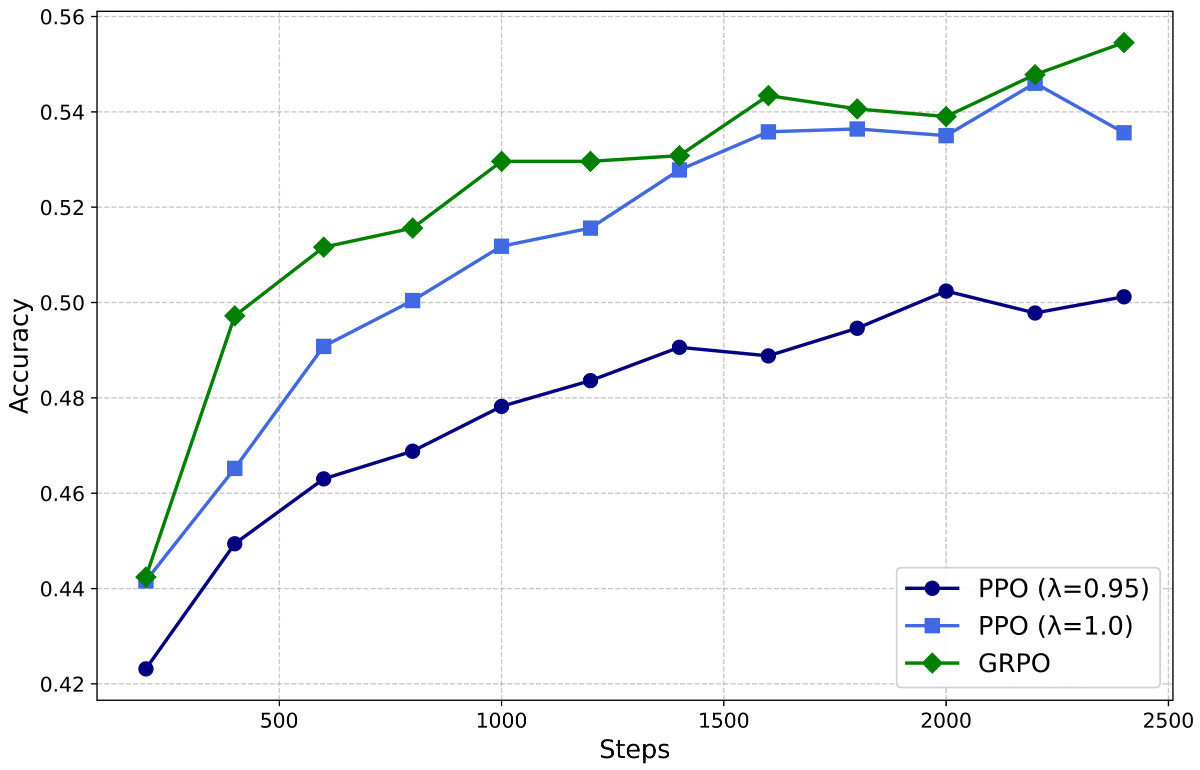

Figure 4 | Performance of PPO and GRPO on the MATH task.

maximize cumulative rewards, PPO's approach penalizes the cumulative KL divergence, which may implicitly penalize the length of the response and thereby prevent the model's response length from increasing. In addition, as we may train thousands of steps in the scenario of training long chain-of-thought reasoning models, the trained policy can diverge significantly from the initial reference policy. In order to balance the scope that the training policy can explore and the stability of the training, we periodically update the reference policy to the latest policy during the actual training process.

Figure 4 compares the performance of PPO and GRPO on the MATH task using DeepSeek-Coder-V2-Lite (16B MoE with 2.4B active parameters). Unlike GRPO, PPO requires additional hyperparameter tuning—particularly of the λ coefficient in GAE—and is highly sensitive to this parameter. When λ is set to 0.95 (the default value in most open-source PPO implementations), PPO performs considerably worse than GRPO. However, with careful tuning (setting λ to 1.0), PPO's performance improves substantially, nearing that of GRPO.

While PPO can achieve comparable performance when appropriately tuned, it demands additional computational cost for hyperparameter optimization. Moreover, considering the memory and computational overhead associated with training an additional value model, GRPO presents a more practical alternative, especially when training large-scale models with constrained resources.

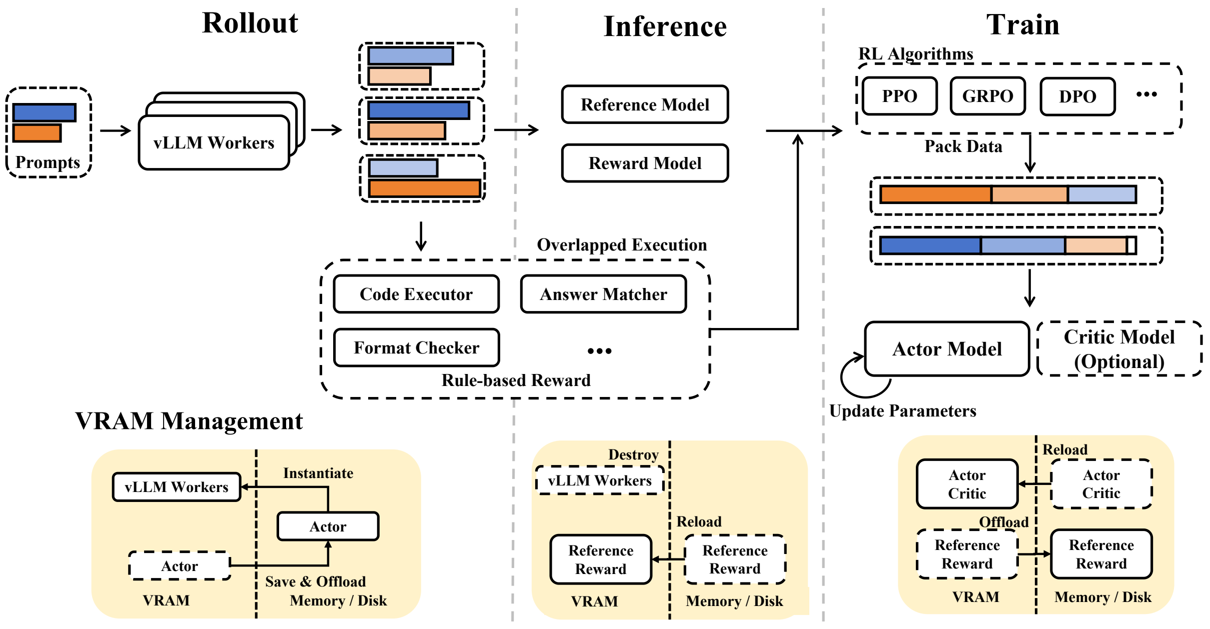

Figure 5 | Overview of our RL framework.

B. Training Details

B.1. RL Infrastructure

Conducting RL training on large models places high demands on the infrastructure. Our RL framework is architected with a decoupled and extensible structure to facilitate seamless integration of diverse models and algorithms. Within this framework, we have incorporated both intra-modular and inter-modular optimization techniques, to ensure training efficiency and scalability.

Specifically, as depicted in Figure 5, the framework is partitioned into four distinct modules, each corresponding to a specific phase of the RL pipeline:

- **Rollout Module:** Prompts are loaded from training dataset and uniformly dispatched across multiple vLLM (Kwon et al., 2023) workers, each equipped with the actor model, to sample multiple responses. For DeepSeek-V3 MoE architecture, we implement an expert parallelism strategy across nodes to reduce memory access overhead, and deploy redundant copies of hotspot experts to balance computational loads among different experts. Multi-Token Prediction (MTP) component is also leveraged for self-speculative decoding, significantly accelerating the decoding speed and effectively minimizing the completion time for the longest samples.
- **Inference Module:** This module loads the reward model and reference to perform a forward pass on the samples generated during the rollout phase, thereby obtaining model-based rewards and other essential information.
- **Rule-based Reward Module:** This module computes rule-based rewards for the model-generated responses. A unified interface has been designed to accommodate diverse implementations (e.g., code executor, answer matcher, format checker, etc.). Although this module does not require loading models into GPU memory, its execution tends to be time-consuming. To tackle this issue, an asynchronous scheduling approach is employed to overlap its execution with the Rollout and Inference modules, effectively hiding the

associated latency.

- **Training Module:** This module loads the actor model and the critic model (if required), to compute loss and update model parameters. It provides flexible support for a variety of RL algorithms (e.g., PPO, GRPO, DPO, etc.). To minimize computational waste caused by sequence padding and balance the workload across devices, we design the following data packing strategy: first, all data in a global batch is sorted by length and distributed across processes within the data parallel group; subsequently, within each process, the Best-Fit strategy is applied to pack the data into fixed-length chunks with minimal padding; finally, the number of chunks is adjusted to be equal across all processes. Additionally, we have integrated the DualPipe algorithm, utilized in DeepSeek-V3 training, to achieve efficient pipeline parallelism.

Notably, upon completion of each module (excluding the Rule-based Reward module), the model instances utilized in that phase are automatically offloaded from VRAM to either system memory or disk storage, thereby freeing up VRAM for the subsequent phase.

B.2. Reward Model Prompt

Please act as an impartial judge and evaluate the quality of the responses provided by two AI assistants to the user prompt displayed below. You will be given assistant A's answer and assistant B's answer. Your job is to evaluate which assistant's answer is better. Begin your evaluation by generating your own answer to the prompt. You must provide your answers before judging any answers.
When evaluating the assistants' answers, compare both assistants' answers with your answer. You must identify and correct any mistakes or inaccurate information.
Then consider if the assistant's answers are helpful, relevant, and concise. Helpful means the answer correctly responds to the prompt or follows the instructions. Note when user prompt has any ambiguity or more than one interpretation, it is more helpful and appropriate to ask for clarifications or more information from the user than providing an answer based on assumptions. Relevant means all parts of the response closely connect or are appropriate to what is being asked. Concise means the response is clear and not verbose or excessive.
Then consider the creativity and novelty of the assistant's answers when needed. Finally, identify any missing important information in the assistants' answers that would be beneficial to include when responding to the user prompt.
After providing your explanation, you must output only one of the following choices as your final verdict with a label:
1. Assistant A is significantly better: [[A≫B]]
2. Assistant A is slightly better: [[A>B]]
3. Tie, relatively the same: [[A=B]]
4. Assistant B is slightly better: [[B>A]]
5. Assistant B is significantly better: [[B≫A]]
Example output: "My final verdict is tie: [[A=B]]".

Table 4 | Description of RL Data and Tasks.

Data Type	# Prompts	Question Type	Output Type
Math	26K	Quantitative Reasoning	Number/Expression/Equation
Code	17K	Algorithm and Bug Fixing	Code Solution
STEM	22K	Multi-Choice	Option
Logic	15K	Choice/Quantitative Reasoning	Option/Number
General	66K	Helpfulness/Harmlessness	Ranked Responses

B.3. Data Recipe

B.3.1. RL Data

Reasoning RL data includes four categories: mathematics, coding, STEM, and logic problems. In addition, we also incorporate general RL data to improve the helpfulness and harmlessness of the model in the training of DeepSeek-R1. All questions are in Chinese or English. The description of the RL data can be found in Table 4, where we will describe the details of each data type one by one as follows:

- **Mathematics** dataset consists of 26k quantitative reasoning questions, including math exam questions and competition problems. The average number of prompt tokens is 122. The dataset covers various mathematical domains such as algebra, calculus, probability, and geometry. Problems range in difficulty from regional contests to international Olympiads. For each problem, the model is expected to produce a step-by-step reasoning process culminating in a final answer, which can be a numerical value (e.g., "5"), a mathematical expression (e.g., "$x^2 + 3x - 2$"), or an equation (e.g., "$y = 2x + 1$"). Mathematical proofs are excluded because it is difficult to determine their correctness. For reinforcement learning purposes, we calculate the reward of a reasoning process by matching the predicted answer with the reference answer. If the answer aligns with the reference, the reward is assigned a value of 1; otherwise, it is assigned a value of 0.
- **Coding** dataset includes 17k algorithm competition questions, along with 8k bug fixing problems. The algorithm competition questions are similar to problems found on platforms like Codeforces or LeetCode. Each problem typically includes a detailed problem description, constraints, and multiple input-output examples. The task is to write a complete function or program that can solve the problem correctly and efficiently, passing a comprehensive set of hidden test cases that assess both correctness and performance. These problems test algorithmic skills, including dynamic programming, graph theory, string manipulation, and data structure usage.
  The bug-fixing problems are extracted from real-world GitHub issues. Each task provides an issue description, a buggy version of the source code, and a set of unit tests that partially or completely fail. The goal is to understand the intent of the issue, locate and fix the defect in the code, and ensure that the corrected version passes all unit tests.
- **STEM** dataset comprises 22k choice questions that cover topics such as physics, chemistry, and biology. Each question in the STEM task presents a subject-specific problem accompanied by four to eight answer options. The model is required to select the most scientifically accurate answer based on the given context and domain knowledge. The average number of prompt tokens is 161. Specifically, the dataset includes 15.5% physics, 30.7% biology, 46.5% chemistry, and 7.3% other topics such as health and medicine. Since all STEM questions are multiple-choice, a binary reward is assigned based on whether the

correct option is matched.

- **Logic** dataset contains 15k questions designed to evaluate a model's reasoning capabilities across a broad spectrum of logical challenges. The dataset includes both real-world and synthetically generated problems. All problems support automatic evaluation, and the average prompt length is approximately 420 tokens. The real-world portion of the dataset comprises a diverse selection of problems sourced from the web, including brain teasers, classical logic puzzles, and knowledge-intensive questions. These questions are presented in a multiple-choice format to ensure objective and consistent assessment. The synthetic portion consists primarily of two categories: code-IO problems and puzzle tasks. Code-IO problems are generated using the data pipeline introduced by Li et al. (2025), which converts competitive coding problems and their corresponding input-output test cases into verifiable logical reasoning problems. The puzzle tasks include problems intended to assess specific reasoning competencies. For example, cryptography puzzles are designed to evaluate a model's ability to identify and apply patterns in cipher schemes or perform string manipulations; logic puzzles focus on deductive reasoning over complex constraints, such as inferring valid conclusions from a fixed set of premises (e.g., the Zebra puzzle); and arithmetic puzzles test the model's numerical reasoning (e.g. probability questions and 24 game).
- **General** dataset consists of 66k questions designed to assess helpfulness, spanning various categories such as creative writing, editing, factual question answering, and role-playing. Additionally, the dataset includes 12,000 questions focused on evaluating harmlessness. To ensure robust verification, two reward models are utilized, each trained on a curated dataset of ranked responses generated by models in relation to helpfulness and harmlessness, respectively. We trained the helpful reward model for a single epoch with a maximum sequence length of 8192 tokens during the training phase. However, when deploying the model to generate reward signals, we did not impose any explicit length constraints on the input sequences being evaluated.

B.3.2. DeepSeek-R1 Cold Start

For DeepSeek-R1, we construct and collect a small amount of long CoT data to fine-tune the model as the initial RL actor. The motivation is primarily product-driven, with a strong emphasis on enhancing user experience. Users tend to find responses more intuitive and engaging when the reasoning process aligns with first-person perspective thought patterns. For example, DeepSeek-R1-Zero is more likely to employ the pronoun 'we' or avoid first-person pronouns altogether during problem solving, whereas DeepSeek-R1 tends to use 'I' more frequently. Furthermore, we acknowledge that such patterns may elicit unwarranted trust from users. Here, we would like to emphasize that the observed vivid reasoning patterns primarily reflect DeepSeek-engineered heuristics, rather than indicating that the model has inherently acquired human-like intelligence or autonomous problem-solving capabilities.

In cold start data creation, we prefer the thinking process that begins with comprehending the problem, followed by detailed reasoning that incorporates reflection and verification. The language employed throughout the thinking process is presented in the first-person perspective. Additionally, maintaining language consistency is crucial for an optimal user experience. Without proper control, model responses may contain a mixture of different languages, regardless of the language used in the query. Such inconsistencies can disrupt comprehension and reduce user satisfaction. Therefore, careful refinement is essential to ensure that responses remain coherent and aligned with user intent. Nevertheless, we acknowledge that the raw Chain-of-Thought (CoT) reasoning produced by DeepSeek-R1-Zero may possess potential that extends beyond the