Leveraging Instruction Tuning and Merging for Reasoning Model Adaptation

Anonymous Author(s)
Affiliation
Address
email

Abstract

Large reasoning models (LRM) have demonstrated impressive performance in domains such as mathematics and coding. These domains permit reliable verification of model outputs, important for enabling the reinforcement learning that drives LRM performance gains. However, training reasoning models on domains that lack reliable verifiers remains challenging. Meanwhile, for both verifiable and unverifiable domains, there exist large amounts of unused instruction-tuning data with human-written solutions. In this work, we show that this instruction-tuning data can be efficiently utilized to further improve reasoning models. For this, we first use classic instruction tuning, without reasoning traces, on the LRM. 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 LRM performance in both verifiable and hard-to-verify domains, including coding and text summarization, while preserving LRM capabilities across other domains. Importantly, our method is highly efficient, enabling such improvements for just a few tens of dollars.

1 Introduction

Reasoning language models (RLMs) have changed the frontier of language model capabilities. These models obtain strong results on tasks such as mathematics and programming [4, 19, 32]. The dominant training recipes behind these gains rely on reinforcement learning with verifiable rewards [4, 9, 18, 22]. However, this approach requires automatic verifiers that can automatically determine whether the proposed answer is objectively correct or not. This leaves a much wider class of tasks in an awkward position. Many domains do not provide reliable verifiers, such as text summarization or coding without available unit tests.

In these settings, large amounts of fine-tuning data are available: inputs paired with high-quality final outputs. However, they usually do not have validated reasoning traces. We are thus interested in whether powerful RLMs can be adapted to such target tasks while preserving and leveraging its reasoning behavior. The most direct approach is standard supervised fine-tuning (SFT) on the final answers. This is cheap and widely applicable, but it creates a distributional mismatch for RLMs, leading the RLM to stop reasoning.

Our Work In this work, we show that this mismatch can be mitigated with a simple two-step procedure. First, we perform standard SFT on the target input-output data, without reasoning traces. Second, we linearly merge the resulting SFT checkpoint with the original reasoning model. The merge coefficient is selected on a small calibration set by searching for the largest amount of the SFT update that preserves a high rate of non-empty reasoning traces. This procedure does not require a verifier, a reward model, or a stronger teacher model. It uses ordinary instruction-tuning data for adaptation and uses the original RLM as an anchor for recovering reasoning behavior.

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

We evaluate this approach on three open reasoning models, OpenThinker 7B, Apriel Nemotron 15B Thinker, and Olmo3 7B Think, across Rust coding and text summarization. These tasks cover both executable and hard-to-verify outputs, while MATH500 is held out as a measure of preserved general reasoning capability. Across settings, standard SFT 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 from SFT. In addition, this method is highly inexpensive, allowing adapting models within 2 hours for around $6.

Our contributions Our key contributions are:

- 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 SFT-and-merge method that adapts an RLM while selecting the merge ratio only from calibration reasoning behavior.
- We evaluate the method across three RLMs and two target tasks, showing that it preserves general reasoning capabilities while retaining target-task improvements.

2 Background

We outline the necessary background relevant to this work.

Language Models and Supervised Fine-Tuning 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:

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

where $\mathbf{c}_{<t} = (c_1, \ldots, c_{t-1})$ represents the preceding tokens. During inference, we split the context $\mathbf{c}$ into a pair $(\mathbf{x}, \mathbf{y})$ of user-provided input $\mathbf{x}$ and model-generated output $\mathbf{y}$. The first inference step samples y_0 from $p_\theta(y_0|\mathbf{x})$, and later steps obtain y_i from $p_\theta(y_i|\mathbf{x}+\mathbf{y}_{<i})$. We refer to $\mathbf{x}$ as the *prompt* and $\mathbf{y}$ as the *answer*. Pre-trained Large Language Models (LLMs) are such models with billions of parameters, trained on trillions of tokens of training data [4, 20, 32]. By such pre-training the models acquire a variety of general skills, in particular language-understanding. However, in practice it is often necessary to adapt the model for particular skills or specialized tasks.

Adapting LLMs to such tasks is commonly achieved through Supervised Fine-Tuning (SFT). SFT is a process that updates a model's parameters using a labeled dataset $\mathcal{D} = \{(\mathbf{x}, \mathbf{y})\}$ of input-output pairs. The training objective is to minimize the cross-entropy loss. The SFT loss function is defined as:

$$\mathcal{L}_{\text{SFT}}(\theta, \mathcal{D}) = -\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}} \left[\sum_{t=1}^{|\mathbf{y}|} \log p_\theta(y_t \mid \mathbf{x} + \mathbf{y}_{<t}) \right]. \tag{2}$$

Large Reasoning Models Recently models are trained on challenging tasks using reinforcement learning, resulting in Large Reasoning Models [4, 19]. The models are prompted to solve a task that permits a reliably verifiable answer, such as a numerical result or executable code. The model is then trained to prefer answers that are validated as correct by the respective verifier. Before producing the answer, the model may generate text in an internal scratch pad, which is referred to as reasoning trace. Empirically, this results in strong solutions for challenging math and code problems [4, 23].

However, there are three limitations. First, self-bootstrapping only works if the model has a non-zero solve rate on the dataset in question [21]. Second, reinforcement learning is an expensive process, that requires several rollouts and many samples [4, 18, 23]. 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 [2, 4, 16]. There are numerous tasks for which such a verifier is not available, for example text summarization.

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 $\mathbf{y}'$ are obtained by sampling an already reasoning-trained RLM, filtering the solutions such that $\mathbf{y}'$ matches the desired solution $\mathbf{y}$. This is usually done using RLMs that are larger than the target model, or to save the cost and effort associated with reinforcement learning [4, 17]. The drawback is that such an RLM with better performance on the desired task has to be available.

Model Merging Model merging combines two or more model checkpoints into a single checkpoint without additional optimization. For two sets of compatible parameters θ_1 and θ_2, a common form is a convex interpolation

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

This can be used for transfer, specialization, and recovering forgotten behavior across related tasks or fine-tuning regimes. 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 [1, 6, 11, 33].

3 Training RLMs via SFT and Model Merging

In this section we describe our core pipeline used to adapt RLMs to a target task using reasoning-free training data.

Adapting RLM In this work we focus on task adaptation, concretely: How to improve the performance of an existing RLM on a target task. The main approach for this is to continue reinforcement learning with verifiable rewards [4, 12], which requires reliable verifiers. Alternatively, developers perform reasoning distillation from stronger models [34]. Both approaches require either a reliable verifier or a stronger RLM on the target task.

Main Challenges The key challenge for training RLM is the requirement to preserve the reasoning trace, which is crucial for the model performance [32]. Thus, it is typically assumed that the training data must provide reasoning traces, which are included in the model training objective [4, 18]. However, if no verifier is available for the given task, or no reasoning model with a higher solve rate on the dataset in question is available, we can not easily obtain relevant reasoning traces for the given task.

Our work resolves these challenges by presenting a method for training RLM without requiring reasoning traces. As such our method is able to leverage largely-available SFT datasets, while preserving the reasoning trace and its associated model performance. Due to the design of this method, it neither requires a robust verifier nor a stronger reasoning model.

Overview Our core method is a two-step pipeline, visualized in §3: Given a base RLM M, we first perform SFT on a task-specific training set $\mathcal{D}_{\text{train}}$ that contains no reasoning traces. This produces a fine-tuned model M_{SFT}. We then linearly merge M_{SFT} with the original model M with coefficient α. The merge coefficient α is selected on a held-out calibration set $\mathcal{D}_{\text{cal}}$ so that the resulting model remains close to M_{SFT} while preserving the model's reasoning behavior, measured by the rate of non-empty reasoning traces. This gives an adapted model that can benefit from the task-specific SFT data without requiring a verifier or a stronger teacher model to generate reasoning traces.

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. We denote an empty reasoning trace by ε.

Finetuning We first fine-tune the base model M on $\mathcal{D}_{\text{train}}$ using standard SFT with LoRA [10]. 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. This reduces the memory footprint and training cost compared with full fine-tuning [29].

A reasoning model is normally trained to produce both a reasoning trace and a final answer. In our setting, however, $\mathcal{D}_{\text{train}}$ contains only (i, o) pairs and does not contain the reasoning trace r. We

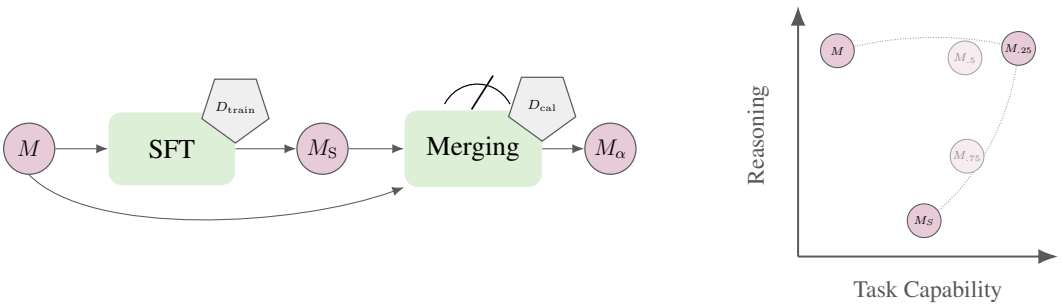

Figure 1: Our core method for RLM training consists of a light-weight two-step pipeline (Left): We first perform standard SFT on D_{train}, which produces M_{SFT}. We then merge M and M_{SFT}, using a calibration dataset D_{cal} to find a merge coefficient α that preserves model reasoning. (Right) This merging produces a point that maintains and sometimes even improves task capability, while preserving the RLM reasoning ability.

therefore serialize each training target as (ε, o): the empty reasoning trace ε followed by the target output. Let $y(o) = \text{serial}(\varepsilon, o)$ denote this serialized response, and let $y_t(o)$ be its t-th token. The SFT objective is the standard next-token prediction loss on this target,

$$\theta_{\text{SFT}} \leftarrow \arg\min_{\theta} \quad - \sum_{(i,o)\in\mathcal{D}_{\text{train}}} \sum_{t=1}^{|y(o)|} \log p_\theta\big(y_t(o) \mid i, y_{<t}(o)\big), \tag{3}$$

where θ_{SFT} denotes the weights of the resulting model M_{SFT}. Because the fine-tuning targets contain an empty reasoning trace, this step can reduce the model's reasoning behavior: after SFT, M_{SFT} may directly output the final answer on examples where the original reasoning model would have produced a non-empty trace. We observe this behavior for most of the reasoning models we evaluated.

Merging To recover reasoning behavior while retaining the task adaptation learned during SFT, we perform a two-way interpolation between the original reasoning model M and its standard-SFT variant M_{SFT}. This choice is motivated by the use of merging to mitigate forgetting [1].

We select α using a search over model reasoning on a 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.

$$\rho(M', \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(i,o)\in\mathcal{D}} \mathbf{1}\{r_i \neq \varepsilon\}, \qquad (r_i, \hat{o}_i) \sim M'(i). \tag{4}$$

We design a search procedure to pick the merge coefficient that is as close as possible to the SFT model, preserving the target task performance gains, while recovering the model reasoning. Let ρ_{min} be the minimum acceptable reasoning rate; in our experiments, $\rho_{\text{min}} = 0.9$. We are then looking for the largest merge ratio whose calibration reasoning rate remains acceptable, i.e.,

$$\alpha^\star = \max_{\alpha\in\mathcal{A}} \{\alpha : \rho_{\text{min}} \leq \rho(M_\alpha, \mathcal{D}_{\text{cal}})\}. \tag{5}$$

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

Algorithm 1 RLM Training Pipeline

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

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

We describe the searching algorithm in Algorithm 1. In practice, we evaluate a small uniform grid of merge coefficients 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 SFT model. After evaluating all grid points on $\mathcal{D}_{\text{cal}}$, we select the largest α whose reasoning rate is at least ρ_{min}.

Optimizations Two optimizations speed up this process even further: 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 usually emits the end-of-reasoning token immediately after the start-of-reasoning token. 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. We can further improve the precision of this estimate by inspecting the model logits directly at the decision point, measuring the probability assigned to immediately closing the reasoning trace instead of relying only on sampled completions.

Second, we reduce the computational effort for the search in Algorithm 1 by converting it into a binary search over merge ratios. For this, we observe that the amount of reasoning monotonously decreases over the merge ratio. This allows us to employ binary search instead of grid search in compute-restrained settings. We employ binary search over the reasoning on the calibration dataset exceeding the minimum reasoning threshold. Since we also observe that model reasoning typically 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 or equal to the minimal ratio.

4 Experimental Evaluation

In this section we demonstrate that our method reliably obtains a strong tradeoff between model performance at the target task and general reasoning capabilities across three different reasoning models and two datasets.

4.1 Experimental Setup

Below, we describe our experimental setup, including the LRMs used, training techniques, target tasks, metrics, and training datasets.

Models We compare three key models: OpenThinker 7B [9], Apriel Nemotron 15B Thinker [22] and Olmo3 7B Think [18]. This covers a diverse set of reasoning models: OpenThinker is a fine-tune based on Qwen2.5 7B [31], distilled on reasoning traces by DeepSeek R1 [4]. Apriel 15B is a fine-tuned version of Apriel 15B Base, post trained through CPT, SFT and GRPO. Olmo3 was trained from scratch with a fully open pipeline, including SFT, DPO and Reinforcement Learning from Verifiable Rewards [18].

Methods We first evaluate the unadapted RLM as *Baseline*. We compare it to the fully fine-tuned variant of the model using LoRA, called *SFT*, trained as described in the first step of our training pipeline without any reasoning traces. Unless otherwise indicated, we finetune the hyperparameters learning rate, batch size and epoch for each combination of model and dataset, and report the hyperparameters in Appendix A. Finally we evaluate *Ours*, the re-merged version of the model at the optimal merge ratio determined by our method using the described binary search method with up to 8 search steps. We further attempt reproducing two related works: *SDFT* [24], which leverages the model for self-guided reasoning synthesization and *VeriFree* [36], which performs direct learning on the golden answers.

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

Rust coding: Given a natural language description of a coding task, implement a function that solves the problem.

Text summarization: Provided with a long-form natural language text, produce a concise, relevant, cohesive and consistent summary.

Mathematical problem solving: Provided with a high-school level mathematical question, derive a numerical answer.

The tasks are highly different in complexity and difficulty. In addition, both tasks do in general not permit reliable verifiers, making it difficult to train on using RLVR. For Rust coding, RLVR is only possible when comprehensive test suites are provided, which is not always possible.

Metrics For Rust coding, we choose a Rust translation of the MBPP dataset [13] and measure the models PASS@1 performance [3], in other words, we measure the percentage of solutions that implement a function described in natural language correctly, as measured by a set of unit tests. For text summarization, we measure fluency, consistency, relevance and coherence on the CNN split of Fabbri et al. [5]. We use Gemini 3 Pro [8] to evaluate the criteria according to a detailed prompt, detailed in Appendix B. To confirm the validity of these results, we compare the Gemini 3 Pro ratings on the dataset of summaries with human annotators provided in Fabbri et al. [5] and establish a Cohens Kappa of over 60% for each metric. We provide the details in Appendix A. For the mathematical problem solving, we measure the performance of the models on the MATH500 dataset [15]. Note that we do not train on this task, and we therefore use it to assess the loss of general reasoning capabilities of the model due to fine-tuning.

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

Training Datasets We devise two datasets for training. To ensure that we measure whether the model preserved its generalization capability, we design the training and test datasets to stem from different distributions, while having similar task formats. (1) *Rust coding*: We train the model on a set of single-function coding tasks in the Rust language, where each data point is a pair of task and code solution that were synthesized from an LLM [28]. We filter the dataset to the subset of code samples that pass the corresponding test suite consistently, arriving at a subset of 6761 training samples. We confirm the obtained datasets to be disjoint from the MBPP dataset by checking for closest matches using cosine similarity and confirming their distinctness. (2) *Text summarization*: We further train the model on the task of text summarization. As training dataset, we use the reddit TLDR; split of Fabbri et al. [5], which is intentionally different to the CNN task split that we use for evaluation.

4.2 Main Results

Next, we present our main results on Rust coding and text summarization.

Rust coding We train all models on both Rust coding and text summarization and show their performance on the respective tasks compared to the performance on MATH500 in Figure 2. As can be seen, standard SFT strongly reduces the performance on MATH500: For OpenThinker 7B, it drops from 79% to 34.2% when training on the Rust dataset, and to 2% on the text summarization dataset. While SFT does not disturb the math reasoning capabilities of Apriel 15B, it decreases its performance *at the target task*. Meanwhile, our method consistently picks a strong trade-off. In Rust, the recovered reasoning even leads to improved performance compared to both the starting model (baseline) and the SFT model, with an average increase of 4.1 percentage points.

Text Summarization For text summarization, we observe that fluency, relevance and coherence are already close to the maximal score for all evaluated models, on average 4.84 out of 5, and remain stable across our training. We therefore focus on the metric of consistency and report the performance on other metrics in Appendix A. Here, the best performance is usually achieved by the SFT model, with model performance on average increasing by 1.10 points. This increase occurs despite as model reasoning decreasing rapidly to 0.0% and 12.9% for OpenThinker 7B and Apriel 15B, respectively. With our method, reasoning is fully recovered, resulting in a parallel recovery of MATH500 performance, while preserving 76.7% and 97.4% of the performance increase for the fully finetuned text consistency.

Runtime and Cost A key benefit of our method is its low cost. Our adaptation method requires on average less than two hours to complete and fits on a single NVIDIA H200 GPU, costing in total less than USD 6 on typical rental services at the time of writing. Most time is split between obtaining the

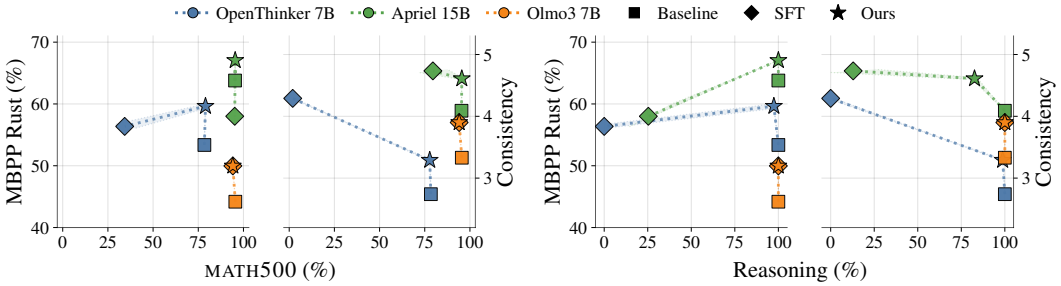

Figure 2: (Left) Our method preserves MATH500 performance while preserving or even increasing the task gains from SFT on Rust coding and text summarization. (Right) The preserved capability correlates with whether the model continues to reason on an out-of-distribution task.

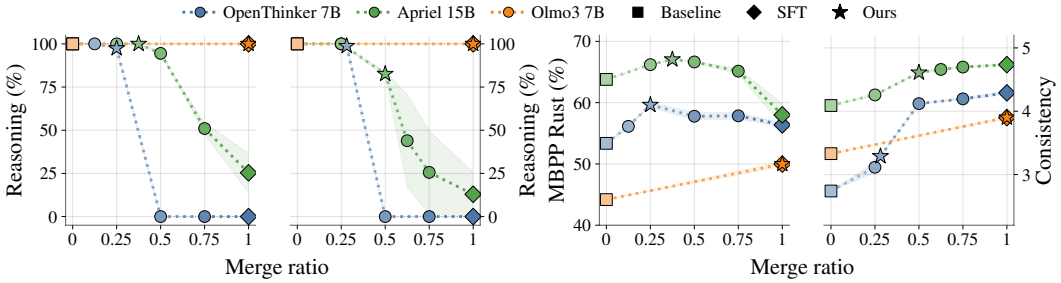

Figure 3: We evaluate all various merge ratios on the final test set. We observe that reasoning on MATH500 tasks drops sharply after a merging threshold is crossed (Left) and target task performance experiences a smoother curve (Right). On Rust, a clear benefit of restored reasoning to task performance is visible.

fine-tuned model, with on average 36 minutes, and evaluating the merge ratios in Algorithm 1, with on average 23 min. Note that a key factor in our cost-efficiency is that our method does not require any reasoning rollouts from the model.

Other Methods We attempted reproducing two popular related methods that are also applicable to our setting of training RLMs using only standard SFT data. The first method, VeriFree [36], requires a significant amount of resources: Based on our initial experiments, we estimated 8 NVIDIA GPUs for a total of 60 hours for a single training run. Given this cost, which applies to each step of a preliminary hyperparameter search for fair comparison, we consider a comparison infeasible. Second, for SDFT [24] we have extensively tried to reproduce SDFT on Olmo3 7B, on the task of Rust coding, exploring over 20 hyperparameter combinations. Despite reaching out to the authors, the best performance we could achieve with SDFT was 41.4%, a slight decrease from the baseline 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 for direct comparisons in our experiments. We report the used hyperparameters in Appendix A.

4.3 Ablation

We ablate over the choice of merge ratios, employed finetuning techniques and training hyperparameters for the supervised finetuning.

Merge ratio We ablate over the merging factor α on the Rust and text summarization dataset, and show their reasoning rate on MATH500 and the target tasks in Figure 3 (left and right, respectively). We notice that OpenThinker 7B loses its reasoning capability quickly for $\alpha \in [0.25, 0.5]$, while Apriel 15B loses it more slowly and Olmo3 7B even maintains full reasoning at $\alpha = 1$. Surprisingly, the performance on Rust coding appears to peak in for $\alpha \in [0.25, 0.5]$, where the model has still knowledge about the task from training and recovered reasoning. Meanwhile, consistency in the text summarization task increases more linearly with the merging ratio. Our method reliably picks

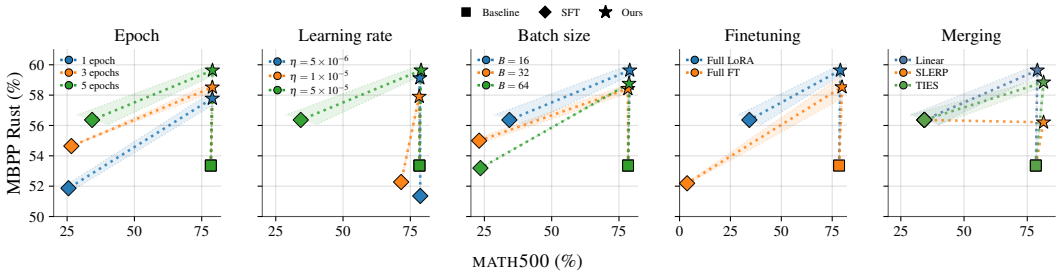

Figure 4: OpenThinker 7B ablations on Rust coding, from left to right: training epochs, learning rate η, batch size B, LoRA vs. full finetuning, and merging techniques. The overall trend of reasoning loss in standard SFT and reasoning and performance recovery is stable across all settings.

a point close to a good trade-off point between reasoning and task performance optimum, despite picking the point only based on the reasoning score on the calibration dataset.

Merging technique We ablate over the employed merging technique for OpenThinker 7B on Rust coding, presenting the results in the right-most panel of Figure 4. We compare the linear merging employed in our main experiments with Spherical Linear intERPolation (SLERP) [7, 25] and TIES [30]. 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 density to 0.5 and search over the α parameter. For all merging techniques, our method obtains the best merging ratio at $\alpha = 0.25$. Compared to linear interpolation, SLERP recovers more MATH500 performance but loses target-task performance, while TIES improves both MATH500 and target-task performance, albeit less than our linear merging.

Finetuning Our main method uses LoRA for light weight and efficient finetuning. We ablate over the choice of LoRA by using full finetuning of all model weights, using the same hyperparameters as for LoRA, and present the results in the second panel from left of Figure 4. We observe that the capability loss on MATH500 is much stronger with standard finetuning, and that our method is unable to obtain as much task specific performance as LoRA.

Hyperparameters We ablate over finetuning hyperparameters for the supervised finetuning stage over OpenThinker 7B in Figure 4. We ablate over varying epochs, learning parameters η, and batch sizes B, keeping the remaining hyperparameters fixed. Importantly, across epoch, learning rate and batch size choices, even though some finetunings result in models with worse performance on the target task than the baseline model, the overall trend is consistent that training disturbs the natural reasoning of the model, while merging can recover the MATH500 performance and maintain improvements on the target task.

5 Related Work

Below, we present an overview of prior work related to our method.

Training Reasoning Models with Verifiers The standard technique for training and adapting reasoning models is via reinforcement learning with verifiable rewards [4, 18, 26]. This works well for tasks that permit such verifiers, however leaves out many domains, such as the explored domain of text summarization. Moreover, while this technique is well-explored for bootstrapping reasoning model performance, continuous adaptation of RLMs to new tasks via RL is not well explored yet.

Training Reasoning Models without Verifiers Verifier-free or verifier-light methods try to enable training reasoning models without reasoning traces, by optimizing likelihood of known answers directly [35, 36] or training teacher models to provide feedback and justifications for reference answers [24]. These methods are closest in motivation to our setting, but they are computationally much more expensive and not designed to work on already trained reasoning models. Our method instead operates on minimal hardware requirements and allows inexpensive, fast task adaptation.

Model merging Model merging combines multiple checkpoints into a single model without additional gradient updates. A common motivation is to combine task-specific skills or mitigate forgetting by merging related models [1, 33]. Prior methods include simple weight averaging, task arithmetic [11], and sparse or sign-based variants such as TIES [30]. Linear interpolation is also connected to linear mode connectivity, where independently trained or fine-tuned networks can lie on low-loss paths in weight space [6]. Most merging work aims to combine capabilities from multiple specialized models. Our use is narrower: we merge an SFT 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 [14]. We build on this observation, but select the merge coefficient using a small calibration set and target the practical setting where only input-output SFT data is available.

6 Limitations and 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 generalist and continuously adapted to new tasks, the question remains how to obtain adaptation to several tasks in parallel. We consider this an exciting avenue for future research.

Interpretability In our experiments we observe that models stop reasoning by immediately producing an end-of-reasoning marker. To address this, during the development of the method, we tried prefilling the model response with a start-of-reasoning marker and an immediately following non-end-of-reasoning token like `'Okay'`, however, the reasoning performance 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 reasoning models, in particular in terms of recovering reasoning tendency.

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 SFT 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—in the same paradigm as in the pre LRM 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—likely due to the lack of supervision signal improving the reasoning process of the model itself.

7 Conclusion

We studied how to adapt reasoning language models when only ordinary input-output supervision is available. Our results show that this failure mode is not merely cosmetic. In several settings, the loss of reasoning coincides with large loss of performance on held-out mathematical reasoning and weaker generalization on the target task. We introduced a simple two-step pipeline to mitigate this issue: first fine-tune the reasoning model with standard SFT, then merge the fine-tuned checkpoint back with the original model. The merge ratio is selected using only a small calibration set and the observed rate of non-empty reasoning traces.

Across Rust coding and text summarization, this procedure recovers most or all of the reasoning behavior lost under SFT while retaining much of the target-task improvement, and does so without a verifier, a reward model, or a stronger teacher model, at a cost comparable to running the initial SFT and a small grid search. Overall, our results indicate that RLMs can be efficiently adapted to target tasks using standard model training techniques.

References

[1] 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 *EMNLP (Findings)*, 2024.

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

[3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating Large Language Models Trained on Code. *arXiv Preprint*, 2021. URL https://arxiv.org/abs/2107.03374.

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

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

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

[7] 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, 2025. URL https://arxiv.org/abs/2403.13257.

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

[9] Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, Ashima Suvarna, Benjamin Feuer, Liangyu Chen, Zaid Khan, Eric Frankel, Sachin Grover, Caroline Choi, Niklas Muennighoff, Shiye Su, Wanjia Zhao, John Yang, Shreyas Pimpalgaonkar, Kartik Sharma, Charlie Cheng-Jie Ji, Yichuan Deng, Sarah Pratt, Vivek Ramanujan, Jon Saad-Falcon, Jeffrey Li, Achal Dave, Alon Albalak, Kushal Arora, Blake Wulfe, Chinmay Hegde, Greg Durrett, Sewoong Oh, Mohit Bansal, Saadia Gabriel, Aditya Grover, Kai-Wei Chang, Vaishaal Shankar, Aaron Gokaslan, Mike A. Merrill, Tatsunori Hashimoto, Yejin Choi, Jenia Jitsev, Reinhard Heckel, Maheswaran Sathiamoorthy, Alexandros G. Dimakis, and Ludwig Schmidt. Openthoughts: Data recipes for reasoning models, 2025. URL https://arxiv.org/abs/2506.04178.

[10] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

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

[12] KRAFTON and SKT. Continual post-training of llms via offline grpo for mathematical reasoning, 2025. URL https://krafton-ai.github.io/blog/llm_post_training_en/. Accessed: 2025-07-28.

[13] Northeastern University Programming Research Lab. Multipl-e (revision 40ca145), 2025. URL https://huggingface.co/datasets/nuprl/MultiPL-E.

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

[15] 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, 2023. URL https://arxiv.org/abs/2305.20050.

[16] Monte MacDiarmid, Benjamin Wright, Jonathan Uesato, Joe Benton, Jon Kutasov, Sara Price, Naia Bouscal, Sam Bowman, Trenton Bricken, Alex Cloud, Carson Denison, Johannes Gasteiger, Ryan Greenblatt, Jan Leike, Jack Lindsey, Vlad Mikulik, Ethan Perez, Alex Rodrigues, Drake Thomas, Albert Webson, Daniel Ziegler, and Evan Hubinger. Natural emergent misalignment from reward hacking in production rl, 2025. URL https://arxiv.org/abs/2511.18397.

[17] NovaSky Team. Sky-t1: Train your own o1 preview model within $450. https://novasky-ai.github.io/posts/sky-t1, 2025. Accessed: 2025-01-09.

[18] Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3, 2025. URL https://arxiv.org/abs/2512.13961.

[19] OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrew Duberstein, Andrew Kondrich, Andrey Mishchenko, Andy Applebaum, Angela Jiang, Ashvin Nair, Barret Zoph, Behrooz Ghorbani, Ben Rossen, Benjamin Sokolowsky, Boaz Barak, Bob McGrew, Borys Minaiev, Botao Hao, Bowen Baker, Brandon Houghton, Brandon McKinzie, Brydon Eastman, Camillo Lugaresi, Cary Bassin, Cary Hudson, Chak Ming Li, Charles de Bourcy, Chelsea Voss, Chen Shen, Chong Zhang, Chris Koch, Chris Orsinger, Christopher Hesse, Claudia Fischer, Clive Chan, Dan Roberts, Daniel Kappler, Daniel Levy, Daniel Selsam, David Dohan, David Farhi, David Mely, David Robinson, Dimitris Tsipras, Doug Li, Dragos Oprica, Eben Freeman, Eddie Zhang, Edmund Wong, Elizabeth Proehl, Enoch Cheung, Eric Mitchell, Eric Wallace, Erik Ritter, Evan Mays, Fan Wang, Felipe Petroski Such, Filippo Raso, Florencia Leoni, Foivos Tsimpourlas, Francis Song, Fred von Lohmann, Freddie Sulit, Geoff Salmon, Giambattista Parascandolo, Gildas Chabot, Grace Zhao, Greg Brockman, Guillaume Leclerc, Hadi Salman, Haiming Bao, Hao Sheng, Hart Andrin, Hessam Bagherinezhad, Hongyu Ren, Hunter Lightman, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian Osband, Ignasi Clavera Gilaberte, Ilge Akkaya, Ilya Kostrikov, Ilya Sutskever, Irina Kofman, Jakub Pachocki, James Lennon, Jason Wei, Jean Harb, Jerry Twore, Jiacheng Feng, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joaquin Quiñonero Candela, Joe Palermo, Joel Parish, Johannes Heidecke, John Hallman, John Rizzo, Jonathan Gordon, Jonathan Uesato, Jonathan Ward, Joost Huizinga, Julie Wang, Kai Chen, Kai Xiao, Karan Singhal, Karina Nguyen, Karl Cobbe, Katy Shi, Kayla Wood, Kendra Rimbach, Keren Gu-Lemberg, Kevin Liu, Kevin Lu, Kevin Stone, Kevin Yu, Lama Ahmad, Lauren Yang, Leo Liu, Leon Maksin, Leyton Ho, Liam Fedus, Lilian Weng, Linden Li, Lindsay McCallum, Lindsey Held, Lorenz Kuhn, Lukas Kondraciuk, Lukasz Kaiser, Luke Metz, Madelaine Boyd, Maja Trebacz, Manas Joglekar, Mark Chen, Marko Tintor, Mason Meyer, Matt Jones, Matt Kaufer, Max Schwarzer, Meghan Shah, Mehmet Yatbaz, Melody Y. Guan, Mengyuan Xu, Mengyuan Yan, Mia Glaese, Mianna Chen, Michael Lampe, Michael Malek, Michele Wang, Michelle Fradin, Mike McClay, Mikhail Pavlov, Miles Wang, Mingxuan Wang, Mira Murati, Mo Bavarian, Mostafa Rohaninejad, Nat McAleese, Neil Chowdhury, Neil Chowdhury, Nick Ryder, Nikolas Tezak, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Patrick Chao, Paul Ashbourne,

Pavel Izmailov, Peter Zhokhov, Rachel Dias, Rahul Arora, Randall Lin, Rapha Gontijo Lopes, Raz Gaon, Reah Miyara, Reimar Leike, Renny Hwang, Rhythm Garg, Robin Brown, Roshan James, Rui Shu, Ryan Cheu, Ryan Greene, Saachi Jain, Sam Altman, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Santiago Hernandez, Sasha Baker, Scott McKinney, Scottie Yan, Shengjia Zhao, Shengli Hu, Shibani Santurkar, Shraman Ray Chaudhuri, Shuyuan Zhang, Siyuan Fu, Spencer Papay, Steph Lin, Suchir Balaji, Suvansh Sanjeev, Szymon Sidor, Tal Broda, Aidan Clark, Tao Wang, Taylor Gordon, Ted Sanders, Tejal Patwardhan, Thibault Sottiaux, Thomas Degry, Thomas Dimson, Tianhao Zheng, Timur Garipov, Tom Stasi, Trapit Bansal, Trevor Creech, Troy Peterson, Tyna Eloundou, Valerie Qi, Vineet Kosaraju, Vinnie Monaco, Vitchyr Pong, Vlad Fomenko, Weiyi Zheng, Wenda Zhou, Wes McCabe, Wojciech Zaremba, Yann Dubois, Yinghai Lu, Yining Chen, Young Cha, Yu Bai, Yuchen He, Yuchen Zhang, Yunyun Wang, Zheng Shao, and Zhuohan Li. Openai o1 system card, 2024. URL https://arxiv.org/abs/2412.16720.

[20] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu,

Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

[21] Jatin Prakash and Anirudh Buvanesh. What can you do when you have zero rewards during rl? *CoRR*, 2025.

[22] Shruthan Radhakrishna, Soham Parikh, Gopal Sarda, Anil Turkkan, Quaizar Vohra, Raymond Li, Dhruv Jhamb, Kelechi Ogueji, Aanjaneya Shukla, Oluwanifemi Bamgbose, Toby Liang, Luke Kumar, Oleksiy Ostapenko, Shiva Krishna Reddy Malay, Aman Tiwari, Tara Bogavelli, Vikas Yadav, Jash Mehta, Saloni Mittal, Akshay Kalkunte, Pulkit Pattnaik, Khalil Slimi, Anirudh Sreeram, Jishnu Nair, Akintunde Oladipo, Shashank Maiya, Khyati Mahajan, Rishabh Maheshwary, Masoud Hashemi, Sai Rajeswar Mudumba, Sathwik Tejaswi Madhusudhan, Torsten Scholak, Sebastien Paquet, Sagar Davasam, and Srinivas Sunkara. Apriel-nemotron-15b-thinker, 2025.

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

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

[25] Ken Shoemake. Animating rotation with quaternion curves. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, page 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.

[26] 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, 2025. URL https://arxiv.org/abs/2505.24760.

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

[28] 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/, 2024. Technical report.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

[30] 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.

[31] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 Technical Report. *arXiv Preprint*, 2024. URL https://doi.org/10.48550/arXiv.2412.15115.

[32] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

[33] 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, 2024. URL https://arxiv.org/abs/2408.07666.

[34] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STar: Bootstrapping reasoning with reasoning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_3ELRdg2sgI.

[35] Xiangxin Zhou, Zichen Liu, Anya Sims, Haonan Wang, Tianyu Pang, Chongxuan Li, Liang Wang, Min Lin, and Chao Du. Reinforcing general reasoning without verifiers. *CoRR*, 2025.

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

Table 1: 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, datasets, and so on.

A.1 Refinement of the Rust dataset

In our study, the Rust corpus from Team [27, 28] 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 LLM-as-a-Judge metric for our evaluation of the Text Summarization task. For this, we run the model on the SummEval dataset [5] and compare the model scores with the human annotations. We present the results in Table 1, showing that the model achieves generally high correlation of around 60% on all scores, and almost 70% in the main inspected metric of consistency.

A.3 SFT Hyperparameters

For the main experiments, we tune the SFT hyperparameters separately for each model and dataset combination. The resulting configurations are shown in Table 2. The main runs use LoRA SFT with the target modules described in §3. We also include the full-finetuning configuration used for the OpenThinker 7B Rust ablation.

For the SDFT [24] training result, we perform 26 runs with the following SFT 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 about five hours using 4 NVIDIA H200 GPUs. The full four-panel text-summarization tradeoff is shown in Figure 5.

Table 2: SFT hyperparameters used for the main model and dataset combinations, plus the full-finetuning ablation. Batch size denotes the effective training batch size.

Dataset	Model	Method	Epochs	Learning rate	Batch size
Rust coding	OpenThinker 7B	LoRA	5	5×10^{-5}	16
Rust coding	OpenThinker 7B	Full FT	5	5×10^{-5}	16
Rust coding	Apriel 15B	LoRA	5	1×10^{-4}	64
Rust coding	Olmo3 7B	LoRA	3	5×10^{-5}	16
Text summarization	OpenThinker 7B	LoRA	1	5×10^{-5}	16
Text summarization	Apriel 15B	LoRA	1	1×10^{-4}	64
Text summarization	Olmo3 7B	LoRA	1	1×10^{-5}	16

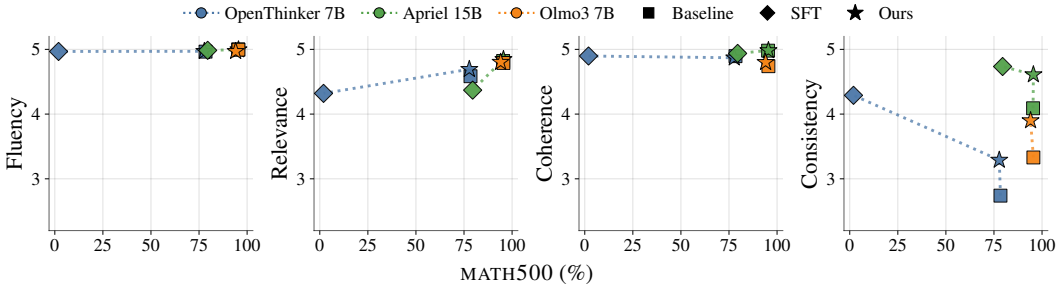

Figure 5: Text summarization tradeoff from SFT/merging experiments: Fluency, Relevance, Coherence, and Consistency against MATH-500.

A.4 Experimental Details on other Text Summarization metrics

In Figure 5, we demonstrate the performance of the trained models across all four metrics of fluency, relevance, coherence and consistency. As can be seen, the performance in the former three categories is already high in the baseline model, and remains stable across our training. The strongest change outside of consistency is due to a loss of relevance in SFT models, which is fully recovered by our method.

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 6 shows the Rust prompt used for both SFT training and MBPP Rust evaluation. Figure 8 shows the text summarization prompt used for both SFT 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 9 for relevance, Figure 10 for fluency, Figure 11 for factual consistency, and Figure 12 for coherence. In this section, we detail all prompts used for the respective models and tasks. Each prompt is shown in the same minipage-based figure format as the prompt examples in the original template.

````
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 6: 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 7: 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 8: 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 9: 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 10: 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 11: 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 12: Prompt used to evaluate summary coherence with Gemini 3 Pro.