

# PuzzleClone: An SMT-Powered Framework for Synthesizing Verifiable Data

Kai Xiong<sup>1\*†</sup>, Yanwei Huang<sup>2\*</sup>, Rongjunchen Zhang<sup>1†</sup>, Kun Chen<sup>1</sup>, Haipang Wu<sup>1</sup>

<sup>1</sup>HiThink Research <sup>2</sup>The Hong Kong University of Science and Technology

## Abstract

High-quality mathematical and logical datasets with verifiable answers are essential for strengthening the reasoning capabilities of large language models (LLMs). While recent data augmentation techniques have facilitated the creation of large-scale benchmarks, existing LLM-generated datasets often suffer from limited reliability, diversity, and scalability. To address these challenges, we introduce PuzzleClone, a formal framework for synthesizing verifiable data at scale using Satisfiability Modulo Theories (SMT). Our approach features three key innovations: (1) encoding seed puzzles into structured logical specifications, (2) generating scalable variants through systematic variable and constraint randomization, and (3) ensuring validity via a reproduction mechanism. Applying PuzzleClone, we construct a curated benchmark comprising over 83K diverse and programmatically validated puzzles. The generated puzzles span a wide spectrum of difficulty and formats, posing significant challenges to current state-of-the-art models. We conduct post training (SFT and RL) on PuzzleClone datasets. Experimental results show that training on PuzzleClone yields substantial improvements not only on PuzzleClone testset but also on logic and mathematical benchmarks. Post training raises PuzzleClone average from 14.4 to 56.2 and delivers consistent improvements across 7 logic and mathematical benchmarks up to 12.5 absolute percentage points (AMC2023 from 52.5 to 65.0). Our code and data are available at <https://github.com/HiThink-Research/PuzzleClone>.

## Introduction

Large-language models (LLMs) have recently demonstrated impressive zero-shot and few-shot performance on a wide spectrum of reasoning tasks, yet consistently achieving robust logical reasoning remains an open challenge. To push the frontier, researchers have released a variety of mathematical-puzzle and formal-logic benchmarks that expose models to carefully crafted, high-difficulty problems. Unfortunately, the manual effort required to compose and validate such items has kept existing corpora relatively small and homogeneous, impeding further progress.

One promising approach to address this bottleneck is *data augmentation*, which involves generating new problems by

\*These authors contributed equally.

†Corresponding authors. Emails: xiongkai@myhexin.com; zhangrongjunchen@myhexin.com

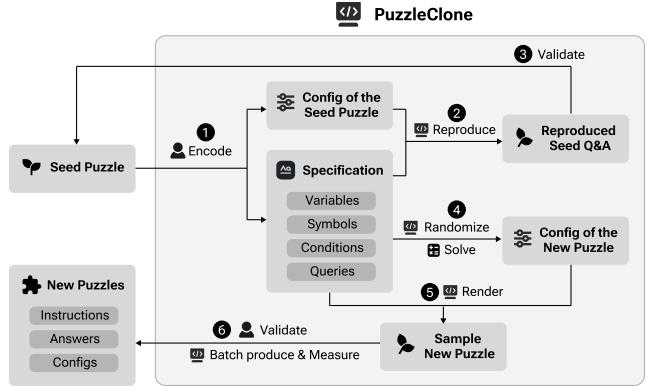


Figure 1: Overview of the PuzzleClone framework.

systematically modifying existing “seed” instances (Feng et al. 2021; Lu et al. 2024). This strategy has the potential to vastly expand the size and diversity of reasoning datasets while reducing the manual effort involved in their creation. However, current augmentation pipelines largely rely on LLMs to annotate new problems, generate solutions, and verify the answers (Lu et al. 2024; Tan et al. 2024; Shah et al. 2024), which introduces several critical limitations. First, without a robust, end-to-end verification approach, it is hard to ensure the data reliability throughout the synthesis pipeline, leading to flawed, inaccurate, or biased data (Wang et al. 2024a). Second, existing pipelines lack formalization and disproportionately depend on the generative capabilities of the underlying LLMs, which threatens data diversity. For example, an individual model typically explores only a narrow range of variations in assumptions, conditions, parameters, and queries. This limited coverage reduces the dataset’s ability to challenge and generalize LLM reasoning capabilities effectively. Finally, the substantial computational costs associated with involving LLMs in every step of the synthesis pipeline also severely constrain scalability (Wang et al. 2024a). Motivated by the need for a scalable path to unbounded yet trustworthy data creation, our work seeks a principled procedure that can, in principle, generate reliable reasoning data indefinitely.

In this work, we introduce PuzzleClone, a novel benchmark of 83,657 challenging, diverse, and fully verifiable

puzzles. PuzzleClone mainly focuses on Satisfiability Modulo Theories (SMT) problems, a representative class of NP-complete puzzles to which many mathematical problems can be reduced. The benchmark is generated through a formal data curation framework proposed by us, as shown in Figure 1. First, each seed puzzle is manually encoded into a structured problem specification that specifies the data synthesis pipeline, with a config file that contains the values of the parameters specific to the seed puzzle (1). Based on them, an instance generator can automatically batch produce new questions with diverse mathematical configurations by systematically varying parameters and combinations of constraints with randomization (4, 5). Meanwhile, it can programmatically derive ground-truth answers for each generated instance with the assistance of a symbolic SMT solver, Z3 (De Moura and Bjørner 2008) (4). To ensure fidelity, our pipeline includes a reproduction step that verifies the original seed questions can be regenerated from the DSL specifications (2, 3), thereby guaranteeing the accuracy of both problem formulations and associated answers.

Our experiments demonstrate that PuzzleClone poses substantial challenges for state-of-the-art large language models (LLMs), including ChatGPT-4o and DeepSeek-R1. We further use PuzzleClone to distill the reasoning capabilities of large models into small ones. After post-training, Qwen2.5-7B-Instruct improves from 14.4 to 56.2 on PuzzleClone, and achieves gains of up to 12.5 absolute percentage points across 7 logic and mathematical benchmarks.

## PuzzleClone

The data synthesis process can be formalized as follows: For each input seed puzzle  $P^0$ , our goal is to generate a set of new puzzles  $P^* = (I^*, A^*)$ , where  $I^*$  is the set of puzzle instructions and  $A^*$  is the set of answers. The process comprises three stages: puzzle encoding, puzzle generation, and config-based validation.

## Puzzle Encoding

Inspired by prior works (Pan et al. 2023), we observe that most puzzles can be conceptualized as a combination of a universal puzzle template, which encapsulates the core logic of the puzzle, and a set of parameters that define its specific details. For example, in the seed puzzle illustrated in Figure 2A, the underscored texts represent the puzzle-specific parameters while the rest describe the template. Thus, data augmentation can be viewed as systematically varying these parameters and embedding them into the universal template to generate new puzzles.

During puzzle encoding, each seed puzzle  $Q^0$  is manually encoded into a structured problem specification  $Q_s$  and a configuration file  $Q_c$ . Contrary to prior works that rely on logic programming languages such as Prolog and SMT-LIB (Clocksin and Mellish 2003; Pan et al. 2023; Barrett et al. 2010), we design a new domain-specific language (DSL) to describe  $Q_s$ , as shown in Figure 2B. This DSL not only captures the core puzzle logic but also encodes constraints that are implicit and only applicable to data augmentation, such as the parameters’ value domains and the num-

ber of constraints. Furthermore, the DSL represents the puzzle in a more human-readable format while still machine-parsable for downstream processing. The DSL specification  $Q_s = (V, S, C, Q, D)$  has the following components<sup>1</sup>:

**Variables**  $V$ : Parameters that can be varied to generate new puzzles. For instance, the variables in the seed puzzle in Figure 2A include the number of students  $s\_num$  and food types  $f\_num$ , as well as their names  $names$  and  $food$ . Each variable  $v$  is characterized by its type ( $type$ ) and value domain ( $domain$ ), or a formula ( $formula$ ) defining how it can be derived from other parameters. For instance,  $get\_faker$  is an internal function which leverages the Faker<sup>2</sup> library to generate random natural-language-based instance names. Finally, each variable is assigned a *difficulty factor* ( $diff\_factor$ ) that specifies how the variable’s value contributes to the puzzle difficulty (+1, -1, 0 for positive, negative, and neutral effect. Default is 0).

**Symbols**  $S$ : Quantities to be solved in the puzzle. Symbols of various types ( $type$ ) can be created by mapping existing variables ( $source$ ), and each symbol can be binded with a natural language (NL) template ( $desc$ ) that outlines how it should be expressed in the puzzle instructions. For instance, a set of boolean symbols  $buy$  are generated for each combination of students’ and food names representing the purchase information, as depicted in Figure 2B.

**Conditions**  $C$ : A pool of all potential constraints that can be applied to the puzzle. Conditions are classified into *static conditions*, which remain the same among all puzzles, and *dynamic conditions*, which are randomly generated from a template and can appear a variable number of times. For example, the statement “[students] have purchased at least one kind of food” in the seed puzzle in Figure 2A can be viewed as a static condition, while the subsequent text under “their choices must meet the following conditions” outlines dynamic conditions which have reusable structures and mutable parameters, making them suitable for replication. Our DSL provides structural declarations for randomly generating such conditions. First, the dynamic parameters within each condition can be randomly determined according to the specified parameter selection pool ( $source$ ) and the allowable range for the number of instances of each condition type ( $domain$ ). Additionally, every condition is associated with a formula template (in Z3, specified by  $formula$ ) and a NL template ( $desc$ ). The randomly chosen parameters are integrated into the templates to generate the complete condition.

**Queries**  $Q$ : The questions in the puzzle. For example, the question in Figure 2B presents a single-choice question ( $query\_type$ ) with four options ( $opt\_num$ ). The structure of these options is defined by two distinct templates ( $templates$ ), each with its own parameter pool ( $source$ ), formula and instruction template ( $opt\_formula$ ,  $opt\_text$ ).

**Description**  $D$ : An NL template assembling the previous components to construct the eventual puzzle ( $desc$ ).

In parallel with the specification, the puzzle-specific values of variables and the parameters within constraints and

<sup>1</sup>DSL schema and specs of puzzles can be found in Appendix.

<sup>2</sup><https://github.com/joke2k/faker>



Figure 2: The data synthesis pipeline of PuzzleClone. (A) An example seed puzzle. (B, C) The DSL specification and the config encoded from the seed puzzle. (D) A randomly generated config produced by the puzzle generator for this DSL specification. (E) A new puzzle instance synthesized by the puzzle renderer using the DSL specification and the generated config.

queries are extracted as a configuration file  $Q_c$ , as shown in Figure 2C. Thus, a puzzle can be uniquely determined by combining a specification and a configuration.

## Puzzle Generation

After encoding the seed puzzle, a puzzle generator will translate the specification into new puzzles. Specifically, it first generates a Python script which automatically decides all variables, conditions, and queries by random while leveraging the Z3 solver to solve the puzzle. Thus, new puzzles can be generated by executing the script.

## Config-based Answer Validation

One of the key goals of PuzzleClone is to keep each synthesis step verifiable for the reliability of the synthesized data. Since the puzzle encoding process relies on careful manual effort, it is necessary to make sure the results of this step is accurate, leading to our design of a config-based answer validator. Specifically, a validation script is generated in parallel with the puzzle generator, which is capable of reproducing the seed puzzle by acquiring the puzzle-specific values directly from the config instead of randomization. As a result, a deterministic reference answer can be computed and compared against the ground-truth solution of the seed puzzle, thereby verifying the correctness of the encoding process.

## Dataset Construction

We constructed our dataset through a multi-stage process designed to ensure quality, diversity, and verifiability. Below we detail the systematic process from raw seed collection to final dataset partitioning.

## Seed Puzzle Collection and Curation

We started by collecting raw puzzles from diverse sources, including mathematics competitions for primary and secondary schools, Olympiad-style problems, logical reasoning books/blogs, and established puzzle benchmarks, e.g., Sudoku and combinatorial optimization problems (Mittal et al. 2024; Chen et al. 2025b; Li et al. 2025; Liu et al. 2025b).

We then employed the Qwen2.5-72B-Instruct model with specially crafted prompts (see Appendix) to select potential seed puzzles based on two key criteria:

- The problem contains variable values or logical constraints that can be replicated by random without altering its semantic validity.
- The puzzle is solvable using SMT solvers within tractable time and memory bounds, and a correct Python solution can be generated by the model.

## Human Verification and Enhancement

The AI-filtered puzzles underwent rigorous manual review, during which we (1) conducted brainstorming sessions to enhance problem complexity through additional variables and novel constraints (e.g., A1-ant vs. A2-athlete in our dataset), (2) introduced diversified questioning approaches to increase variety, and (3) eliminated problems with conceptually similar solutions (e.g., those relying on the same underlying mathematical identity or solving trick) to ensure mathematical diversity.

This process yielded 86 high-quality seed puzzles, categorized as 72 SMT-solvable problems and 14 pure formula/code-based problems (labeled as A-series).

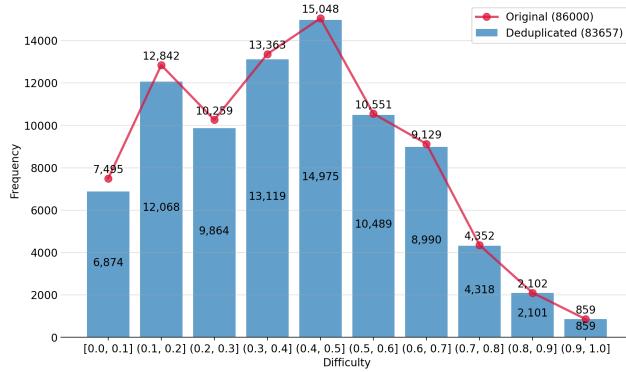


Figure 3: Puzzle difficulty distribution before and after deduplication.

## Puzzle Generation and Deduplication

We utilized PuzzleClone to generate 1K variants for each seed puzzle, resulting in an initial set of 86K puzzle instances. For each generated variant, we assigned a unique identifier, its source puzzle, question type (`qtype`), and evaluation type (`eval_type`). The `qtype` indicates the format of the question, such as multiple-choice, fill-in-the-blank, or short answer. The `eval_type` specifies the answer structure and corresponding evaluation strategy, including formats like numeral, nominal, option, `ordered_array`, and `unordered_array`. Based on the assigned `qtype` and `eval_type`, we designed custom prompt wrappers (see Appendix) and implemented appropriate evaluation operators.

Since each instance was generated independently and randomly, we applied a deduplication process to eliminate logically equivalent puzzles by comparing their `configs` (Figure 2D) rather than surface-level text (Figure 2E). Specifically, we ignored nominal fields (e.g., `names`) and treated value pools as unordered sets in puzzle `configs`, enabling us to identify near-duplicates even when textual details varied (such as wording or ordering differed).

After deduplication, we retained 83,657 unique puzzle instances. Among the 86 seed puzzles, 32 had at least one duplicated variant (see Appendix). Notably, 28-exam exhibited the highest duplication count, with 678 repeated instances. Our analysis revealed that puzzles with limited randomizable space tend to result in higher duplication rates.

## Difficulty Assessment and Analysis

To quantify puzzle complexity, we designed a composite difficulty score based on four features:

- `sym_num`: number of logical symbols.
- `cond_num`: number of logical constraints.
- `desc_len`: the character length of the problem description.
- `var_scale`: a custom metric estimating the difficulty impact of variable domains. It is calculated as the average of normalized variable values, adjusted by their `diff_factor` (see Figure 2B), which encodes the variable's correlation direction with problem difficulty.

Specifically, for a variable  $v$  with domain  $[v_{\min}, v_{\max}]$ , its normalized value is given by  $\hat{v} = (v - v_{\min}) / (v_{\max} - v_{\min})$ . The difficulty-adjusted value,  $v_{adj}$ , is subsequently defined as  $\hat{v}$  or  $1 - \hat{v}$  for positive or negative `diff_factor` values, respectively. Note that variables whose corresponding `diff_factor` is 0 will be excluded. The `vars_scale` is computed as the average of all  $v_{adj}$  for variables where `diff_factor`  $\neq 0$ . The final difficulty score is the mean of the min-max normalized (to  $[0, 1]$ ) values of `sym_num`, `cond_num`, `desc_len`, and `vars_scale`.

The difficulty distribution of the puzzles is shown in Figure 3, revealing that the majority of puzzles cluster in the easy-to-medium range. Hard puzzles are rare due to increased likelihood of being unsolvable when the complexity and diversity (i.e., the variable values and constraint number) increase. Furthermore, duplicates were disproportionately found among easier puzzles due to their narrower randomization space.

## Dataset Partitioning Strategy

To facilitate a comprehensive evaluation of popular LLMs while supporting both supervised fine-tuning (SFT) and reinforcement learning (RL) training, we implemented a stratified dataset partitioning strategy based on puzzle difficulty and seed origin. The splitting protocol was as follows:

- Puzzles were classified into *normal* (difficulty  $\leq 0.5$ ) or *hard* (difficulty  $> 0.5$ ) categories by difficulty scores.
- For each seed puzzle, its variants in each category were randomly split into 90% train and 10% test.
- From the training portion, 25 *normal* and 25 *hard* samples were selected per seed puzzle (falling back to normal in case of insufficient hard samples) to form the SFT set of  $86 \times 25 \times 2 = 4300$  samples.
- A smaller subset (5 *normal* + 5 *hard*) was selected per seed for RL validation (860 in total). All remaining samples from the training set were assigned to RL training.

	Train				Test
	SFT	RL-Train	RL-Val	Total	
Normal	2161	50738	430	51168	5730
Hard	2139	23616	430	24046	2713
Sum	4300	74354	860	75214	8443

Table 1: Dataset partitioning statistics.

The final partitioning (Table 1) ensures comprehensive coverage, with each subset (SFT, RL-Train, RL-Val, and Test) containing puzzles derived from all 86 seed puzzles while maintaining difficulty stratification.

## Experiments

### Experimental Setup

All model training and inference were conducted on NVIDIA H100 GPUs. We trained our models using an 8×NVIDIA H100 GPU cluster, with Qwen2.5-7B-Instruct as the foundation model for all training experiments (detailed training parameter setting can be found in Appendix).

Model	Normal	Hard	Avg.
<b>Proprietary Models</b>			
ChatGPT-4o (OpenAI 2024)	31.6	23.9	27.7
ChatGPT-o3 (OpenAI 2025)	86.8	<b>82.0</b>	<b>84.4</b>
Gemini-2.0-flash (Gemini Team 2025a)	41.7	30.5	36.1
Gemini-2.5-pro (Gemini Team 2025b)	75.8	67.2	71.5
Claude-3.5-sonnet (Anthropic 2024)	37.4	26.5	32.0
Claude-4-sonnet (Anthropic 2025)	62.2	46.3	54.3
<b>GLM Series (THUDM 2025)</b>			
GLM-Z1-9B-0414	63.0	51.9	57.5
GLM-Z1-32B-0414	70.5	59.0	64.8
<b>Qwen2.5 Series (Yang et al. 2024)</b>			
Qwen2.5-7B-Instruct	16.8	11.9	14.4
Qwen2.5-14B-Instruct	24.3	17.4	20.9
Qwen2.5-32B-Instruct	31.1	22.9	27.0
Qwen2.5-72B-Instruct	32.6	24.6	28.6
<b>Qwen3 Series (Yang et al. 2025)</b>			
Qwen3-8B	71.0	57.7	64.4
Qwen3-14B	78.1	65.0	71.6
Qwen3-32B	76.7	66.3	71.5
Qwen3-235B-A22B	82.0	71.2	76.6
<b>DeepSeek Series (Guo et al. 2025)</b>			
DeepSeek-R1-Distill-Qwen-14B	47.5	37.2	42.4
DeepSeek-R1-Distill-Qwen-32B	52.9	41.9	47.4
DeepSeek-R1-0528-Qwen3-8B	75.4	64.7	70.1
DeepSeek-R1-0528	<b>87.7</b>	80.2	84.0

Table 2: Baseline performance on PuzzleClone

## Baseline Results

We evaluate a wide range of models on the PuzzleClone test set, including both open-source and proprietary models. Similar to the training test, the test set is also divided into *Normal* and *Hard* subsets to reflect different levels of reasoning complexity. As shown in Table 2, current LLMs remain limited in handling complex logical reasoning.

Among the open-source models, the Qwen and GLM-Z1 series demonstrate strong performance trends as model size increases. Within the Qwen2.5 family, scores steadily improve from the 7B (14.4 average) to the 72B (28.6 average). Qwen3-8B achieves an average of 64.4, already outperforming many larger models from the previous generation. The Qwen3-14B and Qwen3-32B models achieve average scores of 71.6 and 71.5, respectively, while Qwen3-235B-A22B reaches 76.6, indicating substantial improvements from both scaling and architecture refinement.

The GLM-Z1 models also exhibit strong results. GLM-Z1-9B achieves 57.5 on average, while the larger GLM-Z1-32B outperforms it with a score of 64.8. These results suggest the GLM-Z1 series is competitive with Qwen3 in the same parameter range.

The DeepSeek-R1-Distill models based on Qwen show moderate performance (42.4 and 47.4), while the full DeepSeek-R1-0528-Qwen3-8B model attains a strong 70.1. The most powerful open-source model DeepSeek-R1-0528 achieves an impressive 84.0 average score, surpassing all other open models and even some proprietary models.

Among the proprietary models, ChatGPT-o3 leads (Avg. 84.4) with the smallest *Normal* to *Hard* drop (-4.8), evidencing strong, stable reasoning. Gemini-2.5-pro ranks second (Avg. 71.5) with moderate robustness, while Claude-

3-sonnet is mid-pack (Avg. 54.3) but degrades sharply on *Hard*. Non-flagship models perform notably worse, with Gemini-2.0-flash at 36.1, ChatGPT-4o at 27.7, and Claude-3.5-sonnet at 32.0.

The Qwen3 series outperforms many proprietary models. A key factor is language alignment. PuzzleClone is a Chinese dataset, and Qwen3’s pretraining and instruction tuning include rich Chinese corpora, giving it stronger lexical coverage, idiom handling, and domain term disambiguation. A case study of why Qwen3 excels vs. other proprietary models can be found in Appendix.

## Post Training

To evaluate the quality of the PuzzleClone dataset, we perform post-training (SFT and RL) on the Qwen2.5-7B-Instruct model. During RL (GRPO (Shao et al. 2024)) training, we use Qwen2.5-7B-Instruct as the base model to directly compare the effectiveness of SFT and RL training stages. As shown in Table 1, we use 74,354 training samples for the RL stage. For SFT, we first use DeepSeek-R1-0528 to generate thinking traces for 4,300 training samples to populate the reasoning context. We then apply rule-based filters to remove thinking traces with duplicated generations and multilingual reasoning content. After filtering, 3,574 samples remain for SFT.

To demonstrate its effectiveness in enhancing the logical reasoning capabilities of LLMs, we evaluate the model on a series of standard logic benchmarks, including SATBench (Wei et al. 2025a) and BBEH-mini (Kazemi et al. 2025), as well as mathematical benchmarks such as AIME24 (MAA and users 2024), AIME25 (MAA and users 2025), AMC2023 (MAA and users 2023), MATH500 (Hendrycks et al. 2021a), and OlympiadBench (He et al. 2024). These evaluations demonstrate the capability of SMT-based training data to enhance both logical reasoning and mathematical problem-solving performance.

The main results are presented in Table 3, demonstrating that model performance is significantly improved by training on the PuzzleClone dataset at both the SFT and RL stages.

Training on PuzzleClone brings large gains over Qwen2.5-7B-Instruct. The average score rises from 14.4 to 56.2 after SFT. RL also improves the baseline to 47.7 overall, showing better handling of difficult items than the baseline but behind SFT in absolute terms.

SFT substantially improves SATBench, from 52.8 to 65.0, though it slightly lowers BBEH-mini to 9.8. After RL training stage, SATBench raise to 57.0, while BBEH-mini climbs to 13.3. Interestingly, after the SFT stage, we observe a higher rate of duplicated generations, which likely explains the drop on BBEH-mini (from 11.3 to 9.8). This suggests a distributional skew introduced by SFT data. Adding more diverse SFT samples should rebalance the data and reduce the drop. SFT yields consistent gains across AIME24, AIME25, AMC2023, MATH500, and OlympiadBench. RL partially regresses these improvements, with lower scores than SFT on all five math sets.

Training Stage	PuzzleClone			Logic Benchmarks		Mathematical Benchmarks				
	Normal	Hard	Avg.	SATBench	BBEH-mini	AIME24	AIME25	AMC2023	MATH500	OlympiadBench
Qwen2.5-7B-Instruct	16.8	11.9	14.4	52.8	11.3	16.7	6.7	52.5	75.2	42.5
SFT	<b>63.5</b>	<b>49.0</b>	<b>56.2</b>	<b>65.0</b>	9.8	<b>20.0</b>	<b>23.3</b>	<b>65.0</b>	<b>81.2</b>	<b>46.4</b>
RL (GRPO)	51.8	43.5	47.7	57.0	<b>13.3</b>	13.3	13.3	57.5	77.6	41.2

Table 3: Performance of model trained with PuzzleClone data

## Related Work

### Data generation

Research on data generation mainly focus on *data synthesis* and *data augmentation* (Wang et al. 2024a).

Data synthesis approaches broadly start by extracting instructions from existing data sources or synthesizing them with rule-based or model-based synthesizers (Maiya et al. 2025; Ding et al. 2023; Xu et al. 2024b). Next, responses will be generated for each instruction instance, either manually or automatically. Machine learning has been widely adopted to accelerate the annotation process of various types of data (Zhang, Jafari, and Nagarkar 2021; Lu et al. 2023), despite their limitations in efficiency, data consistency, and domain knowledge (Tan et al. 2024). These limitations are addressed by advanced large language models (LLMs) like ChatGPT and Gemini, whose reasoning abilities make them powerful universal annotators (Zhang, Jafari, and Nagarkar 2021; Lu et al. 2023; Csandy et al. 2024; Zhang et al. 2023). Tan et al. provide a detailed survey of these methods (Tan et al. 2024). However, expert knowledge remains essential for accurate labels. Consequently, semi-automatic approaches have emerged, utilizing human-AI collaborative tools and co-labeling workflows to enhance efficiency and accuracy (Li et al. 2023; Wang et al. 2024b; Zhu et al. 2024).

By contrast, data augmentation focuses on adapting existing data items, called seeds, to new instances with similar structures. In natural language processing, traditional approaches mainly focus on paraphrasing the instructions, leveraging approaches like rule-based transformation and language models (Feng et al. 2021; Xu et al. 2024a; Sugiyama and Yoshinaga 2019; Bayer, Kaufhold, and Reuter 2022). With the advent of LLMs, modern approaches have explored involving LLMs throughout the pipeline (Wang et al. 2024a), allowing LLMs to automatically select promising seeds, generate data and self-improve their reasoning, fine-tune the instructions, and perform evaluation-based filtering to enhance data quality (Lu et al. 2024; Shah et al. 2024; Huang et al. 2024). Some works further involve human validators or LLM-generated scripts in symbolic languages for better accuracy (Leang et al. 2025; Shah et al. 2024). However, there remains a need for robust and scalable strategies that enable end-to-end verification and fine-grained controllability throughout the pipeline.

### Benchmarks for logical reasoning

Numerous public datasets for mathematical and logical reasoning have been collected through academic challenges and crowdsourcing (Hendrycks et al. 2021b; math ai 2025). Curated datasets, however, mostly focus on specific classes

of puzzles. For instance, numerous studies generate equation systems or deductive, inductive, and abductive puzzles by combining unit equations or basic logical clauses while crafting natural language instructions using templates or language models (Chen, Zhang, and Tao 2025; Parmar et al. 2024; Luo et al. 2023; Wei et al. 2025b; Mirzadeh et al. 2024). However, it becomes increasingly difficult to map the abstract logical model to a real-world scenario when it becomes difficult, which in turn jeopardizes the articulation of the instructions. By contrast, another line of research start from classical complicated and challenging mathematical puzzles such as sudoku, hitori, and crosswords, and curate a number of similar puzzles simply through rule-based methods (Gui et al. 2024; Mittal et al. 2024; Chen et al. 2025b,a; Li et al. 2025; Liu et al. 2025b,a). However, a universal framework capable of generating various challenging puzzles is still absent. To address this, we propose the PuzzleClone framework and mainly focuses on SMT puzzles, a representative NP-complete problems, which is a superset of the focus of many existing datasets (e.g., SAT, inductive/deductive reasoning) and also a challenging class of problems to which many mathematical problems can be reduced.

## Discussion

In the process of designing and implementing the PuzzleClone framework, as well as synthesizing variants from seed puzzles, we encountered several key insights, reflections, and limitations worth discussing. These are outlined below.

**Diverse Question and Evaluation Types:** Unlike some existing datasets that typically focus on a single task format, our dataset features a wide range of puzzle types and answer formats (see Section 3.3). Notably, a single puzzle in our dataset may contain multiple sub-questions, each with its own `qtype` and `eval_type`, which further enhances the overall richness and versatility of the dataset. Such diversity enables more comprehensive benchmarking, supports evaluation of specific reasoning capabilities, and promotes model robustness across heterogeneous formats—better reflecting real-world problem-solving scenarios.

**Specification Strategies:** We observed two distinct approaches to puzzle synthesis based on the specification: forward and backward generation. In the forward strategy (e.g., specs for Figure 2, 15-tabletennis, and 23-product), we first define random domains for variables and use symbolic constraints to allow the Z3 solver (De Moura and Bj  rner 2008) to search for feasible solutions. In contrast, the backward strategy (e.g., specs for 7-age and 11-wine) involves randomly generating a feasible solution first, then building symbolic constraints around it to verify its correctness and uniqueness using the

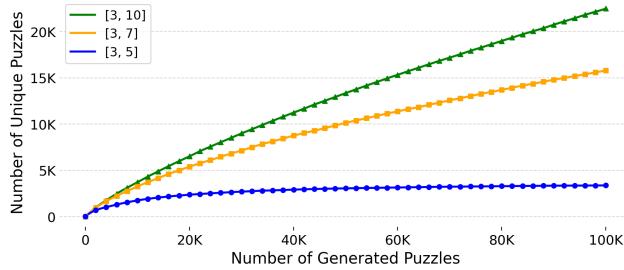


Figure 4: Number of valid puzzles (after de-duplication) synthesized from the seed puzzle 9-vase using three different `p_num` domain settings.

solver. The backward approach significantly improves generation efficiency by avoiding unsolvable instances due to random variable combinations. Therefore, we recommend using the backward strategy for most specification designs.

**Hyperparameter Sensitivity:** While writing specifications, careful tuning of hyperparameters such as variable domains, constraint templates, and their count ranges is crucial to maintain the semantic soundness of generated puzzles. For example, in 2-graduation, where  $m$  out of  $n$  students are selected under constraints, it is necessary to enforce  $0 < m < n$ . However, making  $m$  too small or too large ( $m \rightarrow 0$  or  $m \rightarrow n$ ) reduces puzzle difficulty. Thus, domain ranges should not only be logically valid but also aligned with the intended cognitive challenge.

**Revisiting Difficulty Estimation:** Our current difficulty scoring (Section 3.4) has two shortcomings. First, the `var_scale` metric estimates how variable domains affect difficulty, but uses a simplistic `diff_factor` with only three discrete values to represent correlation direction. This fails to capture nuanced relationships. In future work, we aim to introduce variable-specific weights. Second, not all difficulty-variable relationships are monotonic. For instance, in 2-graduation, as  $m$  increases, puzzle difficulty follows a non-linear trend—first increasing then decreasing. Similarly, the `cond_num` metric may not always correlate with increased difficulty. For example, in 23-product, more constraints can actually make the puzzle easier by narrowing down the solution space more precisely.

**Expressiveness for Optimization Tasks:** Although PuzzleClone supports synthesis of puzzles involving optimization objectives, it is limited to linear optimization due to the capabilities of Z3. This restricts the types of puzzles our system can synthesize. Extending support to nonlinear optimization remains an open challenge.

**Puzzle Difficulty Distribution:** When variable domains are large and constraints are complex, the probability of randomly generating solvable instances diminishes. This leads to a distributional bias in synthesized puzzles (see Figure 3): easier puzzles (associated with simpler parameter combinations) are overrepresented, while harder ones are undersampled. To remedy this, we propose developing dedicated specifications aimed explicitly at generating hard puzzles.

**Upper Bound of Synthesizable Puzzles:** While Puzzle-

Clone can synthesize puzzles at scale, each specification inherently has a finite upper bound on the number of valid (i.e., non-duplicate) puzzles it can generate. This upper bound is determined by the size of the puzzle space defined in the specification. As shown in Figure 4, for 9-vase, we synthesized variants using three specs with decreasing `p_num` domains: [3,10], [3,7], and [3,5]. After generating 100K puzzles for each setting and performing de-duplication, we observed that [3,5] quickly saturated at around 3.3K unique puzzles, while [3,10] continued to grow steadily. The [3,7] setting fell in between and showed signs of approaching saturation. This demonstrates that a larger solution space allows the synthesis of more diverse valid puzzles.

**Preventing Redundancy in Puzzle Statements:** It is critical to ensure that the question and its constraints do not inadvertently leak the answer. For instance, in 23-product, if a condition states that a specific product is in position  $i$  and the question also asks about the position of that product, the answer becomes trivially inferable without reasoning. Such overlap undermines the puzzle’s intent. Specifications should explicitly avoid this redundancy to preserve reasoning difficulty.

**Advanced Features for Practical Synthesis:** To enhance expressiveness and practical utility, PuzzleClone supports several advanced features:

- *Coupled Logic:* Supports joint control of variables and constraints (e.g.,  $A + B = 10$ ), enabling fine-grained coordination beyond independent random sampling.
- *Custom Operators:* Users can define new functions or import scripts (e.g., for A-series puzzles), compensating for limitations in built-in operators.
- *Dynamic Rephrasing:* PuzzleClone supports modifying the textual surface of puzzles (see Appendix), enabling translation into multiple languages or thematic adaptation, thereby increasing data diversity.

## Conclusion and Future Work

We present PuzzleClone, a formal data synthesis framework and a diverse and challenging dataset containing over 83K SMT-verifiable puzzles. Experiments demonstrate that existing state-of-the-art models face significant challenges on this benchmark. Furthermore, SFT and RL applied to Qwen2.5-7B-Instruct show substantial gains in mathematical reasoning, validating the effectiveness of our approach. Looking forward, several promising directions remain open:

- *Automatic puzzle enrichment:* To enrich the diversity of our dataset, future work could explore how to prompt LLMs to automatically rephrase puzzle descriptions without altering their underlying solutions. Beyond rephrasing, we aim to expand puzzle complexity by enabling LLMs to autonomously propose new variables, constraints, or questions grounded in the seed puzzle logic, akin to human brainstorming in problem design.
- *Automatic spec construction:* Another goal is to enable LLMs to parse puzzle texts and generate formal specs, including variable decomposition, range definition, and constraint arrangement. This could significantly reduce the manual effort required in authoring seed puzzles.

- *Towards seed-free generation:* Ultimately, we envision generating puzzles without any seed inputs, by composing constraints directly from a library of logical operators. Achieving this would require improved program synthesis capabilities, possibly in conjunction with interactive or visual interfaces to assist human-in-the-loop validation and correction.

## References

- Anthropic. 2024. Introducing Claude 3.5 Sonnet. <https://www.anthropic.com/news/clause-3-5-sonnet>. Announcement; see also model card addendum.
- Anthropic. 2025. Claude Opus 4 & Claude Sonnet 4: System Card. <https://www.anthropic.com/clause-4-system-card>. Model referenced in paper as Claude-4-sonnet.
- Barrett, C.; Stump, A.; Tinelli, C.; et al. 2010. The smt-lib standard: Version 2.0. In *Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, UK)*, volume 13, 14.
- Bayer, M.; Kaufhold, M.-A.; and Reuter, C. 2022. A survey on data augmentation for text classification. *ACM Computing Surveys*, 55(7): 1–39.
- Chen, G.; Xu, W.; Zhang, H.; Chan, H. P.; Liu, C.; Bing, L.; Zhao, D.; Luu, A. T.; and Rong, Y. 2025a. FINEREA-SON: Evaluating and Improving LLMs’ Deliberate Reasoning through Reflective Puzzle Solving. ArXiv:2502.20238 [cs].
- Chen, J.; Wei, Z.; Ren, Z.; Li, Z.; and Zhang, J. 2025b. LR<sup>2</sup>Bench: Evaluating Long-chain Reflective Reasoning Capabilities of Large Language Models via Constraint Satisfaction Problems. ArXiv:2502.17848 [cs].
- Chen, M. K.; Zhang, X.; and Tao, D. 2025. JustLogic: A Comprehensive Benchmark for Evaluating Deductive Reasoning in Large Language Models. *arXiv preprint arXiv:2501.14851*.
- Clocksin, W. F.; and Mellish, C. S. 2003. *Programming in PROLOG*. Springer Science & Business Media.
- Csanady, B.; Muzsai, L.; Vedres, P.; Nadasdy, Z.; and Lukacs, A. 2024. LlamBERT: Large-scale low-cost data annotation in NLP. *arXiv preprint arXiv:2403.15938*.
- De Moura, L.; and Bj rner, N. 2008. Z3: an efficient SMT solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS’08/ETAPS’08*, 337–340. Berlin, Heidelberg: Springer-Verlag. ISBN 3540787992.
- Ding, N.; Chen, Y.; Xu, B.; Qin, Y.; Zheng, Z.; Hu, S.; Liu, Z.; Sun, M.; and Zhou, B. 2023. Enhancing chat language models by scaling high-quality instructional conversations. *arXiv preprint arXiv:2305.14233*.
- Feng, S. Y.; Gangal, V.; Wei, J.; Chandar, S.; Vosoughi, S.; Mitamura, T.; and Hovy, E. 2021. A Survey of Data Augmentation Approaches for NLP. In Zong, C.; Xia, F.; Li, W.; and Navigli, R., eds., *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, 968–988. Online: Association for Computational Linguistics.
- Gemini Team. 2025a. Gemini 2.0 Flash. <https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash>. Model referenced in paper as Gemini-2.0-flash.
- Gemini Team. 2025b. Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities. Technical report, Google DeepMind. Covers Gemini 2.5 Pro (and 2.5 Flash); model referenced in paper as Gemini-2.5-pro.
- Gui, J.; Liu, Y.; Cheng, J.; Gu, X.; Liu, X.; Wang, H.; Dong, Y.; Tang, J.; and Huang, M. 2024. Logicgame: Benchmarking rule-based reasoning abilities of large language models. *arXiv preprint arXiv:2408.15778*.
- Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; Bi, X.; Zhang, X.; Yu, X.; Wu, Y.; Wu, Z.; Gou, Z.; Shao, Z.; Li, Z.; Gao, Z.; DeepSeek-AI Team; et al. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948*. Covers DeepSeek-R1-R1-Zero and distilled variants.
- He, C.; Luo, R.; Bai, Y.; Hu, S.; Thai, Z. L.; Shen, J.; Hu, J.; Han, X.; Huang, Y.; Zhang, Y.; et al. 2024. Olympiad-bench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*.
- Hendrycks, D.; Burns, C.; Kadavath, S.; Arora, A.; Basart, S.; Tang, E.; Song, D.; and Steinhardt, J. 2021a. Measuring Mathematical Problem Solving With the MATH Dataset. *arXiv preprint arXiv:2103.03874*.
- Hendrycks, D.; Burns, C.; Kadavath, S.; Arora, A.; Basart, S.; Tang, E.; Song, D.; and Steinhardt, J. 2021b. Measuring Mathematical Problem Solving With the MATH Dataset. *NeurIPS*.
- Huang, Y.; Wu, S.; Gao, C.; Chen, D.; Zhang, Q.; Wan, Y.; Zhou, T.; Xiao, C.; Gao, J.; Sun, L.; et al. 2024. Datagen: Unified synthetic dataset generation via large language models. In *The Thirteenth International Conference on Learning Representations*.
- Kazemi, M.; Fatemi, B.; Bansal, H.; Palowitch, J.; Anastasiou, C.; Mehta, S.; Jain, L. D.; Aglietti, V.; Jindal, D.; Chen, P.; Dikkala, N.; Tyen, G.; Liu, X.; Shalit, U. S.; Tay, Y.; Tran, V. Q.; Le, Q. V.; and Firat, O. 2025. BIG-Bench Extra Hard. *arXiv preprint arXiv:2502.19187*.
- Leang, J. O. J.; Hong, G.; Li, W.; and Cohen, S. B. 2025. Theorem Prover as a Judge for Synthetic Data Generation.
- Li, M.; Shi, T.; Ziems, C.; Kan, M.-Y.; Chen, N. F.; Liu, Z.; and Yang, D. 2023. Coannotating: Uncertainty-guided work allocation between human and large language models for data annotation. *arXiv preprint arXiv:2310.15638*.
- Li, N.; Liu, P.; Liu, Z.; Dai, T.; Jiang, Y.; and Xia, S.-T. 2025. Logic-of-Thought: Empowering Large Language Models with Logic Programs for Solving Puzzles in Natural Language. *arXiv preprint arXiv:2505.16114*.
- Liu, J.; Fan, Y.; Jiang, Z.; Ding, H.; Hu, Y.; Zhang, C.; Shi, Y.; Weng, S.; Chen, A.; Chen, S.; Huang, Y.; Zhang, M.; Zhao, P.; Yan, J.; and He, J. 2025a. SynLogic: Synthesizing Verifiable Reasoning Data at Scale for Learning Logical Reasoning and Beyond. *arXiv:2505.19641*.

- Liu, J.; Lin, X.; Bayer, J.; Dillies, Y.; Jiang, W.; Liang, X.; Soletskyi, R.; Wang, H.; Xie, Y.; Xiong, B.; et al. 2025b. CombiBench: Benchmarking LLM capability for combinatorial mathematics. *arXiv preprint arXiv:2505.03171*.
- Lu, Y.; Shen, M.; Wang, H.; Wang, X.; van Rechem, C.; Fu, T.; and Wei, W. 2023. Machine learning for synthetic data generation: a review. *arXiv preprint arXiv:2302.04062*.
- Lu, Z.; Zhou, A.; Ren, H.; Wang, K.; Shi, W.; Pan, J.; Zhan, M.; and Li, H. 2024. MathGenie: Generating Synthetic Data with Question Back-translation for Enhancing Mathematical Reasoning of LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2732–2747. Stroudsburg, PA, USA: Association for Computational Linguistics.
- Luo, M.; Kumbhar, S.; Parmar, M.; Varshney, N.; Banerjee, P.; Aditya, S.; Baral, C.; et al. 2023. Towards logilue: A brief survey and a benchmark for analyzing logical reasoning capabilities of language models. *arXiv preprint arXiv:2310.00836*.
- MAA; and users, B. 2023. AMC 2023 benchmark (American Mathematics Competition 2023). Used as evaluation benchmark in LLM reasoning research.
- MAA; and users, B. 2024. AIME 2024 benchmark (American Invitational Mathematics Examination 2024). Used as evaluation benchmark in LLM reasoning research.
- MAA; and users, B. 2025. AIME 2025 benchmark (American Invitational Mathematics Examination 2025). Used in recent benchmarks like SATBench analysis, etc.
- Maiya, A.; Alghamdi, R.; Pacheco, M. L.; Trivedi, A.; and Somenzi, F. 2025. Explaining Puzzle Solutions in Natural Language: An Exploratory Study on 6x6 Sudoku. *arXiv preprint arXiv:2505.15993*.
- math ai. 2025. AIME25 Dataset. Accessed: 2025-07-01.
- Mirzadeh, I.; Alizadeh, K.; Shahrokhi, H.; Tuzel, O.; Bengio, S.; and Farajtabar, M. 2024. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*.
- Mittal, C.; Kartik, K.; Mausam; and Singla, P. 2024. PuzzleBench: Can LLMs Solve Challenging First-Order Combinatorial Reasoning Problems? ArXiv:2402.02611 [cs] version: 2.
- OpenAI. 2024. GPT-4o System Card. <https://arxiv.org/abs/2410.21276>. Model referenced in paper as ChatGPT-4o.
- OpenAI. 2025. OpenAI o3 and o4-mini System Card. <https://openai.com/index/o3-o4-mini-system-card/>. Model referenced in paper as ChatGPT-o3.
- Pan, L.; Albalak, A.; Wang, X.; and Wang, W. 2023. LogicLM: Empowering Large Language Models with Symbolic Solvers for Faithful Logical Reasoning. In Bouamor, H.; Pino, J.; and Bali, K., eds., *Findings of the Association for Computational Linguistics: EMNLP 2023*, 3806–3824. Singapore: Association for Computational Linguistics.
- Parmar, M.; Patel, N.; Varshney, N.; Nakamura, M.; Luo, M.; Mashetty, S.; Mitra, A.; and Baral, C. 2024. LogicBench: Towards systematic evaluation of logical reasoning ability of large language models. *arXiv preprint arXiv:2404.15522*.
- Shah, V.; Yu, D.; Lyu, K.; Park, S.; Ke, N. R.; Mozer, M.; Bengio, Y.; Arora, S.; and Goyal, A. 2024. AI-Assisted Generation of Difficult Math Questions.
- Shao, Z.; Wang, P.; Zhu, Q.; Xu, R.; Song, J.; Bi, X.; Zhang, H.; Zhang, M.; Li, Y. K.; Wu, Y.; and Guo, D. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint arXiv:2402.03300*. Introduces Group Relative Policy Optimization (GRPO).
- Sugiyama, A.; and Yoshinaga, N. 2019. Data augmentation using back-translation for context-aware neural machine translation. In *Proceedings of the fourth workshop on discourse in machine translation (DiscoMT 2019)*, 35–44.
- Tan, Z.; Li, D.; Wang, S.; Beigi, A.; Jiang, B.; Bhattacharjee, A.; Karami, M.; Li, J.; Cheng, L.; and Liu, H. 2024. Large Language Models for Data Annotation and Synthesis: A Survey. In Al-Onaizan, Y.; Bansal, M.; and Chen, Y.-N., eds., *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 930–957. Miami, Florida, USA: Association for Computational Linguistics.
- THUDM. 2025. GLM-Z1-32B-0414. <https://huggingface.co/THUDM/GLM-Z1-32B-0414>. GLM-Z1 reasoning model series (incl. 9B/32B); model card.
- Wang, K.; Zhu, J.; Ren, M.; Liu, Z.; Li, S.; Zhang, Z.; Zhang, C.; Wu, X.; Zhan, Q.; Liu, Q.; and Wang, Y. 2024a. *A Survey on Data Synthesis and Augmentation for Large Language Models*, volume 1. Association for Computing Machinery.
- Wang, Y.; Stevens, D.; Shah, P.; Jiang, W.; Liu, M.; Chen, X.; Kuo, R.; Li, N.; Gong, B.; Lee, D.; et al. 2024b. Model-in-the-Loop (MILO): Accelerating Multimodal AI Data Annotation with LLMs. *arXiv preprint arXiv:2409.10702*.
- Wei, A.; Wu, Y.; Wan, Y.; Suresh, T.; Tan, H.; Zhou, Z.; Koyejo, S.; Wang, K.; and Aiken, A. 2025a. SATBench: Benchmarking LLMs’ Logical Reasoning via Automated Puzzle Generation from SAT Formulas. *arXiv preprint arXiv:2505.14615*.
- Wei, A.; Wu, Y.; Wan, Y.; Suresh, T.; Tan, H.; Zhou, Z.; Koyejo, S.; Wang, K.; and Aiken, A. 2025b. SATBench: Benchmarking LLMs’ Logical Reasoning via Automated Puzzle Generation from SAT Formulas. *arXiv preprint arXiv:2505.14615*.
- Xu, C.; Sun, Q.; Zheng, K.; Geng, X.; Zhao, P.; Feng, J.; Tao, C.; Lin, Q.; and Jiang, D. 2024a. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations*.
- Xu, Z.; Jiang, F.; Niu, L.; Deng, Y.; Poovendran, R.; Choi, Y.; and Lin, B. Y. 2024b. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing. *arXiv preprint arXiv:2406.08464*.
- Yang, A.; Li, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Gao, C.; Huang, C.; Lv, C.; Zheng, C.; Liu, D.; Zhou, F.; Huang, F.; Hu, F.; Ge, H.; Wei, H.; Lin, H.; Tang, J.; Yang, J.; Tu, J.; Zhang, J.; Yang, J.; Yang, J.; Zhou, J.; Zhou, J.; Lin, J.; Dang, K.; Bao, K.; Yang, K.; Yu, L.; Deng, L.; Li, M.; Xue, M.; Li, M.; Zhang, P.; Wang, P.; Zhu, Q.; Men, R.; Gao, R.; Liu, S.; Luo, S.; Li, T.; Tang, T.; Yin, W.; Ren,

X.; Wang, X.; Zhang, X.; Ren, X.; Fan, Y.; Su, Y.; Zhang, Y.; Zhang, Y.; Wan, Y.; Liu, Y.; Wang, Z.; Cui, Z.; Zhang, Z.; Zhou, Z.; Qiu, Z.; and the Qwen Team. 2025. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*.

Yang, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Li, C.; Liu, D.; Huang, F.; Wei, H.; Lin, H.; Yang, J.; Tu, J.; Zhang, J.; Yang, J.; Yang, J.; Zhou, J.; Lin, J.; Dang, K.; Lu, K.; Bao, K.; Yang, K.; Yu, L.; Li, M.; Xue, M.; Zhang, P.; Zhu, Q.; Men, R.; Lin, R.; Li, T.; Tang, T.; Xia, T.; Ren, X.; Ren, X.; Fan, Y.; Su, Y.; Zhang, Y.; Wan, Y.; Liu, Y.; Cui, Z.; Zhang, Z.; Qiu, Z.; and the Qwen Team. 2024. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115*.

Zhang, R.; Li, Y.; Ma, Y.; Zhou, M.; and Zou, L. 2023. Ll-maaa: Making large language models as active annotators. *arXiv preprint arXiv:2310.19596*.

Zhang, S.; Jafari, O.; and Nagarkar, P. 2021. A survey on machine learning techniques for auto labeling of video, audio, and text data. *arXiv preprint arXiv:2109.03784*.

Zhu, Z.; Wang, Y.; Yang, S.; Long, L.; Wu, R.; Tang, X.; Zhao, J.; and Wang, H. 2024. CORAL: Collaborative Automatic Labeling System Based on Large Language Models. *Proc. VLDB Endow.*, 17(12): 4401–4404.

## Reproducibility Checklist

### Instructions for Authors:

This document outlines key aspects for assessing reproducibility. Please provide your input by editing this `.tex` file directly.

For each question (that applies), replace the “Type your response here” text with your answer.

#### Example: If a question appears as

```
\question{Proofs of all novel claims  
are included} {yes/partial/no}
```

Type your response here

you would change it to:

```
\question{Proofs of all novel claims  
are included} {yes/partial/no}
```

yes

Please make sure to:

- Replace ONLY the “Type your response here” text and nothing else.
- Use one of the options listed for that question (e.g., **yes**, **no**, **partial**, or **NA**).
- **Not** modify any other part of the `\question` command or any other lines in this document.

You can `\input` this `.tex` file right before `\end{document}` of your main file or compile it as a stand-alone document. Check the instructions on your conference’s website to see if you will be asked to provide this checklist with your paper or separately.

### 1. General Paper Structure

1.1. Includes a conceptual outline and/or pseudocode description of AI methods introduced (yes/partial/no/NA) **yes**

1.2. Clearly delineates statements that are opinions, hypothesis, and speculation from objective facts and results (yes/no) **yes**

1.3. Provides well-marked pedagogical references for less-familiar readers to gain background necessary to replicate the paper (yes/no) **yes**

### 2. Theoretical Contributions

2.1. Does this paper make theoretical contributions? (yes/no) **no**

If yes, please address the following points:

2.2. All assumptions and restrictions are stated clearly and formally (yes/partial/no) [Type your response here](#)

2.3. All novel claims are stated formally (e.g., in theorem statements) (yes/partial/no) [Type your response here](#)

2.4. Proofs of all novel claims are included (yes/partial/no) [Type your response here](#)

2.5. Proof sketches or intuitions are given for complex and/or novel results (yes/partial/no) [Type your response here](#)

2.6. Appropriate citations to theoretical tools used are given (yes/partial/no) [Type your response here](#)

2.7. All theoretical claims are demonstrated empirically to hold (yes/partial/no/NA) [Type your response here](#)

2.8. All experimental code used to eliminate or disprove claims is included (yes/no/NA) [Type your response here](#)

### 3. Dataset Usage

3.1. Does this paper rely on one or more datasets? (yes/no) **yes**

If yes, please address the following points:

3.2. A motivation is given for why the experiments are conducted on the selected datasets (yes/partial/no/NA) **yes**

3.3. All novel datasets introduced in this paper are included in a data appendix (yes/partial/no/NA) **yes**

3.4. All novel datasets introduced in this paper will be made publicly available upon publication of the paper with a license that allows free usage for research

purposes (yes/partial/no/NA) **yes**

3.5. All datasets drawn from the existing literature (potentially including authors' own previously published work) are accompanied by appropriate citations (yes/no/NA) **yes**

3.6. All datasets drawn from the existing literature (potentially including authors' own previously published work) are publicly available (yes/partial/no/NA) **yes**

3.7. All datasets that are not publicly available are described in detail, with explanation why publicly available alternatives are not scientificallyatisficing (yes/partial/no/NA) **NA**

4.10. This paper states the number of algorithm runs used to compute each reported result (yes/no) **yes**

4.11. Analysis of experiments goes beyond single-dimensional summaries of performance (e.g., average; median) to include measures of variation, confidence, or other distributional information (yes/no) **yes**

4.12. The significance of any improvement or decrease in performance is judged using appropriate statistical tests (e.g., Wilcoxon signed-rank) (yes/partial/no) **yes**

4.13. This paper lists all final (hyper-)parameters used for each model/algorithm in the paper's experiments (yes/partial/no/NA) **yes**

## 4. Computational Experiments

4.1. Does this paper include computational experiments? (yes/no) **yes**

If yes, please address the following points:

4.2. This paper states the number and range of values tried per (hyper-) parameter during development of the paper, along with the criterion used for selecting the final parameter setting (yes/partial/no/NA) **yes**

4.3. Any code required for pre-processing data is included in the appendix (yes/partial/no) **no**

4.4. All source code required for conducting and analyzing the experiments is included in a code appendix (yes/partial/no) **yes**

4.5. All source code required for conducting and analyzing the experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes (yes/partial/no) **yes**

4.6. All source code implementing new methods have comments detailing the implementation, with references to the paper where each step comes from (yes/partial/no) **yes**

4.7. If an algorithm depends on randomness, then the method used for setting seeds is described in a way sufficient to allow replication of results (yes/partial/no/NA) **NA**

4.8. This paper specifies the computing infrastructure used for running experiments (hardware and software), including GPU/CPU models; amount of memory; operating system; names and versions of relevant software libraries and frameworks (yes/partial/no) **yes**

4.9. This paper formally describes evaluation metrics used and explains the motivation for choosing these metrics (yes/partial/no) **yes**

## Appendix

### Supplement to Dataset Construction

This supplement provides additional details about (1) the prompt templates used for seed puzzle selection (Figure 5) and downstream experiments (Figure 6), and (2) the distribution of duplicate instances across seed puzzles (Figure 7), as referenced in Section 3 of the main paper. Note that each template is presented in both its original Chinese form (used in practice) and its English translation (for readability).

### DSL Schema

This appendix provides a detailed specification of the Domain-Specific Language (DSL) used for puzzle encoding and generation. The DSL is structured around a main `PuzzleTemplate` object, which integrates all necessary components for defining a puzzle’s logic, parameters, and text. The following tables detail each of the core components.

**PuzzleTemplate** The root object, `PuzzleTemplate`, is the top-level container that orchestrates all other elements of the puzzle definition. Its fields are detailed in Table 4.

**Variables** The `Variable` object defines the parameters of a puzzle. A variable can be defined either by its type and domain or by a formula, as shown in Table 5.

**Symbols** Symbols represent the quantities to be solved. They can be directly defined (`DefinedSymbol`) or derived from existing variables (`DerivedSymbol`).

**Conditions** Conditions define the logical constraints of the puzzle, as either static (`StaticCondition`) or dynamically generated (`DynamicCondition`) rules.

**Queries** Queries define the questions posed to the user, supporting open-ended and multiple-choice formats.

**Auxiliary Definitions** The DSL includes objects for post-processing and optimization tasks.

In addition, `PuzzleClone` supports several internal operators and reserved words for expressiveness. For instance, `_opt` and `_sym` refer to the parameter randomization result of each option and symbol, while `get_faker` is an internal operator for fake entity generation.

### Puzzle Specification Examples

This section provides the complete DSL specification files for three representative puzzles from our benchmark. All descriptive texts, originally in Chinese, have been translated to English.

- 1-hamburger (Figure 2): See Listings 1 and 2.
- 2–graduation: See Listing 3.
- 9–vase: See Listing 4.
- 11–wine: See Listing 5.
- 23–product: See Listing 6.
- 28–exam See Listing 7.

### Training Parameters

Table 14 outlines SFT setup: 32K context, cosine LR schedule with warmup, bf16/tf32, gradient checkpointing, ZeRO-3 Offload via DeepSpeed, six epochs, small per-device batch with accumulation, and step-based checkpointing.

Table 15 summarizes key GRPO hyperparameters: KL regularization, vLLM rollouts, fsdp2 distributed training, Qwen2.5-7B-Instruct, five epochs, sixteen H100 across two nodes, dynamic batching and token-limits.

### Supplement to Experimental Results

Figure 8 presents the average accuracy of all evaluated models on the `PuzzleClone` test set, grouped by their originating seed puzzles. For most seed puzzles, the *hard* variants yield lower average accuracy compared to the *normal* ones, demonstrating the effectiveness of our difficulty stratification. However, for certain seed puzzles, the *hard* subset shows unexpectedly higher accuracy. This counterintuitive trend reveals limitations in our current difficulty scoring method, which may not fully capture the underlying reasoning complexity. For a more detailed analysis, please refer to the Revisiting Difficulty Estimation part in Section 6.

### Details of Dynamic Rephrasing

This section introduces *dynamic rephrasing*, a key functionality of `PuzzleClone` that enables transforming the original puzzle into new languages or scenarios in a fully accurate and verifiable way.

Given an original puzzle  $P$  generated by a pair of specification and configuration files ( $Q_s$  and  $Q_c$ ), `PuzzleClone` enables adapting  $P$  to a new Puzzle  $P'$  based on a new specification file  $Q'_s$ , where  $Q'_s$  and  $Q_s$  differ only in the descriptive texts. During generation, the values of all variables and parameters of symbols, conditions, and queries will be directly extracted from  $Q_c$  instead of through randomization. This new puzzle can be validated in a similar manner to the reproduced seed puzzle, ensuring its validity.

It is important to note that in a few cases, the values of some variables or parameters still need to be randomized, such as the names of people mentioned in the puzzle randomized by Faker. To accommodate such cases, `PuzzleClone` enables *partial config-based generation*, enabling users to specify variables for randomization through command-line options.

Field	Type	Description
custom_operator	dict (opt)	A dictionary of custom operators. The key is the operator name (e.g., "double"), and the value is a string containing either a Python lambda function (e.g., "lambda x: x*2") or the path to a Python file defining the operator.
variables	dict	A dictionary defining the puzzle's parameters. The key is the variable name and the value is a Variable object. See Table 5 for details.
symbols	dict (opt)	A dictionary defining the symbols to be solved. Keys are symbol group names. Values can be DefinedSymbol, DerivedSymbol, or DerivedSymbols objects. See Tables 6 and 7.
conditions	dict (opt)	A dictionary of constraints. Keys are condition names. Values can be StaticCondition or DynamicCondition objects. See Tables 8 and 9.
calc_solution	bool	If true (default), the solver will compute the set of valid solutions for the generated puzzle instance.
max_solution	int	The maximum number of valid symbol configurations to find. If the solver exceeds this limit (default 6000), it halts. This refers to satisfying all constraints, not the final answer to a query.
post-generation	PostGen (opt)	Defines operations to be performed after the initial solution is found, such as deriving new variables from the solution. See Table 12.
optimize	Optimize (opt)	Defines an optimization objective for problems that require minimizing or maximizing a certain value. See Table 13.
queries	dict (opt)	A dictionary of questions for the puzzle. The key is the query name and the value is a Query or a selection-based query object. See Tables 10 and 11.
desc	str	A natural language template for the puzzle's introductory text, which can include placeholders for variables.

Table 4: Schema for the PuzzleTemplate Root Object.

Field	Type	Description
type	str (opt)	The data type of the variable, e.g., "int", "bool". Must be defined if formula is not defined.
domain	str (opt)	A string representing the variable's value space. Examples: "[1, 10]" for integers, "['red', 'blue']" for string options. Must be defined if formula is not defined.
formula	str (opt)	A Python expression string to compute the variable's value. Example: "randint(1,6) + randint(1,6)". If defined, type and domain must be omitted.

Table 5: Schema for the Variable Object.

**The prompt for seed puzzle selection**

**CHN**

我是一名数据科学研究员，想通过一道种子题目合成大量与之类似的衍生题。而你是一个数学逻辑问题分析专家，你需要帮助我判断一道题是否适合作为种子题。

**### 判断依据**

1. 是否存在变量值或逻辑条件可以随机替换，并且不会改变题目的语义？
2. 是否能将该题目转换为 SMT（可满足性模理论）问题，并通过 SMT 求解器（如 z3）在可接受的时间和内存范围内求解？

如果以上两个条件均成立，请返回 `is\_seed\_puzzle` 为 `True`，同时返回 `code` 为你的 python 求解代码；否则返回 `is\_seed\_puzzle` 为 `False`，同时返回 `code` 为 `null`。  
`reason` 为你的判断依据，当 `is\_seed\_puzzle` 为 `False` 时，需要在 `reason` 中说明不符合哪项依据。  
例如：给定一个问题和答案：

```
```json
{
  "question": "有一个正整数x加上13是一个平方数，x加上5是一个立方数，那么x的值是多少？",
  "answer": "3"
}
```
期望的回答为：
```

```
```json
{
  "is_seed_puzzle": true,
  "reason": "该题目中存在可以随机替换的变量值（如13和5）和逻辑条件（平方数和立方数），且可以通过SMT求解器在可接受的时间和内存范围内求解。",
  "code": "from z3 import *\ninx = Int('x')\nsolver = Solver()\n# 定义条件\nnsolver.add(x > 0)\nnsolver.add(Exists([y], y * y == x + 13))\nnsolver.add(Exists([z], z * z * z == x + 5))\n\n# 求解\nif solver.check() == sat:\n    m = solver.model()\n    print(m[x])\nelse:\n    print(\"No solution found\")"
}
```
### 待分析的题目
```

```
```json
{
  "question": {question},
  "answer": {answer},
}
```
### 回复格式
```

请直接返回一个 JSON 对象，格式如下：

```
```json
{
  "is_seed_puzzle": <true or false>,
  "reason": <reason>,
  "code": <python code or null>
}
```
注：请确保输出仅为 JSON 数据，不要返回其他文本或解释。
```

**ENG**

I am a data science researcher aiming to generate a large number of derivative puzzles from a single seed puzzle. You are an expert in mathematical logic problem analysis, and your task is to determine whether a given puzzle is suitable to be used as a seed puzzle.

**### Evaluation Criteria**

1. Are there variable values or logical conditions in the puzzle that can be randomly replaced without altering the semantics of the question?
2. Can the puzzle be translated into an SMT (Satisfiability Modulo Theories) problem and solved using an SMT solver (e.g., Z3) within tractable time and memory bounds?

If both criteria are satisfied, return `is\_seed\_puzzle` as `true`, and provide the corresponding Python code in `code`. Otherwise, return `is\_seed\_puzzle` as `false`, and set `code` to `null`. The `reason` field should contain your justification, and if `is\_seed\_puzzle` is `false`, please specify which criterion is not satisfied.

For example, given the following question and answer:

```
```json
{
  "question": "A positive integer x plus 13 is a perfect square, and x plus 5 is a perfect cube. What is the value of x?",
  "answer": "3"
}
```
The expected response would be:
```

```
```json
{
  "is_seed_puzzle": true,
  "reason": "The puzzle contains variable values (e.g., 13 and 5) and logical conditions (perfect square and cube) that can be randomly replaced without changing the semantics. It can also be solved within reasonable time and memory limits using an SMT solver.",
  "code": "from z3 import *\ninx = Int('x')\nsolver = Solver()\n# Define the constraints\nnsolver.add(x > 0)\nnsolver.add(Exists([y], y * y == x + 13))\nnsolver.add(Exists([z], z * z * z == x + 5))\n\n# Solve\nif solver.check() == sat:\n    m = solver.model()\n    print(m[x])\nelse:\n    print(\"No solution found\")"
}
```
### Puzzle to Analyze
```

```
```json
{
  "question": {question},
  "answer": {answer},
}
```
### Response Format
```

Please respond with a JSON object in the following format:

```
```json
{
  "is_seed_puzzle": <true or false>,
  "reason": <reason>,
  "code": <python code or null>
}
```
Note: Ensure the output is strictly a JSON object, without any additional text or explanation.
```

Figure 5: Seed Puzzle Selection Prompt for filtering seed puzzles via the Qwen2.5-72B-Instruct model.

| Field  | Type                    | Description   |
|--------|-------------------------|---|
| source | list [str]              | A list of string expressions that serve as primary keys. For example, if source is `["children"]` and the variable children is `["Alice", "Bob"]`, two symbols are created.                     |
| attr   | list [str] (opt)        | A list of attribute names for the symbol (e.g., `["color", "size"]`). If not provided, the symbol is a simple value. If provided, the symbol is a dictionary-like object with these attributes. |
| type   | str or list [str]       | The Z3 type(s) for the symbol (e.g., `Int`, `Bool`). Must be a single string if attr is not defined. Must be a list of strings matching the length of attr if it is defined.                    |
| desc   | str or list [str] (opt) | A natural language description template. Follows the same format constraints as type based on the presence of attr.   |

Table 6: Schema for the DefinedSymbol Object.

| Field       | Type              | Description  |
|-------------|-------------------|--|
| source      | list [str]        | Data sources for selection. Each string is an expression evaluating to a list (e.g., a variable name or a literal list like "[1, 2, 3]").                        |
| amount      | list [str] (opt)  | The number of items to select from each corresponding source. If omitted, one item is selected from each source. Length must match source.                       |
| order       | list [bool] (opt) | If true for a source, selection order matters (permutation). If false, it does not (combination). Defaults to all true. Length must match source.                |
| duplicate   | list [bool] (opt) | If true for a source, items can be selected more than once. Defaults to all false. Length must match source.   |
| domain      | str (opt)         | The total number of symbols (selections) to generate. Can be an integer literal or a variable name.  |
| domain_cond | bool              | If true (default), identical symbol combinations are disallowed across the entire selection process.   |
| dim         | int               | The number of dimensions for the symbol (default 1). Useful for creating matrices of related symbols.  |
| dim_cond    | list (opt)        | A list of lists specifying inter-dimensional constraints. Each inner list contains source indices whose selected values cannot be identical.                     |
| custom_cond | list (opt)        | A list of custom constraint dictionaries. Each dictionary specifies a scope ("domain" or "dim"), a list of source fields, and a Python lambda constraint string. |
| formula     | str (opt)         | A Python expression string that defines the Z3 constraint for the generated symbol.  |
| desc        | str               | A natural language description template for the set of generated symbols.  |

Table 7: Schema for the `DerivedSymbol` Object.

| Field   | Type      | Description  |
|---------|-----------|--|
| formula | str       | The constraint logic as a Python expression string (e.g., "x + y < 10"). |
| desc    | str (opt) | The corresponding natural language description for the puzzle text.      |

Table 8: Schema for the `StaticCondition` Object.

| Field       | Type             | Description   |
|-------------|------------------|---|
| source      | list [str]       | Data sources for parameter selection, same as in <code>DerivedSymbol</code> .   |
| amount      | list [str] (opt) | Number of items to select from each source, same as in <code>DerivedSymbol</code> .   |
| domain      | str (opt)        | The number of conditions to generate, specified as a range string (e.g., "[1, 5]"). If omitted, one condition is generated. |
| domain_cond | bool             | If true (default), identical parameter combinations are disallowed.   |
| custom_cond | list (opt)       | Custom constraints on parameter selection, same as in <code>DerivedSymbol</code> .  |

Table 9: Additional fields for the `DynamicCondition` Object.

| Field         | Type      | Description  |
|---------------|-----------|--|
| desc          | str       | The natural language text of the question.   |
| ans_formula   | str       | A Python expression to compute the correct answer from the solution.   |
| ans_text      | str       | A template for formatting the display of the answer.   |
| ans_assertion | str (opt) | A Python expression that must evaluate to true for the answer to be considered valid (e.g., to ensure a unique solution exists). |

Table 10: Schema for the `Query` (Open-Ended) Object.

| Field  | Type       | Description  |
|--|------------|--|
| <i>Base fields for all selection queries include desc, query_type, select_type, and opt_num.</i> |            |  |
| source   | list [str] | Data sources for generating option parameters.   |
| cond   | str        | Condition scope for an option to be correct/incorrect. "any": satisfies at least one solution; "all": satisfies all solutions. |
| opt_formula  | str        | A Python expression that evaluates the correctness of a generated option.  |
| opt_text   | str (opt)  | A template for the display text of the option.   |
| custom_cond  | list (opt) | Custom constraints on option parameter selection.  |

Table 11: Schema for the `QuerySelectionTemplate` Object (for multiple-choice options).

| Field               | Type       | Description   |
|---------------------|------------|---|
| post_gen_vars       | dict (opt) | A dictionary to define new variables. Keys are new variable names, values are Python expressions to extract values from the solution. |
| post_gen_conditions | dict (opt) | A dictionary to add new constraints. Keys are new constraint names, values are <code>StaticCondition</code> objects.                  |

Table 12: Schema for the `PostGen` Object.

| Field   | Type | Description   |
|---------|------|---|
| type    | str  | The optimization type: "minimize" or "maximize".    |
| formula | str  | The formula representing the value to be optimized. |

Table 13: Schema for the `Optimize` Object.

| Category                       | Setting                 | Value  |
|--------------------------------|-------------------------|--|
| Objective                      | Task                    | Supervised fine-tuning (prompt: <code>chat_template</code> )           |
| Model & Optimization           | Base model              | <code>Qwen2.5-7B-Instruct</code>                                       |
|                                | Model max length        | 32,768   |
|                                | Per-device batch size   | 1  |
|                                | Grad accumulation steps | 4  |
|                                | Learning rate           | $1 \times 10^{-5}$   |
|                                | Weight decay            | 0.05   |
|                                | Warmup ratio            | 0.03   |
|                                | LR scheduler            | cosine   |
| Regularization & Checkpointing | Gradient checkpointing  | True   |
|                                | Filter by length        | True   |
|                                | Save only model         | True   |
| Precision                      | Numeric formats         | <code>bf16=True; tf32=True</code>                                      |
| Distributed / System           | DeepSpeed               | <code>ZeRO-3 Offload</code>  |
|                                | Dataloader workers      | 1  |
| Training & Logging             | Epochs                  | 6  |
|                                | Evaluation              | <code>eval_strategy=no</code>  |
|                                | Saving                  | <code>save_strategy=steps; save_steps=615; save_total_limit=999</code> |
|                                | Logging                 | <code>logging_steps=1; report_to=None</code>                           |
| Compute                        | GPUs                    | 16 (8 H100 per node $\times$ 2)  |

Table 14: Key hyperparameters for SFT training.

| Category             | Setting                      | Value   |
|----------------------|------------------------------|---|
| Algorithm            | RL type                      | PPO with GRPO advantage estimator   |
|                      | KL regularization            | <code>use_kl_loss=True; kl_loss_coef=0.001; kl_loss_type=low_var_kl; use_kl_in_reward=False; entropy_coeff=0</code> |
|                      | Critic warmup                | 0   |
| Model & Optimization | Base model                   | <code>Qwen2.5-7B-Instruct</code>  |
|                      | Learning rate                | $1 \times 10^{-6}$  |
|                      | Gradient checkpointing       | True  |
|                      | PPO mini-batch size          | 128   |
|                      | Dynamic batch size           | True  |
|                      | Max token len / GPU (PPO)    | 24,000  |
| Rollout / Inference  | Engine                       | <code>vLLM; use_remove_padding=True</code>  |
|                      | Generations per prompt       | $n = 5$   |
|                      | Max batched tokens (rollout) | 5,120   |
|                      | GPU memory utilization       | 0.6   |
|                      | Tensor model parallel size   | 2   |
| Distributed / FSDP   | Strategy                     | <code>fsdp2 (actor / ref / critic / reward model)</code>  |
|                      | Offload                      | <code>actor: param=False, optimizer=False; ref: param=True</code>   |
| Training & Logging   | Epochs                       | 5   |
|                      | Validation                   | <code>val_before_train=True</code>  |
|                      | Save / Test freq (steps)     | <code>save=290; test=80</code>  |
|                      | Logger                       | <code>console, tensorboard</code>   |
| Compute              | GPUs                         | 16 (8 H100 per node $\times$ 2)   |

Table 15: Key hyperparameters for GRPO training.

**The prompt for wrapping puzzles**

**A Single Choice**

**CHN** 本题是单选题（只有一个正确选项），请阅读题目后用中文一步一步推理，并将您认为的正确选项填写在boxed{}中。

**ENG** This is a multiple-choice question (only one correct option). Please read the question carefully and reason step by step in Chinese, then put your final selected option inside \boxed{ }.

**B Short Answer**

**CHN** 本题是简答题，请阅读题目后用中文一步一步推理，并将您的最终精简答案放在boxed{}中。

**ENG** This is a short-answer question. Please read the question carefully and reason step by step in Chinese, then place your concise final answer inside \boxed{ }.

**C Fill In The Blanks**

**CHN** 本题是填空题，请阅读题目后用中文一步一步推理，填补题目中空留的部分，并将这些填补内容全部放在boxed{}中。

**ENG** This is a fill-in-the-blank question. Please read the question carefully and reason step by step in Chinese to fill in the missing parts, then put all filled contents inside \boxed{ }.

**D Multiple Queries**

**CHN** 以下共有{qtype\_len}道小题，分别是{qtype\_list}，请用中文一步一步推理，并将这些小题的最终答案按顺序统一放在回答最末尾，并用boxed{}将最终答案包裹。如果是单选题，请将正确选项作为小题的最终答案；如果是简答题，请将精简后的答案作为小题的最终答案。

**ENG** There are {qtype\_len} sub-questions below, which are {qtype\_list}. Please reason step by step in Chinese, and summarize all final answers in order at the end of your response, wrapped inside \boxed{ }. For single choice questions, provide the correct option; for short-answer questions, give the concise final answer.

Figure 6: Question-Type Prompt Wrappers for LLM reasoning/training.

Listing 1: DSL Specification for the “Hamburger” Puzzle (Part 1/2).

```

variables:
  s_num: {type: int, domain: "[4, 7]", diff_factor: 1}
  f_num: {type: int, domain: "[3, 5]", diff_factor: 1}
  names: {formula: get_faker(s_num, 'name')}
  food: {formula: get_faker(f_num, 'food')}

symbols:
  buy: {source: [names, food], type: bool}

conditions:
  purchased_at_least_one_kind:
    formula: And([Or([buy[(p, f)] for f in food]) for p in names])
  desc: "{s_num} students, {'".join(names)}}, have purchased at least one kind of food: {'.".join(food)})."

if_a_then_not_b:
  source: [food, "[False, True]"]
  amount: ['2', '2']
  formula: >
    And([Implies(buy[(p, _sym[0][0])] if _sym[1][0]
      else Not(buy[(p, _sym[0][0])]), buy[(p, _sym[0][1])] if _sym[1][1] else Not(buy[(p, _sym[0][1])])) for p in names])
  desc: >
    People who {'did not buy' if _sym[1][0] else 'bought'} {_sym[0][0]} {'did not buy' if _sym[1][1] else 'bought'} {_sym[0][1]}.

at_least_one_person_bought:
  source: [food]
  domain: "[1, 2]"
  formula: Or([buy[(p, _sym[0])] for p in names])
  desc: At least one person bought {_sym[0]}.

a_bought_b:
  source: [names, food, "[False, True]"]
  domain: "[s_num*f_num//3, s_num*f_num//2]"
  formula: buy[_sym[0], _sym[1]] == _sym[2]
  desc: "{_sym[0]} {'bought' if _sym[2] else 'did not buy'} {_sym[1]}."

a_b_exclusive:
  source: [names]
  amount: ['2']
  domain: "[1, 2]"
  formula: >
    And([Implies(buy[_sym[0][0], f]), Not(buy[_sym[0][1], f])) for f in food])
  desc: "{_sym[0][1]} did not buy any items that {_sym[0][0]} bought.".

assumption:
  source: [names, "range(2, f_num)"]
  amount: ['2', '1']
  formula: And([Sum([If(buy[(p, f)], 1, 0) for f in food]) == _sym[1][0] for p in _sym[0]])
  desc: "If both {_sym[0][0]} and {_sym[0][1]} bought {_sym[1][0]} kinds of products,".

```

---

**Listing 2: DSL Specification for the “Hamburger” Puzzle (Part 2/2).**

```
# --- (Continuation) Spec of "1-hamburger" ---
queries:
question:
desc: "which of the following must be true?"
opt_num: 4
templates:
- source: [names]
amount: ['2']
cond: all
opt_formula: >
    sum([1 for f in food if get_value(_model, And(
        buy[({_opt[0][0]}, f)], buy[({_opt[0][1]}, f)])))
    ]) == 1
opt_text: >
    There is exactly one food item that both {_opt
        [0][0]} and {_opt[0][1]} bought.
- source: [names, food, "[False, True]"]
amount: ['1', '1', '1']
cond: all
opt_formula: get_value(_model, buy[({_opt[0][0]},
    {_opt[1][0]})] == _opt[2][0])
opt_text: "{_opt[0][0]} {'bought' if {_opt[2][0]}
else 'did not buy'} {_opt[1][0]}."
desc: >
{purchased_at_least_one_kind}Their choices satisfy
these conditions: {if_a_then_not_b}(
at_least_one_person_bought}{a_bought_b}(
a_b_exclusive}{assumption}{question}
```

---



---

**Listing 3: DSL specification for the “Graduation” puzzle.**

```
variables:
p_num: {type: int, domain: "[6, 12]", diff_factor: 1}
select_num: {type: int, domain: "[p_num // 2 - 1,
    p_num // 2 + 1]"}
names: {formula: generate_letters(p_num)}
name_desc: {formula: "", '.join(names)"}
symbols:
events: # Represents whether a person is selected
source: [names]
type: bool
desc: "{_names} was selected for the ceremony"
conditions:
base: # Total number of selected people
formula: gen_event_count_condition(events, 'equal',
    select_num)
desc: "Select {select_num} people for the graduation
ceremony."
cond1: # XOR condition
source: [events]
domain: "[1, 3]"
amount: ['2']
formula: gen_event_count_condition({_sym[0]}, 'equal',
    1)
desc: "Either {get_p({_sym[0][0]}, 'names')} or {get_p(
{_sym[0][1]}, 'names')} must be selected, but not
both."
cond2: # Implication condition
source: [events]
domain: "[1, 3]"
amount: ['2']
formula: Implies({_sym[0][1]}, {_sym[0][0]})
desc: "Unless {get_p({_sym[0][0]}, 'names')} is
selected, {get_p({_sym[0][1]}, 'names')} cannot be
."
queries:
question:
source: [events]
desc: "Which of the following could be a valid
selection?"
opt_num: 5
amount: [select_num]
cond: any
opt_formula: sum([get_value(_model, {_opt[0][i]}) for
    i in range(select_num)]) == select_num
opt_text: "{', '.join(get_p({_opt[0]}, 'names'))}"
q2:
source: [events]
desc: "The selected group must include:"
opt_num: 4
amount: ['2']
cond: all
opt_formula: sum([get_value(_model, {_opt[0][i]}) for
    i in range(2)]) >= 1
opt_text: "{get_p({_sym[0][0]}, 'names')} or {get_p(
{_sym[0][1]}, 'names')}."
q3:
source: [events]
desc: "Which two people cannot be selected at the
same time?"
opt_num: 5
amount: ['2']
cond: all
opt_formula: sum([get_value(_model, {_opt[0][i]}) for
    i in range(2)]) <= 1
opt_text: "{get_p({_sym[0][0]}, 'names')} and {get_p(
{_sym[0][1]}, 'names')}"
desc: "From {p_num} graduates ({name_desc}), {base}. The
selection must satisfy these conditions: {cond1} {
cond2} {queries}"
```

---

---

**Listing 4: DSL specification for the “Vase” puzzle.**

```

variables:
p_num: {type: int, domain: "[3, 10]", diff_factor: 1}
broken_vase_num: {type: int, domain: "[1, round(p_num / 3)]"}
names: {formula: get_faker(p_num, 'name')}
name_desc: {formula: "' , '.join(names)"}

symbols:
names_s: # Who broke the vase
source: [names]
type: bool
desc: "{_names} broke the vase"
speeches_s: # What each person said
source: [names_s, "[True, False]", "'[eq' ]"]
domain: p_num
domain_cond: false # Allow duplicate speeches
formula: make_expr(_sym[2], _sym[0], _sym[1])
desc: >
    {names[_index]} says: "{get_p(_sym[0], 'names')}"
    {'broke' if _sym[1] else 'did not break'} the vase
    "
conditions:
cond1: # Total number of culprits
formula: gen_event_count_condition(names_s, 'equal',
    broken_vase_num)
desc: >
    The mother knows {broken_vase_num} of {p_num}
    children broke the vase.
cond2: # Culprits are liars
formula: "[Implies(names_s[names[i]], Not(speeches_s[i])) for i in range(p_num)]"
desc: "The children who broke the vase are
    definitely lying."
queries:
question:
desc: "Who broke the vase?"
ans_formula: >
    get_p(get_TF_events_for_each_solution(names_s,
        _solutions, True), 'names')
ans_text: "' , '.join(_ans[0])"
ans_assertion: len(_ans) == 1
desc: >
    There are {p_num} children: {name_desc}. {
        broken_vase_num} broke a vase.
    Their statements: {', ' .join(get_desc(speeches_s))}. {
        cond1}, and {cond2}. {question}

```

---



---

**Listing 5: DSL specification for the “Wine” puzzle.**

```

variables:
wine_num: {type: int, domain: "[6, 12]", diff_factor: 1}
beer_num: {type: int, domain: "[1, 2]"}
bought_wine_of_first_customer: {type: int, domain: "[1, wine_num/2]"}
vol_times: {type: int, domain: "[2, 5]"}
wines: {formula: generate_letters(wine_num)}

symbols:
wine_s: # Each barrel has a volume and belonging (0:
    beer, 1:cust1, 2:cust2)
source: [wines]
attr: [volume, belonging]
type: [int, int]

conditions:
wine_belonging: {formula: "And([Or(x==0, x==1, x==2)
    for x in wine_s.get('belonging'))]}"
wine_0:
formula: "Sum([If(x==0, 1, 0) for x in wine_s.get('
    belonging'))] == beer_num"
desc: "{beer_num} barrels contain beer."
wine_1:
formula: "Sum([If(x==1, 1, 0) for x in wine_s.get('
    belonging'))] == bought_wine_of_first_customer"
desc: "The first customer bought {
    bought_wine_of_first_customer} barrels of wine."
wine_times:
formula: >
    Sum([If(x1 == 2,x2,0) for x1,x2 in zip(wine_s.get(
        'belonging'), wine_s.get('volume'))])
    == vol_times * Sum([If(x1==1,x2,0) for x1,x2 in
        zip(wine_s.get('belonging'), wine_s.get('
            volume'))])
desc: "The second customer bought {vol_times} times
    the volume of the first."
wine_volume_domain: {formula: "And([And(x > 0, x <=
    50) for x in wine_s.get('volume'))]"}
wine_volume_distinct: {formula: "
    gen_event_count_condition(wine_s.get('volume'),
        'distinct')"}"
post_generation: # Solve for volumes first, then fix
    them
post_gen_vars:
vol: "get_value(_sol, wine_s.get('volume'))"
post_gen_conditions:
vol_cond:
formula: "And([wine_s[w].get('volume') == vol[i]
    for i, w in enumerate(wines)])"
queries:
question:
source: [range(0, wine_num)]
desc: "Which barrel(s) contain beer? Please provide
    the letters for the correct options."
opt_num: 4
amount: [beer_num]
cond: all
opt_formula: "sum([get_value(_model, wine_s[wines[
    _opt[0][i]]].get('belonging')) == 0 for i in
    range(beer_num)]) == beer_num"
opt_text: "' , '.join([str(vol[_opt[0][i]]) for i in
    range(beer_num)])"
desc: >
A merchant has {wine_num} barrels of wine and beer
with volumes: {', ' .join([str(v) + ' gallons' for v
    in vol])}.
There are {wine_num - beer_num} barrels of wine. {
    wine_0}, {wine_1}, and {wine_times}.
No wine is left. {question}

```

---

**Listing 6: DSL specification for the “Product” puzzle.**

```

variables:
  p_num: {type: int, domain: "[6, 17]", diff_factor: 1}
  products: {formula: get_faker(p_num, 'product')}
  names: {formula: get_faker(1, 'name')}
symbols:
  pos: # The position of each product on a conveyor belt
    source: [products]
    type: int
conditions:
  pos_domain: {formula: "And([And(pos[x] >= 1, pos[x] <=
      p_num) for x in products])"}
  pos_distinct: {formula: "gen_event_count_condition(pos
      , 'distinct')"}
cond1: # Relative distance between products
  source: [products, "range(1, p_num - 2)"]
  amount: ['2', '1']
  domain: "[p_num // 2, p_num]"
  formula: "Or(pos[_sym[0][0]] - pos[_sym[0][1]] ==
      _sym[1][0], pos[_sym[0][1]] - pos[_sym[0][0]] ==
      _sym[1][0])"
  desc: "{_sym[0][0]} and {_sym[0][1]} have {'no' if
      _sym[1][0] == 1 else str(_sym[1][0] - 1)} items
      between them."
cond2: # Adjacent products
  source: [products]
  amount: ['2']
  domain: "[1, p_num // 2]"
  formula: "pos[_sym[0][1]] - pos[_sym[0][0]] == 1"
  desc: "{_sym[0][1]} is placed immediately after {
      _sym[0][0])."
cond3: # Absolute position constraint
  source: [products, "range(1, p_num+1)"]
  amount: ['1', '1']
  domain: "[0, p_num // 3]"
  formula: "pos[_sym[0][0]] != _sym[1][0]"
  desc: "{_sym[0][0]} is not in position {_sym
      [1][0].}"
max_solution: 600
queries:
  question:
    source: [products, "range(1, p_num+1)"]
    desc: "If the statements are true, which of the
        following must be true?"
    opt_num: 6
    amount: ['1', '1']
    cond: all
    opt_formula: "get_value(_model, pos[_opt[0][0]]) ==
        _opt[1][0]"
    opt_text: "{_opt[0][0]} is in position {_opt
        [1][0].}"
desc: >
  After shopping, {names[0]} placed {p_num} items on a
  conveyor belt,
  ordered from front to back (pos 1 to {p_num}). {
  conditions} {question}

```

**Listing 7: DSL specification for the “Exam” puzzle.**

```

variables:
  p_num: {type: int, domain: "[4, 10]", diff_factor: 1}
  exam_num: {type: int, domain: "[2, p_num]",
    diff_factor: 1}
  num_performed_well: {type: int, domain: "[1, p_num -
    1]"}
  names: {formula: get_faker(p_num, 'name')}
  exams: {formula: get_faker(exam_num, 'major')}
symbols:
  performed_well: {source: [names], type: bool}
  passed: {source: [names, exams], type: bool}
conditions:
  num_performed_well_cond:
    formula: gen_event_count_condition(performed_well,
      'equal', num_performed_well)
    desc: "Only {num_performed_well} of {p_num} people
        performed well."
cond1:
  source: [exams]
  amount: ['2']
  domain: "[p_num, p_num]"
  domain_cond: false
  formula: >
    And(Implies(performed_well[names[_index]], passed
      [(names[_index], _sym[0][0])]),
      Implies(Not(performed_well[names[_index]]),
        Not(passed[(names[_index], _sym[0][1])])))
  desc: '{names[_index]} says: If I perform well, I
      will pass {_sym[0][0]}. If not, I will fail {
      _sym[0][1].}'
only_one_who_passed_some_exam:
  source: ["[False', 'True']"]
  amount: ['2']
  domain: "[2, 2]"
  formula: >
    Or([Sum([If(And(passed[(p, e)] == _sym[0][1],
        performed_well[p] != _sym[0][0]),
        1, 0) for p in names]) == 0 for e in exams])
  desc: >
    For one subject, only those who performed {'well'
      if _sym[0][0] else 'poorly'}
      {'if _sym[0][1] else 'did not '}pass.
max_solution: 500
queries:
  question:
    desc: "Who performed well?"
    ans_formula: to_unique([{p for p in names if
        get_value(_model, performed_well[p])} for _model
        in _solutions])
    ans_text: "','.join({_ans[0]})"
    ans_assertion: len(_ans) <= 1
desc: >
  {p_num} people ({', '.join(names)}) took {exam_num}
  exams ({', '.join(exams)}).
  {num_performed_well_cond} Statements: {cond1} Also:
  only_one_who_passed_some_exam} {question}

```

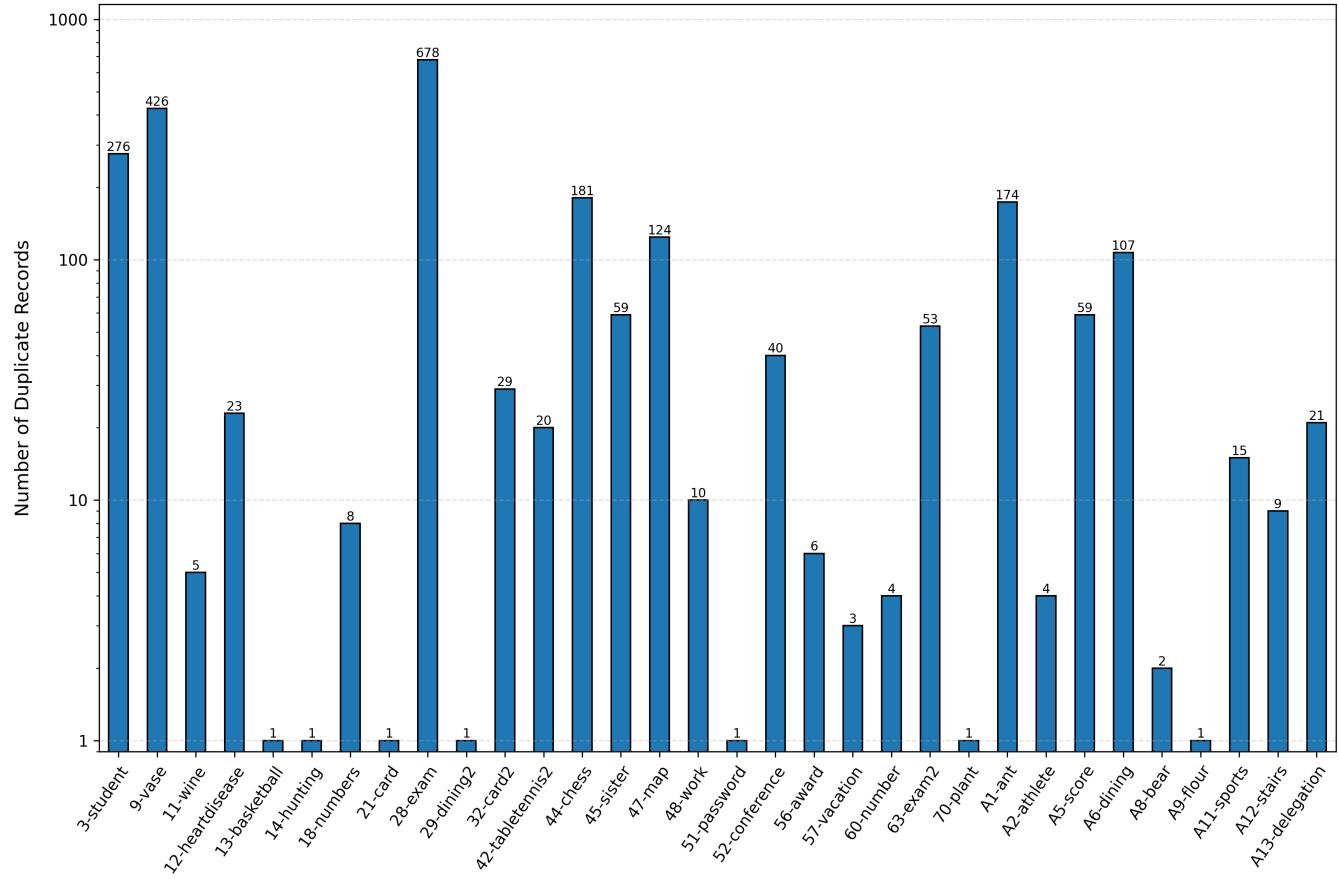


Figure 7: Distribution of duplicated instances across 32 seed puzzles. Only seed puzzles with at least one duplicate are shown.



Figure 8: The average accuracy of all models on the PuzzleClone test set grouped by the seed puzzles.