

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

limitations of current human priors. Specifically, we first engage human annotators to convert the reasoning trace into a more natural, human conversational style. The modified data pairs are then used as examples to prompt an LLM to rewrite additional data in a similar style. All LLM-generated outputs subsequently undergo a second round of human verification to ensure quality and consistency.

Listing 1 | Prompt for producing a human-readable solution.

```
## Question
{question}

## Thought process
{thought_process}

---
Based on the above thought process, provide a clear, easy-to-follow, and well-formatted
solution to the question. Use the same language as the question.

The solution must strictly follow these requirements:
- Stay faithful and consistent with the given thought process. Do not add new reasoning
  steps or conclusions not shown in the original.
- Show key steps leading to final answer(s) in clear, well-formatted LaTeX.
- Use \boxed{} for final answer(s).
- Be clean and concise. Avoid colloquial language. Do not use phrases like "thought
  process" in the solution.

Your response should start with the solution right away, and do not include anything
else. Your task is solely to write the solution based on the provided thought
process. Do not try to solve the question yourself.
```

Specifically, we begin by gathering thousands of high-quality, diverse reasoning prompts. For each prompt, we generate multiple reasoning trajectories using DeepSeek-R1-Zero with a relatively high temperature of 1.0. Next, we filter these generations to retain only those with correct final answers and a readable format. For mathematical outputs, we use `sympy` (<https://www.sympy.org/>) for parsing and expression comparison; and for formatting, we apply rules such as repetition detection and language-mixing filtering. Finally, we prompt DeepSeek-V3 to refine both the reasoning and the summaries to ensure proper formatting and a human-friendly expression. In particular, to resolve language mixing, we instruct DeepSeek-V3 to “Translate the thinking process to the same language as the question.” Since DeepSeek-R1-Zero’s summary only provided the final answer, we use the summary prompt in Listing 1 to produce a concise, human-readable solution that outlines both the reasoning steps and the final result.

For code data, we collect a large set of competitive programming problems. In detail, We have compiled an extensive collection of competitive programming problems from multiple online judge (OJ) platforms, specifically 5151 problems from Codeforces and 2504 problems from AtCoder. Since the original test cases are not publicly available from these platforms, we developed a methodology to create reliable test cases for each problem.

Our approach involves using DeepSeek-V2.5 to generate candidate test cases, followed by a rigorous validation process. Specifically, we prompted DeepSeek-V2.5 to write Python programs that generate test cases tailored to each problem’s requirements as shown in Listing 2.

After obtaining numerous candidate test cases, we implemented a two-phase filtering procedure. First, we used correct submissions to eliminate invalid test cases that produced incorrect outputs. Then, we strategically selected subsets of test cases that successfully identified