Nemotron-Cascade: Scaling Cascaded Reinforcement Learning for General-Purpose Reasoning Models

Boxin Wang[*], **Chankyu Lee**[*], **Nayeon Lee**[*], **Sheng-Chieh Lin**[*], **Wenliang Dai**[*], **Yang Chen**[*], **Yangyi Chen**[*], **Zhuolin Yang**[*], **Zihan Liu**[*], **Mohammad Shoeybi**, **Bryan Catanzaro**, **Wei Ping**[*†]

Abstract

Building general-purpose reasoning models with reinforcement learning (RL) entails substantial cross-domain heterogeneity, including large variation in inference-time response lengths and verification latency. Such variability complicates the RL infrastructure, slows training, and makes training curriculum (e.g., response length extension) and hyperparameter selection challenging. In this work, we propose cascaded domain-wise reinforcement learning (Cascade RL) to develop general-purpose reasoning models, Nemotron-Cascade, capable of operating in both *instruct* and deep *thinking* modes. Departing from conventional approaches that blend heterogeneous prompts from different domains, Cascade RL orchestrates sequential, domain-wise RL, reducing engineering complexity and delivering state-of-the-art performance across a wide range of benchmarks. Notably, RLHF for alignment, when used as a pre-step, boosts the model's reasoning ability far beyond mere preference optimization, and subsequent domain-wise RLVR stages rarely degrade the benchmark performance attained in earlier domains and may even improve it (see an illustration in Figure 1). Our 14B model, after RL, outperforms its SFT teacher, DeepSeek-R1-0528, on LiveCodeBench v5/v6/Pro and achieves silver-medal performance in the 2025 International Olympiad in Informatics (IOI). We transparently share our training and data recipes.

[*]Equal technical contribution, with author names ordered alphabetically by first name.
[†]Leads the effort. Correspondence to: <wping@nvidia.com>.

Contents

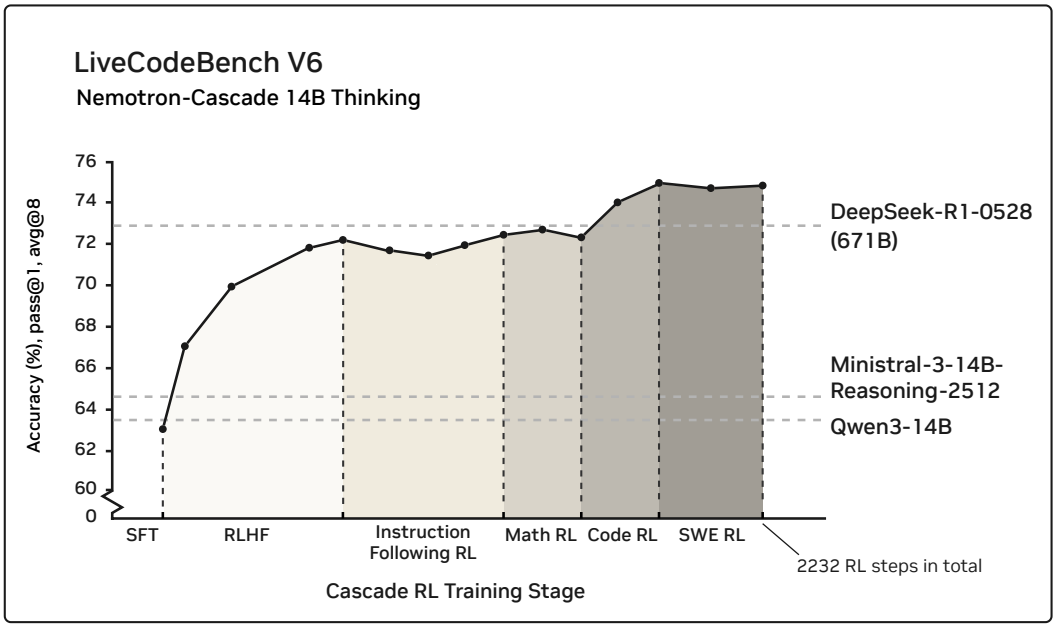


Figure 1: The LiveCodeBench v6 (08/24–05/25) performance of the Nemotron-Cascade-14B-Thinking model throughout the Cascade RL process. Note that DeepSeek-R1-0528 (671B) serves as the teacher model for SFT data curation.

1. Introduction

Reinforcement learning (RL) serves as a cornerstone for developing general-purpose LLMs with advanced reasoning capabilities by substantially improving alignment from human feedback (RLHF) (Ouyang et al., 2022) and enhancing reasoning performance through verifiable rewards (RLVR) (Guo et al., 2025). However, training general-purpose reasoning models with RL involves substantial heterogeneity across different domains, both in response length and reward signal computation. For example, fast symbolic rule-based verification is employed for mathematical reasoning tasks, slow execution-based verification is applied to code generation and software patching, and reward-model–based scores are computed for alignment and creative writing. This domain-specific heterogeneity complicates the RL infrastructure, slows down training, and makes training curriculum (e.g., maximum response length extension) and hyperparameter selection more challenging.

In our previous study (Chen et al., 2025), we proposed performing RL across the math and code domains in a cascaded manner, which first trains on math-only prompts and then on code-only prompts. This cascaded paradigm yields several advantages: *a)* Rule-based math verification can be executed rapidly and is orders of magnitude faster than code verification, allowing the model to be updated immediately without waiting for longer verification cycles required for code prompts; *b)* math RL improves performance on both math and, surprisingly, code benchmarks; and *c)* subsequent code RL significantly enhances code benchmark performance without degrading math results. In this work, we scale up the cascaded cross-domain RL paradigm to a much broader range of domains to build general-purpose reasoning models.

Since the introduction of OpenAI o1 (OpenAI, 2024), model releases in the LLM community have generally fallen into two categories: *thinking* models that generate substantially more reasoning tokens before providing an answer (e.g., DeepSeek-R1 (Guo et al., 2025), OpenAI o3 and o4-mini (OpenAI, 2025), Kimi-K2-Thinking (Kimi-Team, 2025)), and *instruct* or *non-thinking* models that produce instant answers (e.g., DeepSeek-V3 (Liu et al., 2024), GPT-4.5 (OpenAI, 2025), Kimi-K2-Instruct (Kimi-Team et al., 2025)). Meanwhile, it would be ideal to build one *unified reasoning model* that can operate in both *non-thinking* and *thinking* modes while integrating all capabilities together in a single model. This would *i)* greatly simplify model release and production pipelines,

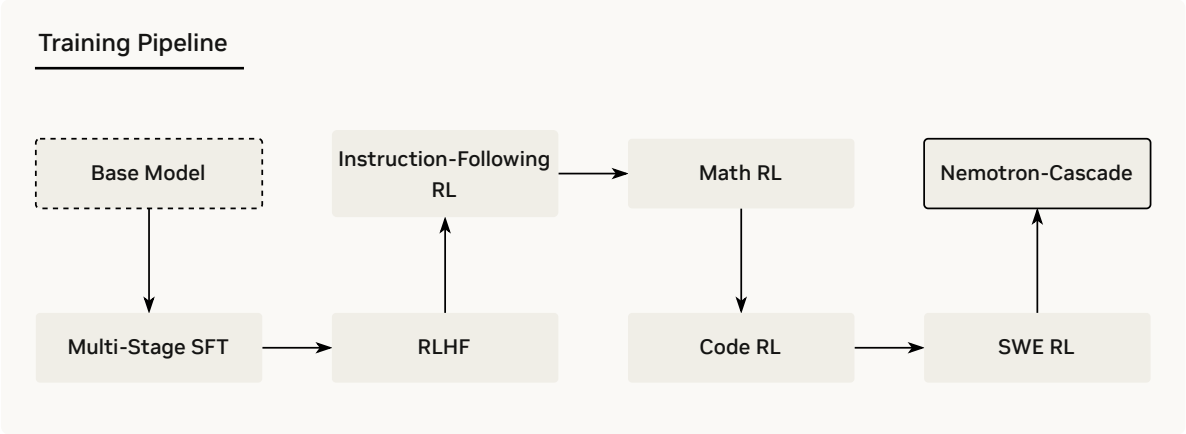


Figure 2: Cascade RL applies sequential, domain-wise reinforcement learning after SFT, leading to substantial improvements across the corresponding domains.

and *ii)* align more closely with the ultimate goal of artificial general intelligence. Consequently, substantial efforts have been devoted to developing a single *unified* model (DeepSeek-AI, 2025; OpenAI, 2025; Yang et al., 2025).

Certain technical challenges have been recognized in efforts to "smoothly integrate everything" (e.g., Altman, 2025), including degradation in reasoning benchmark performance of the *unified* model when operating in *thinking* mode, compared to a dedicated *thinking* model. For example, although the Qwen3 series (Yang et al., 2025) was initially released as a set of unified reasoning models, it was later reverted to separate *thinking* and *instruct* variants, with the dedicated *thinking* models (Qwen-Team, 2025) significantly outperforming the unified models in thinking mode. The GPT-5 release has explored routing between two specialized models, where a standard *instruct* model and a dedicated *thinking* model are used together. However, the ultimate goal remains to integrate them into a single model (OpenAI, 2025). The DeepSeek-V3.1 (DeepSeek-AI, 2025), a unified model, achieves thinking-mode performance comparable to the earlier dedicated reasoning model DeepSeek-R1-0528 on reasoning benchmarks. However, the technical details have not been disclosed, except that DeepSeek-V3.1 and DeepSeek-R1-0528 are based on different base models and were likely trained with distinct data blends.

In this work, we focus on developing an open post-training recipe, using the pretrained Qwen3-8B-Base and Qwen3-14B-Base (Yang et al., 2025) as starting points to support transparent comparison and facilitate knowledge sharing within the community. In particular, we scale up the cascaded reinforcement learning (**Cascade RL**) framework to develop Nemotron-Cascade models, setting new state-of-the-art results across multiple domains. An overview of the training pipeline is shown in Figure 2. Cascade RL trains models sequentially across domains, in contrast to approaches such as DeepSeek-R1 (Guo et al., 2025) and Qwen3 (Yang et al., 2025), which blend diverse prompt distributions from all (reasoning) domains for joint RL training. Moreover, we show that a *unified reasoning model* can operate effectively in both *thinking* and *non-thinking* modes, closing the reasoning gap with the dedicated *thinking* model while ensuring transparency via open data and training recipes.

Specifically, the contributions of our work include:

- We scale up Cascaded Reinforcement Learning (Cascade RL) across a broad spectrum of domains, including human-feedback alignment, strict instruction following, mathematical reasoning, competitive programming, and software engineering. The proposed Cascade RL framework offers notable advantages: *i)* RLHF substantially improves overall response quality (e.g., reduces verbosity), thereby enhancing reasoning performance; *ii)* subsequent domain-specific RL stages rarely degrade the benchmark

performance attained in earlier domains and may even improve it, since RL is resistant to *catastrophic forgetting* (see Figure 1 for a demonstration and the in-depth discussion in Section §4.1.1); and *iii)* RL hyperparameters and training curriculum can be tailored to each specific domain for optimal performance.

- Our 8B/14B models trained using Cascade RL method achieve state-of-the-art, best-in-class performance across a broad range of benchmarks encompassing all these domains. For example, our 14B dedicated Thinking model, with a inference budget of 64K-token, outperforms Gemini-2.5-Pro-06-05, o4-mini (medium), Qwen3-235B-A22B (thinking mode), and DeepSeek-R1-0528 (its SFT teacher) on the LiveCodeBench v5/v6 (Jain et al., 2024) (see Figure 1). It also achieves silver-medal performance on the 2025 International Olympiad in Informatics (IOI).
- We develop Nemotron-Cascade-8B *unified* reasoning model that enable user control over *thinking* and *non-thinking* modes at each conversational turn. We challenge the assumption that LLMs, especially smaller ones, lack the capacity to learn effectively from both non-thinking and thinking data, and demonstrate that the reasoning performance gap between 8B unified model in *thinking* mode and the dedicated 8B-Thinking model can be closed.
- We transparently share our training and data curation recipes, and release the full collection of models and training data at: `https://huggingface.co/collections/nvidia/nemotron-cascade`.

We organize the remainder of this report as follows. We first highlight the main results in Section § 2, and present the technical details in later sections. Section § 3 describes the supervised fine-tuning stage of our post-training recipe. In Section § 4, we present the proposed Cascade RL framework. We highlight the competitive programming results (including IOI 2025) in Section § 5. We offer an in-depth analysis of RLHF in Sections § 6 and a further exploration of software engineering (SWE) tasks in Section § 7. Related work is discussed in Section § 8.

2. Main Results

We evaluate our models and baselines on a comprehensive suite of benchmarks, including MMLU (Hendrycks et al., 2020), MMLU-Pro (Wang et al., 2024), GPQA-Diamond (Rein et al., 2024), IFEval (Zhou et al., 2023), IFBench (Pyatkin et al., 2025), ArenaHard (Li et al., 2024), LiveCodeBench v5 and v6 (Jain et al., 2024), LiveCodeBench Pro (Zheng et al., 2025), SWE-bench Verified (Jimenez et al., 2023; OpenAI, 2024), and BFCL-V3 (Patil et al., 2025). These benchmarks collectively cover general-knowledge reasoning, alignment and instruction following, mathematical reasoning, competitive programming, software engineering, and tool-use proficiency. For baseline models, we use officially reported results whenever available. For Nemotron-Cascade models, we use a maximum generation length of 64K tokens and set the temperature to 0.6 and top-p to 0.95 for reasoning tasks. The benchmarks and detailed evaluation setups are described in detail in Appendix B.

The main results are shown in Table 1. All scores are reported as pass@1, averaged over k generations (avg@k) per prompt, where k is chosen appropriately (typically between 4 and 64, depending on the test set size). Our *unified* reasoning model, Nemotron-Cascade-8B, and our dedicated *thinking* model, Nemotron-Cascade-14B-Thinking, achieve best-in-class performance across almost all benchmarks. We ↑highlight the substantial improvements achieved by applying the complete Cascade RL pipeline to our SFT models, compared with the results in Table 2.

In this work, we also perform comprehensive ablations by evaluating the models after each stage of the Cascade RL pipeline illustrated in Figure 2. The results after each stage of the pipeline are summarized as follows: Table 2 for SFT, Table 4 for RLHF, Table 5 for Instruction-Following RL, Table 6 for Math RL, Table 7 for Code RL, and Table 8 for SWE RL.

Notably, we observe substantial improvements on LiveCodeBench (LCB) and LCB Pro, with Nemotron-Cascade-8B achieving 74.3 on LCB v5 and 71.1 on LCB v6. Despite being only an 8B model, its performance is highly

Table 1: **Main results**. For *unified* reasoning models, we report reasoning-related benchmark results in thinking mode. For IFEval and IFBench, we report the higher score obtained from either *thinking* or *non-thinking* mode. **LCB** stands for LiveCodeBench. When comparing with our inital SFT models in Table 2, ↑number indicates the improvement achieved by applying the Cascade RL to initial SFT models. †Genini-2.5 uses its own scaffolds for SWE evaluation rather than Agentless setup.

Benchmark Metric: pass@1	Qwen3 8B	Nemotron-Nano 9B-v2	Qwen3 14B	DeepSeek-R1 0528 671B	Gemini-2.5 Flash-Thinking	Nemotron Cascade-8B	Nemotron-Cascade 14B-Thinking
Knowledge Reasoning							
MMLU	83.0	82.6	84.9	89.9	–	83.7 **↑0.7**	85.1 **↑0.2**
MMLU-Pro	75.1	73.3	77.6	85.0	81.9	75.7 **↑1.3**	77.0 **↑1.0**
GPQA-Diamond	62.0	64.0	64.0	81.0	82.8	66.5 **↑3.0**	69.6 **↑1.3**
Alignment							
ArenaHard	85.8	74.6	91.7	95.1	95.7	87.9 **↑17.9**	89.5 **↑12.6**
IFEval (strict prompt)	85.0	86.1	85.4	84.1	89.8	90.2 **↑19.4**	81.9 **↑12.1**
IFBench	34.4	37.4	33.7	38.0	36.1	40.8 **↑19.6**	41.7 **↑17.4**
Math							
AIME 2024 (no tools)	76.0	81.9	79.3	91.4	82.3	89.5 **↑5.9**	89.7 **↑2.8**
AIME 2025 (no tools)	67.3	72.0	70.4	87.5	72.0	80.1 **↑7.3**	83.3 **↑2.2**
Code							
LCB v5 (08/24-02/25)	61.2	68.2	65.2	74.8	63.4	74.3 **↑15.1**	**77.5** **↑11.4**
LCB v6 (08/24-05/25)	58.3	65.3	63.5	73.3	61.9	71.1 **↑14.4**	**74.6** **↑11.5**
LCB Pro 25Q2 (Easy)	46.1	59.3	53.6	63.9	47.4	65.7 **↑20.6**	**68.9** **↑11.2**
LCB Pro 25Q2 (Med)	2.2	4.8	2.6	7.0	1.8	6.4 **↑3.8**	**10.5** **↑3.5**
SWE Verified (Agentless)	20.5	–	27.4	57.6	†48.9	37.2 **↑11.1**	**43.1** **↑8.6**
–Test-Time Scaling	–	–	–	–	–	43.6	53.8
Tool Calling							
BFCL V3	68.1	66.9	70.4	67.9	68.6	64.4	67.5

comparable to DeepSeek-R1-0528 (671B), which reports 74.8 on LCB v5 and 73.3 on LCB v6. Note that DeepSeek-R1-0528 (671B) serves as the teacher model during SFT, generating all responses to code prompts used in our SFT data curation (see Section § 3.2.3). Remarkably, our Nemotron-Cascade-14B-Thinking model surpasses DeepSeek-R1-0528 by a clear margin across all splits of LCB and LCB Pro benchmarks. These results highlight the exceptional effectiveness of the proposed Cascade RL framework in enhancing reasoning capabilities. Additional results, including those for IOI 2025, are provided in Section 5.

For SWE-bench Verified, the best general-purpose open 8B and 14B LLMs perform poorly on this challenging benchmark. The specialized model, DeepSWE-32B (Luo et al., 2025), built on Qwen3-32B and specialized for SWE tasks, achieves a pass@1 accuracy of 42.2%. In comparison, our general-purpose 8B and 14B models, attain 37.2% and 43.1%, respectively. Additional details and test-time scaling results are provided in Section §7.

3. Supervised Fine-Tuning

In this section, we describe the training framework and data curation for supervised fine-tuning (SFT), the first stage of our post-training pipeline. This stage equips the model with foundational skills and capabilities, which are then substantially enhanced through cascaded reinforcement learning (Cascade RL) in the subsequent stages.

3.1. Training Framework

3.1.1. Multi-Stage SFT

Our SFT curriculum consists of two stages, spanning a broad spectrum of domains, including math, coding, science, tool use, and software engineering, as well as general domains such as multi-turn dialogue, knowledge-

Chat Template

<|im_start|>system

You are a helpful and harmless assistant.**<|im_end|>**

<|im_start|>user

Write a paragraph about San Francisco. ***/no_think*<|im_end|>**

<|im_start|>assistant

San Francisco, located on the northern coast of California, is a vibrant and iconic city known for its stunning Golden Gate Bridge, steep hills, and eclectic neighborhoods like ..... **<|im_end|>**

<|im_start|>user

What should I prepare for a summer trip to San Francisco? ***/think*<|im_end|>**

<|im_start|>assistant

<think>
Okay, the user is asking about preparing for a summer trip to San Francisco. They've already seen my description of the city, so they're likely planning their visit.
Hmm, summer in SF is famously unpredictable with microclimates and fog, so the key advice should emphasize layers and rain gear. The user might not realize how drastically temperatures can change between neighborhoods or elevations.
I should also highlight practical tips they might overlook: comfy shoes for hills, hydration due to dry air, and booking reservations since summer is peak season. .....
</think>

Here's what to prepare for a summer trip to San Francisco, focusing on essentials and maximizing your experience: ..... **<|im_end|>**

System Prompt for Tool Calling

<|im_start|>system

You are a helpful and harmless assistant.

Tools

You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools> XML tags:
<tools>
{"type": "function", "function": {"name": "FUNCTION_NAME_1", "description": "DESCRIPTION_1", "parameters": { ... } } }
{"type": "function", "function": {"name": "FUNCTION_NAME_2", "description": "DESCRIPTION_2", "parameters": { ... } } }
.....
</tools>

For each function call, return a json object with function name and arguments within <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>**<|im_end|>**

<|im_start|>user

Book a hotel in San Francisco. **/think<|im_end|>**

<|im_start|>assistant

.....

Figure 3: (Left) The chat template employs the */think* and */no_think* tags in the user prompt to control whether the model operates in the *thinking* or *non-thinking* generation mode. (Right) For tool calling, the available tools are listed in the system prompt. The model is instructed to call tools within the *<tool_call>* and *</tool_call>* tags.

intensive question answering, creative writing, role-playing, safety, and instruction following. Details of data curation for these domains are provided in § 3.2. The SFT curriculum is outlined as follows:

- **Stage 1 (16K).** This stage includes general-domain data as well as math, science, and code reasoning data, with a maximum sequence length of 16K tokens. For the general-domain data, each prompt contains parallel responses in both *thinking* and *non-thinking* modes, whereas the math, science, and code data include only *thinking* mode responses. Training is conducted for one epoch.
- **Stage 2 (32K).** This stage further enhances the model's reasoning capabilities with longer responses of up to 32K tokens and equips it with tool use and software engineering skills. To achieve this, we recombine the general-domain data with new Stage-2 math, science, and code reasoning data (up to 32K tokens), along with the tool use and software engineering datasets. Except for the general-domain data, all other domains contain only *thinking*-mode responses. Training is conducted for one epoch.

The SFT training hyperparamters can be found in Appendix D.

3.1.2. Chat Template

We define the model's interaction schema, which is particularly important for our *unified* reasoning model supporting both *thinking* and *non-thinking* generation modes. We adopt the standard ChatML template (OpenAI) and introduce two control flags in the user prompt, */think* and */no_think*, which explicitly instruct the model to generate responses in the corresponding mode.

Although prior work (Bakouch et al., 2025; Yang et al., 2025) adopts similar control flag mechanisms, we introduce several simplifications and enhancements to enable more precise and flexible control over the model's generation behavior. In contrast to Bakouch et al. (2025), which places the */think* and */no_think* flags in the system prompt and therefore controls the entire conversation globally, we instead append the flag to each individual user prompt. This design supports both global and local control: appending the same flag to every user turn enforces consistent global behavior, while varying the flags across multi-turns enables dynamic switching within a single conversation.

In contrast to the Qwen3 reasoning models (Yang et al., 2025), our approach further simplifies mode control. Qwen3 employs a redundant mechanism that enables mode switching in two ways: either through explicit flags or by modifying the template via the *enable_thinking* argument, which implicitly determines the mode (defaulting to the thinking mode, while prepending an empty *<think></think>* block activates the non-thinking mode). Our early experiments show that explicit flags result in more reliable mode transitions than template-based cues. Moreover, the flag-only design covers all use cases without any performance degradation. Consequently, we adopt the flag-based approach exclusively. With this simplification, the empty *<think></think>* block is omitted in the non-thinking mode, as it is no longer needed.

For tool calling task, we specify all available tools in the system prompt within the *<tools>* and *</tools>* tags as illustrated on the right side of Figure 3. We further instruct the model to perform tool calls enclosed within the *<tool_call>* and *</tool_call>* tags.

3.2. SFT Data Curation

3.2.1. General-Domain Data

We curate a comprehensive corpus of 2.8M samples, comprising 3.2B tokens from diverse general-domain datasets, to equip models with foundational skills and robust conversational abilities. This corpus encompasses a wide range of tasks, including daily dialogue, question answering (Lian et al., 2023; Yuan et al., 2024), creative writing (Allal et al., 2025; Xu et al., 2024), safety (Bercovich et al., 2025), instruction following (Lambert et al., 2024), role-playing (Lambert et al., 2024), and others. In addition, for knowledge-intensive tasks (Gema et al., 2024; Hendrycks et al., 2021; Wang et al., 2024) spanning general domains, we collect questions from publicly available datasets (e.g., Khot et al., 2020; Longpre et al., 2023) and further augment them with domain-specific questions from challenging areas such as professional law and ethics, resulting in 1.2M samples comprising 1.5B tokens.

However, directly combining these corpora poses three notable challenges. First, many responses are excessively brief (e.g., single-word or minimal-sentence outputs) and therefore lack sufficient detail and elaboration. Second, response quality varies widely, with some datasets containing inaccurate or suboptimal answers. Third, due to the diverse origins and labeling conventions of these datasets, directly training on them leads to stylistic inconsistencies in model generation. To address these issues, for each prompt, we generate parallel responses using DeepSeek-R1-0528 (DeepSeek-AI, 2025) and DeepSeek-V3-0324 (DeepSeek-AI, 2025) to obtain *thinking* and *non-thinking* format data, respectively, with a maximum sequence length of 16K, thereby ensuring stylistic and qualitative consistency.

To further enhance the training data quality, we apply several post-processing steps. For samples with high-quality annotations, we retain their original responses to preserve diversity. Furthermore, for prompts with verifiable ground-truth answers (e.g., multiple-choice questions), we enhance generation accuracy by discarding responses that deviate from the ground-truth. For samples without ground-truth answers, we cross-validate the generated responses using an auxiliary model (Qwen2.5-32B-Instruct (Yang et al., 2024)) to filter out potentially low-quality generations.

To address data scarcity in domains such as instruction following and creative writing, we generate multiple responses for each prompt using different random seeds, thereby enriching diversity and improving generation quality. To further enhance multi-turn conversational capabilities, we manually augment multi-turn samples in two ways. First, for single-turn samples in the creative writing domain, we add a second turn that instructs the model to rewrite or edit its previous response under specific requirements. Second, we randomly concatenate single-turn samples to construct multi-turn conversations, emulating real-world chatbot interactions.

3.2.2. Math Reasoning Data

We utilize the math reasoning SFT prompts from AceReason-Nemotron-1.1 (Liu et al., 2025) for our Stage-1 SFT training. These prompts encompass a diverse set of data sources, including AceMath (Liu et al., 2024),

NuminaMath (Li et al., 2024), and OpenMathReasoning (Moshkov et al., 2025). The corresponding responses are generated by DeepSeek-R1 (Guo et al., 2025). We set the maximum context length to 16,384 tokens (16K) and filter out samples exceeding this limit to prevent response truncation, following the SFT configuration in AceReason-Nemotron-1.1. In total, we gather 353K unique prompts and generate multiple responses for each, resulting in 2.77M samples with an average of 7.8 responses per prompt. We perform data decontamination by removing any samples that share a 9-gram overlap with test samples from standard math benchmarks.

To further enhance the model's reasoning capability, we employ DeepSeek-R1-0528 (DeepSeek-AI, 2025) to generate responses and construct the Stage-2 math SFT dataset. Compared to the original DeepSeek-R1, the updated DeepSeek-R1-0528 produces longer and more detailed reasoning trajectories, leading to improved performance on challenging problems. We set the maximum context length to 32,768 tokens (32K) to provide a larger reasoning token budget for the model. The Stage-2 prompt set is derived from Stage-1 by filtering out relatively easy questions, specifically those whose DeepSeek-R1 responses contain fewer than 2K tokens. In total, we obtain 163K prompts and generate 1.88M samples, with an average of 11.5 responses per prompt. All math reasoning data in both Stage-1 and Stage-2 SFT are formatted in the *thinking* mode.

3.2.3. Code Reasoning Data

Following a similar procedure as the math reasoning data construction, we adopt the code reasoning SFT prompts from AceReason-Nemotron-1.1 (Liu et al., 2025), which include data from TACO (Li et al., 2023), APPs (Hendrycks et al., 2021), OpenCoder-Stage-2 (Huang et al., 2024), and OpenCodeReasoning (Ahmad et al., 2025). We perform dataset deduplication to ensure all prompts are unique, resulting in 172K distinct prompts. Using DeepSeek-R1 (Guo et al., 2025), we generate 1.42M samples for Stage-1 SFT, with an average of 8.3 responses per prompt and a maximum context length of 16,384 tokens (16K). For data decontamination, we filter out any samples that have a 9-gram overlap with any test sample from coding benchmarks.

To construct the Stage-2 code SFT dataset, we leverage prompts from OpenCodeReasoning (Ahmad et al., 2025) and OpenCoder-Stage2 (Huang et al., 2024). OpenCodeReasoning provides a diverse set of challenging coding prompts, while OpenCoder-Stage2 covers coding tasks with starter code entry points. Similar to the Stage-2 math SFT dataset, we set the maximum context length to 32,768 tokens (32K). In total, we build 79K unique prompts and use DeepSeek-R1-0528 to generate 1.39M samples, averaging 17.6 responses per prompt. All code reasoning data in both Stage-1 and Stage-2 SFT are formatted in the *thinking* mode.

3.2.4. Science Reasoning Data

We curate science-related prompts from S1K (Muennighoff et al., 2025) and the post training datasets used in Llama-Nemotron (Bercovich et al., 2025; Nathawani et al., 2025). Given that many prompts in these sources are multiple-choice, we exclude samples where models focus on analyzing each option rather than directly solving the problem and determining the correct answer. As a result, we retain questions that require strong scientific knowledge and involve substantial reasoning or complex calculations.

To enrich the dataset with less common and more diverse question types, we leverage DeepSeek-R1-0528 (DeepSeek-AI, 2025) to generate rarer questions from each given prompt, following the synthetic question generation strategy used in Liu et al. (2024). Finally, we perform data decontamination and remove any samples that have a 9-gram overlap with any test sample from science benchmark. In total, we collect 226K science prompts, generating 289K samples (up to 16K tokens) for Stage-1 SFT with DeepSeek-R1, and 345K samples (up to 32K tokens) for Stage-2 SFT with DeepSeek-R1-0528. All science reasoning data are formatted in the *thinking* mode, and multiple responses are generated for selected high-quality prompts. The Stage-2 science reasoning data are upsampled by 2× before being blended into the Stage-2 SFT dataset.

3.2.5. Tool Calling Data

We utilize the tool calling dataset from Llama-Nemotron (Nathawani et al., 2025), which is specifically designed to train models for scenarios involving external tool usage, such as function calls. The dataset is comprehensive,

encompassing single-turn, multi-turn, and multi-step interactions. For instance, some prompts require the model to ask clarification questions to gather sufficient information for a tool call, while other prompts may involve using several tools or even performing multiple rounds of tool calls before reaching a final answer. It also includes cases where the model cannot find a suitable tool in the provided tool list. For each conversation, all available tools are included in the system prompt, following the setup used in Qwen3 (Yang et al., 2025). On average, each conversation includes 4.4 available tools. This tool calling SFT dataset is used in Stage-2 SFT training, with responses generated by Qwen3-235B-A22B (Yang et al., 2025). Note that all tool-calling data are formatted in the *thinking* mode. Overall, we collect 310K conversations comprising 1.41M user-assistant turns.

3.3. Software Engineering Task

Software engineering has become one of the most important applications of LLMs (e.g., Anthropic, 2025). For example, SWE-bench Verified (Jimenez et al., 2023) has emerged as a widely used benchmark for software engineering, consisting of real-world GitHub issues paired with their corresponding codebases and descriptions, where the goal is to generate repair patches that successfully resolve the issued problems.

3.3.1. Agentless Framework

To evaluate automated software engineering capabilities, we adopt Agentless (Xia et al., 2024), which decomposes the overall task into three stages (i.e., **localization**, **repair** and **patch validation**) without requiring the LLM itself to plan action sequences or operate external tools.

In this work, we employ a simplified Agentless framework similar to Agentless Mini (Wei et al., 2025), which streamlines the localization process to focus solely on identifying relevant issue files. This approach differs from the original Agentless workflow, which adopts a three-stage hierarchical localization strategy progressing from file-level to class/function-level and finally to line-level identification before proceeding to the repair and patch validation stages. By simplifying the localization process, the LLM can dedicate more reasoning capacity to the repair task itself, while reinforcement learning is consolidated into a focused objective that directly optimizes repair patch generation.

For the code repair stage, the primary objective is to generate effective patch candidates that resolve the identified repository-level issue. Once the relevant files have been localized through earlier stages, the LLM is prompted to generate the repair edits that modify only the necessary portions of the codebase. To maximize contextual understanding, we concatenate multiple localized files and their surrounding code snippets (e.g., imports, class definitions, and dependent functions) into a unified prompt, enabling the model to reason over broader repository-level dependencies. Instead of rewriting entire repair files, the model is guided to produce targeted, diff-style patches that preserve unrelated code structure to reduce hallucinations and syntactic errors.

For the patch validation stage, our framework operates through three phases: regression, reproduction and majority voting. In the regression phase, each candidate patch is initially evaluated through the repository's existing regression test to ensure the compatibility. This step filters out the patches that introduce the failures or disrupt previously correct functionality to ensure that subsequent evaluation focuses exclusively on stable and syntactically valid candidates. In the reproduction phase, the framework generates 10 reproduction tests per issue instance to replicate the original bug behavior on the unmodified repository and to verify functional correctness after patch application. Reproduction tests that fail to trigger the original bug are discarded to maintain high diagnostic precision and the survived tests are then executed on each candidate patch, allowing the system to identify which patches successfully eliminate the reported issues. Finally, in the majority voting phase, we aggregate results across multiple sampled generations to select the most reliable patch. Among the highest-scoring candidates, we first prioritize the most frequently generated patch across test-time samples, reflecting consensus in the model's repair reasoning capabilities. In the event of ties, we favor the patch with a shorter generation sequence or minimal edit distance, promoting conciseness and interpretability. This multi-phase validation framework ensures that the final selected patch is both functionally correct and robust

to repository-level dependencies, balancing precision with efficiency.

We refer readers to Appendix C.2 for the prompts used at each stage. We detail the improved techniques used in this work, including enhanced code file localization and patch validation, compared to prior studies in § 7.

3.3.2. Data Curation

Data source:
Our training data for the software engineering task consist of the following open-source datasets:

- SWE-Bench-Train (Jimenez et al., 2023), the training split generated through the same pipeline as the SWE-Bench evaluation set but without human verification.
- SWE-Fixer-Train (Xie et al., 2025), which consists of Python repositories with more than 100 pull requests, yielding 115K instances after applying heuristic filtering rules.
- SWE-reBench (Badertdinov et al., 2025), a public dataset containing over 21K interactive Python-based SWE tasks, constructed through a novel, automated, and scalable pipeline.
- SWE-Smith (Yang et al., 2025), a synthetic dataset comprising 50K instances from 128 GitHub repositories, generated by automatically injecting bugs into codebases.

To prevent evaluation data contamination, we implement a comprehensive deduplication process against SWE-bench Verified (Jimenez et al., 2023). Specifically, we exclude all instances originating from repositories present in the evaluation dataset. Additionally, we perform deduplication across training data from different sources to eliminate duplicate instances. This deduplication process relies on matching both repository names and base commit identifiers to ensure that identical instances are removed.

Response generation:
We construct SFT datasets for three sub-tasks in Agentless framework:

1. Code localization: Given a problem statement and the corresponding GitHub repository structure, the model identifies and lists the code files that are likely to contain bugs.
2. Code repair: Given a problem statement and the contents of one or more buggy code files, the model generates revised code patches that address the issues described in the problem statement.
3. Test code generation: Given a problem statement, code localization and repair patches, the model generates test code that validates the generated code patches.

To construct the SFT datasets, we employ DeepSeek-R1-0528 (DeepSeek-AI, 2025) to generate multiple responses across the four datasets listed in Data Source. We generate 8 responses per prompt for SWE-Bench-Train, SWE-reBench, and SWE-Smith, while producing 4 responses per prompt for the larger-scale SWE-Fixer-Train dataset. The input prompts are structured to include task specifications, problem statements, content of code files for repair tasks, and desired output formats. See Appendix C.2 for details of the template. The model is instructed to output the comprehensive reasoning chain and the solution. For code localization, the solution contains a prioritized list of potential buggy code file names, ranked from most to least likely to contain bugs. For code repair, the solution contains the code blocks to be replaced, and the code patch used to replace. For test code generation, the solution contains a set of unit tests and reproduction tests designed to verify both bug replication and patch correctness.

Data filtering and splitting for SFT and RL:
To ensure high-quality training data, we validate all generated responses against ground truth annotations. Our filtering strategy varies across the three tasks. For the localization task, we retain only those samples that include all buggy code files required to address the issue (i.e., recall equals 1.0). For repair tasks, we measure the similarity between the generated and ground-truth patches using *Unidiff*, a lightweight Python library for parsing and interacting with unified *diff* data, following the approach of Wei et al. (2025). We adopt a stratified approach: instances demonstrating consistent solution quality—defined as at least 4 out of 8 sampled responses

Table 2: The evaluation results of 8B/14B-Thinking and the *unified* 8B models **after SFT** are presented below. For unified 8B model, we evaluate IFEval in the *non-thinking* mode and all other benchmarks in the *thinking* mode.

Benchmark Metric: pass@1	8B-Thinking SFT	8B (unified) SFT	14B-Thinking SFT
Knowledge and Reasoning			
MMLU (EM)	83.6	83.0	84.9
MMLU-Pro (EM)	74.5	74.4	76.0
GPQA Diamond (avg@8)	64.2	63.5	68.3
Alignment			
ArenaHard (GPT4-turbo-2024-04-09)	71.7	70.0	76.9
IFEval (strict prompt) (avg@8)	66.3	70.8	69.8
IFBench (avg@8)	23.2	21.2	24.3
Math			
AIME 2024 (avg@64)	83.8	83.6	86.9
AIME 2025 (avg@64)	71.6	72.8	81.1
Code			
LiveCodeBench v5 (08/24-02/25) (avg@8)	59.6	59.2	66.1
LiveCodeBench v6 (08/24-05/25) (avg@8)	56.7	56.7	63.1
LiveCodeBench Pro 25Q2 (Easy) (avg@8)	48.5	45.1	57.7
LiveCodeBench Pro 25Q2 (Med) (avg@8)	3.1	2.6	7.0
SWE-bench Verified (avg@4)	30.2	26.1	34.5

exceeding the 0.5 similarity threshold for SWE-Bench-Train, SWE-reBench, and SWE-Smith—are included in the SFT dataset. More challenging instances—defined as those with at least 1 out of 8 sampled responses achieving non-zero similarity, excluding SFT samples—are reserved for the SWE RL training described in § 4.7. Given the scale and diversity of SWE-Fixer-Train dataset, we include all prompt-response pairs surpassing the 0.5 similarity threshold. For test code generation, we retain only the trajectories that can be successfully parsed and executed as reproduction tests without any syntax error.

Dataset composition summary:
The resulting Code Repair dataset comprises 127K instances distributed as follows: 17K from SWE-Bench-Train, 17K from SWE-reBench, 18K from SWE-Smith, and 77K from SWE-Fixer-Train. This composition ensures comprehensive coverage of diverse coding scenarios while maintaining high data quality standards. The resulting datasets for localization and test case generation consist of 92K and 31K samples, respectively. All SWE datasets are upsampled by 3× before being incorporated into the Stage-2 SFT data blend.

3.4. Results after SFT

After the multi-stage SFT process, the results for the 8B *unified* model and the 8B/14B *thinking* models are summarized in Table 2. We find that 8B unified model performs on par with dedicated 8B Thinking model across all reasoning-related benchmarks, while surpassing it on the IFEval benchmark—a task more naturally suited to the *instruct* mode. It is worth noting that both models are trained on the same SFT *thinking* data; however, the unified model further incorporates *non-thinking* data. Due to resource constraints, we only train a 14B-Thinking model and provide stronger results than 8B models. Next, we examine the results across all RL stages, highlighting the robustness and overall effectiveness of the Cascade RL framework.

4. Cascade RL

In this section, we describe the proposed Cascaded Reinforcement Learning (Cascade RL) method. In curating the RL data, we ensure that the **SFT and RL datasets are strictly disjoint in terms of prompts**, so the model cannot leverage memorized answers for given prompts from SFT during RL training.

4.1. Training Framework

As illustrated in Figure 2, the Cascade RL process begins with applying general-domain Reinforcement Learning from Human Feedback (RLHF) to the SFT models described in § 4.3, followed by domain-wise Reinforcement Learning with Verifiable Rewards (RLVR). We first apply RLHF and then RLVR (e.g., Math RL), as RLHF substantially improves the quality of generated responses by reducing *verbosity* and *repetition*, thereby enhancing the reasoning performance within constrained response lengths (e.g., 64K tokens). In Cascade RL, we sequentially apply RLHF (§ 4.3), Instruction-Following RL (§ 4.4), Math RL (§ 4.5), Code RL (§ 4.6), and finally SWE RL (§ 4.7), progressively from more general domains towards more specialized ones.

4.1.1. Why Cascade RL for LLMs Is Resistant to Catastrophic Forgetting

Catastrophic forgetting occurs when a model trained sequentially on multiple domains overwrites previously learned knowledge while acquiring new ones, a common issue in supervised learning, where disjoint training datasets cause updates to push the model toward the new distribution.

Cascaded Cross-Domain RL for LLMs differs in several structural ways that mitigate this issue:

- ***i***) In RL, the training data distribution is policy-dependent; the LLM generates its own experience. When a new objective or task is introduced, the LLM still explores across states, meaning old behaviors are continuously sampled if they remain useful or high-reward. This contrasts with supervised learning (e.g., SFT), where the samples in the previous domain disappear unless explicitly replayed.
- ***ii***) RL optimizes expected cumulative reward, not exact targets for each input. As a result, the updates focus on improving long-term outcomes rather than explicitly fitting a new token-level distribution. Old knowledge that remains reward-relevant naturally persists. When new tasks share structure with old ones, updates tend to generalize rather than overwrite.
- ***iii***) *Catastrophic forgetting* may still occur when the reward of a new domain sharply conflicts with that of a previous one (e.g., optimizing for concise responses versus detailed, step-by-step reasoning), particularly when prompts from different domains are semantically similar. However, the reward structures of RLHF and RLVR overlap substantially across domains, such as math, code, reasoning, and instruction following, since they all aim to make outputs better, more accurate, and more aligned with human preferences or verification signals. For instance, reducing verbosity or hallucination generally benefits all domains.
- ***iv***) In our Cascade RL framework, we further minimize prompt overlap to the greatest extent possible, given that prompts across domains are generally already distinct. For instance, we remove all math and competitive programming–related prompts from the RLHF stage to reduce cross-domain interference. Furthermore, domain-wise RL is organized from more general domains (e.g., RLHF, instruction-following) to more specialized ones (e.g., math, code, SWE), preventing specialized capabilities from being overwritten by generic behaviors.

4.1.2. RL Training Configuration

Throughout the entire Cascade RL process, we use Group Relative Policy Optimization (GRPO) algorithm (Shao et al., 2024) with strict **on-policy** training following AceReason-Nemotron (Chen et al., 2025). We adopt on-policy training for improved stability and higher accuracy. We conduct our training using the verl repository (Sheng et al., 2025).

At each iteration, we generate a group of G rollouts from the current policy π_θ and then perform a *single* gradient update. This ensures that the policy used for data collection always matches the one being updated, making the importance sampling ratio exactly 1. This on-policy setup contributes to stable RL training and mitigates entropy collapse. In addition, we remove KL divergence term entirely, which simplifies the GRPO objective to the standard *REINFORCE* objective (Williams, 1992) with group-normalized rewards and token-level loss (Yu

et al., 2025):

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{(q,a)\sim\mathcal{D},\ \{o_i\}_{i=1}^{G}\sim\pi_\theta(\cdot|q)}\left[\frac{1}{\sum_{i=1}^{G}|o_i|}\sum_{i=1}^{G}\sum_{t=1}^{|o_i|}\hat{A}_{i,t}\right],\ \text{where}\ \hat{A}_{i,t} = \frac{r_i - \text{mean}(\{r_i\}_{i=1}^{G})}{\text{std}(\{r_i\}_{i=1}^{G})}\ \text{for all}\ t, \quad (1)$$

and $\{r_i\}_{i=1}^{G}$ denotes the group of G rewards assigned to the sampled responses $\{o\}_{i=1}^{G}$ for a given question q drawn from the dataset $\mathcal{D}$, verified against the ground-truth answer a in RLVR (e.g., Math RL). For RLHF, r_i is the scalar-output from the reward model for response o_i and question q. Details of the reward functions for different domains will be provided in the corresponding subsections.

4.2. Reward Modeling

In this subsection, we describe the construction of the reward model (RM) used in the RLHF phase presented later in § 4.3.

4.2.1. Data Curation

Our reward modeling preference dataset is a mixture of open-source and in-house data, comprising a total of 82K preference pairs. Specifically, we use the following open-source data:

- HelpSteer2 (Wang et al., 2024) — a 10K high-quality, human-annotated preference dataset with multi-aspect annotations covering helpfulness, correctness, coherence, complexity, and verbosity.
- HelpSteer3 (Wang et al., 2025) — a 40K preference dataset spanning multiple domains, including general, STEM, code, and multilingual. Each sample (pair of response) is annotated with a preference score ranging from –3 (Response 1 is much better than Response 2) to 3 (Response 2 is much better than Response 1). We filter out samples with a score of 0 (Response 1 is similar to Response 2), resulting in 36K remaining samples.

Inspired by Park et al. (2024), we generated additional data to improve our final preference data blend. The core idea is to construct preference pairs with bad responses from the stronger LLM and good responses from the weaker one. Note that, inducing the stronger LLM to generate bad responses is crucial for making the data effective; otherwise, the preference pairs would be too easy for the reward model to distinguish.

One specific approach is to generate a slightly off-topic prompt and use it to obtain a bad response from the stronger model. DeepSeek-V3 was employed to produce these off-topic prompts by rewriting the original prompt, and the quality of the rewrites was verified to be high through both manual inspection and automatic evaluation using LLM-as-a-Judge. For detailed prompts, please refer to Appendix §C.1. We explored different combinations of LLMs as the weaker and stronger models and ultimately selected DeepSeek-V3-0324 and DeepSeek-V3 as the stronger and weaker models, respectively. We also tried explicitly instructing strong LLMs to produce subtly erroneous answers to the given prompts; however, this approach was unsuccessful.

4.2.2. Training Recipe

We train a scalar-output reward model (RM) on pairwise human preference data using the Bradley–Terry objective (Bradley and Terry, 1952):

$$P_{\text{BT}}(y^+ \succ y^- \mid x) = \frac{\exp(r_\theta(x, y^+))}{\exp(r_\theta(x, y^+)) + \exp(r_\theta(x, y^-))}$$

where y^+ is preferred and y^- is dispreferred. Our reward model is initialized with Qwen2.5-72B-Instruct (Yang et al., 2024) with a linear predictor on top of its last hidden layer, and is trained by maximizing the log-likelihood of human preferences:

$$\mathcal{L}(\theta) = \mathbb{E}_{(x,y^+,y^-)\sim\mathcal{D}}\left[\log P_{\text{BT}}(y^+ \succ y^- \mid x)\right]$$

For each prompt, two responses—one preferred and one dispreferred—are compared in a contrastive manner, enabling the RM to learn to assign higher scalar scores to preferred responses and lower scores to dispreferred ones. We treat this scalar score as a proxy metric for the "quality" of the model's response. The training hyperparameters are as follows: batch size 256, learning rate 2e-6, AdamW optimizer (Loshchilov and Hutter, 2017), and 1 epoch. Note that we also experimented with longer training schedules, but found that a single epoch yielded the best results.

RM evaluation:
In this study, we primarily used RewardBench (Lambert et al., 2024) to evaluate and select our reward model (RM) for the RLHF process. In practice, we found that reward models with low RewardBench scores typically lead to poor policy alignment after RLHF. However, the RM with the highest RewardBench score does not necessarily yield the best aligned policy model (i.e., as measured by alignment benchmark), since RewardBench can be an imperfect proxy for identifying the optimal reward model for RLHF, and the RLHF process itself introduces additional variance. There are ongoing research efforts to establish robust RM benchmarks that can serve as more reliable proxy metrics for identifying reward models likely to produce the best final alignment. However, there is still no general consensus on a standard benchmark to use.

Ablation studies on RM:
We conducted ablation studies to determine the choice of backbone LLM, and our key findings are as follows:

- *Model Size*: We trained reward models of various sizes using the Qwen2.5-Instruct series (7B, 14B, 32B, and 72B) and observed that performance scales positively with model size, confirming that the scaling law (Kaplan et al., 2020) holds in the context of reward model training as well (see *(a)-(d)* in Table 3). Larger LLMs demonstrated greater robustness to stylistic artifacts in the preference data, whereas smaller models tended to focus more on the style of a response rather than its overall quality. This observation was further validated by the Arena-Hard scores (Li et al., 2024) when applying the reward models in RLHF, both with and without style control: smaller models exhibited a larger performance drop under style-controlled evaluation compared to their larger, as futher discussed in § 6.2.
- *With vs. Without Large-scale Preference Pre-training*: Wang et al. (2025) released the WorldPM checkpoint, a Qwen2.5-72B model further pre-trained on 15M-scale diverse preference data, which can serve as a strong initialization for reward model training. In our experiments, models initialized from WorldPM perform better during the early stages of training; however, with extended training, models initialized from the vanilla Qwen2.5-72B-Instruct eventually catch up and slightly outperform them on Reward-Bench (Lambert et al., 2024) - see (d) vs (e) in Table 3.
- *Reasoning vs. Instruct Models*: During the development of our post-training pipeline, the Qwen3 unified reasoning models were released (Yang et al., 2025), prompting us to explore the Qwen3 8B and 14B checkpoints as well. However, we empirically found that the Qwen3 reasoning models consistently underperformed their Qwen2.5 *instruct/non-thinking* counterparts of the same size when used as the backbone for reward models trained with BT loss (see (b) vs (f) in Table 3). We suspected this was due to our preference data being less suitable for Qwen3 models operating in *thinking* mode, so we conducted additional experiments with the *non-thinking* mode enabled. Although performance improved considerably, the Qwen3 models in *non-thinking* mode still failed to surpass Qwen2.5-Instruct—their dedicated *non-thinking* counterparts (refer to (b) vs (g) in Table 3). We hypothesize this is because the Qwen3 reasoning models are primarily optimized for reasoning-centric tasks (e.g., math and code) rather than general human preference alignment.

4.3. Reinforcement Learning from Human Feedback (RLHF)

In this subsection, we describe our RLHF recipe, which serves as the first stage of the Cascade RL process. We find that the generalization capability of the reward model plays a crucial role in ensuring stable RLHF training, and that larger reward models (e.g., 72B RM) are more resilient to out-of-distribution (OOD) samples

Table 3: RewardBench (Lambert et al., 2024) performance showing ablation of RM's LLM backbone choice.

	LLM Backbone	Overall	Chat	Chat Hard	Safety	Reasoning
(a)	Qwen2.5-Instruct-7B	91.98	96.65	85.09	90.54	95.64
(b)	Qwen2.5-Instruct-14B	93.22	96.93	87.72	91.62	96.63
(c)	Qwen2.5-Instruct-32B	93.56	96.37	89.04	91.76	97.09
(d)	Qwen2.5-Instruct-72B	**95.15**	**98.60**	**89.69**	**93.92**	**98.40**
(e)	WorldPM-72B	94.08	97.77	86.40	93.78	98.36
(f)	Qwen3-14B (thinking)	46.38	29.05	46.27	60.95	49.24
(g)	Qwen3-14B (non-thinking)	91.72	94.69	87.94	89.73	94.52

generated by the policy LLM.

4.3.1. Data Curation

We find that introducing OOD prompts of the reward model to the RLHF stage often leads to instability or even training collapse due to inaccurate or misleading reward signals. Therefore, during the RLHF stage, we use a subset of prompts from the reward model's preference dataset described in § 4.2. Additionally, we exclude prompts related to mathematics and competitive programming, since the reward model may not provide reward signals as reliable as those produced by rule-based or execution-based verifiers used in subsequent Math RL and Code RL stages. For example, in our early experiments, we observed that failing to exclude math-related prompts during RLHF resulted in a 2% performance drop on the AIME25 benchmark. As a result, our RLHF dataset primarily focuses on improving helpfulness, harmlessness, and alignment with human preferences, while remaining disjoint from the domains that will be enhanced in the subsequent Cascade RL stages.

4.3.2. Training Recipe

RLHF helps LLMs better follow user intentions and align with human preferences. We also observe that RLHF improves overall generation quality and, interestingly, enhances reasoning performance on math and code benchmarks, despite our curated RLHF datasets containing no math or code-related prompts. Moreover, RLHF tends to reduce repetition and verbosity, thereby compressing the number of thinking tokens for simpler questions. This, in turn, enhances reasoning efficiency and training stability in the subsequent Math RL and Code RL stages. Therefore, we design our cascade RL pipeline to begin with the RLHF stage. Our RLHF training initializes from the SFT checkpoint, employs the GRPO algorithm, and follows the unified RL training configuration in § 4.1.2 (e.g., on-policy, token-level loss, no KL divergence). For dedicated *thinking* models, we naturally perform RLHF in *thinking* mode. For *unified* models, we perform RLHF training in both *non-thinking* and *thinking* modes, with an equal split of prompts allocated to each mode within every batch. We provide further studies in § 6.1.

Reward function:

RLHF training uses the reward scores produced by the reward model as the reward function. Specifically, we extract the model's answer, concatenate it with the corresponding question, apply the reward model's chat template, and feed the formatted input into the reward model to obtain a point-wise reward score. We handle answer extraction differently for LLMs operating in the *non-thinking* mode and *thinking* mode: for the *non-thinking* mode, we directly extract the model answer following the assisant role; for the *thinking* mode, we exclude the reasoning traces and only extract the final summary after the thinking process (*i.e.*, model generation after the `<\think>` token). If the thinking process does not terminate properly (*i.e.*, the `<\think>` token is missing), we fall back to sending the entire unfinished response to the reward model. Such incomplete generations typically receive low reward scores because the reward model was not trained on unfinished or unseen reasoning traces, effectively penalizing verbose or incomplete thinking processes. During training, we use a maximum response length of 12K for both the 8B and 14B models in RLHF, without applying overlong filtering, which encourages more succinct generations.

Table 4: The evaluation results of our 8B/14B-Thinking and the *unified* 8B models **after RLHF** are presented below. For unified model, we evaluate IFEval in the non-thinking mode and all other benchmarks in the thinking mode. We compare the results against those obtained after SFT in Table 2, using ↑ to denote improvements and ↓ to mark degradations after RLHF training.

Benchmark **Metric: pass@1**	**8B-Thinking** **RLHF**	**8B (*unified*)** **RLHF**	**14B-Thinking** **RLHF**
Knowledge and Reasoning			
MMLU (EM)	83.9 ↑0.3	83.1 ↑0.1	85.2 ↑0.3
MMLU-Pro (EM)	76.2 ↑1.7	77.8 ↑3.4	77.7 ↑1.7
GPQA Diamond (avg@8)	67.3 ↑3.0	66.8 ↑3.3	71.3 ↑3.0
Alignment			
ArenaHard (GPT4-turbo-2024-04-09)	89.9 ↑18.2	90.1 ↑20.1	93.1 ↑19.2
IFEval (strict prompt) (avg@8)	45.5 ↓20.8	50.1 ↓20.7	56.2 ↓12.4
IFBench (avg@8)	23.9 ↑0.7	24.5 ↑3.3	25.6 ↑1.9
Math			
AIME 2024 (avg@64)	86.4 ↑2.6	86.1 ↑2.5	88.4 ↑1.5
AIME 2025 (avg@64)	75.1 ↑3.5	75.0 ↑2.2	81.8 ↑0.7
Code			
LiveCodeBench v5 (08/24-02/25) (avg@8)	70.3 ↑10.7	70.2 ↑11.0	75.2 ↑9.1
LiveCodeBench v6 (08/24-05/25) (avg@8)	67.3 ↑10.6	67.2 ↑10.5	72.3 ↑9.2
SWE-bench Verified (avg@4)	33.3 ↑3.1	28.2 ↑2.1	38.8 ↑4.3

To prevent language mixing in the generation, we apply an additional *code-switch penalty* when the prompt is purely in English but the generated response, including both thinking traces and summary, contains non-English tokens. As the reward scores from the reward model are unbounded,we adaptively assign the reward for mixed-language generations to be the lowest score in the batch minus 10, ensuring they receive the lowest relative score under the GRPO algorithm and thus the strongest penalty for code-switching behavior. We do not apply additional reward shaping techniques, as the reward signals from our 72B reward model are already of high quality.

Hyperparameters:
For both our 8B and 14B models, we use a maximum response length of 12K during RLHF without applying overlong filtering, which effectively encourages more succinct generations. We use a batch size of 128, generating 8 rollout per prompt with a temperature of 0.6 and a top-p value of 0.95. We adopt a learning rate of 2e-6 with AdamW (Kingma, 2014), and set both the entropy loss coefficient and KL loss coefficient to 0. The training takes around 800 steps. More details of training hyperparamters can be found in Appendix D.

4.3.3. Results after RLHF

After RLHF, the results of our 8B and 14B models are shown in Table 4. One can observe significant improvements on nearly every benchmark, except for IFEval. The main reason is that our RLHF process substantially improves response quality by penalizing overly long, verbose, and repetitive generations, especially for *thinking* mode. For the degraded IFEval performance, the primary cause is the unavoidable semantic overlap between the prompts used during RLHF training and the test prompts in IFEval. Additionally, the reward model used in RLHF encourages human-preferred response qualities that can conflict with the strict instruction-following constraints evaluated by the verifier. We believe this issue can be mitigated by training a stronger reward model (e.g., a large generative RM (Wang et al., 2025)) capable of handling strict instruction-following constraints as well. We leave this for future work. We will revisit this in the next subsection.

4.4. Instruction-Following Reinforcement Learning (IF-RL)

Verifiable instruction following is a crucial aspect of ensuring that LLMs can follow human instructions precisely. Although our SFT data blend already contains instruction-following data, applying IF-RL with verifiable rewards further enhances the accuracy of instruction adherence.

4.4.1. Data Curation

We use the instruction-following dataset from Llama-Nemotron (NVIDIA, 2025), which consists of synthetically generated prompts containing one to ten detailed instruction constraints derived from the IFEval taxonomy (Zhou et al., 2023). However, we found this dataset to be noisy due to its synthetic nature. To improve overall quality, we performed extensive preprocessing and filtering, reducing 56K samples to 40K high-quality ones. We additionally curate 60K custom data samples to enhance the diversity of our data blend, using user prompts from LMSYS-Chat-1M (Zheng et al., 2023) paired with various instruction constraints from the IFEval taxonomy. We also incorporate the IF-RLVR training data from Pyatkin et al. (2025), which is designed to enhance robustness to unseen constraint taxonomies. This dataset comprises prompts paired with instruction constraints drawn from either the IFEval taxonomy or the IF-Bench-Train taxonomy (Pyatkin et al., 2025), with base prompts sampled from Tulu-3-SFT (Lambert et al., 2025).

4.4.2. Training Recipe

The IF-RL training proceeds in two stages, each utilizing a distinct data blend with progressively increasing difficulty. The first stage focuses on instruction constraints from IFEval taxonomy, while the second stage focuses on IF-Bench-Train taxonomy. We find the dynamic filtering (Yu et al., 2025) largely stabilize the IF-RL training and improve the results at both stages by ensuring all prompts in the batch with effective gradients.

One of the major challenges in the IF-RL stage was mitigating the negative impact that IF-RL could have on the human alignment capabilities acquired during the RLHF stage (e.g., measured by ArenaHard). In our early experiments, we observed that naively using a rule-based IF verifier as the reward function degraded human alignment results. This occurs because the rule-based IF verifier focuses solely on whether the response adheres to the constraints specified by the instruction, without considering the overall response quality. For example, a poorly written answer to the prompt "write a summary within 300 words" could still receive a full reward as long as its word count remains below 300.

***Unified* models: IF-RL in the *non-thinking* mode**

An effective strategy emerges for the *unified* reasoning model: we first perform RLHF in both *thinking* and *non-thinking* modes, and then apply IF-RL only in the *non-thinking* mode. This approach minimizes negative mutual interference between RLHF and IF-RL, while still yielding substantial improvements in the model's instruction-following capability in the *thinking* mode (i.e., our 8B unified model achieves IFEval 85.3 in thinking mode). We hypothesize that applying IF-RL to an RLHF-trained model in the *non-thinking* mode is far less likely to generate low-quality responses than applying it in the *thinking* mode, and therefore is much less prone to reward-hacking the rule-based IF verifier. We also experimented with reversing the order of RLHF and IF-RL, but observed much worse results. At both first-stage and second-stage IF-RL training, we set the maximum response length to 8K tokens and do not apply overlong filtering for unified reasoning model.

***Thinking model*: IF-RL with combined reward function**

Another approach is to design a reward function in IF-RL that jointly accounts for both human preference and precise instruction-following capability. For dedicated *thinking* models, this is crucial for mitigating the negative impact of IF-RL on benchmarks such as ArenaHard (Li et al., 2024).

We combine signals from both the rule-based instruction-following verifier and the human preference reward model, achieving the best of both worlds. For a given prompt q and a group of generated response $\{o_i\}_{i=1}^{G}$, the

reward for each response o_i is defined as,

$$r_i = \begin{cases} R_{\text{IF}}(o_i) + \text{sigmoid}(\hat{R}_{\text{RM}}(o_i)), & \text{if } R_{\text{IF}}(o_i) = 1 \\ 0, & \text{otherwise} \end{cases}, \text{therein } \hat{R}_{\text{RM}}(o_i) = \frac{R_{\text{RM}}(o_i) - \text{mean}(\{R_{\text{RM}}(o_i)\}_{i=1}^{G})}{\text{std}(\{R_{\text{RM}}(o_i)\}_{i=1}^{G})}$$

where $R_{\text{IF}}(o_i) \in \{0, 1\}$ refers to the binary reward from instruction-following verifier, $\hat{R}_{\text{RM}}(o_i)$ is the group-normalized reward (mean = 0, standard deviation = 1) from the same reward model used in the RLHF stage. The sigmoid function is applied to $\hat{R}_{\text{RM}}$ to scale its values to the range $(0, 1)$, ensuring it is on the same scale as R_{IF} before aggregation.

We then perform IF-RL using the same GRPO objective described in § 4.1.2, incorporating the combined reward. For the dedicated *thinking* model, we set the maximum response length to 8K tokens with overlong filtering during the first-stage IF-RL training, and increase it to 16K tokens with overlong filtering in the second stage to accommodate the longer reasoning required for difficult prompts in *thinking* mode.

Hyperparameters:
For both 8B and 14B models, we use a batch size of 128, sampling 8 responses per prompt with temperature 0.6, top-p 0.95, and top-k 20. We adopt a learning rate of 2e-6 with AdamW (Kingma, 2014), and set both the entropy loss coefficient and KL loss coefficient to 0. For *non-thinking* mode IF-RL, the first-stage training takes around 2000 steps, and the second-stage training takes 1000 steps. For *thinking* mode IF-RL, the first-stage training takes around 500 steps, and the second-stage training takes around 300 steps. More details of training hyperparamters can be found in Appendix D.

4.4.3. Results after IF-RL

The results after IF-RL are presented in Table 5. We observe significant improvements on IFEval and IFBench, with controlled small degradation on ArenaHard when applying the improved techniques to both the *unified* models and the dedicated *thinking* model. We also find that IF-RL generally reduces model entropy and shortens the average length of reasoning tokens (see Figure 8 for an illustration). On the negative side, this introduces minor degradations on reasoning benchmarks, although most of them (except ArenaHard) are fully recoverable and are further improved after the subsequent Math RL, Code RL, and SWE RL stages. On the positive side, it compresses the reasoning trace and improves token efficiency. Overall, the *unified* reasoning model achieves a stronger balance than dedicated *thinking* model, delivering robust performance on both ArenaHard and IFEval.

4.5. Math RL

In this subsection, we describe the Math RL stage, which focuses on enhancing the model's mathematical reasoning and problem-solving capabilities through reinforcement learning. In our final model training, Math RL is applied after the Instruction-Following RL stage. Applying Math RL directly to RLHF checkpoints yielded very similar results.

4.5.1. Data Curation

We mainly use the AceReason-Math dataset (Chen et al., 2025) and filter out overly simple problems, retaining 18K high-quality math problems for RL training. The dataset merges the DeepScaleR blend (Gao et al., 2024; Luo et al., 2025; Min et al., 2024) and NuminaMath (Li et al., 2024), encompassing topics such as algebra, geometry, combinatorics, and number theory. We apply 9-gram filtering to prevent contamination with common math benchmarks, such as AIME 2024/2025 and MATH (Hendrycks et al., 2021). We further exclude questions unsuitable for RL with symbolic rule-based verification, such as multiple-choice or true/false questions (where answers can be easily guessed), proof-based problems (which are difficult to verify for correctness), those containing multiple sub-questions, non-English questions (which increase language mixing), and questions referencing figures. Because NuminaMath contains OCR and parsing errors, each problem is verified by the DeepSeek-R1 model with up to eight attempts. A rule-based verifier retains only problems with majority-voted

Table 5: The evaluation results of 8B/14B-Thinking and the *unified* 8B models **after IF-RL** are presented below. For unified model, we evaluate IFEval in the non-thinking mode and all other benchmarks in the thinking mode. We compare the results against those obtained after RLHF in Table 4, using ↑ to denote improvements and ↓ to mark degradations after IF-RL training.

Benchmark **Metric: pass@1**	**8B-Thinking IF-RL**	**8B (unified) IF-RL**	**14B-Thinking IF-RL**
Knowledge and Reasoning			
MMLU (EM)	83.8 ↓0.1	83.4 ↑0.3	85.0 ↓0.2
MMLU-Pro (EM)	74.8 ↓1.4	74.5 ↓3.3	76.4 ↓1.3
GPQA Diamond (avg@8)	65.2 ↓2.1	66.1 ↓0.7	70.1 ↓1.2
Alignment			
ArenaHard (GPT4-turbo-2024-04-09)	86.3 ↓3.6	88.0 ↓2.1	90.2 ↓2.9
IFEval (strict prompt) (avg@8)	83.3 ↑37.8	90.4 ↑40.3	81.3 ↑25.1
IFBench (avg@8)	42.1 ↑18.2	40.5 ↑16.0	40.4 ↑14.8
Math			
AIME 2024 (avg@64)	85.6 ↓0.8	86.2 ↑0.1	89.2 ↑0.8
AIME 2025 (avg@64)	72.3 ↓2.8	75.2 ↑0.2	82.3 ↑0.5
Code			
LiveCodeBench v5 (avg@8)	69.0 ↓1.3	70.2 ↑0.0	75.3 ↑0.1
LiveCodeBench v6 (avg@8)	65.9 ↓1.4	66.7 ↓0.5	72.7 ↑0.4
SWE-bench Verified (avg@4)	32.4 ↓0.9	28.3 ↑0.1	38.4 ↓0.4

correct answers, while ambiguous or noisy items are discarded. Finally, we remove overly simple problems that AceReason-Nemotron-7B (Chen et al., 2025) can solve with a $\geq$ 75 % success rate over 16 samples, reducing the dataset size from the original 49K to 14K problems.

4.5.2. Training Recipe

Our goal is to develop a general math RL recipe that can be applied across different base models and scaled efficiently for large-scale RL training. We build on the AceReason-Nemotron (Chen et al., 2025) training strategy, which strictly adheres to **on-policy training** under the GRPO objective, removes KL regularization entirely, and combines **length extension training** with **dynamic filtering** to stabilize optimization. We find that **initializing Math RL from RLHF-trained models** plays a crucial role in achieving better performance. Throughout the development cycle, we applied this training recipe to five different 8B checkpoints and consistently achieved accuracies of around 90% on AIME24 within 500 RL steps, demonstrating the robustness of the approach across models with different training dynamics. Below, we describe each component in detail.

Initialization from models that have undergone RLHF:
In an early study, we explored an approach that first applied Math RL and Code RL, followed by RLHF and IF-RL. Later, we found that initializing Math RL from RLHF-trained models is highly beneficial because it *(i)* provides a much stronger initial math reasoning capability than SFT checkpoints—response quality is substantially improved, and reasoning becomes more token-efficient after RLHF (e.g., less verbosity and repetition); and *(ii)* significantly reduces the number of steps required for math RL training.

In practice, we insert IF-RL between RLHF and Math RL, as IF-RL reduces model entropy and shortens the reasoning trace, which can temporarily hurt reasoning-related benchmarks. Applying Math RL and Code RL with high temperature after IF-RL restores the model entropy to a normal level.

Reward function:
Rewards are assigned strictly based on answer correctness, which is determined by extracting the boxed answer (`\\boxed{}`) that follows the `<\think>` token, and verifying it using the AceMath (Liu et al., 2024) rule-based verifier (1 for correct and 0 for incorrect). To prevent language mixing during the reasoning process,

we apply a code-switching penalty by assigning a reward of -1 whenever tokens from a language (e.g., Chinese) different from the original prompt's language (e.g., English) are detected in the reasoning chain.

Response length extension training:
The key driver of performance improvement lies in the model's ability to think more deeply and produce longer reasoning chains. We adopt a staged response length extension curriculum with a custom configuration (24K $\rightarrow$ 32K $\rightarrow$ 40K) where each stage plays a distinct role: compressing overlong reasoning, stabilizing reasoning length, and finally extending longer reasoning chains, respectively. One key benefit of starting with the compression stage (i.e., 24K) is that it brings different initial models into a consistent reasoning length range (around 16K on the full training set), which enables subsequent training stages to work effectively across diverse initial models without extensive hyperparameter tuning.

- **24K (Compression Stage).** We begin by training with a 24K token budget to address a key issue observed in small and medium sized SFT checkpoints: these models tend to generate *overlong reasoning chains*, leading to incomplete ratios of 15–20% on the AIME benchmark under a 32K token budget. This overgeneration wastes tokens and often leaves solutions unfinished. By starting with a shorter 24K budget, we encourage the model to *compress and refine* its reasoning. At this stage, models typically exhibit very high incomplete ratios (30–50%) initially, but after around 100 steps of training, this drops to around 15% on training set. Importantly, we deliberately apply the overlong filtering (i.e., we skip generations exceeding 24K tokens rather than assigning a reward of 0), as doing so may excessively penalize long reasoning on difficult problems, causing a sharp performance drop during compression (Liu et al., 2025) and leading to unstable training due to noisy rewards in the high–incomplete-ratio regime.
- **32K (Extension Stage).** Once the reasoning chains are stabilized at 24K, we extend the token budget to 32K. However, checkpoints emerging from the 24K stage vary considerably in how efficiently they use tokens: some start the 32K phase with as little as 5% incomplete ratio, while others hover around 10%. This variability motivates treating the 32K phase as a *controlled extension stage*. Here, we do not apply overlong filtering to regularize reasoning length to fit the 32K context (i.e., assigning a reward of 0 for overlong generation). As training progresses, models not only adapt to the larger budget but also begin to surpass their starting accuracy, reflecting a balanced trade-off between length and correctness.
- **40K (Long Reasoning Stage).** After 32K training, model accuracy on easy and medium problems[1] nearly saturates (99% and 85% respectively) on AIME24/25, but hard problems remain challenging, with accuracy plateauing below 30%. Since our evaluation is performed with a 64K token budget, we observed that models were not fully exploiting the available context even with YARN length extension (factor of 2). To address this gap, we push the model on a final 40K training stage. This extension explicitly incentivizes the model to leverage more tokens during reasoning. As a result, performance on hard AIME problems improves significantly from 30 to 40%, while performance on other problems remains at high level.

Dynamic filtering:
To simplify development, we fix a seed dataset across all math RL experiments. However, since model capabilities vary, overly simple or unsolvable problems provide no useful policy gradient signal when using the group-normalized advantage function. To address this, after each epoch, we filter out problems that achieve either 100% or 0% accuracy based on the verification results from that epoch's RL training. Hard problems that were filtered out are re-sampled into the dataset with a 10% probability, as the policy may learn to solve these problems during subsequent updates within the same epoch. Easy problems are re-sampled into the dataset with a 1% probability to stabilize training, as the policy may forget how to solve them within an epoch. This ensures that $\sim$90% of training samples contribute meaningful learning signals and significantly stabilizes model accuracy during training, especially at later training stage when more problems are 100% solved. Note that this epoch-based dynamic filtering can be viewed as a more efficient alternative to batch-based dynamic

[1]We categorize each problem in AIME24/25 into easy (80-100%)/medium (30-80%)/hard (0-30%) based on the problem accuracy of an early checkpoint.

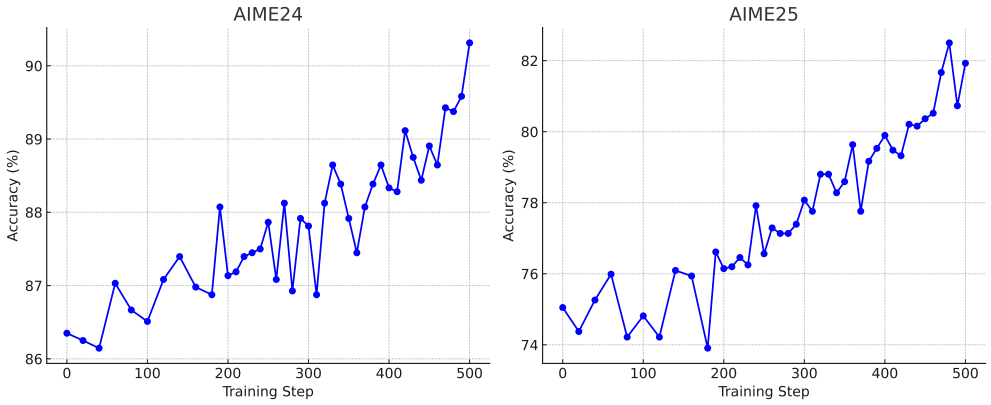


Figure 4: Training curve of math RL for 8B *unified* model on AIME24 and AIME25 (max response length 64K, avg@64).

Table 6: The evaluation results of 8B/14B-Thinking and the *unified* 8B models **after Math RL** are presented below. For unified model, we evaluate IFEval in the non-thinking mode and all other benchmarks in the thinking mode. We compare the results against those obtained after IF-RL in Table 5, using ↑ to denote improvements and ↓ to mark degradations after Math RL training.

Benchmark **Metric: pass@1**	**8B-Thinking** **Math RL**	**8B (unified)** **Math RL**	**14B-Thinking** **Math RL**
Knowledge and Reasoning			
MMLU (EM)	83.8 ↑0	83.4 ↑0	84.8 ↓0.2
MMLU-Pro (EM)	75.0 ↑0.2	75.0 ↑0.5	76.9 ↑0.5
GPQA Diamond (avg@8)	63.8 ↓1.4	65.7 ↓0.4	67.6 ↓2.6
Alignment			
ArenaHard (GPT4-turbo-2024-04-09)	84.0 ↓2.3	87.0 ↓1.0	89.3 ↓0.9
IFEval (strict prompt) (avg@8)	84.7 ↑1.3	92.1 ↑1.7	84.3 ↑3.0
IFBench (avg@8)	39.8 ↓2.3	40.4 ↓0.1	41.0 ↑0.6
Math			
AIME 2024 (avg@64)	90.2 ↑4.6	90.2 ↑4.0	90.4 ↑1.3
AIME 2025 (avg@64)	80.2 ↑7.9	81.9 ↑6.7	83.3 ↑1.0
Code			
LiveCodeBench v5 (avg@8)	71.2 ↑2.2	70.6 ↑0.4	75.2 ↓0.1
LiveCodeBench v6 (avg@8)	68.1 ↑2.2	67.4 ↑0.7	72.4 ↓0.3
SWE-bench Verified (avg@4)	32.5 ↑0.1	30.6 ↑2.3	39.7 ↑1.3

sampling (Xiaomi et al., 2025; Yu et al., 2025), as the latter requires substantially more rollouts to construct a fixed-size batch free of overly simple or unsolvable prompts.

Hyperparameters:
We use a batch size of 128, sampling 8 responses per prompt with temperature 1 and top-p 0.95. We adopt a learning rate of 2 or 2.5×10^{-6} with AdamW (Kingma, 2014), and set both the entropy loss coefficient and KL loss coefficient to 0. Each stage of training takes around 100 to 200 steps depends on how fast the clip-ratio reaches 10%. For 8B models, we adopt the 3 stage training with length extension from 24K $\rightarrow$ 32K $\rightarrow$ 40K. For the 14B model, as the initial policy already achieve high accuracy, we start with 28K max token length to avoid accuracy drop in the first stage and then extend to 40K directly. More details of training hyperparamters can be found in Appendix D.

4.5.3. Results after Math RL

We monitor the training dynamics of Math RL for the 8B unified model by tracking its performance on AIME24 and AIME25, as shown in Figure 4. The results after Math RL are presented in Table 6. We observe noticeable improvements on AIME 2024 and 2025. Overall, Math RL has minimal effect on knowledge reasoning and alignment benchmarks. Most of the observed differences can be attributed to evaluation variance and checkpoint selection. Math RL improves the coding benchmarks, including both LiveCodeBench and SWE. Although the gains are not as pronounced as those reported in AceReason-Nemotron (Chen et al., 2025), this is largely because our starting models already exhibit strong general reasoning capabilities prior to Math RL.

4.6. Code RL

In this subsection, we describe the Code RL process, which focuses on improving the model's competitive programming performance through reinforcement learning. We apply Code RL to the model checkpoint obtained after Math RL.

4.6.1. Data Curation

We construct our Code RL training dataset based on the AceReason-Nemotron coding corpus (Chen et al., 2025), which is primarily curated from open-source datasets containing unit tests, including TACO (Li et al., 2023), APPS (Hendrycks et al., 2021), DeepCoder (Luo et al., 2025), and others. These problems cover a wide range of algorithmic topics commonly found in modern competitive programming. We apply strict filtering rules to exclude problems that are incompatible with standard output comparison (e.g., interactive formats or those requiring special judges), as well as problems with insufficient unit test coverage for edge and corner cases. This filtering process substantially reduces false-positive and false-negative reward signals during training, which are known to degrade Code RL performance (Chen et al., 2025).

We further perform rigorous validation of the training set to remove duplicates and prevent benchmark contamination, using 9-gram filtering and raw problem URL matching. To calibrate problem difficulty, we employ AceReason-Nemotron-7B (NVIDIA, 2025) to exclude trivial problems (solved in all 8 out of 8 rollouts) and DeepSeek-R1-0528 (DeepSeek-AI, 2025) to filter out intractable or overly difficult ones (unsolved in all 8 rollouts), resulting in a final training set of 9.8K samples.

4.6.2. Training Recipe

We conduct Code RL after Math RL, as the Math RL stage serves as an effective warm-up that stabilizes future RL training and enhances the model's general reasoning capabilities (Chen et al., 2025). Following the AceReason-Nemotron recipe, we perform single-stage, on-policy Code RL (without KL regularization, using token-level loss as in § 4.1.2) initialized from the final Math-RL model checkpoint. During training, the maximum response length is set in the range of 44K–48K, with no overlong filtering applied.

Reward function:
Code RL adopts a strict binary rule-based reward function, where a reward of 1 is assigned only when the generated code passes all test cases for the given problem; otherwise, a reward of 0 is assigned. For efficient and robust evaluation, we employ the parallelized code verifier from the AceReason Evaluation Toolkit to verify the correctness of the model's generated code. Furthermore, we apply asynchronous reward computation in VeRL (Sheng et al., 2024), as code verification incurs significant overhead. This asynchronous computation substantially reduces the averaged code verification time per batch. For example, when training Code RL on 8 DGX H100 nodes with a batch size of 128 and a rollout of 8, the verification time drops from 1172.4 seconds to 416.2 seconds.

Similar to Math RL, we also apply a code-switching penalty, assigning a reward of 0 whenever tokens from a language different from the original prompt's language are detected in the reasoning trace. Unlike in Math RL, we find that assigning a reward of -1 for code-switching negatively impacts coding performance. This is likely

Table 7: The evaluation results of 8B/14B-Thinking and the *unified* 8B models **after Code RL** are presented below. For unified model, we evaluate IFEval in the non-thinking mode and all other benchmarks in the thinking mode. We compare the results against those obtained in the previous stage (Math RL) in Table 6, using ↑ to denote improvements and ↓ to indicate degradations after Code RL training.

Benchmark Metric: pass@1	8B-Thinking Code RL	8B (unified) Code RL	14B-Thinking Code RL
Knowledge and Reasoning			
MMLU (EM)	84.3 ↑0.5	83.7 ↑0.3	85.1 ↑0.3
MMLU-Pro (EM)	75.4 ↑0.4	75.3 ↑0.3	77.6 ↑0.7
GPQA Diamond (avg@8)	67.7 ↑3.9	67.4 ↑1.7	70.3 ↑2.7
Alignment			
ArenaHard (GPT4-turbo-2024-04-09)	85.4 ↑1.4	87.8 ↑0.8	89.8 ↑0.5
IFEval (strict prompt) (avg@8)	83.1 ↓1.6	90.7 ↓1.4	81.8 ↓2.5
IFBench (avg@8)	41.8 ↑2.0	38.1 ↓2.3	41.0 ↑0.0
Math			
AIME 2024 (avg@64)	88.3 ↓1.9	89.1 ↓1.1	90.4 ↑0.0
AIME 2025 (avg@64)	81.8 ↑1.6	80.5 ↓1.4	83.5 ↑0.2
Code			
LiveCodeBench v5 (avg@8)	74.3 ↑3.1	75.3 ↑4.7	78.0 ↑2.8
LiveCodeBench v6 (avg@8)	71.0 ↑2.9	71.5 ↑4.1	74.8 ↑2.4
SWE-bench Verified (avg@4)	33.3 ↑0.8	31.6 ↑1.0	39.6 ↓0.1

because the additional penalty encourages the model to produce incorrect answers without code-switching in GRPO training when all rollouts in the group are either incorrect or contain language mixing.

Hyperparameter:
We set the batch size to 128, the learning rate to 4×10^{-6} with the AdamW optimizer, and use 8 rollouts per training prompt. We set sampling temperature as 1.0 and `top_p` as 0.95 as we found that Code RL is sensitive to the temperature configuration. The detailed hyperparameters can be found in Appendix D.

4.6.3. Results after Code RL

The results after Code RL are presented in Table 7. We observe strong gains on LiveCodeBench (LCB). For instances, our unified 8B achieves 75.3 on LCB v5 and 71.5 on LCB v6, matching the performance of DeepSeek-R1-0528 (671B), which achieves 74.8 and 73.3, respectively. Our 14B-Thinking achieves 78.0 on LCB v5 and 74.8 on LCB v6, outperforming DeepSeek-R1-0528 by a clear margin. Since DeepSeek-R1-0528 (671B) is the teacher model used during SFT, these results highlight how remarkably effective Cascade RL is at strengthening code-reasoning ability—even for small 8B and 14B models. Code RL also has minimal impact on benchmarks from other domains, aside from normal checkpoint and evaluation variance.

The superb coding capability of our Nemotron-Cascade models is further examined in Section §5.

4.7. SWE RL

In Section § 3.3.2, we employ the Agentless framework for SWE-bench (Jimenez et al., 2023), decomposing the SWE task into three sub-tasks: localization, repair, and patch validation. We construct the SFT data for each of these sub-tasks accordingly. Among these sub-tasks, code repair is the most critical one, requiring the highest level of reasoning and model capability to generate revised code patches that fix bugs and address underlying issues. As a result, our RL process for SWE is primarily designed to enhance code repair accuracy.

4.7.1. Data Curation

As described in § 3.3.2, the RL dataset for code repair consists of more challenging instances than those used in the SFT stage. Specifically, we retain prompts for which fewer than four of the eight sampled responses exceed

the 0.5 similarity threshold, while at least one of the eight responses from DeepSeek-R1-0528 (DeepSeek-AI, 2025) attains non-zero similarity (indicating the prompt is not too hard or unsolvable).

During the SFT stage, our models are fine-tuned with a maximum total sequence length of 32K. Accordingly, we construct prompts that contain only the ground-truth localization files—i.e., all files that include bugs or require modifications to resolve the issue—as references for code repair. However, when evaluating model performance under the agentless framework, we provide the model with file contents retrieved from the localization stage as input for code repair. This setup introduces a discrepancy between the SFT training and the final evaluation. To ensure that the ground-truth localization files are included in the repair prompts, we incorporate the top-k ($k \geq 4$) localized files and extend the maximum prompt length to 60K with YaRN scaling factor 3.

This design naturally introduces out-of-distribution contexts for the SFT model in two ways: *i*) the total input length during code repair exceeds the maximum sequence length used in SFT; and *ii*) the inclusion of top-k localized files may bring in irrelevant files, making the code repair task more challenging than during SFT.

To address this issue, we construct and combine two subsets of long prompts (up to l tokens) for RL training:

1. Ground-truth only: Similar to SFT, prompts are constructed using only the ground-truth localization files.
2. Mixed localization: Augmented prompts are built using both the files localized by DeepSeek-R1-0528 and the ground-truth localization files. We include up to five files in total and ensure that all ground-truth files are present. Specifically, the initial prompt contains only the ground-truth files. We then add noisy files one by one until the total prompt length would exceed l; if the limit is exceeded before adding any noisy files, we discard the instance. To enhance robustness, we also randomize the order of files within each prompt.

To further enhance training efficiency, for both subsets, we discard prompts whose total length is shorter than 8K tokens. In § 7.3, we will ablate the effectiveness of RL training with various l.

4.7.2. Training Recipe

We perform reinforcement learning for software engineering (SWE RL) as the final stage of Cascade RL, since it represents a more specialized task compared to the general domains. Starting from the checkpoint obtained after code RL, we conduct on-policy RL using the GRPO algorithm with a token-level loss, while removing KL regularization (see detailed configurations in § 4.1.2).

Reward function:

Previous studies (Jain et al., 2025; Luo et al., 2025) perform RL by executing model-generated code patches within Docker environments to obtain rewards. However, running and managing numerous Docker instances significantly limits scalability, constraining prior work to training datasets of around 10k unique instances. To overcome this limitation, we design a execution-free verifier as the reward model, enabling scalable RL training for code repair generation. That is, we define the reward r as the similarity between the generated patch $\hat{p}$ and the human-annotated ground-truth patch p^*:

$$r(\hat{p}, p^*) = \begin{cases} 1, & \text{if } s_{\text{lex}}(\hat{p}, p^*) = 1, \\ 0, & \hat{p} \text{ is identical to the original code snippet} \\ -1, & \text{if } \hat{p} \text{ cannot be parsed,} \\ s_{\text{sem}}(\hat{p}, p^*), & \text{otherwise,} \end{cases} \tag{2}$$

where $s_{\text{lex}}(\hat{p}, p^*)$ denotes the lexical similarity computed with *Unidiff* library following Wei et al. (2025), and $s_{\text{sem}}(\hat{p}, p^*)$ represents the semantic similarity score produced by a LLM. Specifically, we prompt the Kimi-Dev-72B model[2] (Kimi-Team et al., 2025) with a yes/no question to assess the semantic similarity between the

[2] https://huggingface.co/moonshotai/Kimi-Dev-72B

generated and golden patches (see the reward modeling prompt in Appendix C.2). The probability assigned to the "YES" token is directly used as the reward score. Note that we assign a reward of -1 when the model's generated patch fails to parse, and a reward of 0 when the generated patch is identical to the original code snippet. We refer readers to § 7.2 for ablation studies on reward functions.

Multi-stage RL training for input context extension:
Our preliminary experimental investigations reveal a strong positive correlation between input context length and SWE task performance, specifically demonstrating that the inclusion of additional retrieved files for analysis yields substantial performance improvements. This finding motivates the design of our training strategy, which exploits this relationship through controlled context expansion. To optimize the utilization of extended context while maintaining training stability, we implement a carefully designed two-stage curriculum that progressively expands the input context length from 16K to 24K tokens while maintaining a constant output length of 16K tokens. This approach ensures robust learning and avoids the degradation effects observed with immediate long-context training, which is particularly effective for 8B models, given that smaller models have limited long-context capability.

- **16K Context Initialization (Warmup Stage).** The training process begins with a conservative 16K input token budget, which serves as an essential warmup stage. Our empirical analysis shows that directly initializing training with a 24K context length leads to suboptimal convergence and degraded final performance—a phenomenon we attribute to the model's initial difficulty in attending to and synthesizing information across extended sequences. During this stage, the model learns fundamental long-context utilization skills and develops stable attention mechanisms for multi-file analysis within a manageable context window.
- **24K Context Extension.** Once the 16K setup reaches a reward plateau, with little improvement over successive iterations, we extend the context to 24K tokens. The timing of this transition is important: the model has already built strong multi-file analysis skills at 16K, forming a solid basis for scaling to longer context. During the extended phase, we observe steady gains in long-context understanding, including more advanced cross-file reasoning and improved synthesis of information across retrieved files. The model demonstrates increasing proficiency in leveraging the expanded context window, effectively using the additional retrieved files to produce more accurate solutions.

Hyperparameters:
We set the batch size to 128 and the learning rate to 2.5×10^{-6} using the AdamW optimizer. For each prompt, we generate 16 rollouts with a sampling temperature of 1 and set maximum response length to 16K. We apply the overlong filtering for the trajectories that reach maximum response length. The detailed hyperparamters can be found in Appendix D.

4.7.3. Results after SWE RL

The results after applying SWE RL are shown in Table 8. SWE RL yields substantial gains on SWE-bench Verified, while its positive or negative impact on benchmarks from other domains remains minimal and is largely attributable to checkpoint and evaluation variance. Our 14B-Thinking model achieves a pass@1 resolve rate of 43.1, already outperforming the recent open 32B specialized models, DeepSWE-32B (42.2) (Luo et al., 2025) and SWE-agent-LM-32B (40.2) (Yang et al., 2025). It also performs significantly better than other 14B general-purpose open LLMs, such as Qwen3-14B (27.4) and Ministral-3-14B-Reasoning-2512 (mistralai, 2025) (25.5). Interestingly, we find that the performance gap on SWE-bench Verified between the dedicated 8B *thinking* SFT model and the 8B *unified* SFT model (30.2 vs. 26.1 in Table 2) is largely mitigated after the full Cascade RL process (38.5 vs. 37.2). In conclusion, the unified Nemotron-Cascade-8B performs comparably to Nemotron-Cascade-8B-Thinking on all reasoning-related tasks, while performing substantially better on instruction-following tasks.

Table 8: The evaluation results of our final 8B/14B-Thinking and the *unified* 8B models **after SWE RL** are presented below (Note that they are the final models). For unified model, we evaluate IFEval in the non-thinking mode and all other benchmarks in the thinking mode. We compare the results against those obtained in the previous stage (Code RL) in Table 7, using ↑ to denote improvements and ↓ to indicate degradations after SWE RL training.

Benchmark Metric: pass@1	8B-Thinking SWE RL	8B (unified) SWE RL	14B-Thinking SWE RL
Knowledge and Reasoning			
MMLU (EM)	84.0 ↓0.3	83.7 ↑0.0	85.1 ↑0.0
MMLU-Pro (EM)	75.5 ↑0.1	75.7 ↑0.4	77.0 ↓0.6
GPQA Diamond (avg@8)	66.7 ↓1.0	66.5 ↓0.9	69.6 ↓0.7
Alignment			
ArenaHard (GPT4-turbo-2024-04-09)	85.8 ↑0.4	87.9 ↑0.1	89.5 ↓0.3
IFEval (strict prompt) (avg@8)	83.7 ↑0.6	90.2 ↓0.5	81.9 ↑0.1
IFBench (avg@8)	41.4 ↓0.4	40.8 ↑2.7	41.7 ↑0.7
Math			
AIME 2024 (avg@64)	88.8 ↑0.5	89.5 ↑0.4	89.7 ↓0.7
AIME 2025 (avg@64)	81.4 ↓0.4	80.1 ↓0.4	83.3 ↓0.2
Code			
LiveCodeBench v5 (avg@8)	74.5 ↑0.2	74.3 ↓1.0	77.5 ↓0.5
LiveCodeBench v6 (avg@8)	71.4 ↑0.4	71.1 ↓0.4	74.6 ↓0.2
SWE-bench Verified (avg@4)	38.5 ↑5.2	37.2 ↑5.6	43.1 ↑3.5

5. Deep Dive on Competitive Coding

We evaluate the performance of our Nemotron-Cascade models on challenging competitive programming benchmarks, including LiveCodeBench (Jain et al., 2024), which contains recently released AtCoder and LeetCode problems, and LiveCodeBench Pro (Zheng et al., 2025), which includes newly released Codeforces problems. To avoid benchmark contamination, we report accuracy only on problems released after our training data cutoff of 08/2024. For LiveCodeBench, we evaluate on subsets v5 (08/2024–02/2025, **279** problems) and v6 (08/2024–05/2025, **454** problems). For LiveCodeBench Pro, we use the two most recent subsets: 2025Q1 (01/2025–04/2025, **166** problems) and 2025Q2 (04/2025–07/2025, **167** problems). We perform our evaluation under avg@8 setting with thinking budgets as 64K tokens. We also evaluate model ELO score based on **51** Codeforces Rounds from LiveCodeBenchPro {2025Q1, 2025Q2} split. We also leave more details and analysis related to Elo Rating calculation in Appendix E.

As shown in Table 9, the Nemotron-Cascade models demonstrate strong performance across multiple competitive coding benchmarks, including the latest splits of LiveCodeBench and LiveCodeBench-Pro. Nemotron-Cascade-8B significantly outperforms nearly all recently released reasoning LLMs of comparable size and achieves comparable performance to the previous state-of-the-art distilled model, OpenReasoning-Nemotron-32B (Ahmad et al., 2025), despite using far fewer parameters. Notably, the Nemotron-Cascade-14B-Thinking model even outperforms DeepSeek-R1-0528—its SFT teacher, Qwen3-235B-A22B, and Qwen3-Next-80B-A3B-Thinking across all competitive coding benchmarks, demonstrating the exceptional effectiveness of Cascade RL.

5.1. Test-Time Scaling in Practice: IOI 2025

Beyond standard competitive coding platform benchmarks, we further conduct evaluations on one of the most challenging competitive programming contest: the International Olympiad in Informatics (IOI) 2025. IOI imposes a strict limit of at most 50 submissions per problem, each with official judgment feedback, but does not explicitly constrain the number of model generations to construct those submissions. To fully exploit the reasoning capabilities of our strongest Nemotron-Cascade model, we deploy our Nemotron-Cascade-14B-Thinking with the total 128K token thinking budget and propose a feedback-driven, test-time scaling pipeline

Table 9: Competitive programming results on comprehensive benchmarks, evaluated against a significantly expanded set of proprietary and open-source baseline models.

Models	LiveCodeBench		LiveCodeBench Pro				Codeforces	
	v5	v6	25Q1		25Q2		2501 - 2507	
	2408 - 2502	2408 - 2505	Easy	Med	Easy	Med	ELO	Percentile
o4-mini (high)	**82.8**	**80.2**	**85.4**	**51.7**	**84.5**	**29.8**	**2169**	**99.1**
o3 (high)	78.1	75.8	79.8	35.0	82.5	26.3	2094	98.4
o4-mini (medium)	76.3	74.2	-	-	-	-	-	-
Gemini-2.5-Pro-06-05	73.5	73.6	76.4	26.7	77.3	21.1	2034	98.1
DeepSeek-R1-0528	74.8	73.3	57.3	6.7	63.9	7.0	-	-
Qwen3-235B-A22B-Thinking-2507	81.6	78.7	75.8	18.8	77.6	17.5	1979	97.7
Qwen3-235B-A22B (thinking mode)	70.7	67.3	55.6	9.2	52.3	3.1	1516	86.2
Qwen3-Next-80B-A3B-Thinking	76.0	73.2	68.5	**16.3**	**69.1**	7.5	1800	95.8
Magistral-Medium-1.2-2509	75.0	-	-	-	-	-	-	-
OpenReasoning-Nemotron-32B	71.8	70.2	66.9	11.7	66.2	8.3	1766	95.3
Llama-3.3-Nemotron-Super-49B-v1.5	70.3	68.1	63.5	11.3	61.9	6.1	1716	94.1
NVIDIA-Nemotron-Nano-9B-v2	68.2	65.3	61.0	7.9	59.3	4.8	1642	91.2
Meta-CWM-32B	-	63.5	-	-	-	-	-	-
Klear-Reasoner-8B	66.0	63.0	50.3	8.8	49.7	2.2	1517	86.2
AReaL-Boba-2-14B	70.3	67.4	56.2	4.2	48.7	1.8	1446	82.2
AceReason-Nemotron-1.0-14B	61.1	58.7	50.0	5.0	47.9	2.2	1437	81.3
AceReason-Nemotron-1.1-7B	57.2	55.2	40.2	2.5	42.3	1.8	1340	70.2
Nemotron-Cascade-8B	74.3	71.1	65.0	12.1	65.7	6.4	1789	95.7
Nemotron-Cascade-8B-Thinking	74.5	71.4	64.2	12.5	64.8	6.1	1740	94.7
Nemotron-Cascade-14B-Thinking	**77.5**	**74.6**	**71.6**	**16.3**	68.9	**10.5**	**1932**	**97.2**

as follows.

The whole pipeline can be viewed as a multi-round *generate–select–submit* process, with up to 50 rounds per problem (one for each submission). In each round, for every subtask of a problem, the model generates 20 candidate responses with different random seeds. We then filter out ***(i)*** incomplete responses that do not contain code, and ***(ii)*** generated code that fails to pass the provided sample test cases (if any). Among the remaining candidate generations for each subtask, we apply the Tail-10 selection heuristic from Fu et al. (2025) to obtain the final high-quality response, and submit this response to the official judge to obtain verdicts and (for partial-score tasks) scores.

After each round, we update each subtask's generation prompt to incorporate the new feedback from official judge, so the later generations are conditioned on the history of failed submissions. Specifically, for each unsolved subtask in *classic* problem, we append to the next-round prompt with up to 5 most recent submission codes to this subtask, and their corresponding official verdicts. We intentionally cap this history cache size at 5 to avoid overfitting to earlier failed attempts while still encouraging the model to analyze and improve upon past incorrect attempts. For *partial-score* problems, we instead append up to 3 highest-scoring prior submissions and encourage the model to keep improving the scores.

Beyond submission history, we also bring cross-subtask insights: once a subtask is solved, its correct solution code is appended as *insight* when prompting the model for other unsolved subtasks with different constraints of the same problem. This encourages the model to reason about relationships between constraints and to transfer good insights across subtasks. The complete prompt template is provided in Appendix C.3.

With this effective and explicitly self-improving test-time scaling strategy, our 14B-Thinking model achieves the overall score of **343.37** on IOI 2025, corresponding to a solid silver medal performance with at most 1000 generations (20 generations $\times$ 50 rounds), and no more than 50 official submissions to each problem. Notably, on IOI2025 Problem 2 *Triples*, which contains a constructive subtask that requires proposing and iteratively

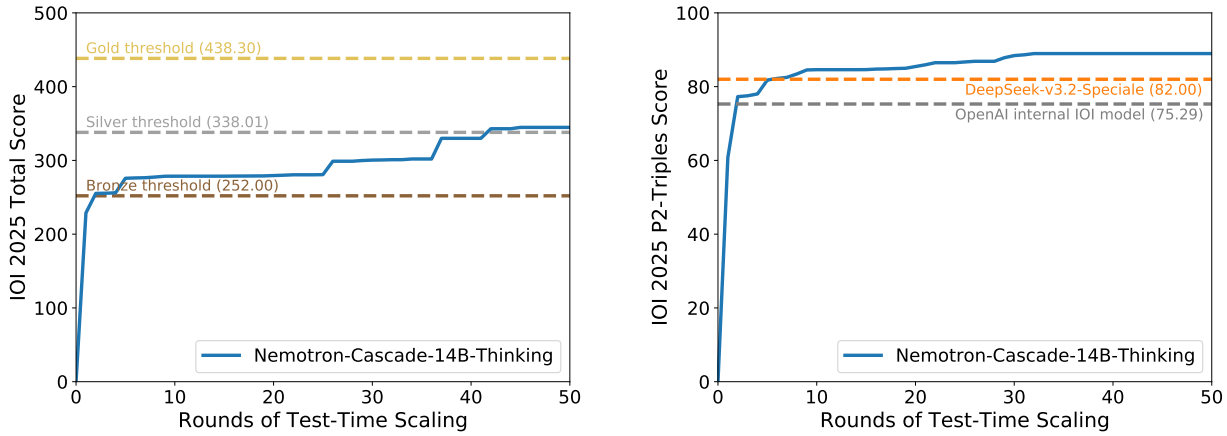


Figure 5: IOI 2025: Nemotron-Cascade-14B-Thinking's score on **(Left)** Full problem set; **(Right)** Problem 2 *Triples* after rounds of our proposed test-time scaling pipeline.

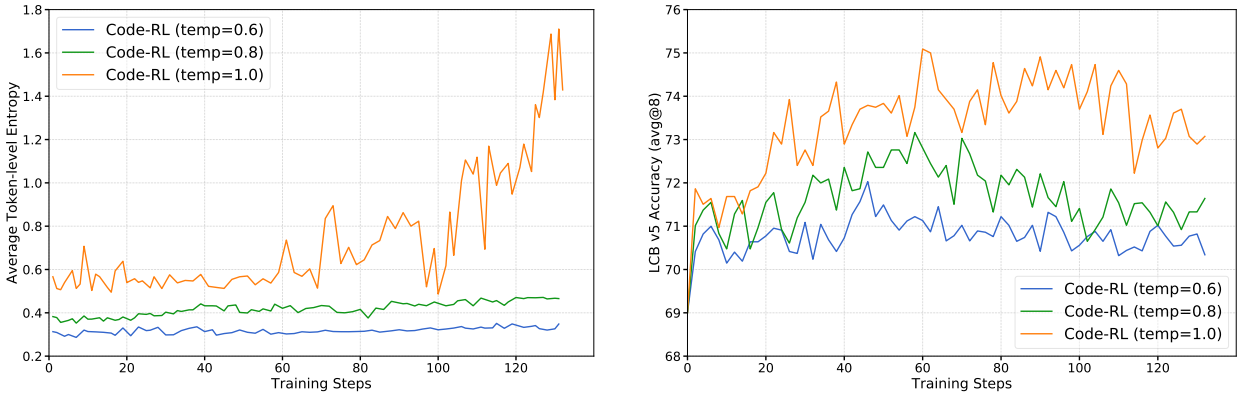


Figure 6: **(Left)** Average Token Entropy **(Right)** Model accuracy curves of our 8B *unified* model during Code RL training under different temperatures {0.6, 0.8, 1.0}. Training with high temperature yields better model accuracy but may suffer from training instability.

refining a construction algorithm, our pipeline achieves **90.37** points, outperforming both OpenAI's internal IOI-gold model (75.29 points) and DeepSeek-V3.2-Speciale (82 points) (Liu et al., 2025). This proves the effectiveness of our feedback-driven, self-evolving test-time scaling approach, on real, high-stakes competitive programming problems. We also show our rounds of progress in Figure 5.

We further analyze Code-RL training through the following ablations:

5.2. The Role of Training Temperature in Code RL

To identify the most suitable temperature for Code RL training, we conduct ablation experiments using temperatures of 0.6, 0.8, and 1.0 on 8B unified model (RL curves shown in Figure 6). While lower temperatures produce more stable entropy curves, they lead to degraded code-reasoning performance compared with higher-temperature settings. This pattern suggests that in large, noisy sampling spaces such as code generation, higher temperatures encourage exploration and improve sample efficiency under limited rollout budgets. However, high temperatures may also induce training instability, leading to entropy explosion. Designing training frameworks that retain the benefits of high-temperature sampling while ensuring entropy stability is a promising direction for future work.

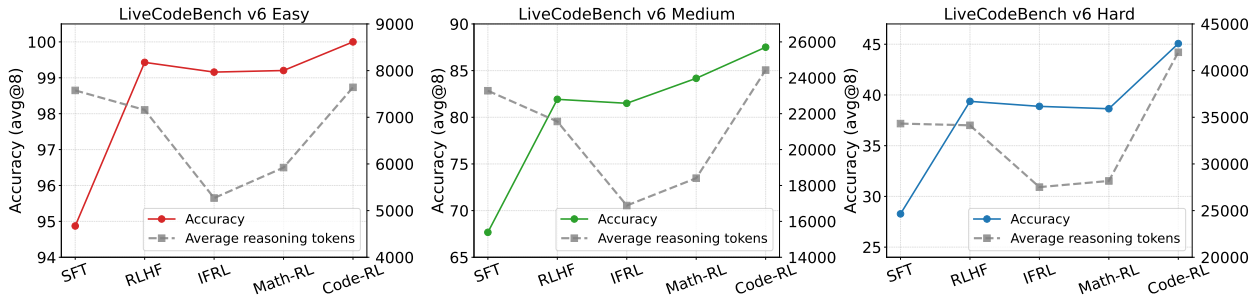


Figure 8: Nemotron-Cascade-8B accuracy (avg@8) and average reasoning token counts after cascade RL stages on each difficulty splits of LiveCodeBench v6 (2408-2505). Curves indicate the improvements of reasoning capability on solving algorithmic coding problems after each stage.

5.3. How Cascade RL Improves Code Reasoning

To assess the progressive effectiveness of our Cascade RL pipeline, we analyze the average reasoning-token usage and model accuracy across each difficulty split of LiveCodeBench v6 for our unified 8B model after successive cascaded RL stages—SFT, RLHF, IF-RL, Math RL, and Code RL (Figure 8). As shown in the figure, the initial RLHF stage provides a strong foundation: it substantially improves reasoning-token efficiency and mitigates the verbosity of the SFT model, evidenced by sharply reduced reasoning tokens alongside significant accuracy gains across all difficulty splits. The subsequent IF-RL stage further encourages conciseness, yielding an additional 20% reduction in token usage with only a negligible accuracy drop (0.5%).

We further observe that performance on easy problems saturates (>99%) after the initial stages, shifting the room for improvement to the medium and hard splits. Math RL enhances reasoning by increasing token usage, improving accuracy on medium problems, while Code-RL provides the final performance lift on both medium and hard problems through substantially expanded reasoning traces.

Additionally, we conduct ablation experiments to analyze how Cascade RL improves coding capability at the topic level. We annotate the LiveCodeBench v6 problems with five sub-categories—Math, String, Graph, Data Structure, Geometry—and report the topic-wise accuracy of our unified 8B model after each Cascade RL stage in Figure 7. While RLHF provides strong initial gains across all sub-categories, Math RL primarily benefits math-related topics (Math, Graph, Geometry) and yields limited improvements on more computer-science-oriented ones (String, Data Structure). Code RL delivers the largest accuracy boost, improving performance across nearly all topics.

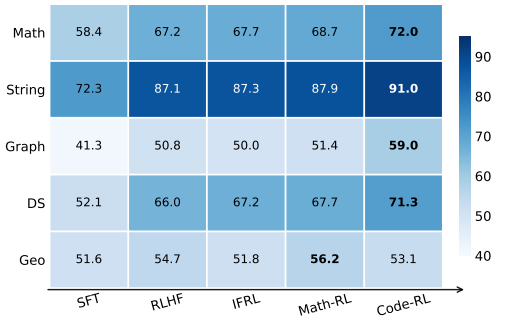


Figure 7: Topic-wise accuracy of unified Nemotron-Cascade-8B after Cascade RL stages on LCB v6 set. DS refers to Data Structure and Geo refers to Geometry.

6. Deep Dive on RLHF

In this section, we present our findings on selecting effective reward models and designing a robust RLHF recipe. We show that RLHF trained with the largest reward model yields the strongest performance on the ArenaHard benchmark, particularly under style control (Chiang et al., 2024), which helps disentangle substance from stylistic preferences in LLM responses. We also find that smaller reward models tend to generate noisier reward signals, necessitating additional techniques such as reward shaping and KL regularization to preserve training stability. For larger reward models, these techniques are unnecessary: their reward signals are sufficiently accurate and consistent on their own, enabling stable RLHF training and better performance on other tasks.

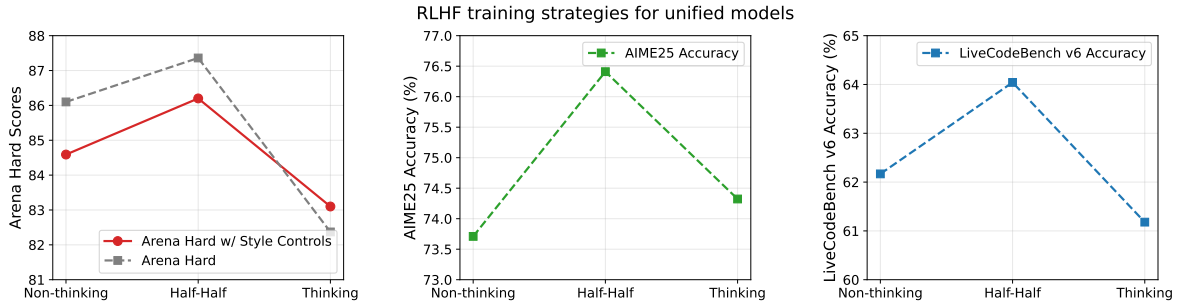


Figure 9: RLHF training results for unified 8B SFT model using the 72B reward model with different training strategies. Training RLHF in both *non-thinking* and *thinking* modes, with an equal split of prompts allocated to each mode within every batch ("Half-Half"), is significantly better than training the unified model in the *non-thinking* mode only ("Non-thinking") and the *thinking* mode only ("Thinking"), although the evaluation of ArenaHard, AIME, and LiveCodeBench is in the *thinking* mode only.

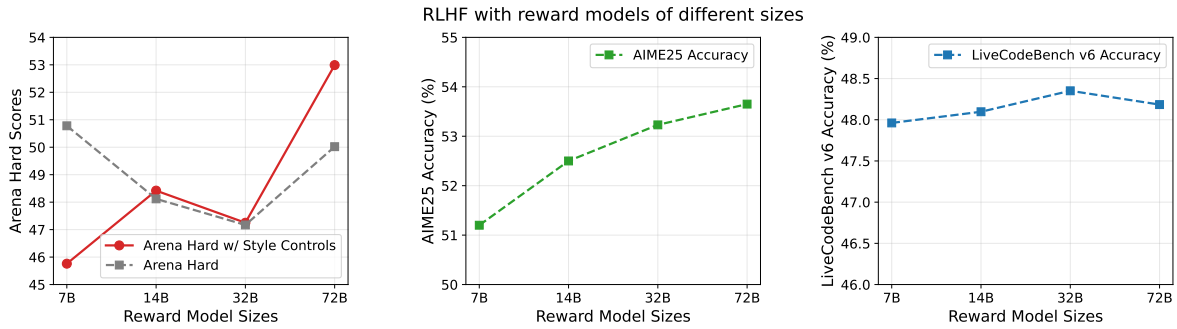


Figure 10: RLHF training results for AceReason-Nemotron-1.0-7B (Chen et al., 2025) using reward models ranging from 7B to 72B. Corresponding RewardBench scores for the reward models are provided in Table 3. Using the largest reward model yields the best ArenaHard performance under style control, while smaller reward models lead to noticeable degradation in ArenaHard scores as well as math and code capabilities.

6.1. RLHF Training Strategies for Unified Models

Since our unified model can respond in both *thinking* and *non-thinking* modes, a natural research question arises: which mode should we use for RLHF training, especially given that many benchmarks favor the *thinking* mode? To investigate this, we apply RLHF to our 8B unified SFT model (performance reported in Table 2) using the same training recipe described in Section 4.3.2, but vary the training mode. Specifically, the "Non-thinking" setting uses only the *non-thinking* mode during RLHF; the "Thinking" setting uses only the *thinking* mode; and the "Half-Half" setting splits each batch evenly between the two modes. The results, shown in Figure 9, reveal a clear trend: although ArenaHard, AIME, and LiveCodeBench are all evaluated in the *thinking* mode, training in the "Half-Half" setting provides the strongest overall performance, yielding the highest ArenaHard scores as well as improved performance on math and code benchmarks. This indicates that including samples in the *non-thinking* mode during RLHF can improve cross-mode transfer and alignment, leading to stronger general capabilities across both reasoning and non-reasoning settings.

6.2. Impact of Reward Model Size on RLHF Performance

A key research question in RLHF is how to select the most effective reward model. To study this systematically, we train a series of reward models ranging from 7B to 72B, and apply the same RLHF recipe described in § 4.3.2 to the AceReason-Nemotron-1.0-7B policy model [3] (Chen et al., 2025). We report ArenaHard scores as well as performance on math and code benchmarks in Figure 10. Our key findings are summarized below.

[3]We run early RLHF experiments with AceReason-Nemotron-1.0-7B as the policy model and perform ablations on it.

Table 10: RLHF training results for AceReason-Nemotron-1.0-7B using the 7B reward model. ArenaHard (SC) refers to ArenaHard score with style control. We select the best checkpoint for each method from its training trajectory before collapse and report the corresponding training step.

Techniques	Step	ArenaHard	ArenaHard (SC)	LCB v5	LCB v6	AIME24	AIME25
KL=1e-3 + token-level loss	350	46.63	43.05	50.81	47.47	64.58	52.55
KL=1e-3 + Seq-level loss	550	48.71	46.60	50.13	47.38	65.99	53.23
KL=1e-3 + Seq-level loss + Reward shaping	950	50.78	45.76	50.76	47.96	65.05	51.20

1. Larger reward models yield stronger ArenaHard performance. RLHF trained with the largest reward model achieves the highest ArenaHard scores, under style control (Chiang et al., 2024) that disentangles substance and style in the ArenaHard leaderboard. Notably, we observe a substantial gap for the 7B reward model depending on whether style control is enabled. This suggests that the 7B reward model is prone to reward hacking, e.g., improving ArenaHard scores primarily by increasing response length. We also examine the RLHF taining curves, and confirm that RLHF with the 7B reward model tends to improve reward scores by generating longer outputs, whereas training with the 72B reward model produces much more stable response lengths.
2. RewardBench is a useful proxy but not always predictive of RLHF quality. While RewardBench scores correlate with reward model quality overall, higher RewardBench performance does not necessarily translate into better ArenaHard scores. We hypothesize that RewardBench is relatively saturated (typically above 90), so marginal gains beyond this level do not meaningfully improve downstream helpfulness. In contrast, model-specific behaviors, such as vulnerability to reward hacking, play a more decisive role in determining RLHF effectiveness.
3. Larger reward models also improve performance on other tasks such as math. For example, RLHF trained with the 72B reward model achieves around 3% higher AIME25 accuracy than training with the 7B reward model. For code benchmarks, the choice of reward model has minimal impact, with performance differences within 1%.

6.3. Bag of Tricks for Stablizing RLHF Training

Although RL algorithms are crucial for enabling long-form chain-of-thought reasoning, RL training can be unstable and susceptible to early collapse. This issue is further amplified in RLHF, where training depends on model-based rewards that may be noisy or out-of-distribution. In this subsection, we summarize the set of techniques ("bag of tricks") we found effective for stabilizing RLHF training:

1. **KL penalty loss:** The KL penalty loss constrains the divergence between the online policy and the frozen reference policy, ensuring that the policy does not drift too far from the initial model. We find that when RLHF training collapses early, introducing this KL term is an effective way to maintain training stability.
2. **Policy gradient loss aggregation:** Standard GRPO uses a *sequence-level loss*, where token-level losses are first averaged within each sample and then aggregated across the batch. For long-CoT RL, *token-level loss*, where all token losses in the batch are averaged directly, is typically recommended. However, when RLHF shows signs of early collapse, switching from token-level to sequence-level aggregation helps suppress significant increases in response length and stabilizes training.
3. **Reward shaping:** Because our reward model is trained with a Bradley–Terry objective, its raw reward signals are unbounded. When training RLHF with unbounded rewards, noisy or outlier rewards can lead to unstable training. To address this, we design a reward-shaping mechanism: for each group of rewards, we compute the mean and standard deviation, and then normalize each reward by subtracting the mean and dividing by the standard deviation, producing a centered and scaled reward. Finally, we apply a `tanh` transformation. This bounds the shaped rewards within $[-1, 1]$, effectively mitigating the impact of outliers and noisy reward signals within the group and leading to more stable RLHF updates.

Table 11: RLHF training results for our unified 8B SFT model (performance reported in Table 2) using the 72B reward model. ArenaHard (SC) refers to ArenaHard score with style control. We select the best checkpoint for each method from its training trajectory before collapse and report the corresponding training step.

Techniques	Step	ArenaHard	ArenaHard (SC)	LCB v5	LCB v6	AIME24	AIME25
KL=1e-3 + Seq-level loss + Reward shaping	500	90.03	89.11	68.59	65.66	86.20	73.80
KL=0 + Token-level loss	600	91.04	89.37	68.46	65.86	86.33	75.03

In our early experiments with RLHF using the 7B reward model, we found that applying these "bag-of-tricks" techniques significantly improved training stability, extending the number of stable RL steps from 350 to 950, and led to better ArenaHard scores (Table 10). However, when using a stronger reward model (e.g., the 72B reward model), RLHF training is already stable, and omitting these techniques yields comparable—and in some cases slightly better—downstream performance than using them as shown in Table 11. Our takeaway is that these techniques should be viewed as a toolbox to deploy only when training shows signs of instability. Otherwise, the RLHF recipe described in § 4.3.2 is sufficient.

7. Deep Dive on SWE

In this section, we present the improved techniques for SWE tasks and provide the corresponding ablation results.

7.1. Generation–Retrieval Approach for Code Localization

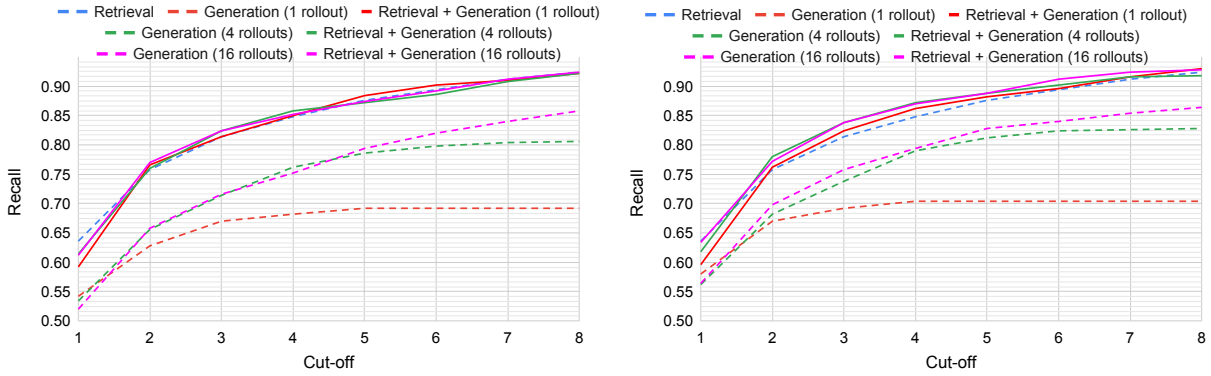
For the file-localization stage, we adopt a dual approach that combines generation-based and retrieval-based methods. In the generation-based approach, the model is guided to infer potentially buggy files based on the issue description and repository structure, as shown in Appendix C.2. To further enhance this method, we aggregate results from multiple rollouts and rank candidate files by their frequency of appearance, with higher-ranked files appearing more consistently across rollouts (Wang et al., 2023).

However, this generation-based approach only has access to the repository structure (i.e., folder and file names) rather than code contents. To complement this, we employ a code embedding model, NV-Embed-Code (Sohrabizadeh et al., 2025), to retrieve candidate files whose code contents are semantically similar to the problem context.[4] The final set of relevant files is then determined by aggregating the results from both approaches using reciprocal rank fusion (Cormack et al., 2009) with the hyperparameter k set to 0, which effectively integrates the complementary strengths of the two localization signals.

To evaluate code localization performance, we measure recall at various cutoffs (top-k). Specifically, a localization is considered successful (recall = 1) for a instance if all ground-truth files requiring fixes appear within the top-k retrieved candidates; otherwise, the recall is defined as 0 for this instance. Figure 11 illustrates the performance of different approaches for code localization on the SWE-bench Verified benchmark.

First, we observe that the retrieval-based method outperforms the generation-based ones. This improvement is likely because the retrieval-based approach encodes the full source code content of each repository, whereas the generation-based approach relies only on repository structure when identifying potentially relevant files. Second, the generation-based approach demonstrates consistent gains at both top and higher ranks when results from multiple rollouts are aggregated. This indicates that aggregation not only improves top-ranking accuracy but also promotes ranking diversity in code localization. Finally, combining the generation- and retrieval-based methods with reciprocal rank fusion yields slight additional improvements, particularly at cutoffs below 5. In all our experiments, we directly use reciprocal fusion from generation- (with 16 rollouts) and retrieval-based methods as default.

[4] https://build.nvidia.com/nvidia/nv-embedcode-7b-v1



(a) Nemotron-Cascade-8B

(b) Nemotron-Cascade-14B-Thinking

Figure 11: Ablation studies on code localization. The retrieval-based method uses NV-Embed-Code, while the generation-based method prompts the model to reason and generate the final ranking list (see Appendix C.2).

Table 12: Ablation results for different reward functions and reward shaping strategies on SWE-bench. Init. model is our intermediate 14B model without any code or math RL training.

Cond.	Reward Function		Repair w/ Ground-truth Loc.		Repair w/ Top-4 Loc.	
	Similarity	Reward shaping	avg@4	pass@4	avg@4	pass@4
0	Init. 14B model		41.7%	55.8%	38.8%	54.4%
1	Lexical Sim.	no	42.6%	57.0%	41.2%	54.0%
2		yes	42.8%	58.0%	41.6%	55.2%
3	Semantic Sim.	no	**43.0%**	**58.6%**	**42.3%**	56.4%
4		yes	42.9%	57.0%	42.1%	**56.8%**

7.2. Execution-Free Reward Model for SWE RL

As mentioned in § 4.7.2, we use a execution-free reward defined in Eq. (2) for code repair RL training. That is, given a human-written golden patch, we compare its similarity to the model-generated patch using either lexical similarity—computed with the *Unidiff* library following Wei et al. (2025)—or a semantic similarity score produced by the Kimi-Dev-72B model.

In our ablation study, we compare these two approaches to compute the similarity. We initialize from one of our intermediate 14B models without math and code RL (cond. 0 in Table 12) and conduct RL training for code repair using different similarity scores as reward functions. We follow the hyperameters in § 4.7.2 except for setting rollouts to 8, and for the ablation of reward models, we use the training data with the maximum prompt length of 24K. We evaluate the trained models under two settings: *i*) when ground-truth localized files are provided in the prompt, and *ii*) when the top-4 localized files are obtained from generation–retrieval method.

For the reward model based on semantic similarity, we directly apply the original reward function defined in Eq. (2). For lexical similarity, we replace $s_{\text{sem}}(\hat{p}, p^*)$ with $s_{\text{lex}}(\hat{p}, p^*)$ in this reward function. Table 12 reports the resolve rates averaged over four runs with sampling temperature set to 0.6, along with pass@4; that is, an instance is considered resolved if it is successfully fixed in at least one of the four generations.

First, we observe that RL training generally enhances the model's effectiveness in code repair, and using semantic similarity as the reward model yields better effectiveness than using lexical similarity (cond. 4 vs. 2). Second, we apply reward shaping to both reward models by setting the reward to 0 when it falls below 0.5. This adjustment improves the effectiveness of the lexical similarity reward model (cond. 2 vs. 1), suggesting that reward shaping helps filter out noisy supervision signals. In particular, when lexical similarity is below 0.5, the reward tends to provide unreliable guidance for model training. However, we do not observe the same

Table 13: Ablation studies on SWE RL for code repair using training data with various maximum prompt lengths. Init. model is our intermediate 14B model without any code or math RL training.

Cond.	Max Prompt Len.	Repair w/ Ground-truth Loc.		Repair w/ Top-4 Loc.	
		avg@4	pass@4	avg@4	pass@4
0	Init. 14B model	41.7%	55.8%	38.8%	54.4%
1	16K	44.0%	**58.8%**	41.6%	54.6%
2	24K	43.0%	58.6%	42.3%	**56.4%**
3	32K	**44.1%**	57.6%	**42.7%**	56.2%
4	40K	42.8%	57.6%	41.5%	55.4%

effect when applying reward shaping to semantic similarity (cond. 4 vs. 3), indicating that semantic similarity continues to provide meaningful training signals even when code similarity is low. As a result, we apply the default reward function setup (cond. 3 in Table 12) for SWE RL training.

Overall, we demonstrate that using an LLM-based, execution-free reward model is a promising direction for scaling SWE RL training. We leave the exploration of reward model training as future work.

7.3. Improving Long-Context Analysis

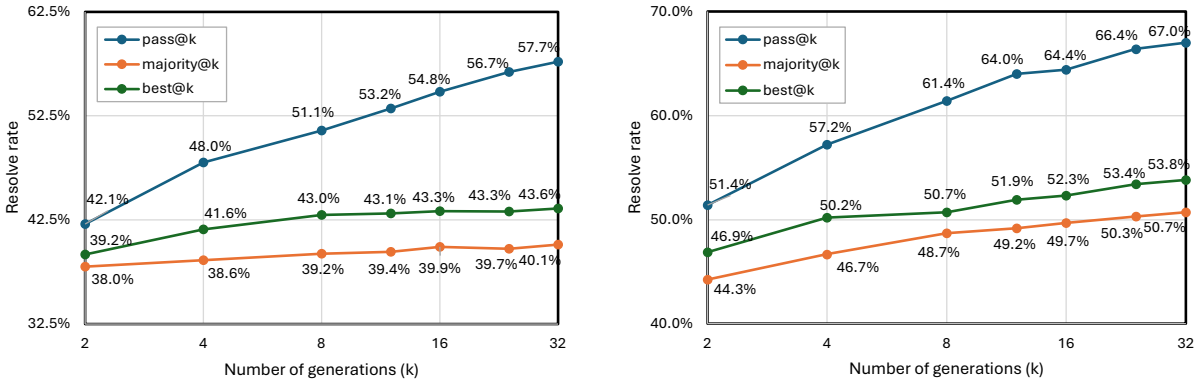
To ensure prompts including all buggy code patches, we form a long prompt with code contents from multiple retrieved files. However, our preliminary study shows that the code resolve rate drops significantly when the input prompt length exceeds 24K, alongside a response length of 16K. We hypothesize the suboptimal code resolve rate is due to the 32K maximum sequence length used during SFT, inherited from the 32K context window of the Qwen3-8B/14B-Base models. Thus, at RL stage, we create training data with longer prompts by mixing models' retrieved noisy files and ground-truth files.

In Table 13, we ablate training with different data which is created with various maximum prompt length (see more detail in § 4.7.1). We observe that from 16K to 32K, training with longer prompts helps improve model's repairing capability. We contribute the improvement to model's capability to prompts with longer context, which is especially important in the repairing task since during repairing, the models' need to identify and repair buggy code patches among all the retrieved code contents. However, when extending the maximum prompt length to 40K, the training shows less effective. We hypothesize that the model performs worse under such long prompts, causing the sampled trajectories to contain more noise for RL training, or that the pretrained Qwen3-14B-Base has limited long-context capability at 32K. As a result, for training data, we set the maximum prompt length to 24K and 32K for the final 8B and 14B models, respectively.

7.4. Test-Time Scaling and Patch Validation

To further improve code-repair accuracy, we employ a test-time scaling (TTS) strategy that enhances model performance by aggregating and filtering multiple candidate patches during inference. As outlined in Section §3.3.1, the model generates a diverse set of candidate repair patches and reproduction tests using temperature-based and top-p decoding. Each candidate patch is then evaluated through a patch-validation stage that applies regression and reproduction tests to identify the most reliable fixes.

For SWE-bench Verified benchmark, our TTS pipeline generates k candidate repair patches along with 40 reproduction tests per instance, then filters and ranks these candidates by first assessing how many existing regression tests each patch passes, followed by executing a curated subset of generated reproduction tests to identify the most promising repairs. The patch with the highest combined pass rate is ultimately selected, with ties resolved first by majority voting and then by minimal solution length. We refer to this ranking and selection procedure as best@k. This approach broadens the solution search space, enhances robustness by exploring multiple reasoning trajectories, and substantially increases the likelihood of producing a correct repair.



(a) Nemotron-Cascade-8B

(b) Nemotron-Cascade-14B-Thinking

Figure 12: Ablation studies on test-time-scaling (TTS) that our best@k approach and majority voting can substantially boost resolve rate on SWE-bench Verified.

Figure 12 presents results for (a) Nemotron-Cascade-8B and (b) Nemotron-Cascade-14B-Thinking evaluated on SWE-bench Verified using test-time scaling (TTS) combined with our patch-validation pipeline. As shown in Figure 12, we plot pass@k, majority@k, and best@k across $k \in 2, 4, 8, 12, 16, 24, 32$. Pass@$k$ improves monotonically with increasing k, while majority voting grows more slowly and saturates earlier. Best@k consistently outperforms majority@k by a clear margin, demonstrating the effectiveness of our patch-validation pipeline. For Nemotron-Cascade-14B-Thinking, the improvements are more pronounced across all metrics, reflecting stronger reasoning ability and greater diversity in generated repair patches. Overall, both Nemotron-Cascade-8B and Nemotron-Cascade-14B-Thinking benefit substantially from the TTS strategy, with the 14B model achieving results competitive with larger open-weight models such as DeepSWE (Luo et al., 2025) (resolve rate: 52.4% by performing TTS with execution-based verifier). These gains demonstrate that downstream filtering and validation remain powerful mechanisms for boosting patch-repair performance without modifying model weights.

As shown in Figure 12(a), Nemotron-Cascade-8B achieves a best@32 resolve rate of 43.6%, improving from 39.2% at $k = 2$ and gradually increasing as more samples are considered. With TTS and patch validation, the model reaches a pass@k score of 57.7% at $k = 32$, indicating a 15.6-point gap that reflects additional room for potential improvement toward best@32. Majority voting provides a simpler alternative but plateaus around 39–40%, showing only marginal gains as k increases. These results demonstrate that, even for a smaller model, structured test-time scaling combined with validation can substantially enhance repair accuracy. As shown in Figure 12(b), the overall metrics for Nemotron-Cascade-14B-Thinking improve more substantially. The model begins with a majority@k resolve rate of 50.7%, already surpassing the 8B variant's best@32 score of 43.6%. Under the TTS strategy, best@k provides further gains, plateauing around 53.8%. Moreover, the pass@k curve continues to rise as k increases, highlighting the considerable potential for developing more effective TTS strategies for the 14B model.

8. Related Work

In this section, we briefly review related work and position our study within the existing literature.

8.1. Reinforcement Learning for LLMs

Reinforcement learning from human feedback (RLHF) plays a critical role in further aligning large language models (LLMs) following supervised fine-tuning or instruction tuning (Bai et al., 2022; Ouyang et al., 2022). Traditionally, a pretrained reward model is utilized to be the surrogate of human judge and provide instant

reward signal in online training (e.g., Liu et al., 2024; Wang et al., 2024). In contrast to teacher-forced training in supervised fine-tuning (SFT), which requires high-quality and costly annotations, RLHF offers a more cost-effective and generalizable approach to capturing the subtleties of human intent and the nuances of linguistic expression. More recently, reward-model-based reinforcement learning has also been explored for math reasoning (e.g., Shao et al., 2024; Yang et al., 2024). However, its success has been limited due to the inherent challenges of reward modeling in the mathematical domain (Lightman et al., 2023; Wang et al., 2023; Zhang et al., 2025).

Large-scale reinforcement learning with verifiable rewards (RLVR)—which employs objective and deterministic criteria (e.g., symbolic rule-based verification for math reasoning) to provide reward signals—has achieved remarkable success in developing frontier reasoning models (e.g., Guo et al., 2025; Kimi-Team et al., 2025; Yang et al., 2025). Open RLVR recipes with publicly available datasets have been developed, such as AceReason-Nemotron (Chen et al., 2025; Liu et al., 2025), DeepScaleR (Luo et al., 2025), DeepCoder (Luo et al., 2025), DAPO (Yu et al., 2025), and Skywork-OR1 (He et al., 2025). However, such open-recipe models focus primarily on math and code reasoning, differing from the general-purpose frontier models.

The RL training of general-purpose DeepSeek-R1 and Qwen3 follows a two-stage process: an initial reasoning-oriented RL stage, followed by a second stage covering all domains. In each stage, diverse prompts are used for joint training. However, due to the substantial heterogeneity across tasks, this design complicates the RL infrastructure, training curriculum, and hyperparameter tuning, ultimately leading to suboptimal performance.

In this work, we present the Cascade RL framework and release open training recipes and datasets for developing general-purpose LLMs with strong reasoning capabilities across diverse domains, including math, coding, science, instruction following, software engineering, and general domain. In particular, we systematically investigate the interplay between RLHF and RLVR—a topic that has been underexplored in existing literature.

Various RL algorithms have been explored for both RLHF and RLVR, including PPO (Schulman et al., 2017), DPO (Rafailov et al., 2023), GRPO (Shao et al., 2024), and their variants—such as on-policy methods (e.g., AceReason-Nemotron (Chen et al., 2025)) versus off-policy approaches (e.g., clipping strategies in Su et al. (2025); Yu et al. (2025))—as well as sample-level versus token-level policy gradient losses (Yu et al., 2025). Moreover, various techniques have been proposed to enhance the stability and efficiency of RL training, such as curriculum learning with gradually increasing maximum response lengths (e.g., Luo et al., 2025) and overlength filtering to mitigate excessive penalties from truncated generations under inference budgets (Yu et al., 2025).

8.2. Supervised Fine-Tuning and Distillation

Supervised fine-tuning (SFT) serves as an indispensable preparatory stage to adapt a pretrained base LLM for general conversation, instruction following, and a variety of other tasks prior to RL-based alignment (Adler et al., 2024; Dai et al., 2024; Dubey et al., 2024; Ouyang et al., 2022). To build compact reasoning models, another approach is to distill large teacher models (DeepSeek-AI, 2025; Guo et al., 2025; Yang et al., 2025)—originally trained via RL—into smaller ones. A popular strategy is off-policy distillation (Ahmad et al., 2025; Bercovich et al., 2025; Moshkov et al., 2025; NVIDIA, 2025), in which synthetic responses or teacher outputs are first sampled from the teacher model, and the student model is then trained to predict the teacher-sampled tokens or logits. Although generating synthetic data from large teacher models is computationally expensive, such methods enable efficient training once the SFT data have been created and allow the same data to be reused for training other models. On-policy distillation, in which samples are generated from the student model, is also explored on top of off-policy distillation to reduce the performance gap to models trained with on-policy RL (Yang et al., 2025).

In general, RL is applied to SFT models to develop state-of-the-art reasoning models. In our previous work (Liu et al., 2025), we studied the synergy between SFT and RL. We found that the performance gap between initial SFT models narrows significantly during a well-designed RL process, provided that an appropriate balance

between exploration and exploitation is achieved.

8.3. Unified Reasoning Models

Building general-purpose models with strong reasoning capabilities has been a central goal of recent LLM development. Over the past year, many dedicated *thinking* models have been released, including OpenAI's o1 (OpenAI, 2024), o3, o4-mini (OpenAI, 2025), DeepSeek-R1 (Guo et al., 2025), Qwen3-Thinking (Qwen-Team, 2025), MiniMax-M1 (Chen et al., 2025), gpt-oss (Agarwal et al., 2025), and Kimi-K2-Thinking (Kimi-Team, 2025). These models emphasize deep reasoning through long chain-of-thought (CoT) generation (Yeo et al., 2025), involving problem analysis, idea sketching, enumeration of alternative solution strategies, as well as verification and correction of answers.

Several recent efforts aim to unify *instruct* and *thinking* models into a single model. Llama-Nemotron (Bercovich et al., 2025) enables global control over the thinking or instruct mode through the system prompt. Qwen3 (Yang et al., 2025), GLM-4.5 (GLM-4.5-Team et al., 2025) and DeepSeek-V3.1 (DeepSeek-AI, 2025) provide more flexible user control, allowing mode switching between *thinking* and *instruct* at each conversational turn. GPT-5 (OpenAI, 2025) employs an automatic routing mechanism that circumvents, rather than resolves, this challenge.

Appendix

A. Acknowledgments

We would like to extend our gratitude to the NVIDIA Nemo team for the valuable discussion and collaboration on building reasoning models. We especially wish to thank Boris Ginsburg, Oleksii Kuchaiev, Igor Gitman, Olivier Delalleau, Zhilin Wang, Olivier Delalleau, Banghua Zhu, Tugrul Konuk, Wei Du, Somshubra Majumdar, Siddhartha Jain, Jiaqi Zeng, Yi Dong, Alexander Bukharin, Vahid Noroozi, Khushi Bhardwaj, Sugam Dipak Devare, Jian Zhang, and Jonathan Cohen.

We thank Ying Lin for helpful discussions and useful input in building the knowledge-intensive SFT dataset. We also thank Atefeh Sohrabizadeh, Jialin Song, and Jonathan Raiman for valuable discussions on SWE-bench.

B. Benchmarks and Evaluation Setups

For knowledge reasoning tasks, we include:

- **MMLU** (Hendrycks et al., 2020) is a benchmark designed to assess an LLM's broad world knowledge and problem-solving ability. It contains 14,079 test questions across 57 subjects spanning STEM, the humanities, social sciences, and professional domains such as law and ethics. For both unified reasoning model and dedicated thinking model, we evaluate the models in *thinking* mode and, due to the large test set size, report exact match (EM) accuracy based on a single generation per question.
- **MMLU-Pro** (Wang et al., 2024) is an enhanced version of the original MMLU benchmark that mitigates model saturation by expanding to over 12,000 graduate-level questions and increasing answer choices from four to ten. We report EM accuracy in *thinking* mode using one generation per question.
- **GPQA-Diamond** (Rein et al., 2024) is a benchmark for assessing an LLM's scientific reasoning capability. It consists of the highest quality 198 GPQA questions covering graduate-level physics, biology, and chemistry. We report pass@1 accuracy in *thinking* mode, averaged over 8 generations per question (avg@8) to reduce variance.

For Nemotron-Cascade models evaluated on MMLU, MMLU-Pro, and GPQA-Diamond in *thinking* mode, we use a temperature of 0.6, a top-p value of 0.95, and a 64K-token thinking budget (maximum response length) with a YaRN scaling factor (Peng et al., 2023) of 2.

For alignment tasks, we include:

- **IFEval** (Zhou et al., 2023) is a benchmark for evaluating an LLM's instruction-following capability, focusing on verifiable instructions. It contains 541 prompts and 25 verifiable instructions, with each prompt including one or more instructions. We use *prompt strict*, which measures the percentage of prompts where all instructions are satisfied. In contrast, prior work (NVIDIA, 2025) adopts *instruct strict*, which measures the percentage of individual instructions that are satisfied. We evaluate the unified reasoning model in *non-thinking* mode, and the dedicated thinking model in *thinking* mode. We report pass@1 accuracy, using an average over 8 generations per question (avg@8).
- **IFBench** (Pyatkin et al., 2025) extends IFEval by introducing 58 new, diverse, and challenging verifiable out-of-domain instruction constraints. It provides a separate constraint list to ensure no overlap between training and test constraints, enabling evaluation of an LLM's generalization ability. The test set contains 294 prompts. We report pass@1 accuracy in *thinking* mode, averaged over 8 generations (avg@8).
- **ArenaHard 1.0** (Li et al., 2024) is a human-preference alignment benchmark consisting of 500 diverse and challenging real user prompts. It uses an automatic LLM-as-Judge approach to estimate human preferences relative to a baseline model, enabling fully automated, low-cost, and fast evaluation without human intervention. In our experiments, we report results without style control to allow for straightforward comparison with the officially reported numbers of other models. We evaluate the models in *thinking* mode, and use GPT4-Turbo-2024-0409 as the judge.

For Nemotron-Cascade models evaluated on IFEval in *non-thinking* mode, on IFBench and ArenaHard in *thinking* mode, we use a temperature of 0.6, a top-p value of 0.95, and a maximum response length of 32K tokens. For baseline models, we use officially reported results whenever available; if such results are absent, we evaluate them using their recommended inference configuration or the same settings as ours.

For math reasoning tasks, we include

- **AIME 2024** (MAA, 2024) consists of 30 problems from American Invitational Mathematics Examination at 2024.
- **AIME 2025** (MAA, 2025) consists of 30 problems from American Invitational Mathematics Examination at 2025.

For Nemotron-Cascade models on AIME 2024 and 2025, we set the thinking budget (maximum response length) to 64K tokens, the sampling temperature to 0.6, the top-p value to 0.95, and the YaRN scaling factor to 2. For baseline models, we follow their recommended inference settings with a thinking budget of at least 64K tokens.

For code generation tasks, we include

- **LiveCodeBench** (Jain et al., 2024) contains diverse algorithm coding problems with unit tests, collected from AtCoder, LeetCode platforms. We evaluate models competitive coding capability on its two subsets: LiveCodeBench v5 (2024/08/01-2025/02/01, **279** problems in total) and v6 (2024/08/01-2025/05/01, **454** problems in total). We report pass@1 accuracy in *thinking* mode, averaged over 8 generations (avg@8).
- **LiveCodeBench Pro** (Zheng et al., 2025) contains daily-updated challenging competitive coding problems with strong unit tests, collected mainly from top-tier coding contests. We report pass@1 accuracy on Easy/Med difficulty splits in *thinking* mode, averaged over 8 generations (avg@8) on two recently released subsets: 2025Q1 (2025/01-2025/04, **166** problems in total) and 2025Q2 (2025/04-2025/07, **167** problems in total).
- **SWE-bench Verified** (OpenAI, 2024) is a subset of the original test set from SWE-bench (Jimenez et al., 2023), consisting of 500 samples verified to be non-problematic by human annotators. We evaluate models in *thinking* mode and report pass@1 accuracy, averaged over 4 generations per prompt (avg@4).

For Nemotron-Cascade models evaluated on LiveCodeBench (v5/v6) and LiveCodeBench Pro, we use a 64K-

token thinking budget, a sampling temperature of 0.6, a top-p of 0.95, and a YaRN scaling factor of 2. We evaluate baseline models with their recommended inference configurations, ensuring a thinking budget of at least 64K tokens. For SWE-bench Verified, we use 32K-token thinking budget and a sampling temperature of 0.6. We set maximum input prompt length to 32K and 64K tokens, and set YaRN scaling factor of 2 and 3 for 8B and 14B models, respectively.

C. Prompt Templates

C.1. Unpreferrable Response Generation for RM data

Step 1: Generate offtopic prompts

Given an user input (called "given input"), please generate a new user input (called "generated input") such that:

(1) The generated input is highly relevant to but different from the given input.
(2) The correct response to the generated input superficially resembles the correct response to the given input as much as possible.
(3) But actually, the correct response to the generated input should not be a correct response to the given input.

Given input:
`{instruction}`

Generated input:

Step 2: Judge if the offtopic prompts are really different to the original

There are two instructions, Instruction A and Instruction B. Are the two instructions asking the same thing? Please answer in ‘YES‘ or ‘NO‘.
Instruction A:
`{instruction A}`

Instruction B:
`{instruction B}`

C.2. Prompts and Templates for SWE Task

Code Localization

Please look through a given GitHub issue and repository structure and provide a list of files that one would need to edit or look at to solve the issue.

GitHub Problem Description
`{problem_statement}`
###

Repository Structure
`{structure}`

```
###

Below are some code segments, each from a relevant file. One or more of these files may contain bugs.
Only provide the full path and return at most n files. The returned files should be separated by new
lines ordered by most to least important and wrapped with ```. For example:

```
most/important/file1.xx
less/important/file2.yy
least/important/file3.zz
```
```

Code Repair

We are currently solving the following issue within our repository. Here is the issue text:
— BEGIN ISSUE —
`{problem_statement}`
— END ISSUE —
Below are some code segments, each from a relevant file. One or more of these files may contain bugs.
— BEGIN FILE —

```
{content}
```

— END FILE —
Please first localize the bug based on the issue statement, and then generate **SEARCH/REPLACE** edits to fix the issue.
Every **SEARCH/REPLACE** edit must use this format:

1. Start with "'`diff\n` to indicate a diff block, and end the whole block with "'.
2. The file path
3. The start of search block: `<<<<< SEARCH`
4. A contiguous chunk of lines to search for in the existing source code
5. The dividing line: `=======`
6. The lines to replace into the source code
7. The end of the replace block: `>>>>> REPLACE`

Here is an example:

```
```diff
### mathweb/flask/app.py
<<<<<<< SEARCH
from flask import Flask
=======
import math
from flask import Flask
>>>>>>> REPLACE
```
```

Please note that the **SEARCH/REPLACE** edit REQUIRES PROPER INDENTATION. If you would like to add the line `' print(x)'`, you must fully write that out, with all those spaces before the code!
Wrap each **SEARCH/REPLACE** edit in a code block as shown in the example above. If you have multiple **SEARCH/REPLACE** edits, use a separate code block for each one.

Output format requirement: Please put your reasoning tokens in a separate code block, starting with `<think>` and ending with `</think>`, and the solution tokens in a separate code block, starting with `<solution>` and ending with `</solution>`.

Test Code Generation

We are currently solving the following issue within our repository. Here is the issue text:
— BEGIN ISSUE —
`{problem_statement}`
— END ISSUE —
Several candidate repair patches have been generated to address this issue. You must carefully examine them and select the one that best matches the issue description when creating the test so that it specifically validates the behavior before and after applying the patch:
— BEGIN PATCH —

```
{model_patch}
```

— END PATCH —
Below are some code segments, each from a relevant file. One or more of these files may contain bugs.
— BEGIN FILE —

```
{content}
```

— END FILE —
Please generate a complete test that can be used to reproduce the issue.
The complete test should contain the following:

1. Includes all necessary imports
2. Reproduces the issue described in the issue text before the patch is applied
3. Exercises the exact functions, classes, or lines modified in the repair patch
4. Contains assertions or checks that confirm the issue is reproduced without the patch
5. Contains assertions or checks that confirm the issue is resolved after the patch is applied
6. Uses meaningful assertions tied to the patch changes (e.g., expected outputs, raised exceptions, or altered return values)
7. Print "Issue reproduced" if the outcome indicates that the issue is reproduced
8. Print "Issue resolved" if the outcome indicates that the issue has been successfully resolved
9. Print "Other issues" if the outcome indicates there are other issues with the source code

The test should not be generic; it must directly validate the correctness of the patch.
Here is an example:

````
```python
from sqlfluff import lint

def test__rules__std_L060_raised() -> None:
    try:
        sql = "SELECT   IFNULL(NULL, 100),
            NVL(NULL,100);"
        result = lint(sql, rules=["L060"])
        assert len(result) == 2
    except:
        print("Other issues")
        return
````
````

```
    try:
        assert result[0]["description"] == "Use 'COALESCE' instead of 'IFNULL'."
        assert result[1]["description"] == "Use 'COALESCE' instead of 'NVL'."
        print("Issue resolved")
    except AssertionError:
        print("Issue reproduced")
        return

    return

test_rules_std_L060_raised()
```

Reward Modeling

System Prompt
You are an expert judge evaluating AI assistant interactions. Your task is to determine if the assistant successfully resolved the user's request given a reference golden solution.

Key evaluation criteria:
1. Did the assistant complete the main task requested by the user?
2. Were there any errors or issues in the final solution?

Respond only with "<judgement>YES</judgement>" or "<judgement>NO</judgement>".

User Prompt
We are currently solving the following issue within our repository. Here is the issue text:
— BEGIN ISSUE —
`{problem_statement}`
— END ISSUE —
Below are some code segments, each from a relevant file. One or more of these files may contain bugs.
— BEGIN FILE —

```
{content}
```

— END FILE —
Please first localize the bug based on the issue statement, and then generate **SEARCH/REPLACE** edits to fix the issue.
Every **SEARCH/REPLACE** edit must use this format:
1. Start with ```` ```diff\n ```` to indicate a diff block, and end the whole block with ```` ``` ````.
2. The file path
3. The start of search block: `<<<<< SEARCH`
4. A contiguous chunk of lines to search for in the existing source code
5. The dividing line: `=======`
6. The lines to replace into the source code
7. The end of the replace block: `>>>>> REPLACE`

Here is the reference golden git diff solution:

`{golden_patch}`

Here is the solution from the assistance:

```
{model_patch}
```

Please compare the assistance's solution to the reference golden git diff solution and judge whether the assistance's solution successfully resolve the issue. Note that the solution is not required to be exactly the same as reference golden solution. Use your own knowledge to judge whether the assistance's solution successfully resolve the issue. Respond with "<judgement>YES</judgement>" or "<judgement>NO</judgement>".

C.3. Prompt Templates for Test-Time Scaling on IOI 2025

Write Python code to solve the problem. Please place the solution code in the following format:
```python
# Your solution code here
```
```
{problem_statement}
```
Below you are provided the accepted correct solutions but with different input constraints. You may use them as a reference for your insights.
==========
Different Constraints (for reference only):
```
{subtask_constraints}
```
Accepted Code:
```
[CODE]
```
==========
Different Constraints (for reference only):
...
==========
From here, you are also given your submission history containing **incorrect** code and their corresponding official judgement verdicts as reference – Official judgement verdicts and problem statement/-conditions are 100% reliable. You should make improvements from them if they could help:
==========
Incorrect Code
```
[CODE]
```
Judgement Verdict: `[VERDICT]`, Score: `[SCORE]`
==========
Incorrect Code
...
==========

D. Training Hyperparameters

D.1. Multi-Stage SFT

We list the hyperparameters for the multi-stage SFT of the 8B and 14B models in Table 14.

D.2. RLHF

We present the RLHF hyperparameters for the 8B and 14B models in Table 15.

Table 14: Training hyperparameters of 8B and 14B models in **multi-stage SFT**. Both the *unified* and the *thinking* models share the same hyperparameters.

Hyperparameters	8B (Stage-1 / Stage-2)	14B (Stage-1 / Stage-2)
Global batch size	256	256
Max learning rate	$5e^{-5}$ / $2e^{-5}$	$5e^{-5}$
Min learning rate	0	0
Scheduler	cosine	cosine
Max Steps	100K	100K
Optimizer	AdamW	AdamW
Optimizer config	$\beta_1 = 0.9$, $\beta_2 = 0.95$	$\beta_1 = 0.9$, $\beta_2 = 0.95$
Weight decay	$1e^{-4}$	$1e^{-4}$
# of training steps	20K / 32K	22K / 41K

Table 15: Training hyperparameters of 8B/14B models in **RLHF**. Both the *unified* and *thinking* models share the same hyperparameters. *Unified* models are trained in both *non-thinking* and *thinking* modes, with an equal split of prompts allocated to each mode.

Hyperparameters	8B	14B
Max response length	12K	12K
Batch size	256	256
# Rollout size	8	8
Learning rate	$2e^{-6}$	$2e^{-6}$
Steps	800	900
Optimizer	AdamW	AdamW
Optimizer config	$\beta_1 = 0.9$, $\beta_2 = 0.95$	$\beta_1 = 0.9$, $\beta_2 = 0.95$
Temperature	0.6	0.6
Top-p	0.95	0.95
Overlong filtering	False	False

D.3. IF-RL

The hyperparameters of 8B and 14B models in **IF-RL** training are in Table 16.

Table 16: Training hyperparameters of 8B/14B models in **IF-RL**. *Unified* models are trained in the *non-thinking* mode.

Hyperparameters	8B (unified)	8B-Thinking	14B-Thinking
Max response length (stage 1)	8K	8K	8K
Max response length (stage 2)	8K	16K	16K
Batch size	256	256	256
# Rollout size	8	8	8
Learning rate	$2e^{-6}$	$2e^{-6}$	$2e^{-6}$
Steps (stage 1)	2300	550	800
Steps (stage 2)	800	300	120
Optimizer	AdamW	AdamW	AdamW
Optimizer config	$\beta_1 = 0.9, \beta_2 = 0.95$	$\beta_1 = 0.9, \beta_2 = 0.95$	$\beta_1 = 0.9, \beta_2 = 0.95$
Temperature	0.6	0.6	0.6
Top-p	0.95	0.95	0.95
Overlong filtering (stage 1)	False	False	False
Overlong filtering (stage 2)	False	True	True

D.4. Math RL

The hyperparameters for the 8B and 14B models used in **Math RL** training are listed in Table 17 and Table 18, respectively.

Table 17: Training hyperparameters of our 8B models in **Math RL**. Both the *unified* and *thinking* models share the same hyperparameters. Unified models are trained in the *thinking* mode.

Hyper-parameters	8B (Stage-1)	8B (Stage-2)	8B (Stage-3)
Max response length	24K	32K	40K
Batch size	128	128	128
# Rollout size	8	8	8
Learning rate	$2e^{-6}$	$2e^{-6}$	$2e^{-6}$
Steps (start-end)	0-190	190-430	430-500
Optimizer	AdamW	AdamW	AdamW
Optimizer config	$\beta_1 = 0.9, \beta_2 = 0.95$	$\beta_1 = 0.9, \beta_2 = 0.95$	$\beta_1 = 0.9, \beta_2 = 0.95$
Temperature	1	1	0.8
Top-p	0.95	0.95	0.95
Overlong filtering	True	False	False

D.5. Code RL

The hyperparameters of 8B-Thinking, 8B unified and 14B-Thinking models in **Code RL** are in Table 19.

Table 18: Training hyperparameters of 14B-Thinking model in **Math RL**.

Hyper-parameters	14B (Stage-1)	14B (Stage-2)
Max response length	28K	40K
Batch size	128	128
# Rollout size	8	8
Learning rate	$2.5e^{-6}$	$2.5e^{-6}$
Steps (start-end)	0-120	120-220
Optimizer	AdamW	AdamW
Optimizer config	$\beta_1 = 0.9$, $\beta_2 = 0.95$	$\beta_1 = 0.9$, $\beta_2 = 0.95$
Temperature	1.2	1.1
Top-p	0.95	0.95
Overlong filtering	True	False

Table 19: Training hyperparameters of 8B-Thinking, 8B unified, and 14B-Thinking models in **Code RL**. 8B unified model are trained in the *thinking* mode.

Hyper-parameters	8B-Thinking	8B (unified)	14B-Thinking
Max response length	44k	44K $\rightarrow$ 48K	56K
Batch size	128	128	128
# Rollout size	8	8	8
Learning rate	$4e^{-6}$	$4e^{-6}$	$4e^{-6}$
Steps	64	90	64
Optimizer	AdamW	AdamW	AdamW
Optimizer config	$\beta_1 = 0.9$, $\beta_2 = 0.95$	$\beta_1 = 0.9$, $\beta_2 = 0.95$	$\beta_1 = 0.9$, $\beta_2 = 0.95$
Temperature	1.2	1.0 $\rightarrow$ 0.8	1.0
Top-p	0.95	0.95	0.95
Overlong filtering	False	False	False

D.6. SWE RL

The hyperparameters for the 8B unified, 8B-Thinking, and 14B-Thinking models used in **SWE RL** training are listed in Table 20.

Table 20: Training hyperparameters of 8B unified, 8B-Thinking, and 14B-Thinking models in **SWE RL**. Both the 8B *unified* and Thinking models share the same hyperparameters. Unified models are trained in the *thinking* mode.

Hyper-parameters	8B (Stage-1)	8B (Stage-2)	14B (single stage)
Input response length	16K	24K	32K
Max response length	16K	16K	16K
Batch size	128	128	128
# Rollout size	16	16	16
Learning rate	$2.5e^{-6}$	$2.5e^{-6}$	$2.5e^{-6}$
Steps (start-end)	0-30	30-60	0-120
Optimizer	AdamW	AdamW	AdamW
Optimizer config	$\beta_1 = 0.9$, $\beta_2 = 0.95$	$\beta_1 = 0.9$, $\beta_2 = 0.95$	$\beta_1 = 0.9$, $\beta_2 = 0.95$
Temperature	1	1	1
Top-p	0.95	0.95	0.95
Overlong filtering	True	False	True

E. ELO Rating Analysis

In this section, we present the details of the reported Codeforces Elo ratings for the Nemotron-Cascade-8B and Nemotron-Cascade-14B-Thinking models, based on **51** recent Codeforces contests held between 2501–2507. Problems and evaluations are provided by LiveCodeBench Pro (Zheng et al., 2025). For each contest, we simulate participation by allowing the model up to N independent submissions per problem (with N set to 8 by default) and generate the model's responses using a temperature of 0.6, `top-p` of 0.95, and a maximum token budget of 128K. Let k denote the number of correct submissions among these N attempts, and $N - k$ the number of incorrect submissions ($0 \leq k \leq N$). In a real contest, submissions are made sequentially and the penalty submission counts is defined by the number of incorrect submissions prior to the first correct submission. To estimate the submission penalty, we assume the ordering of the k correct and $N - k$ incorrect submissions are uniformly distributed over $\binom{N}{k}$ permutations and the expected number of penalties can be derived as:

$$\mathbb{E}[\#\text{ of penalties}] = \frac{N-k}{k+1}$$

We adopt the standard codeforces contest rules: for regular codeforces round, we apply score penalty as 50 for each expected penalty, and for ICPC style round (e.g. educational rounds, Div.3 rounds), we add time penalty as 10 for each incorrect instead. Penalties on problems which remained unsolved will not take into consideration. We rank our model's contest performance against n real human participants as m ($1 \leq m \leq n+1$) based on the final score, and compute implied performance rating R_{model} following standard Elo rating definition (Glickman and Jones, 1999; Quan et al., 2025) by solving:

$$m = \sum_{i=1}^{n} \frac{1}{1 + 10^{(R_{\text{model}} - R_i)/400}}$$

where R_i refers to the Elo rating of human contestant i before each contest. We report the averaged performance

rating over 51 codeforces rounds as our final Elo score and present the performance details of our Nemotron-Cascade-8B and Nemotron-Cascade-14B-Thinking model in Table 21 and Table 22, respectively.

We observe large variance in the model's estimated performance rating across contests. For instances, the Nemotron-Cascade-14B-Thinking model achieves the estimated performance rating above 2600 on Codeforces Round 1015, yet fails to solve any problems (even with 8 attempts) and receives Elo rating below 1000 on Round 1024 Div.1. We also find inconsistent behavior on coding problem solving: while the model is sometimes able to solve very difficult problems, it can also become stuck on relatively easy ones, even within the same contest. Furthermore, the model tends to perform well on problems solvable by standard techniques, heavy implementation, or straightforward intuition, but often struggles on problems that require hypothesis-driven exploration on small-scale data or ad hoc ideas, such as constructive or interactive problems. It could be an interesting direction for understanding and improving such reasoning abilities in the future.

Table 21: Nemotron-Cascade-8B performance details on 51 Codeforces Rounds ranging from 2501-2507. We attempt each problem with $N = 8$ times in total. For regular codeforces rounds, we present the score after considering expected penalties for each problem. For ICPC style rounds, we mark passed/failed problems as + and - correspondingly. We compute the estimated rank to human contestants and the corresponding Elo score as shown in rightmost two columns.

Contest Name	Contest Problems											Score	Penalty	Est. Rank	ELO
Hello 2025	A 493.75	B 1000.00	C 1400.00	D 0.0	E1 0.0	E2 0.0	F 0.0	G 0.0	H 0.0			2893.75	-	1695/16703	1942
Codeforces Round 996 (Div. 2)	A 460.00	B 985.71	C 1485.71	D 0.0	E 0.0	F 0.0						2931.43	-	749/21232	1979
Codeforces Round 997 (Div. 2)	A 0.0	B 1250.00	C 1485.71	D 0.0	E 0.0	F1 0.0	F2 0.0					2735.71	-	2025/18823	1695
Codeforces Round 998 (Div. 3)	A +	B +	C +	D +	E +	F +	G -					6	23.00	22/24247	1699
IAEPC Preliminary Contest (Codeforces Round 999, Div. 1 + Div. 2)	A 500.00	B 937.50	C 1493.75	D 1325.00	E 0.0	F1 0.0	F2 0.0	G 0.0	H1 0.0	H2 0.0	I 0.0	4256.25	-	1200/12647	1988
Codeforces Round 1000 (Div. 2)	A 500.00	B 985.71	C 1500.00	D 2243.75	E 2575.00	F1 0.0	F2 0.0					7804.46	-	14/17169	2200
Ethflow Round 1 (Codeforces Round 1001, Div. 1 + Div. 2)	A 500.00	B 900.00	C 937.50	D 0.0	E1 0.0	E2 0.0	F 0.0	G 0.0	H 0.0			2337.50	-	2336/16234	1771
Codeforces Round 1002 (Div. 2)	A 500.00	B 825.00	C 0.0	D 0.0	E1 0.0	E2 0.0						1325.00	-	4240/19443	1454
Codeforces Round 1003 (Div. 4)	A +	B +	C1 +	C2 +	D +	E +	F +	G +	H +			9	45.86	1/28033	1522
Codeforces Round 1004 (Div. 1)	A 0.0	B 650.00	C 1075.00	D1 650.00	D2 0.0	E 0.0	F 0.0					2375.00	-	252/1030	2482
Codeforces Round 1004 (Div. 2)	A 500.00	B 0.0	C 0.0	D 0.0	E 1650.00	F 2075.00	G 0.0					4225.00	-	174/16749	2098
Codeforces Round 1005 (Div. 2)	A 0.0	B 900.00	C 0.0	D 0.0	E 0.0	F 0.0						900.00	-	9718/17621	1021
Educational Codeforces Round 174 (Rated for Div. 2)	A +	B +	C +	D -	E -	F -						3	4.11	1298/16701	1810
Codeforces Round 1006 (Div. 3)	A +	B +	C +	D +	E +	F +	G +					7	21.25	1/24140	1699
Educational Codeforces Round 175 (Rated for Div. 2)	A +	B +	C +	D +	E -	F -						4	2.86	234/16060	2195
Codeforces Round 1007 (Div. 2)	A 500.00	B 937.50	C 0.0	D1 1725.00	D2 0.0	E 2325.00	F 0.0					5487.50	-	4/16254	2198
Codeforces Round 1008 (Div. 1)	A 437.50	B 0.0	C 1493.75	D 0.0	E 0.0	F 0.0	G 0.0					1931.25	-	371/909	2294
Codeforces Round 1008 (Div. 2)	A 500.00	B 725.00	C 1187.50	D 1650.00	E 0.0	F 2743.75	G 0.0					6806.25	-	9/14641	2008
Codeforces Round 1009 (Div. 3)	A +	B +	C +	D +	E -	F -	G +					5	22.86	178/23635	1708
Educational Codeforces Round 176 (Rated for Div. 2)	A +	B -	C +	D +	E +	F -						4	41.25	73/18159	2198
Codeforces Round 1011 (Div. 2)	A 500.00	B 1187.50	C 1075.00	D 0.0	E 0.0	F1 1825.00	F2 0.0					4587.50	-	170/15906	2200
Codeforces Round 1012 (Div. 1)	A 0.0	B1 0.0	B2 0.0	C1 0.0	C2 0.0	D 0.0	E 0.0					0	-	653/653	977
Codeforces Round 1012 (Div. 2)	A 500.00	B 960.00	C 0.0	D 0.0	E1 0.0	E2 0.0	F1 0.0	F2 0.0				1460.00	-	2197/8536	1466
Codeforces Round 1013 (Div. 3)	A +	B +	C +	D +	E +	F -	G -					5	0.00	852/24379	1715
Codeforces Round 1014 (Div. 2)	A 500.00	B 750.00	C 1250.00	D 1650.00	E 2250.00	F 0.0						6400.00	-	2/15842	2213
Teza Round 1 (Codeforces Round 1015, Div. 1 + Div. 2)	A 750.00	B 1000.00	C 1485.71	D 1650.00	E 0.0	F 0.0	G1 0.0	G2 0.0	H 0.0			4885.71	-	489/11206	2320
Codeforces Round 1016 (Div. 3)	A +	B +	C +	D +	E +	F +	G -					6	36.25	67/21249	1699
Codeforces Round 1017 (Div. 4)	A +	B +	C +	D +	E +	F +	G +	H +				8	52.50	1/22234	1503
Neowise Labs Contest 1 (Codeforces Round 1018, Div. 1 + Div. 2)	A 485.71	B 743.75	C 1493.75	D 0.0	E 0.0	F 0.0	G 0.0	H 0.0				2723.21	-	1382/12771	1929
Codeforces Round 1019 (Div. 2)	A 500.00	B 993.75	C 1460.00	D 0.0	E 0.0	F 0.0						2953.75	-	849/14465	1903
Codeforces Round 1020 (Div. 3)	A +	B +	C +	D +	E +	F -	G1 -	G2 -				5	43.00	311/17451	1708
Codeforces Round 1021 (Div. 1)	A 0.0	B 0.0	C 0.0	D 0.0	E 0.0	F 0.0						0	-	651/651	982
Codeforces Round 1021 (Div. 2)	A 500.00	B 1235.71	C 0.0	D 0.0	E 0.0	F 0.0						1735.71	-	1799/5824	1431
Educational Codeforces Round 178 (Rated for Div. 2)	A +	B +	C +	D +	E +	F -	G -					5	50.36	76/11706	2215
Codeforces Round 1022 (Div. 2)	A 500.00	B 0.0	C 1325.00	D 0.0	E 0.0	F 0.0						1825.00	-	3110/11127	1454
Codeforces Round 1023 (Div. 2)	A 250.00	B 743.75	C 0.0	D 0.0	E 0.0	F1 0.0	F2 0.0					993.75	-	2848/11636	1485
Codeforces Round 1024 (Div. 1)	A 0.0	B 0.0	C 0.0	D 0.0	E 0.0	F 0.0						0	-	857/857	938
Codeforces Round 1024 (Div. 2)	A 250.00	B 500.00	C 0.0	D 0.0	E 0.0	F 0.0						750.00	-	4640/11201	1246
Codeforces Round 1025 (Div. 2)	A 500.00	B 985.71	C1 0.0	C2 0.0	C3 0.0	D 0.0	E 2460.00	F 0.0				3945.71	-	270/15945	2156
Codeforces Round 1026 (Div. 2)	A 500.00	B 743.75	C 1437.50	D 0.0	E 2075.00	F 2825.00						7581.25	-	15/17668	2198
Codeforces Round 1027 (Div. 3)	A +	B +	C +	D +	E +	F +	G -					6	9.11	12/22295	1709
Codeforces Round 1028 (Div. 1)	A 400.00	B 0.0	C 0.0	D 0.0	E 0.0	F1 0.0	F2 0.0					400.00	-	803/956	1840
Codeforces Round 1028 (Div. 2)	A 400.00	B 750.00	C 1150.00	D 0.0	E 0.0	F 0.0						2300.00	-	339/18314	2018
Educational Codeforces Round 179 (Rated for Div. 2)	A +	B +	C +	D -	E -	F -	G -					3	37.86	4100/12301	1371
Codeforces Round 1029 (Div. 3)	A +	B +	C +	D +	E -	F -	G +	H -				5	15.36	460/20324	1707
Codeforces Round 1030 (Div. 2)	A 500.00	B 0.0	C 825.00	D1 1243.75	D2 0.0	E 0.0	F 0.0					2568.75	-	1960/18335	1715
Codeforces Round 1031 (Div. 2)	A 493.75	B 0.0	C 0.0	D 0.0	E 0.0	F 0.0						493.75	-	5469/11032	1134
Codeforces Round 1032 (Div. 3)	A +	B +	C +	D +	E +	F +	G +	H -				7	20.00	17/22170	1733
Codeforces Round 1033 (Div. 2) and CodeNite 2025	A 500.00	B 750.00	C 1187.50	D 0.0	E 2325.00	F 0.0	G 0.0					4762.50	-	183/12948	2216
Educational Codeforces Round 180 (Rated for Div. 2)	A +	B +	C +	D -	E +	F -						4	37.50	345/17128	2114
Codeforces Round 1035 (Div. 2)	A 500.00	B 1000.00	C 1485.71	D 0.0	E 0.0	F 0.0						2985.71	-	587/15624	2008

Table 22: Nemotron-Cascade-14B-Thinking performance details on 51 Codeforces Rounds ranging from 2501-2507. We attempt each problem with $N = 8$ times in total. For regular codeforces rounds, we present the score after considering expected penalties for each problem. For ICPC style rounds, we mark passed/failed problems as + and - correspondingly. We compute the estimated rank to human contestants and the corresponding Elo score as shown in rightmost two columns.

Contest Name	Contest Problems											Score	Penalty	Est. Rank	ELO
Hello 2025	A	B	C	D	E1	E2	F	G	H			4975.00	-	679/16703	2290
	500.00	1000.00	1400.00	2075.00	0.0	0.0	0.0	0.0	0.0						
Codeforces Round 996 (Div. 2)	A	B	C	D	E	F						2887.50	-	755/21232	1977
	475.00	937.50	1475.00	0.0	0.0	0.0									
Codeforces Round 997 (Div. 2)	A	B	C	D	E	F1	F2					5110.71	-	41/18823	2198
	437.50	1250.00	1485.71	1937.50	0.0	0.0	0.0								
Codeforces Round 998 (Div. 3)	A	B	C	D	E	F	G					6	30.86	22/24247	1699
	+	+	+	+	+	+	-								
IAEPC Preliminary Contest (Codeforces Round 999, Div. 1 + Div. 2)	A	B	C	D	E	F1	F2	G	H1	H2	I	4397.50	-	1170/12647	1998
	500.00	937.50	1500.00	1460.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0				
Codeforces Round 1000 (Div. 2)	A	B	C	D	E	F1	F2					6825.00	-	39/17169	2200
	500.00	0.0	1500.00	2250.00	2575.00	0.0	0.0								
Ethflow Round 1 (Codeforces Round 1001, Div. 1 + Div. 2)	A	B	C	D	E1	E2	F	G	H			2445.71	-	1737/16234	1895
	500.00	985.71	960.00	0.0	0.0	0.0	0.0	0.0	0.0						
Codeforces Round 1002 (Div. 2)	A	B	C	D	E1	E2						1325.00	-	4240/19443	1454
	500.00	825.00	0.0	0.0	0.0	0.0									
Codeforces Round 1003 (Div. 4)	A	B	C1	C2	D	E	F	G	H			9	6.61	1/28033	1522
	+	+	+	+	+	+	+	+	+						
Codeforces Round 1004 (Div. 1)	A	B	C	D1	D2	E	F					1950.00	-	373/1030	2339
	0.0	0.0	1225.00	725.00	0.0	0.0	0.0								
Codeforces Round 1004 (Div. 2)	A	B	C	D	E	F	G					3625.00	-	282/16749	2087
	500.00	900.00	0.0	0.0	0.0	2225.00	0.0								
Codeforces Round 1005 (Div. 2)	A	B	C	D	E	F						2543.75	-	1433/17621	1806
	493.75	975.00	1075.00	0.0	0.0	0.0									
Educational Codeforces Round 174 (Rated for Div. 2)	A	B	C	D	E	F						3	0.00	1298/16701	1810
	+	+	+	-	-	-									
Codeforces Round 1006 (Div. 3)	A	B	C	D	E	F	G					7	12.11	1/24140	1699
	+	+	+	+	+	+	+								
Educational Codeforces Round 175 (Rated for Div. 2)	A	B	C	D	E	F						4	0.00	234/16060	2195
	+	+	+	+	-	-									
Codeforces Round 1007 (Div. 2)	A	B	C	D1	D2	E	F					6910.71	-	2/16254	2198
	485.71	975.00	1325.00	1725.00	0.0	2400.00	0.0								
Codeforces Round 1008 (Div. 1)	A	B	C	D	E	F	G					1985.71	-	359/909	2307
	485.71	0.0	1500.00	0.0	0.0	0.0	0.0								
Codeforces Round 1008 (Div. 2)	A	B	C	D	E	F	G					6885.71	-	8/14641	2008
	500.00	750.00	1235.71	1650.00	0.0	2750.00	0.0								
Codeforces Round 1009 (Div. 3)	A	B	C	D	E	F	G					6	44.25	15/23635	1708
	+	+	+	+	-	+	+								
Educational Codeforces Round 176 (Rated for Div. 2)	A	B	C	D	E	F						4	29.25	73/18159	2198
	+	-	+	+	+	-									
Codeforces Round 1011 (Div. 2)	A	B	C	D	E	F1	F2					8670.71	-	1/15906	2200
	500.00	1075.00	1210.00	1735.71	2325.00	1825.00	0.0								
Codeforces Round 1012 (Div. 1)	A	B1	B2	C1	C2	D	E					1650.00	-	365/653	2135
	0.0	825.00	0.0	825.00	0.0	0.0	0.0								
Codeforces Round 1012 (Div. 2)	A	B	C	D	E1	E2	F1	F2				6737.50	-	12/8536	2007
	500.00	937.50	1650.00	0.0	1825.00	0.0	1825.00	0.0							
Codeforces Round 1013 (Div. 3)	A	B	C	D	E	F	G					6	6.96	22/24379	1715
	+	+	+	+	+	+	-								
Codeforces Round 1014 (Div. 2)	A	B	C	D	E	F						6441.25	-	2/15842	2213
	500.00	743.75	1243.75	1710.00	2243.75	0.0									
Teza Round 1 (Codeforces Round 1015, Div. 1 + Div. 2)	A	B	C	D	E	F	G1	G2	H			7037.50	-	186/11206	2631
	743.75	1000.00	1493.75	1725.00	2075.00	0.0	0.0	0.0	0.0						
Codeforces Round 1016 (Div. 3)	A	B	C	D	E	F	G					7	52.50	1/21249	1699
	+	+	+	+	+	+	+								
Codeforces Round 1017 (Div. 4)	A	B	C	D	E	F	G	H				8	24.00	1/22234	1503
	+	+	+	+	+	+	+	+							
Neowise Labs Contest 1 (Codeforces Round 1018, Div. 1 + Div. 2)	A	B	C	D	E	F	G	H				4398.21	-	493/12771	2312
	485.71	750.00	1475.00	1687.50	0.0	0.0	0.0	0.0							
Codeforces Round 1019 (Div. 2)	A	B	C	D	E	F						2931.25	-	852/14465	1902
	500.00	993.75	1437.50	0.0	0.0	0.0									
Codeforces Round 1020 (Div. 3)	A	B	C	D	E	F	G1	G2				6	25.00	54/17451	1708
	+	+	+	+	+	+	-	-							
Codeforces Round 1021 (Div. 2)	A	B	C	D	E	F						3068.75	-	162/5824	2019
	500.00	1243.75	1325.00	0.0	0.0	0.0									
Codeforces Round 1021 (Div. 1)	A	B	C	D	E	F						325.00	-	625/651	1568
	325.00	0.0	0.0	0.0	0.0	0.0									
Educational Codeforces Round 178 (Rated for Div. 2)	A	B	C	D	E	F	G					4	2.86	1810/11706	1661
	+	+	-	+	+	-	-								
Codeforces Round 1022 (Div. 2)	A	B	C	D	E	F						1900.00	-	3073/11127	1459
	500.00	0.0	1400.00	0.0	0.0	0.0									
Codeforces Round 1023 (Div. 2)	A	B	C	D	E	F1	F2					4337.50	-	79/11636	2209
	250.00	750.00	1437.50	1900.00	0.0	0.0	0.0								
Codeforces Round 1024 (Div. 1)	A	B	C	D	E	F						0	-	857/857	938
	0.0	0.0	0.0	0.0	0.0	0.0									
Codeforces Round 1024 (Div. 2)	A	B	C	D	E	F						750.00	-	4640/11201	1246
	250.00	500.00	0.0	0.0	0.0	0.0									
Codeforces Round 1025 (Div. 2)	A	B	C1	C2	C3	D	E	F				3885.71	-	304/15945	2131
	500.00	985.71	0.0	0.0	0.0	0.0	2400.00	0.0							
Codeforces Round 1026 (Div. 2)	A	B	C	D	E	F						7725.00	-	10/17668	2198
	500.00	750.00	1500.00	0.0	2150.00	2825.00									
Codeforces Round 1027 (Div. 3)	A	B	C	D	E	F	G					6	28.00	12/22295	1709
	+	+	+	+	+	+	-								
Codeforces Round 1028 (Div. 2)	A	B	C	D	E	F						4610.00	-	5/18314	2018
	325.00	750.00	1210.00	0.0	2325.00	0.0									
Codeforces Round 1028 (Div. 1)	A	B	C	D	E	F1	F2					2035.00	-	135/956	2673
	460.00	0.0	1575.00	0.0	0.0	0.0	0.0								
Educational Codeforces Round 179 (Rated for Div. 2)	A	B	C	D	E	F	G					4	76.25	998/12301	1848
	+	+	+	+	-	-	-								
Codeforces Round 1029 (Div. 3)	A	B	C	D	E	F	G	H				5	32.50	460/20324	1707
	+	+	+	+	-	-	+	-							
Codeforces Round 1030 (Div. 2)	A	B	C	D1	D2	E	F					5025.00	-	35/18335	2205
	500.00	0.0	900.00	1187.50	0.0	2437.50	0.0								
Codeforces Round 1031 (Div. 2)	A	B	C	D	E	F						500.00	-	5433/11032	1138
	500.00	0.0	0.0	0.0	0.0	0.0									
Codeforces Round 1032 (Div. 3)	A	B	C	D	E	F	G	H				7	3.75	17/22170	1733
	+	+	+	+	+	+	+	-							
Codeforces Round 1033 (Div. 2) and CodeNite 2025	A	B	C	D	E	F	G					6481.25	-	31/12948	2216
	493.75	750.00	1225.00	1575.00	2437.50	0.0	0.0								
Educational Codeforces Round 180 (Rated for Div. 2)	A	B	C	D	E	F						5	43.00	8/17128	2253
	+	+	+	+	+	-									
Codeforces Round 1035 (Div. 2)	A	B	C	D	E	F						2985.71	-	587/15624	2008
	500.00	1000.00	1485.71	0.0	0.0	0.0									

References

[1] Bo Adler, Niket Agarwal, Ashwath Aithal, Dong H Anh, Pallab Bhattacharya, Annika Brundyn, Jared Casper, Bryan Catanzaro, Sharon Clay, Jonathan Cohen, et al. Nemotron-4 340b technical report. *arXiv preprint arXiv:2406.11704*, 2024. 38

[2] Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025. 39

[3] Wasi Uddin Ahmad, Sean Narenthiran, Somshubra Majumdar, Aleksander Ficek, Siddhartha Jain, Jocelyn Huang, Vahid Noroozi, and Boris Ginsburg. Opencodereasoning: Advancing data distillation for competitive coding. *arXiv preprint arXiv:2504.01943*, 2025. 10, 28, 38

[4] Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, et al. SmolLM2: When Smol Goes Big–Data-Centric Training of a Small Language Model. *arXiv preprint arXiv:2502.02737*, 2025. 9

[5] Sam Altman. Change of Plans, 2025. URL https://x.com/sama/status/1908167621624856998. 5

[6] Anthropic. Claude Code, 2025. 11

[7] Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. Swe-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. *arXiv preprint arXiv:2505.20411*, 2025. 12

[8] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022. 37

[9] Elie Bakouch, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, Lewis Tunstall, Carlos Miguel Patiño, Edward Beeching, Aymeric Roucher, Aksel Joonas Reedi, Quentin Gallouédec, Kashif Rasul, Nathan Habib, Clémentine Fourrier, Hynek Kydlicek, Guilherme Penedo, Hugo Larcher, Mathieu Morlon, Vaibhav Srivastav, Joshua Lochner, Xuan-Son Nguyen, Colin Raffel, Leandro von Werra, and Thomas Wolf. SmolLM3: smol, multilingual, long-context reasoner. https://huggingface.co/blog/smollm3, 2025. 8

[10] Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach Moshe, Tomer Ronen, Najeeb Nabwani, et al. Llama-Nemotron: Efficient reasoning models. *arXiv preprint arXiv:2505.00949*, 2025. 9, 10, 38, 39

[11] Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952. 15

[12] Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, et al. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025. 39

[13] Yang Chen, Zhuolin Yang, Zihan Liu, Chankyu Lee, Peng Xu, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Acereason-nemotron: Advancing math and code reasoning through reinforcement learning. *Advances in neural information processing systems*, 2025. 4, 14, 20, 21, 24, 32, 38

[14] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: An open platform for evaluating llms by human preference, 2024. 31, 33

[15] Gordon V. Cormack, Charles L. A. Clarke, and Stefan Buettcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proc. SIGIR*, pages 758–759, 2009. 34

[16] Wenliang Dai, Nayeon Lee, Boxin Wang, Zhuolin Yang, Zihan Liu, Jon Barker, Tuomas Rintamaki, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. NVLM: Open frontier-class multimodal LLMs. *arXiv preprint arXiv:2409.11402*, 2024. 38

[17] DeepSeek-AI. DeepSeek-R1-0528, 2025. URL https://huggingface.co/deepseek-ai/DeepSeek-R1-0528. 9, 10, 12, 24, 26, 38

[18] DeepSeek-AI. DeepSeek-V3-0324, 2025. URL https://huggingface.co/deepseek-ai/DeepSeek-V3-0324. 9

[19] DeepSeek-AI. DeepSeek-V3.1, 2025. URL https://huggingface.co/deepseek-ai/DeepSeek-V3.1. 5, 39

[20] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The Llama 3 herd of models. *arXiv e-prints*, 2024. 38

[21] Yichao Fu, Xuewei Wang, Yuandong Tian, and Jiawei Zhao. Deep think with confidence. *arXiv preprint arXiv:2508.15260*, 2025. 29

[22] Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, et al. Omni-math: A universal olympiad level mathematic benchmark for large language models. *arXiv preprint arXiv:2410.07985*, 2024. 20

[23] Aryo Pradipta Gema, Joshua Ong Jun Leang, Giwon Hong, Alessio Devoto, Alberto Carlo Maria Mancino, Rohit Saxena, Xuanli He, Yu Zhao, Xiaotang Du, Mohammad Reza Ghasemi Madani, Claire Barale, Robert McHardy, Joshua Harris, Jean Kaddour, Emile van Krieken, and Pasquale Minervini. Are we done with mmlu?, 2024. 9

[24] Mark E Glickman and Albyn C Jones. Rating the chess rating system. *CHANCE-BERLIN THEN NEW YORK-*, 12:21–28, 1999. 49

[25] GLM-4.5-Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. GLM-4.5: Agentic, reasoning, and coding (ARC) foundation models. *arXiv preprint arXiv:2508.06471*, 2025. 39

[26] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. 4, 5, 10, 38, 39

[27] Jujie He, Jiacai Liu, Chris Yuhao Liu, Rui Yan, Chaojie Wang, Peng Cheng, Xiaoyu Zhang, Fuxiang Zhang, Jiacheng Xu, Wei Shen, Siyuan Li, Liang Zeng, Tianwen Wei, Cheng Cheng, Bo An, Yang Liu, and Yahui Zhou. Skywork open reasoner series, 2025. Notion Blog. 38

[28] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020. 6, 39

[29] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with apps. *NeurIPS*, 2021. 24

[30] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021. 9

[31] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021. 10, 20

[32] Siming Huang, Tianhao Cheng, Jason Klein Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J Yang, Jiaheng Liu, Chenchen Zhang, Linzheng Chai, et al. Opencoder: The open cookbook for top-tier code large language models. *arXiv preprint arXiv:2411.04905*, 2024. 10

[33] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024. 6, 28, 40

[34] Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2e-gym: Procedural environments and hybrid verifiers for scaling open-weights swe agents, 2025. URL https://arxiv.org/abs/2504.07164. 26

[35] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023. 6, 11, 12, 25, 40

[36] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. 16

[37] Tushar Khot, Peter Clark, Michal Guerquin, Peter Jansen, and Ashish Sabharwal. Qasc: A dataset for question answering via sentence composition. *arXiv:1910.11473v2*, 2020. 9

[38] Kimi-Team. Kimi-K2-Thinking, 2025. URL https://huggingface.co/moonshotai/Kimi-K2-Thinking. 4, 39

[39] Kimi-Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi K2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025. 4, 26

[40] Kimi-Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi K1.5: Scaling reinforcement learning with LLMs. *arXiv preprint arXiv:2501.12599*, 2025. 38

[41] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 18, 20, 23

[42] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tülu 3: Pushing frontiers in open language model post-training. 2024. 9

[43] Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, et al. RewardBench: Evaluating reward models for language modeling. *arXiv preprint arXiv:2403.13787*, 2024. 16, 17

[44] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training, 2025. URL https://arxiv.org/abs/2411.15124. 19

[45] Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 2024. URL https://huggingface.co/datasets/AI-MO/NuminaMath-CoT. 10, 20

[46] Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. Taco: Topics in algorithmic code generation dataset. *arXiv preprint arXiv:2312.14852*, 2023. 10, 24

[47] Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E Gonzalez, and Ion Stoica. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. *arXiv preprint arXiv:2406.11939*, 2024. 6, 16, 19, 40

[48] Wing Lian, Guan Wang, Bleys Goodson, Eugene Pentland, Austin Cook, Chanvichet Vong, and "Teknium". Slimorca: An open dataset of gpt-4 augmented flan reasoning traces, with verification, 2023. URL https://https://huggingface.co/Open-Orca/SlimOrca. 9

[49] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023. 38

[50] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024. 4

[51] Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025. 30

[52] Chris Yuhao Liu, Liang Zeng, Jiacai Liu, Rui Yan, Jujie He, Chaojie Wang, Shuicheng Yan, Yang Liu, and Yahui Zhou. Skywork-Reward: Bag of Tricks for Reward Modeling in LLMs. *arXiv preprint arXiv:2410.18451*, 2024. 38

[53] Zihan Liu, Yang Chen, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. AceMath: Advancing frontier math reasoning with post-training and reward modeling. *ACL*, 2024. 9, 10, 21

[54] Zihan Liu, Zhuolin Yang, Yang Chen, Chankyu Lee, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Acereason-nemotron 1.1: Advancing math and code reasoning through sft and rl synergy. *arXiv preprint arXiv:2506.13284*, 2025. 9, 10, 22, 38

[55] Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. The flan collection: Designing data and methods for effective instruction tuning. *arXiv preprint arXiv:2301.13688*, 2023. 9

[56] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 16

[57] Michael Luo, Naman Jain, Jaskirat Singh, Sijun Tan, Ameen Patel, Qingyang Wu, Alpay Ariyak, Colin Cai, Tarun Venkat, Shang Zhu, Ben Athiwaratkun, Manan Roongta, Ce Zhang, Li Erran Li, Raluca Ada Popa, Koushik Sen, and Ion Stoica. DeepSWE: Training a Fully Open-sourced, State-of-the-Art Coding Agent by Scaling RL, 2025. URL https://www.together.ai/blog/deepswe. 7, 26, 27, 37

[58] Michael Luo, Sijun Tan, Roy Huang, Xiaoxiang Shi, Rachel Xin, Colin Cai, Ameen Patel, Alpay Ariyak, Qingyang Wu, Ce Zhang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepcoder: A fully open-source 14b coder at o3-mini level, 2025. Notion Blog. 24, 38

[59] Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. DeepScaleR: Surpassing O1-Preview with a 1.5B Model by Scaling RL, 2025. Notion Blog. 20, 38

[60] MAA. American Invitational Mathematics Examination - AIME 2024, 2024. 40

[61] MAA. American Invitational Mathematics Examination - AIME 2025, 2025. 40

[62] Yingqian Min, Zhipeng Chen, Jinhao Jiang, Jie Chen, Jia Deng, Yiwen Hu, Yiru Tang, Jiapeng Wang, Xiaoxue Cheng, Huatong Song, Wayne Xin Zhao, Zheng Liu, Zhongyuan Wang, and Ji-Rong Wen. Imitate, explore, and self-improve: A reproduction report on slow-thinking reasoning systems. *arXiv preprint arXiv:2412.09413*, 2024. 20

[63] mistralai. Ministral-3-14B-Reasoning-2512, 2025. URL https://huggingface.co/mistralai/Ministral-3-14B-Reasoning-2512. 27

[64] Ivan Moshkov, Darragh Hanley, Ivan Sorokin, Shubham Toshniwal, Christof Henkel, Benedikt Schifferer, Wei Du, and Igor Gitman. Aimo-2 winning solution: Building state-of-the-art mathematical reasoning models with openmathreasoning dataset. *arXiv preprint arXiv:2504.16891*, 2025. 10, 38

[65] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling, 2025. URL https://arxiv.org/abs/2501.19393. 10

[66] Dhruv Nathawani, Igor Gitman, Somshubra Majumdar, Evelina Bakhturina, Ameya Sunil Mahabaleshwarkar, , Jian Zhang, and Jane Polak Scowcroft. Nemotron-Post-Training-Dataset-v1, 2025. URL https://huggingface.co/datasets/nvidia/Nemotron-Post-Training-Dataset-v1. 10

[67] NVIDIA. Llama-Nemotron-Post-Training-Dataset, May 2025. URL https://huggingface.co/datasets/nvidia/Llama-Nemotron-Post-Training-Dataset. 19

[68] NVIDIA. AceReason-Nemotron-7B, 2025. URL https://huggingface.co/nvidia/AceReason-Nemotron-7B. 24

[69] NVIDIA. Nvidia nemotron nano 2: An accurate and efficient hybrid mamba-transformer reasoning model. *arXiv preprint arXiv:2508.14444*, 2025. 38, 40

[70] OpenAI. ChatML format. URL https://github.com/openai/openai-python/blob/release-v0.28.0/chatml.md. 8

[71] OpenAI. Learning to reason with LLMs, 2024. 4, 39

[72] OpenAI. Introducing SWE-bench Verified, 2024. 6, 40

[73] OpenAI. Introducing GPT-4.5, 2025. 4

[74] OpenAI. Introducing GPT-5, 2025. 5, 39

[75] OpenAI. Introducing OpenAI o3 and o4-mini, 2025. 4, 39

[76] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *NeurIPS*, 35, 2022. 4, 37, 38

[77] Junsoo Park, Seungyeon Jwa, Meiying Ren, Daeyoung Kim, and Sanghyuk Choi. Offsetbias: Leveraging debiased data for tuning evaluators, 2024. 15

[78] Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models. In *ICML*, 2025. 6

[79] Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*, 2023. 39

[80] Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following, 2025. URL https://huggingface.co/datasets/allenai/IF_multi_constraints_upto5. 6, 19, 40

[81] Shanghaoran Quan, Jiaxi Yang, Bowen Yu, Bo Zheng, Dayiheng Liu, An Yang, Xuancheng Ren, Bofei Gao, Yibo Miao, Yunlong Feng, et al. Codeelo: Benchmarking competition-level code generation of llms with human-comparable elo ratings. *arXiv preprint arXiv:2501.01257*, 2025. 49

[82] Qwen-Team. Qwen3-235B-A22B-Thinking-2507 , 2025. URL https://huggingface.co/Qwen/Qwen3-235B-A22B-Thinking-2507. 5, 39

[83] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 2023. 38

[84] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. 6, 39

[85] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 38

[86] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. DeepseekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. 14, 38

[87] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024. 24

[88] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297, 2025. 14

[89] Atefeh Sohrabizadeh, Jialin Song, Mingjie Liu, Rajarshi Roy, Chankyu Lee, Jonathan Raiman, and Bryan Catanzaro. Nemotron-cortexa: Enhancing llm agents for software engineering tasks via improved localization and solution diversity. In *Forty-second International Conference on Machine Learning*, 2025. 34

[90] Zhenpeng Su, Leiyu Pan, Xue Bai, Dening Liu, Guanting Dong, Jiaming Huang, Wenping Hu, Fuzheng Zhang, Kun Gai, and Guorui Zhou. Klear-reasoner: Advancing reasoning capability via gradient-preserving clipping policy optimization. *arXiv preprint arXiv:2508.07629*, 2025. 38

[91] Binghai Wang, Runji Lin, Keming Lu, Le Yu, Zhenru Zhang, Fei Huang, Chujie Zheng, Kai Dang, Yang Fan, Xingzhang Ren, et al. Worldpm: Scaling human preference modeling. *arXiv preprint arXiv:2505.10527*, 2025. 16

[92] Peiyi Wang, Lei Li, Zhihong Shao, RX Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *arXiv preprint arXiv:2312.08935*, 2023. 38

[93] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw. 34

[94] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*, 2024. 6, 9, 39

[95] Zhilin Wang, Yi Dong, Olivier Delalleau, Jiaqi Zeng, Gerald Shen, Daniel Egert, Jimmy J. Zhang, Makesh Narsimhan Sreedhar, and Oleksii Kuchaiev. Helpsteer2: Open-source dataset for training top-performing reward models, 2024. 15, 38

[96] Zhilin Wang, Jiaqi Zeng, Olivier Delalleau, Ellie Evans, Daniel Egert, Hoo-Chang Shin, Felipe Soares, Yi Dong, and Oleksii Kuchaiev. Rlbff: Binary flexible feedback to bridge between human feedback & verifiable rewards. *arXiv preprint arXiv:2509.21319*, 2025. 18

[97] Zhilin Wang, Jiaqi Zeng, Olivier Delalleau, Hoo-Chang Shin, Felipe Soares, Alexander Bukharin, Ellie Evans, Yi Dong, and Oleksii Kuchaiev. Helpsteer3-preference: Open human-annotated preference data across diverse tasks and languages, 2025. URL https://arxiv.org/abs/2505.11475. 15

[98] Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I Wang. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution. *arXiv preprint arXiv:2502.18449*, 2025. 11, 12, 26, 35

[99] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992. 14

[100] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying llm-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024. 11

[101] LLM Xiaomi, Bingquan Xia, Bowen Shen, Dawei Zhu, Di Zhang, Gang Wang, Hailin Zhang, Huaqiu Liu, Jiebao Xiao, Jinhao Dong, et al. MiMo: Unlocking the Reasoning Potential of Language Model–From Pretraining to Posttraining. *arXiv preprint arXiv:2505.07608*, 2025. 23

[102] Chengxing Xie, Bowen Li, Chang Gao, He Du, Wai Lam, Difan Zou, and Kai Chen. Swe-fixer: Training open-source llms for effective and efficient github issue resolution. *arXiv preprint arXiv:2501.05040*, 2025. 12

[103] Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing. *ArXiv*, abs/2406.08464, 2024. URL https://api.semanticscholar.org/CorpusID:270391432. 9

[104] 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 arXiv:2412.15115*, 2024. 9, 15

[105] An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2.5-Math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024. 38

[106] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. 5, 8, 9, 11, 16, 38, 39

[107] John Yang, Kilian Lieret, Carlos E Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents. *arXiv preprint arXiv:2504.21798*, 2025. 12, 27

[108] Edward Yeo, Yuxuan Tong, Morry Niu, Graham Neubig, and Xiang Yue. Demystifying long chain-of-thought reasoning in llms. *arXiv preprint arXiv:2502.03373*, 2025. 39

[109] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025. 14, 19, 23, 38

[110] Lifan Yuan, Ganqu Cui, Hanbin Wang, Ning Ding, Xingyao Wang, Jia Deng, Boji Shan, Huimin Chen, Ruobing Xie, Yankai Lin, Zhenghao Liu, Bowen Zhou, Hao Peng, Zhiyuan Liu, and Maosong Sun. Advancing llm reasoning generalists with preference trees, 2024. 9

[111] Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*, 2025. 38

[112] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric. P Xing, Joseph E. Gonzalez, Ion Stoica, and Hao Zhang. Lmsys-chat-1m: A large-scale real-world llm conversation dataset, 2023. 19

[113] Zihan Zheng, Zerui Cheng, Zeyu Shen, Shang Zhou, Kaiyuan Liu, Hansen He, Dongruixuan Li, Stanley Wei, Hangyi Hao, Jianzhu Yao, et al. Livecodebench pro: How do olympiad medalists judge llms in competitive programming? *arXiv preprint arXiv:2506.11928*, 2025. 6, 28, 40, 49

[114] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models, 2023. URL https://arxiv.org/abs/2311.07911. 6, 19, 40