL. The reason for the low cost of our method is that it only evaluates merged candidates on a small calibration set, and does not require reasoning rollouts on an additional dataset (OPD) or additional teacher inference on part of the training data (KL). We provide a detailed runtime breakdown in §A.5 and a runtime estimate for more expensive, unevaluated related work in §A.6.

4.3. Ablation

We ablate the choice of merge ratios, fine-tuning techniques, and hyperparameters. Overall, we find that the general trend of our method is stable across settings.

Merge ratio We ablate over the merging factor α on the Rust and text summarization datasets, and show the target-task reasoning rates and target-task performance in Figure 2 (left and right, respectively). We notice that OpenThinker 7B loses its target-task reasoning capability quickly for $\alpha \in [0.25, 0.5]$, while Qwen 7B R1-D loses it mostly only for text summarization for $\alpha \in [0.5, 0.75]$. Meanwhile Apriel 15B loses reasoning more slowly and Olmo3 7B even maintains full reasoning at $\alpha = 1$. Surprisingly, the performance on Rust coding appears to peak for $\alpha \in [0.25, 0.75]$, where the models still has knowledge about the task from training as well as recovered reasoning. Meanwhile, SumEval changes more smoothly with the merging ratio. Our method reliably picks a point close to a good trade-off between reasoning and task performance, despite picking the point only based on the target-task reasoning score on the calibration dataset.

Merging technique We ablate the choice of merging technique for OpenThinker 7B on Rust coding, presenting the results in the rightmost panel of Figure 3. We compare our linear merging with Spherical Linear Interpolation (SLERP) (Shoemake, 1985; Goddard et al., 2025) and TIES (Yadav et al., 2023). For TIES, since fixing $\alpha = 1$ and searching over the density parameter would result in complete loss of reasoning even for a density of less than 0.1, we fix the density to 0.5 and search over the α parameter. For all merging techniques, our method obtains the best merge ratio at $\alpha = 0.25$. Compared to linear interpolation, SLERP and TIES recover more MATH500 performance but improve target-task performance less.

Fine-tuning Our main method uses LoRA for lightweight and efficient fine-tuning. We ablate the use of LoRA by

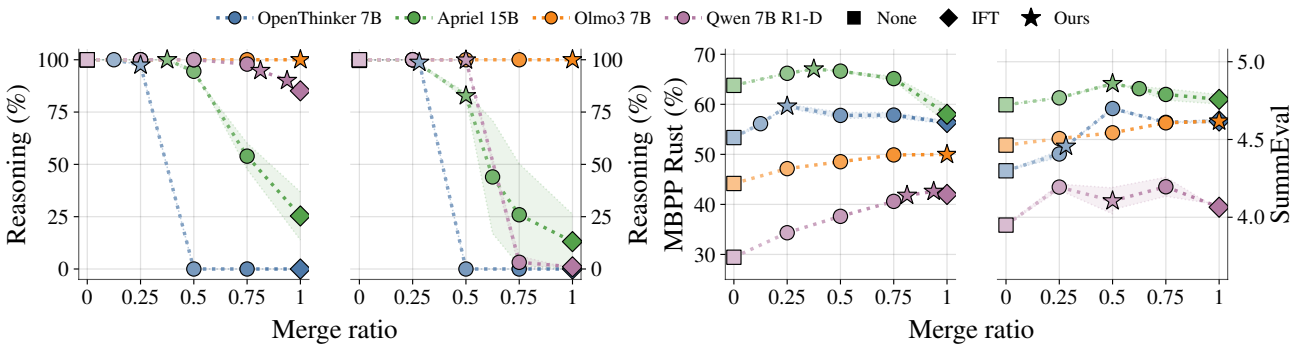

Figure 2. We evaluate various merge ratios on the final test set. We observe that reasoning on the target task drops sharply after a merging threshold is crossed (left), while target-task performance follows a smoother curve (right). For text summarization, task performance is measured by SummEval. On Rust, the benefit of restored reasoning for task performance is clearly visible.

fully fine-tuning all model weights with the same hyperparameters and present the results in the second panel from the right of Figure 3. We observe that the capability loss on MATH500 is much stronger with full fine-tuning and that our method is unable to obtain as much task-specific performance as with LoRA.

Hyperparameters We ablate fine-tuning hyperparameters for the supervised fine-tuning stage on OpenThinker 7B in Figure 3. We vary the number of epochs, learning rate η, and batch size B while keeping the remaining hyperparameters fixed. Importantly, even though some parameters result in models with worse performance on the target task than the untuned model, training consistently disturbs the reasoning behavior of the model, while our merging recovers MATH500 performance while maintaining target-task improvements.

5. Related Work

Reasoning distillation An alternative to reinforcement learning is reasoning distillation. In this approach, there is a training dataset $\mathcal{D} = \{(\mathbf{x}, \mathbf{y})\}$. 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 (Deng et al., 2025; DeepSeek-AI, 2025; Guha 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.

Training reasoning models with verifiers The standard technique for training and adapting reasoning models uses RLVR (DeepSeek-AI, 2025; Olmo et al., 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; Olmo et al., 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., 2025a;b) 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. We report our attempts to reproduce these methods in our setting in §A.6. In any case, they require substantially more resources than our approach. For example, SDFT used over 20 times our GPU resources: 4.5 hours on four H200 GPUs versus 40 minutes on one. Another approach to training reasoning models uses On-Policy Distillation to recover reasoning behavior after IFT (Agarwal et al., 2024). As we show in our experiments, our method is less expensive and does not depend on an external dataset.

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).

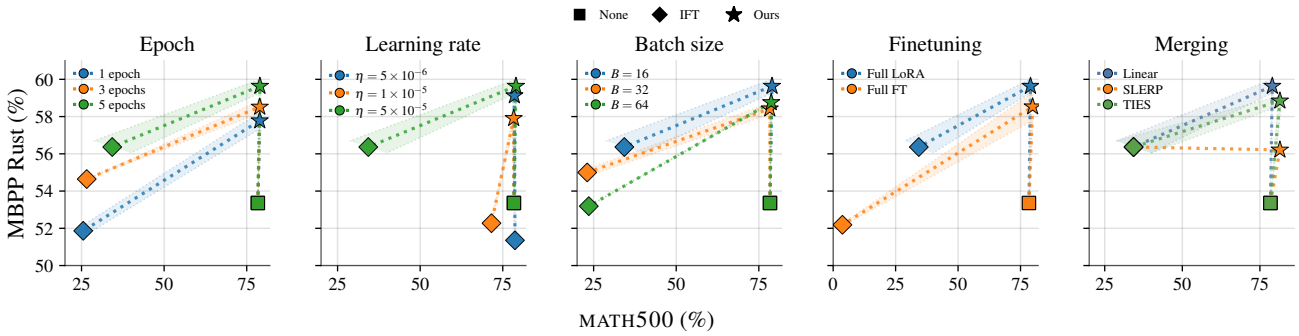

Figure 3. OpenThinker 7B ablations on Rust coding, from left to right: training epochs, learning rate η, batch size B, LoRA vs. full fine-tuning, and merging techniques. The overall trend of reasoning loss in standard IFT and recovery of reasoning and performance is stable across all settings.

6. Discussion

Inconsistent reasoning loss In our experiments, we observe that OpenThinker 7B loses its reasoning behavior in all settings, while Apriel 15B and Qwen 7B R1-D lose it in one setting but not all, and Olmo3 7B never loses its reasoning behavior. This might be related to the way these models were trained: OpenThinker and Qwen 7B R1-D are based on Qwen2.5 and only trained to reason via distillation from RLM traces (Guha et al., 2025), while Apriel and Olmo3 were both trained using RLVR (Radhakrishna et al., 2025; Olmo et al., 2025). We consider a study of the mechanisms that cause loss of reasoning capabilities an important direction for future work.

Composability and continual learning Our method demonstrates that for single-task adaptation, a simple train-and-merge pipeline is enough to preserve general capabilities and obtain task-specific improvements. However, as LLMs are trained to be generalists and continuously adapted to new tasks, the question remains how to obtain adaptation to several tasks in parallel. In exploratory experiments, we observe that applying our method sequentially leads to quicker loss of reasoning and partial forgetting of learned capabilities. We consider improvements in this direction an exciting avenue for future research.

Interpretability In our experiments, we observe that models stop reasoning by immediately producing an end-of-reasoning marker. It appears that there is a sharp drop in reasoning behavior around a specific merging ratio. We hypothesize that this is due to the probability of the end-of-reasoning token at some point outweighing the probabilities of any other follow-up token. To investigate this, we tried prefilling the model response with a start-of-reasoning marker and an immediately following non-end-of-reasoning token like `'Okay'`. However, while the model produced reasoning, the performance of the untuned model was not recovered. An interesting direction for future work is to mechanistically understand how reasoning tendency is encoded in the model. We hope that potential insights could then be directly leveraged to design better adaptation methods for RLMs, in particular in terms of recovering reasoning.

Lightweight adaptation on reasoning domains Our experiments focus on domains where the reasoning ability of the model itself is not crucial for achieving top performance; instead, the IFT data already provides the necessary signal for the model to perform well. This is still crucial, especially when patching the model for knowledge gaps; our method provides a lightweight way to insert new information into reasoning models simply by collecting instruction-completion pairs, following the same paradigm as in the pre-RLM era. However, on domains where the reasoning gains themselves define performance improvements (e.g., mathematics), final-output-based methods such as ours could fall short. Indeed, in preliminary experiments we have tried to apply our method to mathematical proofs, but we failed to improve the models' performance and hypothesize that such training would require supervision on the reasoning.

7. Conclusion

We studied how to adapt RLMs with standard input-output pairs. We observed that standard IFT can result in the loss of target-task reasoning and performance on held-out mathematical reasoning. We introduce a simple two-step pipeline to mitigate this issue: first fine-tune the RLM with standard IFT, then merge the fine-tuned checkpoint with the untuned model. The merge ratio is selected using a target-task calibration set and the observed rate of reasoning on that task. Across Rust coding and text summarization, this procedure recovers reasoning behavior while retaining much of the target-task improvement, and does so without a verifier, a reward model, or a stronger teacher model, at lower cost than comparable baselines. In ablations, we demonstrate that our method is robust to variations of merging algorithms, fine-tuning approaches, and hyperparameter choices.

Acknowledgements

This work has been done as part of the grant SAFEAI (Certified Safe, Fair and Robust Artificial Intelligence). The work has received funding from the Swiss State Secretariat for Education, Research and Innovation (SERI), contract no. MB22.00088.

References

Agarwal, R., Vieillard, N., Zhou, Y., Stanczyk, P., Ramos, S., Geist, M., and Bachem, O. On-policy distillation of language models: Learning from self-generated mistakes, 2024. URL https://arxiv.org/abs/2306.13649.

Alexandrov, A., Raychev, V., Müller, M. N., Zhang, C., Vechev, M. T., and Toutanova, K. Mitigating catastrophic forgetting in language transfer via model merging. In *EMNLP (Findings)*, 2024.

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., and Mané, D. Concrete problems in ai safety, 2016. URL https://arxiv.org/abs/1606.06565.

Cassano, F., Gouwar, J., Nguyen, D., Nguyen, S., Phipps-Costin, L., Pinckney, D., Yee, M.-H., Zi, Y., Anderson, C. J., Feldman, M. Q., Guha, A., Greenberg, M., and Jangda, A. Multipl-e: A scalable and extensible approach to benchmarking neural code generation, 2022. URL https://arxiv.org/abs/2208.08227.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating Large Language Models Trained on Code. *arXiv Preprint*, 2021. URL https://arxiv.org/abs/2107.03374.

Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, Y., Wang, X., Dehghani, M., Brahma, S., Webson, A., Gu, S. S., Dai, Z., Suzgun, M., Chen, X., Chowdhery, A., Castro-Ros, A., Pellat, M., Robinson, K., Valter, D., Narang, S., Mishra, G., Yu, A., Zhao, V., Huang, Y., Dai, A., Yu, H., Petrov, S., Chi, E. H., Dean, J., Devlin, J., Roberts, A., Zhou, D., Le, Q. V., and Wei, J. Scaling instruction-finetuned language models, 2022. URL https://arxiv.org/abs/2210.11416.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Deng, C., Tsai, Y.-D., Liu, G.-T., Yu, Z., and Ren, H. Scalertl: Scaling llms with reasoning data and test-time compute for accurate rtl code generation, 2025. URL https://arxiv.org/abs/2506.05566.

Fabbri, A. R., Kryściński, W., McCann, B., Xiong, C., Socher, R., and Radev, D. Summeval: Re-evaluating summarization evaluation, 2021. URL https://arxiv.org/abs/2007.12626.

Farn, H., Su, H., Kumar, S. H., Sahay, S., Chen, S.-T., and yi Lee, H. Safeguard fine-tuned llms through pre- and post-tuning model merging, 2025. URL https://arxiv.org/abs/2412.19512.

Frankle, J., Dziugaite, G. K., Roy, D. M., and Carbin, M. Linear mode connectivity and the lottery ticket hypothesis, 2020. URL https://arxiv.org/abs/1912.05671.

Goddard, C., Siriwardhana, S., Ehghaghi, M., Meyers, L., Karpukhin, V., Benedict, B., McQuade, M., and Solawetz, J. Arcee's mergekit: A toolkit for merging large language models, 2025. URL https://arxiv.org/abs/2403.13257.

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

Guha, E., Marten, R., Keh, S., Raoof, N., Smyrnis, G., Bansal, H., Nezhurina, M., Mercat, J., Vu, T., Sprague, Z., Suvarna, A., Feuer, B., Chen, L., Khan, Z., Frankel, E., Grover, S., Choi, C., Muennighoff, N., Su, S., Zhao, W., Yang, J., Pimpalgaonkar, S., Sharma, K., Ji, C. C.-J., Deng, Y., Pratt, S., Ramanujan, V., Saad-Falcon, J., Li, J., Dave, A., Albalak, A., Arora, K., Wulfe, B., Hegde, C., Durrett, G., Oh, S., Bansal, M., Gabriel, S., Grover, A., Chang, K.-W., Shankar, V., Gokaslan, A., Merrill, M. A., Hashimoto, T., Choi, Y., Jitsev, J., Heckel, R., Sathiamoorthy, M., Dimakis, A. G., and Schmidt, L. Openthoughts: Data recipes for reasoning models, 2025. URL https://arxiv.org/abs/2506.04178.

Hu, E. J., yelong shen, Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Ilharco, G., Ribeiro, M. T., Wortsman, M., Gururangan, S., Schmidt, L., Hajishirzi, H., and Farhadi, A. Editing models with task arithmetic, 2023. URL https://arxiv.org/abs/2212.04089.

Lambert, N., Morrison, J., Pyatkin, V., Huang, S., Ivison, H., Brahman, F., Miranda, L. J. V., Liu, A., Dziri, N., Lyu, S., Gu, Y., Malik, S., Graf, V., Hwang, J. D., Yang, J., Bras, R. L., Tafjord, O., Wilhelm, C., Soldaini, L., Smith, N. A., Wang, Y., Dasigi, P., and Hajishirzi, H. Tülu 3: Pushing frontiers in open language model post-training. 2024.

Lan, X., Zheng, Y., Cao, S., and Li, Y. The thinking spectrum: An empirical study of tunable reasoning in llms through model merging, 2025. URL https://arxiv.org/abs/2509.22034.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Liu, J., Xia, C. S., Wang, Y., and ZHANG, L. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 21558–21572. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/43e9d647ccd3e4b7b5baab53f0368686-Paper-Conference.pdf.

Lobo, E., Agarwal, C., and Lakkaraju, H. On the impact of fine-tuning on chain-of-thought reasoning, 2025. URL https://arxiv.org/abs/2411.15382.

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

MacDiarmid, M., Wright, B., Uesato, J., Benton, J., Kutasov, J., Price, S., Bouscal, N., Bowman, S., Bricken, T., Cloud, A., Denison, C., Gasteiger, J., Greenblatt, R., Leike, J., Lindsey, J., Mikulik, V., Perez, E., Rodrigues, A., Thomas, D., Webson, A., Ziegler, D., and Hubinger, E. Natural emergent misalignment from reward hacking in production rl, 2025. URL https://arxiv.org/abs/2511.18397.

Matena, M. S. and Raffel, C. Merging models with fisher-weighted averaging. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 17703–17716. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/70c26937fbf3d4600b69a129031b66ec-Paper-Conference.pdf.

Olmo, T., Ettinger, A., Bertsch, A., Kuehl, B., Graham, D., Heineman, D., Groeneveld, D., Brahman, F., Timbers, F., Ivison, H., Morrison, J., Poznanski, J., Lo, K., Soldaini, L., Jordan, M., Chen, M., Noukhovitch, M., Lambert, N., Walsh, P., Dasigi, P., Berry, R., Malik, S., Shah, S., Geng, S., Arora, S., Gupta, S., Anderson, T., Xiao, T., Murray, T., Romero, T., Graf, V., Asai, A., Bhagia, A., Wettig, A., Liu, A., Rangapur, A., Anastasiades, C., Huang, C., Schwenk, D., Trivedi, H., Magnusson, I., Lochner, J., Liu, J., Miranda, L. J. V., Sap, M., Morgan, M., Schmitz, M., Guerquin, M., Wilson, M., Huff, R., Bras, R. L., Xin, R., Shao, R., Skjonsberg, S., Shen, S. Z., Li, S. S., Wilde, T., Pyatkin, V., Merrill, W., Chang, Y., Gu, Y., Zeng, Z., Sabharwal, A., Zettlemoyer, L., Koh, P. W., Farhadi, A., Smith, N. A., and Hajishirzi, H. Olmo 3, 2025. URL https://arxiv.org/abs/2512.13961.

OpenAI, :, Jaech, A., Kalai, A., Lerer, A., Richardson, A., El-Kishky, A., Low, A., Helyar, A., Madry, A., Beutel, A., Carney, A., Iftimie, A., Karpenko, A., Passos, A. T., Neitz, A., Prokofiev, A., Wei, A., Tam, A., Bennett, A., Kumar, A., Saraiva, A., Vallone, A., Duberstein, A., Kondrich, A., Mishchenko, A., Applebaum, A., Jiang, A., Nair, A., Zoph, B., Ghorbani, B., Rossen, B., Sokolowsky, B., Barak, B., McGrew, B., Minaiev, B., Hao, B., Baker, B., Houghton, B., McKinzie, B., Eastman, B., Lugaresi, C., Bassin, C., Hudson, C., Li, C. M., de Bourcy, C., Voss, C., Shen, C., Zhang, C., Koch, C., Orsinger, C., Hesse, C., Fischer, C., Chan, C., Roberts, D., Kappler, D., Levy, D., Selsam, D., Dohan, D., Farhi, D., Mely, D., Robinson, D., Tsipras, D., Li, D., Oprica, D., Freeman, E., Zhang, E., Wong, E., Proehl, E., Cheung, E., Mitchell, E., Wallace, E., Ritter, E., Mays, E., Wang, F., Such, F. P., Raso, F., Leoni, F., Tsimpourlas, F., Song, F., von Lohmann, F., Sulit, F., Salmon, G., Parascandolo, G., Chabot, G., Zhao, G., Brockman, G., Leclerc, G., Salman, H., Bao, H., Sheng, H., Andrin, H., Bagherinezhad, H., Ren, H., Lightman, H., Chung, H. W., Kivlichan, I., O'Connell, I., Osband, I., Gilaberte, I. C., Akkaya, I., Kostrikov, I., Sutskever, I., Kofman, I., Pachocki, J., Lennon, J., Wei, J., Harb, J., Twore, J., Feng, J., Yu, J., Weng, J., Tang, J., Yu, J., Candela, J. Q., Palermo, J., Parish, J., Heidecke, J., Hallman, J., Rizzo, J., Gordon, J., Uesato, J., Ward, J., Huizinga, J., Wang, J., Chen, K., Xiao, K., Singhal, K., Nguyen, K., Cobbe, K., Shi, K., Wood, K., Rimbach, K., Gu-Lemberg, K., Liu, K., Lu, K., Stone, K., Yu, K., Ahmad, L., Yang, L., Liu, L., Maksin, L., Ho, L., Fedus, L., Weng, L., Li, L., McCallum, L., Held, L., Kuhn, L., Kondraciuk, L., Kaiser, L., Metz, L., Boyd, M., Trebacz, M., Joglekar, M., Chen, M., Tintor, M., Meyer, M., Jones, M., Kaufer, M., Schwarzer, M., Shah, M., Yatbaz, M., Guan, M. Y., Xu, M., Yan, M., Glaese, M., Chen, M., Lampe, M., Malek, M., Wang, M., Fradin, M., McClay, M., Pavlov, M., Wang, M., Wang, M., Murati, M., Bavarian, M., Rohaninejad, M., McAleese, N., Chowdhury, N., Chowdhury, N., Ryder, N., Tezak, N., Brown, N., Nachum, O., Boiko, O., Murk, O., Watkins, O., Chao, P., Ashbourne, P., Izmailov, P., Zhokhov, P., Dias, R., Arora, R., Lin, R., Lopes, R. G., Gaon, R., Miyara, R., Leike, R., Hwang, R., Garg, R., Brown, R., James, R., Shu, R., Cheu, R., Greene, R., Jain, S., Altman, S., Toizer, S., Toyer, S., Miserendino, S., Agarwal, S., Hernandez, S., Baker, S., McKinney, S., Yan, S., Zhao, S., Hu, S., Santurkar, S., Chaudhuri, S. R., Zhang, S., Fu, S., Papay, S., Lin, S., Balaji, S., Sanjeev, S., Sidor, S., Broda, T., Clark, A., Wang, T., Gordon, T., Sanders, T., Patwardhan, T., Sottiaux, T., Degry, T., Dimson, T., Zheng, T., Garipov, T., Stasi, T., Bansal, T., Creech, T., Peterson, T., Eloundou, T., Qi, V., Kosaraju, V., Monaco, V., Pong, V., Fomenko, V., Zheng, W., Zhou, W., McCabe, W., Zaremba, W., Dubois, Y., Lu, Y., Chen, Y., Cha, Y., Bai, Y., He, Y., Zhang, Y., Wang,

Y., Shao, Z., and Li, Z. Openai o1 system card, 2024a. URL https://arxiv.org/abs/2412.16720.

OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., Bello, I., Berdine, J., Bernadett-Shapiro, G., Berner, C., Bogdonoff, L., Boiko, O., Boyd, M., Brakman, A.-L., Brockman, G., Brooks, T., Brundage, M., Button, K., Cai, T., Campbell, R., Cann, A., Carey, B., Carlson, C., Carmichael, R., Chan, B., Chang, C., Chantzis, F., Chen, D., Chen, S., Chen, R., Chen, J., Chen, M., Chess, B., Cho, C., Chu, C., Chung, H. W., Cummings, D., Currier, J., Dai, Y., Decareaux, C., Degry, T., Deutsch, N., Deville, D., Dhar, A., Dohan, D., Dowling, S., Dunning, S., Ecoffet, A., Eleti, A., Eloundou, T., Farhi, D., Fedus, L., Felix, N., Fishman, S. P., Forte, J., Fulford, I., Gao, L., Georges, E., Gibson, C., Goel, V., Gogineni, T., Goh, G., Gontijo-Lopes, R., Gordon, J., Grafstein, M., Gray, S., Greene, R., Gross, J., Gu, S. S., Guo, Y., Hallacy, C., Han, J., Harris, J., He, Y., Heaton, M., Heidecke, J., Hesse, C., Hickey, A., Hickey, W., Hoeschele, P., Houghton, B., Hsu, K., Hu, S., Hu, X., Huizinga, J., Jain, S., Jain, S., Jang, J., Jiang, A., Jiang, R., Jin, H., Jin, D., Jomoto, S., Jonn, B., Jun, H., Kaftan, T., Łukasz Kaiser, Kamali, A., Kanitscheider, I., Keskar, N. S., Khan, T., Kilpatrick, L., Kim, J. W., Kim, C., Kim, Y., Kirchner, J. H., Kiros, J., Knight, M., Kokotajlo, D., Łukasz Kondraciuk, Kondrich, A., Konstantinidis, A., Kosic, K., Krueger, G., Kuo, V., Lampe, M., Lan, I., Lee, T., Leike, J., Leung, J., Levy, D., Li, C. M., Lim, R., Lin, M., Lin, S., Litwin, M., Lopez, T., Lowe, R., Lue, P., Makanju, A., Malfacini, K., Manning, S., Markov, T., Markovski, Y., Martin, B., Mayer, K., Mayne, A., McGrew, B., McKinney, S. M., McLeavey, C., McMillan, P., McNeil, J., Medina, D., Mehta, A., Menick, J., Metz, L., Mishchenko, A., Mishkin, P., Monaco, V., Morikawa, E., Mossing, D., Mu, T., Murati, M., Murk, O., Mély, D., Nair, A., Nakano, R., Nayak, R., Neelakantan, A., Ngo, R., Noh, H., Ouyang, L., O'Keefe, C., Pachocki, J., Paino, A., Palermo, J., Pantuliano, A., Parascandolo, G., Parish, J., Parparita, E., Passos, A., Pavlov, M., Peng, A., Perelman, A., de Avila Belbute Peres, F., Petrov, M., de Oliveira Pinto, H. P., Michael, Pokorny, Pokrass, M., Pong, V. H., Powell, T., Power, A., Power, B., Proehl, E., Puri, R., Radford, A., Rae, J., Ramesh, A., Raymond, C., Real, F., Rimbach, K., Ross, C., Rotsted, B., Roussez, H., Ryder, N., Saltarelli, M., Sanders, T., Santurkar, S., Sastry, G., Schmidt, H., Schnurr, D., Schulman, J., Selsam, D., Sheppard, K., Sherbakov, T., Shieh, J., Shoker, S., Shyam, P., Sidor, S., Sigler, E., Simens, M., Sitkin, J., Slama, K., Sohl, I., Sokolowsky, B., Song, Y., Staudacher, N., Such, F. P., Summers, N., Sutskever, I., Tang, J., Tezak, N., Thompson, M. B., Tillet, P., Tootoonchian, A., Tseng, E., Tuggle, P., Turley, N., Tworek, J., Uribe, J. F. C., Vallone, A., Vijayvergiya, A., Voss, C., Wainwright, C., Wang, J. J., Wang, A., Wang, B., Ward, J., Wei, J., Weinmann, C., Welihinda, A., Welinder, P., Weng, J., Weng, L., Wiethoff, M., Willner, D., Winter, C., Wolrich, S., Wong, H., Workman, L., Wu, S., Wu, J., Wu, M., Xiao, K., Xu, T., Yoo, S., Yu, K., Yuan, Q., Zaremba, W., Zellers, R., Zhang, C., Zhang, M., Zhao, S., Zheng, T., Zhuang, J., Zhuk, W., and Zoph, B. Gpt-4 technical report, 2024b. URL https://arxiv.org/abs/2303.08774.

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.

Prakash, J. and Buvanesh, A. What can you do when you have zero rewards during rl? *CoRR*, 2025.

Radhakrishna, S., Parikh, S., Sarda, G., Turkkan, A., Vohra, Q., Li, R., Jhamb, D., Ogueji, K., Shukla, A., Bamgbose, O., Liang, T., Kumar, L., Ostapenko, O., Malay, S. K. R., Tiwari, A., Bogavelli, T., Yadav, V., Mehta, J., Mittal, S., Kalkunte, A., Pattnaik, P., Slimi, K., Sreeram, A., Nair, J., Oladipo, A., Maiya, S., Mahajan, K., Maheshwary, R., Hashemi, M., Mudumba, S. R., Madhusudhan, S. T., Scholak, T., Paquet, S., Davasam, S., and Sunkara, S. Apriel-nemotron-15b-thinker, 2025.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y. K., Wu, Y., and Guo, D. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Shenfeld, I., Damani, M., Hübotter, J., and Agrawal, P. Self-distillation enables continual learning, 2026. URL https://arxiv.org/abs/2601.19897.

Shoemake, K. Animating rotation with quaternion curves. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, 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.

Stiennon, N., Ouyang, L., Wu, J., Ziegler, D. M., Lowe, R., Voss, C., Radford, A., Amodei, D., and Christiano, P. Learning to summarize from human feedback, 2022. URL https://arxiv.org/abs/2009.01325.

Stojanovski, Z., Stanley, O., Sharratt, J., Jones, R., Adefioye, A., Kaddour, J., and Köpf, A. Reasoning gym: Reasoning

environments for reinforcement learning with verifiable rewards, 2025. URL https://arxiv.org/abs/2505.24760.

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

Twist, L., Yannakoudakis, H., and Zhang, J. M. Reasoning-trace collapse: Evaluating the loss of explicit reasoning during fine-tuning, 2026. URL https://arxiv.org/abs/2605.21127.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Wei, J., Bosma, M., Zhao, V. Y., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. Finetuned language models are zero-shot learners, 2022. URL https://arxiv.org/abs/2109.01652.

Yadav, P., Tam, D., Choshen, L., Raffel, C., and Bansal, M. TIES-merging: Resolving interference when merging models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=xtaX3WyCj1.

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., et al. Qwen2.5 Technical Report. *arXiv Preprint*, 2024a. URL https://doi.org/10.48550/arXiv.2412.15115.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Zhang, P., Wang, P., Zhu, Q., Men, R., Gao, R., Liu, S., Luo, S., Li, T., Tang, T., Yin, W., Ren, X., Wang, X., Zhang, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Zhang, Y., Wan, Y., Liu, Y., Wang, Z., Cui, Z., Zhang, Z., Zhou, Z., and Qiu, Z. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Yang, E., Shen, L., Guo, G., Wang, X., Cao, X., Zhang, J., and Tao, D. Model merging in llms, mllms, and beyond: Methods, theories, applications and opportunities, 2024b. URL https://arxiv.org/abs/2408.07666.

Zhao, B., Srikanth, D., Wu, Y., and Jiang, Z. Specbench: Measuring reward hacking in long-horizon coding agents, 2026. URL https://arxiv.org/abs/2605.21384.

Zhou, X., Liu, Z., Sims, A., Wang, H., Pang, T., Li, C., Wang, L., Lin, M., and Du, C. Reinforcing general reasoning without verifiers. *CoRR*, 2025a.

Zhou, X., Liu, Z., Sims, A., Wang, H., Pang, T., Li, C., Wang, L., Lin, M., and Du, C. Reinforcing general reasoning without verifiers, 2025b. URL https://arxiv.org/abs/2505.21493.

Table 2. SummEval correlations for `Gemini-3-pro-preview`.

Metric	Spearman ρ	p-value
Fluency	0.6034	6.29×10^{-16}
Relevance	0.5616	1.69×10^{-14}
Coherence	0.6785	8.31×10^{-22}
Consistency	0.6791	7.44×10^{-22}

A. Experimental Details, Ablations and Case Study

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

A.1. Refinement of the Rust dataset

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`).

A.2. Validation of Gemini 3 Pro as Text Summarization Judge

As mentioned in §4, we validated the use of Gemini 3 Pro Preview as an LLM-as-a-judge metric for our evaluation of the text summarization task. For this validation, we ran the model on the SummEval dataset (Fabbri et al., 2021) and compared its ratings with the human annotations. We present the results in Table 2, showing that the model achieves generally high correlations of around 60% across the metrics and almost 70% on the consistency dimension.

A.3. Hyperparameters

For the main experiments, we tune the IFT hyperparameters separately for each model and dataset combination. The resulting configurations are shown in Table 3. The main runs use LoRA IFT with the target modules described in §3. We report the hyperparameters for the ablation runs in Table 4. The IFT+OPD rows only report the training parameters of the subsequent OPD training; the IFT part is identical to the IFT parameters in Table 3.

A.4. Main Result Score and Cost Table

A.5. Runtime Details

We report the stage-wise runtime breakdown for our method in Table 5. The averages are computed over two seeds where complete merge-search timing logs are available; for Olmo3, the selected configuration is the IFT checkpoint itself, so no merge-search timing is needed. All measurements use one NVIDIA H200 GPU.

Table 3. Hyperparameters used for the main model and dataset combinations. Batch size denotes the effective training batch size.

Dataset	Model	Epochs	Learning rate	Batch size	LoRA rank
Rust	OpenThinker 7B	5	5×10^{-5}	16	32
Rust	Olmo3 7B	3	5×10^{-5}	16	32
Rust	Apriel 15B	5	1×10^{-4}	64	32
Rust	R1-Distill-Qwen 7B	5	2×10^{-4}	16	64
Text sum.	OpenThinker 7B	1	5×10^{-5}	16	32
Text sum.	Olmo3 7B	1	1×10^{-5}	16	32
Text sum.	Apriel 15B	1	1×10^{-4}	64	32
Text sum.	R1-Distill-Qwen 7B	3	2×10^{-4}	32	64

Table 4. Hyperparameters used for the ablation and baseline runs. Batch size denotes the effective training batch size.

Dataset	Model	Method	Training length	Learning rate	Batch size	Extra
Rust	OpenThinker 7B	IFT+KL		IFT config		$\lambda = 5$
Rust	Olmo3 7B	IFT+KL		IFT config		$\lambda = 5$
Rust	Apriel 15B	IFT+KL		IFT config		$\lambda = 5$
Rust	R1-Distill-Qwen 7B	IFT+KL	5 epochs	5×10^{-5}	16	$\lambda = 5$
Text sum.	OpenThinker 7B	IFT+KL		IFT config		$\lambda = 5$
Text sum.	Olmo3 7B	IFT+KL		IFT config		$\lambda = 0$
Text sum.	Apriel 15B	IFT+KL		IFT config		$\lambda = 5$
Text sum.	R1-Distill-Qwen 7B	IFT+KL		IFT config		$\lambda = 5$
Rust	OpenThinker 7B	IFT+OPD	25 steps	2×10^{-4}	8	$r = 64$
Rust	Olmo3 7B	IFT+OPD	0 steps	–	–	–
Rust	Apriel 15B	IFT+OPD	50 steps	4×10^{-4}	4	$r = 32$
Rust	R1-Distill-Qwen 7B	IFT+OPD	25 steps	2×10^{-4}	8	$r = 64$
Text sum.	OpenThinker 7B	IFT+OPD	25 steps	2×10^{-4}	8	$r = 64$
Text sum.	Olmo3 7B	IFT+OPD	0 steps	–	–	–
Text sum.	Apriel 15B	IFT+OPD	50 steps	4×10^{-4}	4	$r = 32$
Text sum.	R1-Distill-Qwen 7B	IFT+OPD	25 steps	2×10^{-4}	8	$r = 64$
Rust	OpenThinker 7B	Full FT	5 epochs	5×10^{-5}	16	all weights

A.6. 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.

The first method, VeriFree (Zhou et al., 2025b), requires substantial resources. Based on our initial experiments, we estimated that a single training run would require eight NVIDIA GPUs for 60 hours. 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 7B 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 $\alpha \in \{0.01, 0.02, 0.05\}$. We set `max_token_length` $= 4096$ and discarded training samples exceeding this budget, leaving roughly one-third of the examples. The best-performing run used epoch 2, learning rate 5×10^{-6}, batch size 64, and EMA $\alpha = 0.05$, and completed in 4.5 hours using 4 NVIDIA H200 GPUs. The best performance we could achieve with SDFT was 41.4%, a slight decrease from the untuned performance of 44.2%, while performance on MATH500 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. We report the hyperparameters used in §A.

Table 5. Runtime breakdown for our method, in minutes, across IFT training (IFT), model merging (Merge), and evaluation of reasoning rates at candidate merge ratios for calibration (Cal. eval).

Task	Model	IFT	Merge	Cal. eval	Total
Rust	OpenThinker 7B	45.9	0.04	0.85	46.8
Rust	Apriel 15B	106.8	0.00	5.82	112.7
Rust	Olmo3 7B	31.4	–	–	31.4
Rust	R1-Distill-Qwen 7B	87.6	0.00	1.55	89.1
Text sum.	OpenThinker 7B	14.7	0.04	0.85	15.6
Text sum.	Apriel 15B	33.5	0.00	5.82	39.3
Text sum.	Olmo3 7B	16.9	–	–	16.9
Text sum.	R1-Distill-Qwen 7B	66.5	0.00	1.55	68.1
Average	Rust	67.9	0.01	2.06	70.0
Average	Text sum.	32.9	0.01	2.06	35.0
Average	All	50.4	0.01	2.06	52.5

B. Prompts

We use one task prompt for Rust code generation, one task prompt for text summarization, and four judge prompts for the text summarization evaluation. Figure 7 shows the Rust prompt used for both IFT training and MBPP Rust evaluation. Figure 9 shows the text summarization prompt used for both IFT training on the TLDR split and evaluation on the CNN split. For the LLM-based text summarization metrics, Gemini 3 Pro is prompted separately for each criterion: Figure 10 for relevance, Figure 11 for fluency, Figure 12 for factual consistency, and Figure 13 for coherence. In this section, we detail all prompts used for the respective models and tasks.

C. Case Study

In this section, we provide detailed outputs for a Rust MBPP example where the model is tasked with producing code that finds the correct insertion point in a sorted array. The base model Apriel 15B, shown in Figure 4, 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 5, 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 6, reasons like the base 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 base 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.

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 4. The non-tuned Apriel 15B 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 5. 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 6. 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 7. 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 8. 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 9. 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 10. Prompt used to evaluate summary relevance with Gemini 3 Pro. 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 11. Prompt used to evaluate summary fluency with Gemini 3 Pro.

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 12. Prompt used to evaluate summary consistency with Gemini 3 Pro.

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 13. Prompt used to evaluate summary coherence with Gemini 3 Pro.