

SAT-RL: Training Language Models for Logical Reasoning with Symbolic Solvers

Jialiang Sun and Kuldeep Meel

University of Toronto

sjl@cs.toronto.edu

Abstract

Large language models (LLMs) have achieved strong performance in mathematical and logical reasoning, yet they still suffer from the curse of complexity and struggle as problem difficulty increases with the number of constraint interactions and the size of the search space. In contrast, satisfiability solvers such as Z3 provide scalable and verifiable reasoning through constraint solving under formal theories, but lack the flexibility to interpret natural-language problem descriptions. In this work, we propose SAT-RL, a reinforcement-learning (RL) framework that trains LLMs to interact with symbolic solvers through multi-turn tool execution. SAT-RL leverages synthetic data generation with verifiable solutions, supervised warm-up on solver-augmented traces, and solver-grounded reinforcement learning with tool-aware rewards. Experiments show that SAT-RL substantially improves symbolic reasoning with tools on both in-domain and out-of-domain logic tasks. Our code is available on [GitHub](#).

1 Introduction

Large language models (LLMs) have demonstrated strong capabilities on mathematical and logical reasoning tasks, including the Law School Admission Test (LSAT), the International Mathematical Olympiad (IMO), and broad AGI datasets such as AGI-Eval (Zhong et al., 2021; Ren et al., 2025; Zhong et al., 2024). In particular, reinforcement learning (RL) has played a central role in these advances by enabling long-horizon reasoning, self-verification, and agentic tool calling (Schulman et al., 2017; Guo et al., 2025; Shao et al., 2024; Yu et al., 2025). Despite this progress, recent work on logical reasoning benchmarks (Lin et al., 2025; Wei et al., 2025; Chen et al., 2025) shows that LLMs suffer from a curse of complexity, in which accuracy drops significantly as the number of constraints and the size of the search space increase. In-

corporating symbolic checking further reveals that this accuracy degradation is related to systematic constraint violations and a persistent gap between reasoning rationales and verifiable correctness (Qi et al., 2025).

On the other hand, symbolic solvers such as Boolean satisfiability (SAT) solvers, satisfiability modulo theories (SMT) solvers, and first-order theorem provers (TPs) can scale to large combinatorial spaces and produce reliable logical deductions by finding satisfying assignments or proofs of unsatisfiability under well-defined theories. Commonly used solvers include CaDiCaL (Ignatiev et al., 2018), Z3 (de Moura and Bjørner, 2008), cvc5 (Barbosa et al., 2022), and Vampire (Riazanov and Voronkov, 2001), most of which have Python interfaces available through PyPI packages. Inspired by this reliability, a line of research has combined LLMs with symbolic solvers. For example, SAT-LM (Ye et al., 2023) and Logic-LM (Pan et al., 2023) encode natural-language logic tasks into formal constraints and invoke Z3 in a one-shot declarative query, while other systems such as SymProLM (Mittal et al., 2024) and MCP-Solver (Szeidler, 2024) support agentic multi-turn interactions with solver feedback and self-correction. However, these methods operate only at inference time and rely on solver-specific knowledge implicitly learned during pretraining, making them vulnerable to program errors. Moreover, they offer limited means of aligning solver encodings with the constraints in the original problem statements.

Rather than treating symbolic solvers as fixed inference-time heuristics, a promising research direction is to integrate LLMs with symbolic solvers through training that ensures the reliability of reasoning traces and optimizes interactions between LLMs and symbolic solvers. To close this gap, we introduce SAT-RL, a framework that enables LLMs to plan and iteratively refine symbolic solver calls using reinforcement learning. The framework

consists of two major components: (i) a stage for generating synthetic logic data with ground-truth solvers and verifiers to support supervised fine-tuning (SFT) warm-up, and (ii) an RL stage with verifiable rewards and instance-conditioned solver-tool sampling to guide exploration. Overall, SAT-RL aims to produce hybrid reasoning systems that retain natural-language accessibility while achieving scalable, faithful, and verifiable logical reasoning across diverse problem domains.

2 Related Work

2.1 Benchmarks for Logical Reasoning

Benchmarks for logical reasoning span natural-language comprehension, formal validity, and verifiable constraint puzzles. Reading-comprehension datasets such as LogiQA (Liu et al., 2020, 2023) and ReClor (Yu et al., 2020) test multiple-choice logical inference over passages, while CLUTRR (Sinha et al., 2019) targets relational induction over narratives. Formal corpora such as FOLIO and P-FOLIO (Han et al., 2024a,b), RuleTaker (Clark et al., 2020), and LogicNLI (Tian et al., 2021) emphasize first-order validity or entailment with more explicit logical structure and, in the case of P-FOLIO, proof-chain annotations. A complementary line of work uses programmatically generated puzzles with exact solution checking: SAT-Bench (Wei et al., 2025) randomly samples SAT formulas and creates instances with LLM annotations, ZebraLogic and BoardgameQA (Lin et al., 2025; Kazemi et al., 2023) provide scalable generators with difficulty control, and Enigmata (Chen et al., 2025) broadens coverage across 36 puzzle families. Recent work emphasizes robustness to memorization through automatic generation in AutoLogi (Zhu et al., 2025) and periodic refreshes in NPHardEval (Fan et al., 2024), and broadens puzzle coverage with long-tail game settings in HardcoreLogic (Liang et al., 2025). Complementary evaluation efforts such as LogicBench and Multi-LogiEval (Parmar et al., 2024; Patel et al., 2024) assemble multi-step logical test suites, while CFBench (Zhang et al., 2025) evaluates whether models can satisfy explicit constraints in generated outputs.

2.2 Logical Reasoning with LLMs

Prompting techniques, such as chain-of-thought (CoT), tree-of-thought (ToT), and self-consistency sampling, improve decomposition and search over

candidate responses (Wei et al., 2022; Yao et al., 2023; Wang et al., 2022). Several studies focus on coupling LLMs with external symbolic solvers to enable verifiable reasoning steps: SAT-LM (Ye et al., 2023) issues a one-shot declarative query to Z3 (de Moura and Bjørner, 2008), while MCP-Solver (Szeider, 2024) supports iterative constraint-model construction and validation through a structured interface for Z3, PySAT (Ignatiev et al., 2018), and MiniZinc (Nethercote et al., 2007). FCoReBench (Mittal et al., 2024) targets first-order combinatorial reasoning with scalable instances.

RL can improve reasoning by directly optimizing model behavior under preference signals or reward functions. Proximal Policy Optimization (PPO) remains a common RLHF backbone due to its stable on-policy updates in large action spaces (Schulman et al., 2017). Direct Preference Optimization (DPO) avoids an explicit reward model by optimizing a closed-form objective over pairwise preferences (Rafailov et al., 2023). Group Relative Policy Optimization (GRPO) estimates advantages from within-group comparisons over multiple sampled responses, reducing reliance on value-function learning (Shao et al., 2024). Recent variants refine policy-gradient baselines or update rules for long-horizon reasoning, including REINFORCE++ (Hu, 2025) and OPO (Hao et al., 2025). Self-play-based methods, such as SPIN and SPPO (Chen et al., 2024; Wu et al., 2024), generate preference data from model-model interactions to iteratively strengthen policies. DAPO emphasizes stabilizing long-horizon training through improved optimization and regularization (Yu et al., 2025).

3 Methods

Overall, we integrate long-horizon reinforcement learning with multi-turn tool calls to symbolic solvers. The SAT-RL pipeline comprises three stages: (i) a data generation and preprocessing stage for supervised fine-tuning; (ii) an SFT stage that warms up the base policy model; and (iii) an RL stage that fine-tunes the model using DAPO (Yu et al., 2025), a variant of GRPO (Shao et al., 2024), while interleaving code execution via Sandbox Fusion (Bytedance-Seed-Foundation-Code-Team et al., 2025) as the MCP backend. We first introduce the notation and problem formulation and then describe the proposed methods in detail.

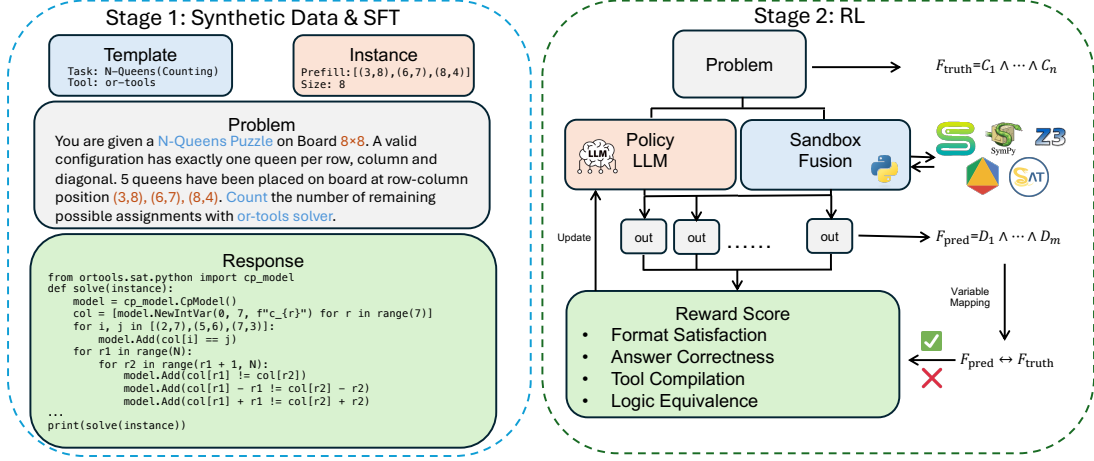


Figure 1: Workflow of the SAT-RL framework. Stage 1 (**left**) illustrates template-based data generation for the SFT phase: tasks and output programs are instantiated with randomized parameters to produce natural-language problems paired with ground-truth solver programs and executable responses. Stage 2 (**right**) illustrates the RL training phase and its reward components.

3.1 Preliminaries

Notation. We denote the trainable policy model by π_θ , where θ denotes the model parameters. The conditional distribution over outputs is given by

$$\pi_\theta(y \mid x) = \prod_{t=1}^T P(y_t \mid y_{<t}, x, \theta)$$

where x is the input query and $y = (y_1, \dots, y_T)$ is the output token sequence.

Throughout the paper, we use the term symbolic solvers to collectively refer to the following PyPI distributions: z3-solver, pysmt, python-sat, ortools, and sympy. In particular, Z3 (de Moura and Bjørner, 2008) and PySMT are SMT interfaces supporting theories such as first-order arithmetic over integers, reals, and bit-vectors; PySAT (python-sat) provides Python bindings to state-of-the-art SAT and MaxSAT solvers; OR-Tools is an optimization library for combinatorial and linear programming; and SymPy is a computer algebra system for symbolic mathematics with limited logical reasoning support. A tool call refers to an executable program generated by the model and executed in a sandboxed environment during rollout.

Problem Formulation. We study natural-language (NL) logical reasoning tasks that admit a reduction to a satisfiability problem modulo a background theory. Following prior work on solver-grounded reasoning (Ye et al., 2023; Qi et al., 2025), we formalize each problem instance P as a 4-tuple $P = (\mathcal{T}, V, \Phi, q)$, where $\mathcal{T}$ is a first-order background theory, $V = \{v_1, \dots, v_n\}$

is a finite set of free variables, Φ is a first-order formula over the signature of $\mathcal{T}$ whose free variables are contained in V , and q is a query whose free variables are also contained in V . The theory $\mathcal{T}$ determines the semantics of the symbols and predicates appearing in Φ and q , including propositional logic, linear integer arithmetic, and linear real arithmetic.

Let $\mathcal{M}_{\mathcal{T}}(\Phi, V)$ denote the set of $\mathcal{T}$ -models that satisfy Φ and interpret exactly the variables in V : $\mathcal{M}_{\mathcal{T}}(\Phi, V) = \{M \mid M \models_{\mathcal{T}} \Phi \text{ and } \text{dom}(M) = V\}$, where each model M assigns values to the variables in V that satisfy Φ . We consider the following query types:

- **Decision.** The decision task $\text{Decide}(P)$ asks whether $\mathcal{M}_{\mathcal{T}}(\Phi, V)$ is non-empty. The output is True if and only if at least one model exists.
- **Solving.** Given an optional optimization objective $\mathcal{O}$, the solving task $\text{Solve}(P, \mathcal{O})$ asks for a model $M \in \underset{M' \in \mathcal{M}_{\mathcal{T}}(\Phi, V)}{\text{argmax}} \mathcal{O}(M')$, or any model $M \in \mathcal{M}_{\mathcal{T}}(\Phi, V)$ if no objective is given. Note that when Φ is a Boolean formula in conjunctive normal form (CNF), a typical MaxSAT problem sets $\mathcal{O}$ to be the number of satisfied clauses, whereas a SAT problem does not specify an objective.
- **Counting.** The counting task $\text{Count}(P)$ asks for the number of satisfying assignments, which equals the size of the model set $|\mathcal{M}_{\mathcal{T}}(\Phi, V)|$.

Example. Consider the question: In how many distinct ways can a factory package exactly 50 widgets using only 4-widget and 6-widget boxes? To model this instance using the 4-tuple $P = (\mathcal{T}, V, \Phi, q)$ defined above, let $\mathcal{T}$ be linear integer arithmetic and $V = \{x, y\}$, where x and y represent the numbers of 4-widget and 6-widget boxes, respectively. Define Φ as $(4x + 6y = 50) \wedge (x \geq 0) \wedge (y \geq 0)$. The corresponding counting query asks for $\text{Count}(P)$. Since the satisfying models are $\{(2, 7), (5, 5), (8, 3), (11, 1)\}$, the answer is 4. A concrete OR-Tools implementation is provided in Appendix D.

3.2 Data Generation and Processing

In this section, we describe the data processing procedures for SFT and RL training. Our synthetic dataset, Synthetic-Logic, is used to warm up the policy model during the SFT phase. We leverage SAT-Bench (Wei et al., 2025) for RL training and Enigmata (Chen et al., 2025) for evaluation.

Synthetic Data Generation for SFT. We construct the Synthetic-Logic dataset for SFT warm-up by systematically selecting task families and generating verifiable puzzle instances under controlled difficulty. Our dataset covers 16 logical reasoning tasks, including 9 drawn from Karp’s list of NP-complete problems (Karp, 2009), such as Subset Sum and the Traveling Salesman Problem (TSP). In addition, we include 7 further NP-hard tasks, such as binarized neural network (BNN) verification and N-Queens puzzles, with full task descriptions provided in Appendix B.1. Each task is instantiated for each of the three query types—decision, solving, and counting—as formalized in Section 3.1. Table 2 summarizes the coverage of task categories.

For each combination of task type, query type, and symbolic solver, we predefine a structured template consisting of a natural-language problem description and a ground-truth solver program implementing the formal constraint encoding. Ground-truth answers for decision and counting queries are obtained by executing the corresponding solver program, whereas responses to solving queries are validated using task-specific verifiers that check the correctness of candidate witnesses produced by the model. This design ensures that every instance is paired with an executable and verifiable reference trace grounded in the intended solver semantics.

We randomly sample 10 instances for each fixed combination of task, query type, and solver across

different difficulty parameter settings. For decision problems, we target a balanced 50%/50% SAT/UNSAT split and discard instances that fail due to solver timeouts during ground-truth execution. This procedure yields 1,399 synthetic SFT $\mathcal{D}_{\text{SFT}}$ instances after removing timeout instances, and they are split into 1,260 training and 139 test instances that are fully verifiable with ground-truth responses and verifiers.

Data Preprocessing for RL. We focus on logical reasoning datasets whose difficulty and size can be controlled. For RL training, we use SAT-Bench (Wei et al., 2025), which consists of logic puzzles automatically generated from CNF formulas. An LLM verbalizes each clause as a natural-language constraint, and each problem poses a yes-or-no decision question asking whether the constraints are satisfiable. Difficulty is controlled at generation time by the number of clauses and variables, with more clauses yielding harder puzzles. SAT-Bench includes a built-in mapping between each sampled formula variable and the corresponding natural-language entity, enabling SAT-RL to compute the logic-equivalence reward during training.

Data Preprocessing for Evaluation. For out-of-distribution evaluation, we test the trained models on Enigmata (Chen et al., 2025), a large collection of synthetic and verifiable logical and constraint-based puzzles, such as Tic-Tac-Toe and the 24 Game. Each instance requires the explicit construction of a satisfying assignment and includes a built-in correctness verifier. The final dataset sizes are reported in Table 3.

3.3 Supervised Fine-tuning

We initialize the trainable policy model π_θ by supervised fine-tuning on the solver-augmented dataset $\mathcal{D}_{\text{SFT}}$ constructed in Section 3.2. Each example $(x, y) \in \mathcal{D}_{\text{SFT}}$ pairs an input prompt x with a target response y that includes a valid solver program embedded within `<tool_call>...</tool_call>`. We train the policy model using the objective:

$$\mathcal{J}_{\text{SFT}}(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}_{\text{SFT}}} \left[\sum_{t=1}^{|y|} \log \pi_\theta(y_t \mid x, y_{<t}) \right]$$

This warm start teaches π_θ basic solver-tool API usage, planning, and coding, and stabilizes subsequent RL by reducing early-stage invalid tool executions.

Resource	Tasks	Controlled Difficulty	Scalable	NP-hardness	Verifier	Solution Program
SAT-Bench (Wei et al., 2025)	1	✓	✗	✓	✓	✗
ZebraLogic (Lin et al., 2025)	1	✓	✓	✓	✓	✗
FCoReBench (Mittal et al., 2024)	40	✓	✓	✗	✗	✗
Enigmata (Chen et al., 2025)	36	✓	✓	✓	✓	✗
Synthetic-Logic (Ours)	48	✓	✓	✓	✓	✓

Table 1: Comparison of logical reasoning benchmarks across different features.

Data Type	Count
Total	48
Number of Tasks	16
Number of Queries	3
Primary Task Category	
Puzzle Games	5
Program Verification	2
Integer Programming	4
Graph Theory	5

Table 2: Task statistics for Synthetic-Logic. Task types are assigned based on the primary problem structure, and each task is counted under exactly one category.

Dataset / Partition	Train	Test	Total
Synthetic-Logic (SFT)	1,260	139	1,399
SAT-Bench (RL)	1,575	525	2,100
Enigmata (Evaluation)	-	1,901	1,901

Table 3: Sizes of the training and test splits for each dataset.

3.4 Reinforcement Learning

We fine-tune the SFT-initialized policy model with DAPO (Yu et al., 2025) using multi-turn tool execution.

Tool Call Interface. We follow the ReTool paradigm (Feng et al., 2025) to interleave natural-language reasoning with executable solver calls. The LLM emits reasoning tokens r_i within `<think>...</think>` blocks and executable code or CLI commands c_i within `<tool_call>...</tool_call>` blocks. A sandbox compiles and executes c_i and returns the resulting stdout and stderr as feedback f_i within a `<tool_response>...</tool_response>` block, which immediately follows the corresponding `<tool_call>` block. The rollout thus interleaves reasoning, tool calls, and tool responses as

$$[r_1, c_1, f_1, \dots, r_n, c_n, f_n, a]$$

where a is the final answer.

We implement training using the verl (Sheng et al., 2025) pipeline with agentic mode enabled.

For multi-turn tool execution, we deploy an asynchronous code server via Sandbox Fusion to support parallel, high-throughput solver execution. We use the prompt in Appendix C to incentivize the policy model to use symbolic solvers and align variables with the problem context through the built-in variable mapping in SAT-Bench (Wei et al., 2025).

Reward Design. Overall, we aim to incentivize the policy model to solve diverse logical reasoning problems by learning to query various symbolic solvers using faithful formula encodings. The reward should focus not only on the correctness of the final answer but also on the quality of the encoded formula. Thus, we define the total reward r as the sum of four verifiable components: format r_{format} , correctness r_{correct} , compilability r_{tool} , and quality r_{equiv} . All components are initialized to 0.

For each instance, we uniformly sample one of the five solvers (Z3, PySMT, PySAT, OR-Tools, and SymPy) as the required solver, thereby familiarizing the model with diverse solvers and avoiding collapse to a single solver backend during training (Li et al., 2026). To enable final-answer parsing, we require the final answer a in the response to be enclosed in triple backticks using the ````answer```` format and assign a format reward $r_{\text{format}} = 0.1$ when parsing succeeds. For final-answer evaluation, we reward correct answers, encourage an honest *I don’t know* response under uncertainty, and penalize incorrect answers to discourage random guessing. We allow an explicit abstention token (rendered as ````idk````) and define the adjusted correctness reward as follows:

$$r_{\text{correct}}(a) = \begin{cases} 1 & \text{if } a \text{ is correct} \\ 0 & \text{if } a \text{ is idk} \\ -1 - \varepsilon & \text{if } a \text{ is incorrect} \end{cases} \quad (1)$$

Because the SAT-Bench training set has only two possible final answers (yes/no), this design makes uniform random guessing strictly suboptimal in expectation. The penalty for an incorrect a naturally generalizes to $\frac{-1}{n-1} - \varepsilon$ for multiple-choice problems with n choices.

We assign a tool-syntax reward $r_{\text{tool}} = 0.1$ if the final tool call executes without compilation errors and uses the sampled solver. Beyond compilability, we also judge the quality of the solver call based on the logical similarity between the ground-truth formula and the proposed formula extracted from the last tool call. This approach is inspired by a body of work on autoformalization (Murphy et al., 2024; Liu et al., 2025). Suppose the ground-truth and proposed CNF formulas have aligned variables and are represented by

$$\begin{aligned} F_{\text{truth}} &= C_1 \wedge \dots \wedge C_n \\ F_{\text{pred}} &= D_1 \wedge \dots \wedge D_m \end{aligned}$$

We verify the equivalence $F_{\text{truth}} \leftrightarrow F_{\text{pred}}$ using Z3 to check whether

$$(F_{\text{pred}} \wedge \neg F_{\text{truth}}) \vee (\neg F_{\text{pred}} \wedge F_{\text{truth}})$$

is unsatisfiable. We further assign $r_{\text{equiv}} = 1$ if the formulas are equivalent. The overall reward score is therefore

$$r = r_{\text{format}} + r_{\text{correct}} + r_{\text{tool}} + r_{\text{equiv}}$$

Training Objective. We leverage Decoupled Clip and Dynamic Sampling Policy Optimization (DAPO) (Yu et al., 2025), a variant of Group Relative Policy Optimization (GRPO) (Shao et al., 2024), as our RL training algorithm. Given an input prompt x , DAPO samples a group of G completions $\{o_i\}_{i=1}^G$ from the old policy $\pi_{\theta_{\text{old}}}$, computes the reward R_i for each completion, and estimates the corresponding advantages:

$$\hat{A}_{i,t} = \frac{R_i - \text{mean}(\{R_j\}_{j=1}^G)}{\text{std}(\{R_j\}_{j=1}^G)}$$

It then optimizes θ with the following training objective:

$$\mathcal{J}_{\text{DAPO}}(\theta) = \mathbb{E} \left[\frac{1}{G} \sum_{i=1}^G \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \left(\min \left(r_{i,t}(\theta) \hat{A}_{i,t}, \text{clip}(r_{i,t}(\theta), 1 - \epsilon_{\text{low}}, 1 + \epsilon_{\text{high}}) \hat{A}_{i,t} \right) \right) \right]$$

where the expectation is taken over $x \sim \mathcal{D}$ and $\{o_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot | x)$. Here, $|o_i|$ denotes the number of tokens in o_i , ϵ_{low} and ϵ_{high} denote the lower

and upper clipping thresholds, respectively, and the importance ratio $r_{i,t}$ is defined as

$$r_{i,t} = \frac{\pi_{\theta}(o_{i,t} | x, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} | x, o_{i,<t})}$$

Complete rollout examples with reward computation can be found in Appendices E.1 and E.2.

4 Evaluation

In this section, we evaluate SAT-RL across multiple models and datasets to examine its in-domain performance and out-of-domain generalization. In particular, we design our empirical evaluation to answer the following research questions:

RQ1. In-Domain Improvements. Does SAT-RL improve accuracy on in-domain logical reasoning benchmarks?

RQ2. Out-of-Domain Generalization. Does training with SAT-RL on logic puzzles generalize to out-of-domain logical or mathematical tasks?

RQ3. Ablation Studies. How do key components of SAT-RL, including SFT with synthetic data, tool sampling, and logic-equivalence checking, affect test accuracy and training behavior?

Hyperparameter Settings. We use verl v0.8.0 as our SFT and RL training framework (Sheng et al., 2025). We use DAPO (Yu et al., 2025) with a rollout group size of $G = 8$ as the RL training objective. During training, the maximum input and output token limits are set to 4096 and 8192, respectively. The detailed setup is provided in Appendix B.

Model Setup. We evaluate cutting-edge proprietary models, including OpenAI GPT-5.6 Luna (Singh et al., 2025), Claude Sonnet 5, and Claude Haiku 4.5 (Anthropic, 2026, 2025), with the reasoning-effort parameter set to “high.” We train 4B- and 9B-parameter variants of the Qwen3.5 (Team, 2026) model family. We compare the SAT-RL regime with the ReTool (Feng et al., 2025) baseline regime, in which the model can interact with tools without being incentivized to use symbolic solvers, and the reward includes only format, correctness, and successful compilation of tools. We train each model under each regime on the SAT-Bench training split (Wei et al., 2025) and

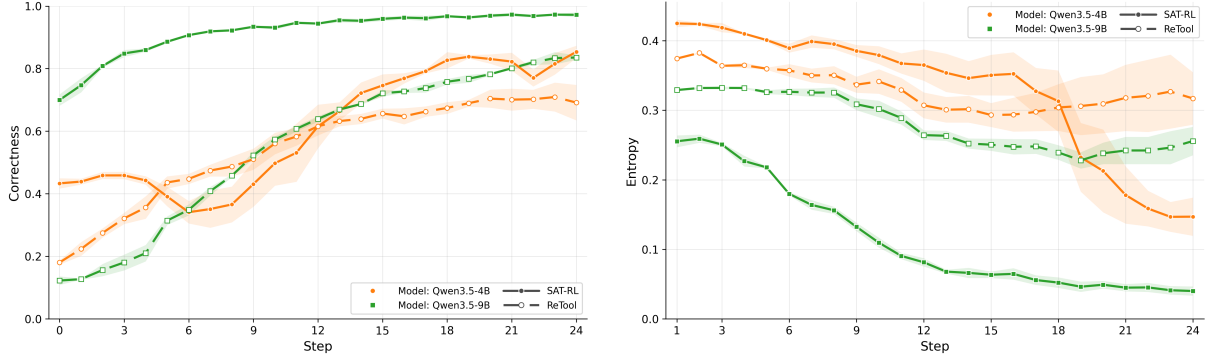


Figure 2: Training trajectories on SAT-Bench for ReTool and SAT-RL (mean $\pm$ population standard deviation across three runs). **Left:** validation correctness. **Right:** policy entropy.

evaluate in-domain performance on its test split. We use Enigmata (Chen et al., 2025) and AIME 2025 to evaluate out-of-domain generalization in logic and mathematics. For each setup, we conduct three training runs ($N = 3$) in parallel with different random seeds to assess training stability. We report test accuracy, with a correct response scoring 1.0 and an incorrect response scoring 0.0.

4.1 RQ 1: In-Domain Performance

Model	Accuracy (%)
Base LLMs	
GPT-5.6 Luna	92.0
Claude Sonnet 5	93.0
Claude Haiku 4.5	56.0
Qwen3.5-4B	19.8
Qwen3.5-9B	14.1
ReTool-Trained LLMs	
Qwen3.5-4B	69.1 ± 5.7
Qwen3.5-9B	83.5 ± 0.7
SAT-RL-Trained LLMs	
Qwen3.5-4B (After-SFT)	43.3 ± 1.6
Qwen3.5-4B (After-RL)	85.4 ± 2.0
Qwen3.5-9B (After-SFT)	70.0 ± 1.8
Qwen3.5-9B (After-RL)	97.2 ± 0.8

Table 4: In-domain Avg@8 test accuracy (%) on the SAT-Bench test set, which contains 525 instances. For trained models, accuracy is reported as the mean $\pm$ standard deviation.

We evaluate in-domain test accuracy on SAT-Bench and observe that SAT-RL enables the Qwen3.5-9B model to outperform cutting-edge proprietary models, as detailed in Table 4 and Figure 2. The base Qwen3.5-9B model performs poorly on

the SAT-Bench dataset and has difficulty following the ``answer`` format, with fewer than 20% of its responses yielding parseable answers. Its test accuracy reaches 70.0% after SFT and 97.2% after RL, surpassing Claude Sonnet 5 (93.0%) and the ReTool-trained Qwen3.5-9B model (83.5%). A similar pattern appears for Qwen3.5-4B: SAT-RL improves its accuracy from 19.8% to 85.4%, outperforming the larger ReTool-trained Qwen3.5-9B model. Overall, SAT-RL yields consistent in-domain accuracy improvements across model sizes.

4.2 RQ 2: Out-of-Domain Generalization

We evaluate Qwen3.5-9B on the Enigmata benchmark across difficulty and category partitions after training on SAT-Bench under ReTool and SAT-RL. For each regime, we select the final checkpoint with the highest SAT-Bench accuracy across three runs (Table 5). SAT-RL raises overall accuracy from 33.9% to 59.9%, outperforming Claude Haiku 4.5 and the strongest flagship models in categories such as Crypto. Across the easy, medium, and hard subsets, the gains are 19.6%, 31.6%, and 25.9%, respectively. SAT-RL improves all seven categories, with the largest gains on Crypto and Logic, where accuracy reaches 100.0% and 82.8%, respectively. These results demonstrate substantial transfer to tasks that emphasize constraint satisfaction and multi-step deductive reasoning.

On AIME 2025 (30 problems), SAT-RL modestly improves Avg@8 test accuracy from 86.7% (208/240) to 88.3% (212/240); both the base and SAT-RL-trained Qwen3.5-9B models solve 28/30 problems. Overall, under this configuration, SAT-RL generalizes to out-of-domain logic tasks and provides limited but non-negative gains in mathematical reasoning.

Model	Overall	Difficulty			Category						
		Easy	Medium	Hard	Arithmetic	Graph	Grid	Logic	Sequential	Search	Crypto
Base LLMs											
GPT-5.6 Luna	70.9	78.2	68.5	67.1	100.0	74.5	81.4	63.9	57.7	52.2	91.7
Claude Sonnet 5	72.5	72.0	72.3	73.2	100.0	66.8	86.1	75.6	52.0	62.2	86.7
Claude Haiku 4.5	49.7	52.0	49.7	47.8	84.2	44.1	56.6	65.6	26.1	45.0	61.7
Qwen3.5-9B	33.9	47.0	33.2	24.2	70.0	37.3	42.7	45.0	18.4	23.8	15.0
Trained LLMs											
Qwen3.5-9B (ReTool)	47.9	61.6	49.7	35.2	92.5	55.9	38.6	83.3	22.9	41.6	77.5
Qwen3.5-9B (SAT-RL)	59.9	66.6	64.8	50.1	100.0	59.5	53.4	82.8	46.0	45.6	100.0

Table 5: Out-of-domain Avg@1 accuracies (%) on the Enigmata test set (1,901 instances), reported overall and across difficulty levels and puzzle categories. Details about the categories are provided in Appendix A.

4.3 RQ 3: Ablation Studies

We perform ablation studies to quantify the impact of (i) SFT, (ii) the logic-equivalence reward r_{equiv} , and (iii) tool sampling, as shown in Table 6. (i) Removing the SFT stage causes the largest degradation in SAT-Bench accuracy, from 97.2% to 54.9%; the latter is comparable to random guessing when responses follow the required parseable answer format. (ii) After removing r_{equiv} from training, inspection of generated rollouts reveals reward-hacking behavior, including degenerate tool calls such as “from z3 import *\nprint('z3 imported ')", which superficially obtain the r_{tool} reward. This reduces the in-domain accuracy to 91.2%, with only 80.6% of solver calls having formula encodings equivalent to the ground-truth formulas. (iii) Ablating tool sampling incurs a slight accuracy drop and causes a severe collapse in solver diversity: over 99% of tool calls are routed to z3, as illustrated in Figure 4. It also reduces Enigmata accuracy to 52.7%. In contrast, enabling tool sampling yields a substantially more balanced distribution across symbolic solvers. Together, these ablations demonstrate that logic-equivalence checking, SFT, and tool sampling are all necessary for robust end-to-end performance in SAT-RL. Figure 3 summarizes the corresponding logic-equivalence trajectories.

5 Conclusion

We present SAT-RL, an RL framework that integrates synthetic data and symbolic solver interaction into LLM training, enabling accurate logical reasoning and faithful solver use. Experiments show that SAT-RL improves in-domain logical reasoning and generalizes to unseen out-of-domain benchmarks, outperforming base LLMs and ReTool baselines.

Setup	Accuracy	Equivalence
SAT-RL	97.2 $\pm$ 0.8	94.0 $\pm$ 1.7
w/o SFT	54.9 $\pm$ 8.4	48.1 $\pm$ 6.1
w/o TS	97.0 $\pm$ 2.2	93.9 $\pm$ 4.3
w/o r_{equiv}	91.2 $\pm$ 2.0	80.6 $\pm$ 3.1

Table 6: In-domain final-answer accuracy and logic-equivalence rate of the encoded formulas (%) on SAT-Bench for Qwen3.5-9B under ablations. “TS” stands for Tool Sampling.

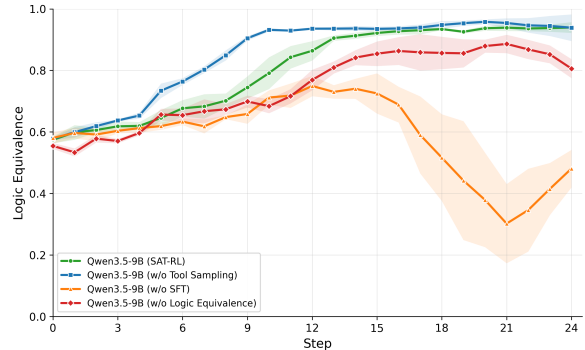


Figure 3: Logic-equivalence trajectories on SAT-Bench. In the “w/o Logic Equivalence” ablation, r_{equiv} is computed but excluded from the training reward.

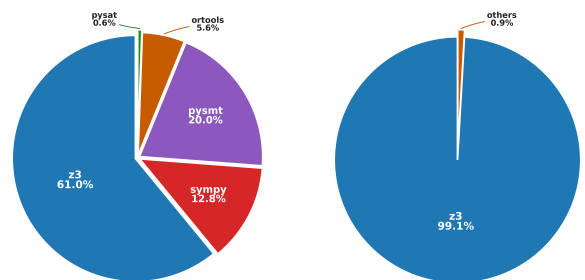


Figure 4: Distribution of symbolic solver usage with tool sampling enabled (left) and disabled (right).

Limitations

Our current study focuses on logical reasoning and a fixed set of widely used symbolic solvers, which limits the scope of this work. Broader program synthesis and open-ended natural-language reasoning remain to be explored. Future work will extend the framework to richer tool-using agent systems and more general coding and mathematical reasoning domains.

References

- Anthropic. 2025. [Claude Haiku 4.5 System Card](#). System card, Anthropic.
- Anthropic. 2026. [Claude Sonnet 5 System Card](#). System card, Anthropic. Published June 30, 2026.
- Haniel Barbosa, Clark Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, et al. 2022. cvc5: A versatile and industrial-strength smt solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 415–442. Springer.
- Bytedance-Seed-Foundation-Code-Team, :, Yao Cheng, Jianfeng Chen, Jie Chen, Li Chen, Liyu Chen, Wentao Chen, Zhengyu Chen, Shijie Geng, Aoyan Li, Bo Li, Bowen Li, Linyi Li, Boyi Liu, Jiaheng Liu, Kaibo Liu, Qi Liu, Shukai Liu, and 37 others. 2025. [Fullstack bench: Evaluating llms as full stack coders](#). Preprint, arXiv:2412.00535.
- Jiangjie Chen, Qianyu He, Siyu Yuan, Aili Chen, Zhicheng Cai, Weinan Dai, Hongli Yu, Jiaze Chen, Xuefeng Li, Qiying Yu, et al. 2025. Enigmata: Scaling logical reasoning in large language models with synthetic verifiable puzzles. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. 2024. Self-play fine-tuning converts weak language models to strong language models. *arXiv preprint arXiv:2401.01335*.
- Peter Clark, Oyvind Taffjord, and Kyle Richardson. 2020. [Transformers as soft reasoners over language](#). In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 3882–3890. ijcai.org.
- Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: An efficient smt solver. In *TACAS*.
- Lizhou Fan, Wenyue Hua, Lingyao Li, Haoyang Ling, and Yongfeng Zhang. 2024. [NPHardEval: Dynamic benchmark on reasoning ability of large language models via complexity classes](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4092–4114, Bangkok, Thailand. Association for Computational Linguistics.
- Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536*.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, Lucy Sun, Alexander Wardle-Solano, Hannah Szabó, Ekaterina Zubova, Matthew Burtell, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, and 16 others. 2024a. [FOLIO: natural language reasoning with first-order logic](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 22017–22031. Association for Computational Linguistics.
- Simeng Han, Aaron Yu, Rui Shen, Zhenting Qi, Martin Riddell, Wenfei Zhou, Yujie Qiao, Yilun Zhao, Semih Yavuz, Ye Liu, Shafiq Joty, Yingbo Zhou, Caiming Xiong, Dragomir Radev, Rex Ying, and Arman Co-han. 2024b. [P-FOLIO: evaluating and improving logical reasoning with abundant human-written reasoning chains](#). In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pages 16553–16565. Association for Computational Linguistics.
- Yaru Hao, Li Dong, Xun Wu, Shaohan Huang, Zewen Chi, and Furu Wei. 2025. On-policy rl with optimal reward baseline. *arXiv preprint arXiv:2505.23585*.
- Jian Hu. 2025. Reinforce++: A simple and efficient approach for aligning large language models. *arXiv preprint arXiv:2501.03262*.
- Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. 2018. Pysat: A python toolkit for prototyping with sat oracles. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 428–437. Springer.

- Richard M Karp. 2009. Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art*, pages 219–241. Springer.
- Mehran Kazemi, Quan Yuan, Deepti Bhatia, Najoung Kim, Xin Xu, Vaiva Imbrasaite, and Deepak Ramachandran. 2023. [Boardgameqa: A dataset for natural language reasoning with contradictory information](#). In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.
- Long Li, Zhijian Zhou, JIARAN HAO, Jason Klein Liu, Yanting Miao, Wei Pang, Xiaoyu Tan, Wei Chu, Zhe Wang, Shirui Pan, Chao Qu, and Yuan Qi. 2026. [The choice of divergence: A neglected key to mitigating diversity collapse in reinforcement learning with verifiable reward](#). In *The Fourteenth International Conference on Learning Representations*.
- Jingcong Liang, Shijun Wan, Xuehai Wu, Yitong Li, Qianglong Chen, Duyu Tang, Siyuan Wang, and Zhongyu Wei. 2025. Hardcorelogic: Challenging large reasoning models with long-tail logic puzzle games. *arXiv preprint arXiv:2510.12563*.
- Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. 2025. Zebralogic: On the scaling limits of llms for logical reasoning. In *International Conference on Machine Learning*, pages 37889–37905. PMLR.
- Hanmeng Liu, Jian Liu, Leyang Cui, Zhiyang Teng, Nan Duan, Ming Zhou, and Yue Zhang. 2023. [Logiqa 2.0 - an improved dataset for logical reasoning in natural language understanding](#). *IEEE ACM Trans. Audio Speech Lang. Process.*, 31:2947–2962.
- Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. 2020. [Logiqa: A challenge dataset for machine reading comprehension with logical reasoning](#). In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 3622–3628. ijcai.org.
- Qi Liu, Xinhao Zheng, Xudong Lu, Qinxiang Cao, and Junchi Yan. 2025. Rethinking and improving auto-formalization: Towards a faithful metric and a dependency retrieval-based approach. In *The Thirteenth International Conference on Learning Representations*.
- Chinmay Mittal, Krishna Kartik, Parag Singla, et al. 2024. Fcorebench: Can large language models solve challenging first-order combinatorial reasoning problems? *arXiv preprint arXiv:2402.02611*.
- Logan Murphy, Kaiyu Yang, Jialiang Sun, Zhaoyu Li, Anima Anandkumar, and Xujie Si. 2024. Autoformalizing Euclidean Geometry. In *Proceedings of the International Conference on Machine Learning*.
- Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. 2007. Minizinc: Towards a standard cp modelling language. In *International conference on principles and practice of constraint programming*, pages 529–543. Springer.
- Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. 2023. [Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning](#). *Preprint*, arXiv:2305.12295.
- Mihir Parmar, Nisarg Patel, Neeraj Varshney, Mutsumi Nakamura, Man Luo, Santosh Mashetty, Arindam Mitra, and Chitta Baral. 2024. [Logicbench: Towards systematic evaluation of logical reasoning ability of large language models](#).
- Nisarg Patel, Mohith Kulkarni, Mihir Parmar, Aashna Budhiraja, Mutsumi Nakamura, Neeraj Varshney, and Chitta Baral. 2024. [Multi-logieval: Towards evaluating multi-step logical reasoning ability of large language models](#).
- Chengwen Qi, Ren Ma, Bowen Li, He Du, Binyuan Hui, Jinwang Wu, Yuanjun Laili, and Conghui He. 2025. Large language models meet symbolic provers for logical reasoning evaluation. *arXiv preprint arXiv:2502.06563*.
- Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741.
- ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, et al. 2025. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*.
- Alexandre Riazanov and Andrei Voronkov. 2001. Vampire 1.1. In *International Joint Conference on Automated Reasoning*, pages 376–380. Springer.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. [Proximal policy optimization algorithms](#). *Preprint*, arXiv:1707.06347.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.
- Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. [Hybridflow: A flexible and efficient rlhf framework](#). In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys ’25, page 1279–1297. ACM.

- Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. 2025. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*.
- Koustuv Sinha, Shagun Sodhani, Jin Dong, Joelle Pineau, and William L. Hamilton. 2019. [CLUTRR: A diagnostic benchmark for inductive reasoning from text](#). In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4506–4515, Hong Kong, China. Association for Computational Linguistics.
- Stefan Szeider. 2024. Mcp-solver: Integrating language models with constraint programming systems. *arXiv preprint arXiv:2501.00539*.
- Qwen Team. 2026. Qwen3. 5-omni technical report. *arXiv preprint arXiv:2604.15804*.
- Jidong Tian, Yitian Li, Wenqing Chen, Liqiang Xiao, Hao He, and Yaohui Jin. 2021. [Diagnosing the first-order logical reasoning ability through logicnli](#). In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 3738–3747. Association for Computational Linguistics.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.
- Anjiang Wei, Yuheng Wu, Yingjia Wan, Tarun Suresh, Huanmi Tan, Zhanke Zhou, Sanmi Koyejo, Ke Wang, and Alex Aiken. 2025. Satbench: Benchmarking llms’ logical reasoning via automated puzzle generation from sat formulas. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 33820–33837.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.
- Yue Wu, Zhiqing Sun, Huizhuo Yuan, Kaixuan Ji, Yiming Yang, and Quanquan Gu. 2024. Self-play preference optimization for language model alignment. *arXiv preprint arXiv:2405.00675*.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.
- Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. 2023. Satlm: Satisfiability-aided language models using declarative prompting. *Advances in Neural Information Processing Systems*, 36:45548–45580.
- Qiyong Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. 2025. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.
- Weihao Yu, Zihang Jiang, Yanfei Dong, and Jiashi Feng. 2020. [Reclor: A reading comprehension dataset requiring logical reasoning](#). In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.
- Tao Zhang, Chenglin Zhu, Yanjun Shen, Wenjing Luo, Yan Zhang, Hao Liang, Fan Yang, Mingan Lin, Yujing Qiao, Weipeng Chen, et al. 2025. CFBench: A comprehensive constraints-following benchmark for LLMs. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32926–32944.
- Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2024. Agieval: A human-centric benchmark for evaluating foundation models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 2299–2314.
- Wanjun Zhong, Siyuan Wang, Duyu Tang, Zenan Xu, Daya Guo, Jiahai Wang, Jian Yin, Ming Zhou, and Nan Duan. 2021. Ar-lsat: Investigating analytical reasoning of text. *arXiv preprint arXiv:2104.06598*.
- Qin Zhu, Fei Huang, Runyu Peng, Keming Lu, Bowen Yu, Qinyuan Cheng, Xipeng Qiu, Xuanjing Huang, and Junyang Lin. 2025. [Autologi: Automated generation of logic puzzles for evaluating reasoning abilities of large language models](#). *CoRR*, abs/2502.16906.

A Components of the Enigmata Test Set

We report statistics for the Enigmata test set (Section 4.2), grouped by difficulty, puzzle category, and task, in Table 7.

Data Type	Number of Instances
All instances	1901
Difficulty	
Easy	560
Medium	642
Hard	699
Puzzle category	
Arithmetic	120
countdown	60
game24	60
Crypto	120
crypto_KKA	60
crypto_KPA	60
Graph	220
hamiltonian_cycle	60
hamiltonian_path	60
maze	60
natural_language_navigation	40
Grid	539
binario	60
campsite	60
magic_square	60
skyscraper	60
star_battle	60
sudoku	60
sudoku2	60
sum_skyscraper	60
full_crosswords	40
symbolic_hard	19
Logic	180
FOLIO	60
knights_and_knives	60
zebra_logic	60
Search	320
kakurasu	60
minesweeper	60
slant	60
hitori	40
light_up	40
tic_tac_toe	40
checkmate_in_one	20
Sequential	402
car_painting	60
eight_puzzle	60
fifteen_puzzle	60
nine_puzzle	60
sixteen_puzzle	60
stack_permutation	60
twiddle	22
big_bench_symbolic	20

Table 7: Number of instances in the Enigmata test set, grouped by difficulty, puzzle category, and puzzle name.

B Hyperparameter Settings

The SFT and RL training configurations are shown in Table 8. Training Qwen3.5-9B on SAT-Bench

require around 24 hours.

<i>Software Environment</i>	
Python	3.12.13
CUDA Toolkit	12.9
PyTorch	2.11.0
verl	0.8.0
vLLM	0.19.1
transformers	5.5.4
<i>Model and Context</i>	
Base Model	Qwen3.5 Model Family (bfloat16)
Max Prompt Length	4096
Max Response Length	8192
<i>SFT Hyperparameters</i>	
Train Batch Size	64
Micro Batch Size	4
Learning Rate	1×10^{-5}
Total Epochs	1
<i>RL Hyperparameters</i>	
Train Batch Size	128
Micro Batch Size (Log Probabilities)	4
Micro Batch Size (PPO per GPU)	4
Learning Rate	1×10^{-6}
KL Loss	0.0 (Disabled)
Clip Ratio ϵ_{low}	0.2
Clip Ratio ϵ_{high}	0.28
Clip Ratio c	10.0
<i>Rollout Settings</i>	
DAPO Group Size (roll-out.n)	8
Rollout Backend	vLLM
Temperature	1.0
GPU Memory Utilization	0.6
<i>Training Schedule</i>	
Total Epochs	2
Test Frequency (steps)	1
<i>Distributed Setup</i>	
Actor Strategy	FSDP2
Number of Nodes	2
GPUs per Node	$4 \times \text{L40S (48 GB)}$
CPUs per Node	64 Intel Xeon Gold 6448Y

Table 8: Training configuration.

B.1 Description of Synthetic Puzzles

Clique (K-Clique). Given an undirected graph, the task is to select a set of vertices that forms a clique of a target size. Hyperparameters: number of vertices N , clique size K , and edge list edges $\subseteq \{(u, v) : 0 \leq u < v < N\}$.

Feedback Node Set (Directed Feedback Vertex Set). Given a directed graph, the task is to remove

a small set of vertices so that the remaining graph becomes acyclic. Hyperparameters: number of vertices N , removal budget K , and directed edge list edges $\subseteq V \times V$.

Graph K -Coloring. Given an undirected graph, the task is to assign one of K colors to each vertex such that adjacent vertices have different colors. Hyperparameters: number of vertices N , number of colors K , and edge list edges.

Max Cut (Threshold Formulation). Given an undirected graph, the task is to find a binary partition of vertices whose cut size meets a target threshold. Hyperparameters: number of vertices N , threshold T (required number of crossing edges), and edge list edges.

Partition. Given positive integers, the task is to split them into two groups with equal sums (equivalently, find a subset summing to half the total). Hyperparameters: number of items N and the integer array arr (with total sum determining the implied target).

Subset Sum. Given positive integers and a target, the task is to choose a subset whose sum equals the target exactly. Hyperparameters: number of items N , target sum T , and integer array arr .

Traveling Salesman Problem (Threshold Variant). Given a complete weighted graph, the task is to find a Hamiltonian tour whose total cost satisfies a threshold constraint. Hyperparameters: number of nodes N , tour-cost threshold T , and weight matrix $W \in \mathbb{Z}^{N \times N}$.

Set Covering. Given a universe and a family of subsets, the task is to select a small number of sets whose union covers the entire universe. Hyperparameters: universe size U , number of sets M , budget parameter K , and set family sets $= [S_0, \dots, S_{M-1}]$ with $S_i \subseteq \{0, \dots, U-1\}$.

Set Packing. Given a universe and a family of subsets, the task is to select a prescribed number of pairwise disjoint sets. Hyperparameters: universe size U , number of sets M , packing size K , and set family sets $= [S_0, \dots, S_{M-1}]$.

NFA Path Problems. Given an NFA (as an explicit transition relation), the task is to find an accepted path of a fixed length from a start state to an accepting state. Hyperparameters: number of states S , alphabet size A , path length L , start state

start, accepting set accepts, and transition list transitions $= [(s, a, t)]$.

Binary Neural Network (BNN) Robustness. Given a threshold-gate BNN and a seed input, the task is to find an input within a Hamming ball that flips the network output. Hyperparameters: input dimension n_{in} , network structure layers (neurons with threshold k and signed literals), and property dict prop containing seed, radius $\varepsilon = \text{eps}$, and y_{seed} .

Partial N-Queens. Given a partially filled $N \times N$ chessboard, the task is to place one queen in each row while respecting the pre-placed queens and nonattacking constraints. Hyperparameters: board size N and fixed queen positions $\text{pre} = [(row, col)]$.

Generalized Sudoku. Given an $N \times N$ Sudoku grid (with $N = n^2$), the task is to fill in the blanks while satisfying row, column, and sub-block uniqueness constraints. Hyperparameters: grid size N and the partially filled grid given_grid (where 0 or . denotes a blank).

Magic Square. Given an $N \times N$ grid, the task is to assign integers so that every row, every column, and both main diagonals sum to the same magic constant. Hyperparameters: order N , value domain (typically $\{1, \dots, N^2\}$ with an all-different constraint), and an optional set of pre-filled cells $\text{givens} = [(r, c, v)]$.

Latin Square. Given an $N \times N$ grid over symbols $\{1, \dots, N\}$, the task is to fill the grid so that each symbol appears exactly once in every row and every column. Hyperparameters: order N and an optional partially filled grid $\text{given_grid} \in (\{0, 1, \dots, N\})^{N \times N}$, where 0 denotes blanks.

Futoshiki. Given an $N \times N$ Latin-square-style grid together with inequality clues between neighboring cells, the task is to fill the grid with symbols $\{1, \dots, N\}$ satisfying both row/column uniqueness and all inequality constraints. Hyperparameters: order N , an optional partially filled grid given_grid , and a set of directed inequality edges between adjacent cells $\text{ineq} = [((r_1, c_1), (r_2, c_2))]$, where each edge denotes $(r_1, c_1) < (r_2, c_2)$.

C Prompt Templates

Below, we present the LLM template used for SAT-RL training on SAT-Bench.

Input

Solve the problem step by step and put your final answer in the format
`<answer>`.

If the evidence is genuinely insufficient after using the tool, you may output `idk`.

For this instance, you must import and use the `{{sampled_tool}}` as the solver/tool call.

Variable Mapping:
`{{variable_mapping}}`

Problem Statement:
`{{problem}}`

Below, we present the LLM template used for training in the other settings. This template is also used for all evaluations.

Input

Solve the problem step by step and put your final answer in the format
`<answer>`.

Problem Statement:
`{{problem}}`

D OR-Tools Example

Below is an example OR-Tools program.

```
from ortools.sat.python import cp_model

class Counter(cp_model.CpSolverSolutionCallback):
    def __init__(self, x, y, want_print=True):
        super().__init__()
        self._x, self._y = x, y
        self.count = 0
        self.want_print = want_print

    def on_solution_callback(self):
        self.count += 1
        if self.want_print:
            print(f"Solution {self.count}:")
            x={self.Value(self._x)},
            y={self.Value(self._y)}")

model = cp_model.CpModel()

# Variables (nonnegative integers)
x = model.NewIntVar(0, 50, "x") # upper
    bounds can be loose
y = model.NewIntVar(0, 50, "y")

# Constraint: 4x + 6y = 50
model.Add(4 * x + 6 * y == 50)
```

```
solver = cp_model.CpSolver()
cb = Counter(x, y, want_print=True)
solver.SearchForAllSolutions(model, cb)

print("Number of solutions:", cb.count)
```

E Solved Examples

Below, we show examples solved during training and evaluation.

E.1 Solved SAT-Bench Example

Below, we show an example SAT-Bench instance successfully solved by Qwen3.5-9B during SAT-RL training. This instance requires the use of the Z3 solver. We omit the repeated system prompt template and directly show the instance below.

Input. The input instance is as follows:

Scenario:

A team of three scientists—Anna, Ben, and Clara—are conducting research on two phenomena (0 = gravitational waves, 1 = dark matter). They conduct each experiment in two different environments (0 = laboratory, 1 = field). For each scientist, phenomenon, and environment, we decide whether the experiment was successfully conducted. Each decision is independent.

Conditions:

1. Either Clara successfully conducts an experiment on dark matter in the field, or Anna does not successfully conduct an experiment on dark matter in the laboratory.
2. Either Clara successfully conducts an experiment on dark matter in the field, or Ben does not successfully conduct an experiment on gravitational waves in the laboratory.
3. Either Clara does not successfully conduct an experiment on dark matter in the field, or Ben does not successfully conduct an experiment on gravitational waves in the laboratory.
4. Either Clara successfully conducts an experiment on dark matter in the field, or Ben successfully conducts an experiment on gravitational waves in the laboratory.

Question:

Can all these experimental conditions be simultaneously satisfied?

Output. The model's output is as follows:

```
<think>
Let me analyze this problem step by step.

First, I need to understand the variables:
- x(i, j, k) means scientist i successfully
  conducts an experiment on phenomenon j in
  environment k
- i in {0, 1, 2} (Anna, Ben, Clara)
- j in {0, 1} (gravitational waves, dark matter)
- k in {0, 1} (laboratory, field)
```

Now let me translate the conditions:

Condition 1: Either Clara successfully conducts an experiment on dark matter in the field, or Anna does not successfully conduct an experiment on dark matter in the laboratory.

- Clara = $i=2$, dark matter = $j=1$, field = $k=1$: $x(2, 1, 1)$
- Anna = $i=0$, dark matter = $j=1$, laboratory = $k=0$: $x(0, 1, 0)$
- Condition 1: $x(2, 1, 1) \vee \neg x(0, 1, 0)$

Condition 2: Either Clara successfully conducts an experiment on dark matter in the field, or Ben does not successfully conduct an experiment on gravitational waves in the laboratory.

- Clara = $i=2$, dark matter = $j=1$, field = $k=1$: $x(2, 1, 1)$
- Ben = $i=1$, gravitational waves = $j=0$, laboratory = $k=0$: $x(1, 0, 0)$
- Condition 2: $x(2, 1, 1) \vee \neg x(1, 0, 0)$

Condition 3: Either Clara does not successfully conduct an experiment on dark matter in the field, or Ben does not successfully conduct an experiment on gravitational waves in the laboratory.

- Clara = $i=2$, dark matter = $j=1$, field = $k=1$: $x(2, 1, 1)$
- Ben = $i=1$, gravitational waves = $j=0$, laboratory = $k=0$: $x(1, 0, 0)$
- Condition 3: $\neg x(2, 1, 1) \vee \neg x(1, 0, 0)$

Condition 4: Either Clara successfully conducts an experiment on dark matter in the field, or Ben successfully conducts an experiment on gravitational waves in the laboratory.

- Clara = $i=2$, dark matter = $j=1$, field = $k=1$: $x(2, 1, 1)$
- Ben = $i=1$, gravitational waves = $j=0$, laboratory = $k=0$: $x(1, 0, 0)$
- Condition 4: $x(2, 1, 1) \vee x(1, 0, 0)$

Let me use Z3 to check if these conditions can be simultaneously satisfied.

For Z3, I'll use $x_{2,1,1}$ for $x(2,1,1)$ and $x_{1,0,0}$ for $x(1,0,0)$.

```
</think>
<tool_call>
from z3 import Bool, Or, Not, And, Solver, sat,
    unsat
```

```
# Define the variables
x_2_1_1 = Bool("x_2_1_1") # Clara, dark
    matter, field
x_1_0_0 = Bool("x_1_0_0") # Ben, gravitational
    waves, laboratory
```

```
# Create the conditions
condition1 = Or(x_2_1_1, Not(x_0_1_0)) if False
    else Or(x_2_1_1, Not(Bool("x_0_1_0")))
condition2 = Or(x_2_1_1, Not(x_1_0_0))
condition3 = Or(Not(x_2_1_1), Not(x_1_0_0))
condition4 = Or(x_2_1_1, x_1_0_0)
```

```
# Create solver and add constraints
solver = Solver()
```

```
solver.add(And(condition2, condition3,
    condition4))
```

```
# Check satisfiability
result = solver.check()
print(result)
</tool_call>
```

```
<tool_response>
stdout: sat
</tool_response>
<think>
```

The solver returned "sat" which means the conditions are satisfiable. This means all conditions can be simultaneously satisfied.

Therefore, the answer is yes.

```
</think>
```

```
```yes```
```

**Reward.** The reward score is as follows:

```
{"format": 0.1, "correctness": 1.0,
 "tool_success": 0.1, "equivalence": 1.0}
```

**Justification.** The model's response contains a correctly parsed answer and a compilable Z3 tool call. The variable mapping is as follows:  $x_{2,1,1}$  is mapped to "Clara conducts the dark-matter experiment in the field",  $x_{1,0,0}$  is mapped to "Ben conducts the gravitational-wave experiment in the laboratory", and  $x_{0,1,0}$  is mapped to "Anna conducts the dark-matter experiment in the laboratory".

The ground-truth formula is  $F_{\text{truth}} = (x_{2,1,1} \vee \neg x_{0,1,0}) \wedge (x_{2,1,1} \vee \neg x_{1,0,0}) \wedge (\neg x_{2,1,1} \vee \neg x_{1,0,0}) \wedge (x_{2,1,1} \vee x_{1,0,0})$ .

The extracted formula is  $F_{\text{pred}} = (x_{2,1,1} \vee \neg x_{1,0,0}) \wedge (\neg x_{2,1,1} \vee \neg x_{1,0,0}) \wedge (x_{2,1,1} \vee x_{1,0,0})$ . Although  $F_{\text{pred}}$  does not include the clause  $(x_{2,1,1} \vee \neg x_{0,1,0})$ , it is still logically equivalent to  $F_{\text{truth}}$  (that is,  $F_{\text{truth}} \leftrightarrow F_{\text{pred}}$ ). In fact, the truth table is

$x_{2,1,1}$	$x_{1,0,0}$	$x_{0,1,0}$	$F_1$	$F_2$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0

## E.2 Solved Enigmata Example

**Input.** The input instance is as follows:

1. **\*\*NOTATIONS:\*\***

The initial board consists of empty cells and blocked cells.

Empty cells are denoted by '.', blocked cells are denoted by 'X', and stars are denoted by '\*'.

- \*\*STAR MUST BE PLACED IN EMPTY CELL:\*\***  
Each star must be placed in an EMPTY cell. Blocked cells cannot contain stars. You can only change cells denoted by '.', and must not change cells denoted by 'X'.
- \*\*EXACTLY 1 STAR IN EACH ROW AND COLUMN:\*\***  
Each row and column must contain EXACTLY one star.  
No two stars can be in the same row or column.  
There shouldn't be rows or columns without stars.
- \*\*NO ADJACENT STARS ROW-WISE, COLUMN-WISE, OR DIAGONALLY:\*\***  
No two stars can be adjacent to each other, even diagonally.
  - Row-wise adjacency: two stars are in the same row, and there is no empty cell between them.
  - Column-wise adjacency: two stars are in the same column, and there is no empty cell between them.
  - Diagonal adjacency: two stars are in the same diagonal, and there is no empty cell between them.
 Note that you are prone to make mistakes in diagonal adjacency, so be extra careful. Here are 2 examples of diagonal adjacency in a partial grid:
 

```

...
. * . .
. . * .
...
...
. . * .
. * . .
...

```

 In both cases, the two stars are diagonally adjacent. You should avoid this situation.
- \*\*CHECK FOR CONSTRAINTS AND BACKTRACK:\*\***  
In each step, you should check if it violates the constraints in 2., 3., and 4. If you find inconsistencies, you should backtrack and try a different placement. If you find that you can't place a star anywhere without violating the constraints, you should backtrack and try a different placement.  
Since there is only one solution, there is great chance your first placement is incorrect, which means you might need to start over.
- \*\*RESPONSE FORMAT:\*\***  
Your response should include the final board.  
The final board is a revised version of the initial board in a way that if you need to place a star in an empty cell, replace the '.' with a '\*'.  
Don't change the blocked cells denoted by 'X'.  
Your final board should be wrapped between <begin\_board> and <end\_board> tags.  
Use the following structure:  
  
Final Board:  
...

```

<begin_board>
[Final Board]
<end_board>
...

```

Here is the puzzle:

```

. X X . X . .
. X X . . X .
. X X . X X X
.
. . . . X X .
. . X . X X X
.

```

**Output.** The model's output is as follows:

```

<think>
The user wants me to solve a star-battle puzzle.

```

Let's start by setting up the model with pysat, then solve, then print the result:

```

Step 1: Import pysat and define binary
variables `star[r][c]`.
Step 2: Add constraints for each row/col to
have exactly one star (clauses like
`sum(star[r][c] for all c) == 1`).
Step 3: Add constraints to prevent
adjacent/diagonal stars (no variable equals
any neighbor in the 3x3 area).
Step 4: Add constraints on blocked cells (each
star variable for a blocked cell must be
false).
Step 5: Solve the instance.
Step 6: Extract the positions where
`star[r][c]` is true and construct the
final board.
</think>

```

```

<tool_call>
from pysat.formula import CNF, IDPool
from pysat.solvers import Glucose3

def solve():
 puzzle = [
 [".", "X", "X", ".", "X", ".", "."],
 [".", "X", "X", ".", ".", "X", "."],
 [".", "X", "X", ".", "X", "X", "X"],
 [".", ".", ".", ".", ".", ".", "."],
 [".", ".", ".", ".", "X", "X", "."],
 [".", ".", "X", ".", "X", "X", "X"],
 [".", ".", ".", ".", ".", ".", "."],
]
 rows = len(puzzle)
 cols = len(puzzle[0])
 vpool = IDPool()
 stars = [[vpool.id(f"star_{r}_{c}") for c
 in range(cols)] for r in range(rows)]
 cnf = CNF()
 for r in range(rows):
 cnf.append(list(stars[r]))
 for c in range(cols):
 cnf.append([~stars[r][c]])
 for c in range(cols):
 cnf.append(list(stars[r][c] for r in
 range(rows)))
 for r in range(rows):
 for c in range(cols):
 for dr in (-1, 0, 1):
 for dc in (-1, 0, 1):

```

```

 if dr == 0 and dc == 0:
 continue
 nr, nc = r + dr, c + dc
 if 0 <= nr < rows and 0 <=
nc < cols:

cnf.append([-stars[r][c], -stars[nr][nc]])
for r in range(rows):
 for c in range(cols):
 if puzzle[r][c] == "X":
 cnf.append([-stars[r][c]])

with Glucose3(bootstrap_with=cnf.clauses)
as solver:
 if not solver.solve():
 return "Unsolvable"
 model = set(solver.get_model())

planted = []
for r in range(rows):
 for c in range(cols):
 if stars[r][c] in model:
 planted.append((r, c))
final_board = []
for r in range(rows):
 row_chars = []
 for c in range(cols):
 if puzzle[r][c] == "X":
 row_chars.append("X")
 elif (r, c) in planted:
 row_chars.append("*")
 else:
 row_chars.append(".")
 final_board.append("".join(row_chars))
return "\n".join(final_board)

print(solve())
</tool_call>
<tool_response>
stdout: Unsolvable
</tool_response>
<think>
The previous solver returned "Unsolvable",
which should not be possible for a
well-formed puzzle with a unique solution.
I must have made an error in how the
constraints were modeled—specifically the
adjacency constraint may be too restrictive
or incorrectly implemented. Let me
carefully reconsider the diagonal adjacency
model and simplify the exact-1 constraints
using linear expressions, then retry the
solver call.
</think>

<tool_call>
from pysat.formula import CNF, IDPool
from pysat.solvers import Glucose3
from pysat.card import CardEnc, EncType

def solve():
 puzzle = [
 [".", "X", "X", ".", "X", ".", "."],
 [".", "X", "X", ".", "X", ".", "."],
 [".", "X", "X", ".", "X", "X", "X"],
 [".", ".", ".", ".", ".", ".", "."],
 [".", ".", ".", ".", "X", "X", "."],
 [".", ".", "X", ".", "X", "X", "X"],
 [".", ".", "X", ".", "X", "X", "X"],
]

```

```

 [".", ".", ".", ".", ".", ".", "."],
]
 rows = len(puzzle)
 cols = len(puzzle[0])
 vpool = IDPool()
 stars = [[vpool.id(f"star_{r}_{c}") for c
in range(cols)] for r in range(rows)]
 cnf = CNF()
 for r in range(rows):

 cnf.extend(CardEnc.equals(lits=stars[r],
bound=1, vpool=vpool,
encoding=EncType.seqcounter).clauses)
 for c in range(cols):

 cnf.extend(CardEnc.equals(lits=[stars[r][c]
for r in range(rows)], bound=1,
vpool=vpool,
encoding=EncType.seqcounter).clauses)
 for r in range(rows):
 for c in range(cols):
 for dr in (-1, 0, 1):
 for dc in (-1, 0, 1):
 if dr == 0 and dc == 0:
 continue
 nr, nc = r + dr, c + dc
 if 0 <= nr < rows and 0 <=
nc < cols:

cnf.append([-stars[r][c], -stars[nr][nc]])
for r in range(rows):
 for c in range(cols):
 if puzzle[r][c] == "X":
 cnf.append([-stars[r][c]])

with Glucose3(bootstrap_with=cnf.clauses)
as solver:
 if not solver.solve():
 return "Unsolvable"
 model = set(solver.get_model())

planted = []
for r in range(rows):
 for c in range(cols):
 if stars[r][c] in model:
 planted.append((r, c))
final_board = []
for r in range(rows):
 row_chars = []
 for c in range(cols):
 if puzzle[r][c] == "X":
 row_chars.append("X")
 elif (r, c) in planted:
 row_chars.append("*")
 else:
 row_chars.append(".")
 final_board.append("".join(row_chars))
return "\n".join(final_board)

print(solve())
</tool_call>
<tool_response>
stdout: .XX.X*.
.XX*.X.
*XX.XXX
..*....
....XX*
.*X.XXX
....*..

```

</tool\_response>

<think>

The solver returned a valid solution:

...

```
.XX.X*.
.XX*.X.
*XX.XXX
..*....
....XX*
.*X.XXX
....*..
...
```

This should be formatted according to the required output structure with the <begin\_board> and <end\_board> tags. Let me verify this solution is correct:

- Row 0: star at column 5
- Row 1: star at column 3
- Row 2: star at column 0
- Row 3: star at column 2
- Row 4: star at column 6
- Row 5: star at column 1
- Row 6: star at column 4

This gives all columns [5,3,0,2,6,1,4], which are all distinct (0-6 exactly once). No row or column has more than one star. Checking diagonally adjacent stars:

- Row 0 col 5 and Row 1 col 3: not adjacent
- Row 1 col 3 and Row 2 col 0: not adjacent
- Row 2 col 0 and Row 3 col 2: not adjacent
- Row 3 col 2 and Row 4 col 6: not adjacent
- Row 4 col 6 and Row 5 col 1: not adjacent
- Row 5 col 1 and Row 6 col 4: not adjacent

None are adjacent horizontally, vertically, or diagonally. The placement also respects the blocked cells ('X'). This is the correct and unique solution.

</think>

Final Board:

...

<begin\_board>

```
.XX.X*.
.XX*.X.
*XX.XXX
..*....
....XX*
.*X.XXX
....*..
<end_board>
```

...

## F Additional Training Trajectories

Additional training trajectories of detailed reward component, the test correctness and entropy is shown to Figure 5.

**Justification:** The first PySAT call creates variables for all 49 cells and adds an at-least-one clause for each row, but this formulation produces an UNSAT result that contradicts the task’s stated unique-solution premise. The model then corrects the formula and obtains a satisfying solution. The star columns by row are (5, 3, 0, 2, 6, 1, 4). These placements ensure that every row and column has exactly one star, that each star replaces an empty cell, and that the star columns in adjacent rows differ by more than one. Therefore, the solution is correct, as confirmed by the verifier.

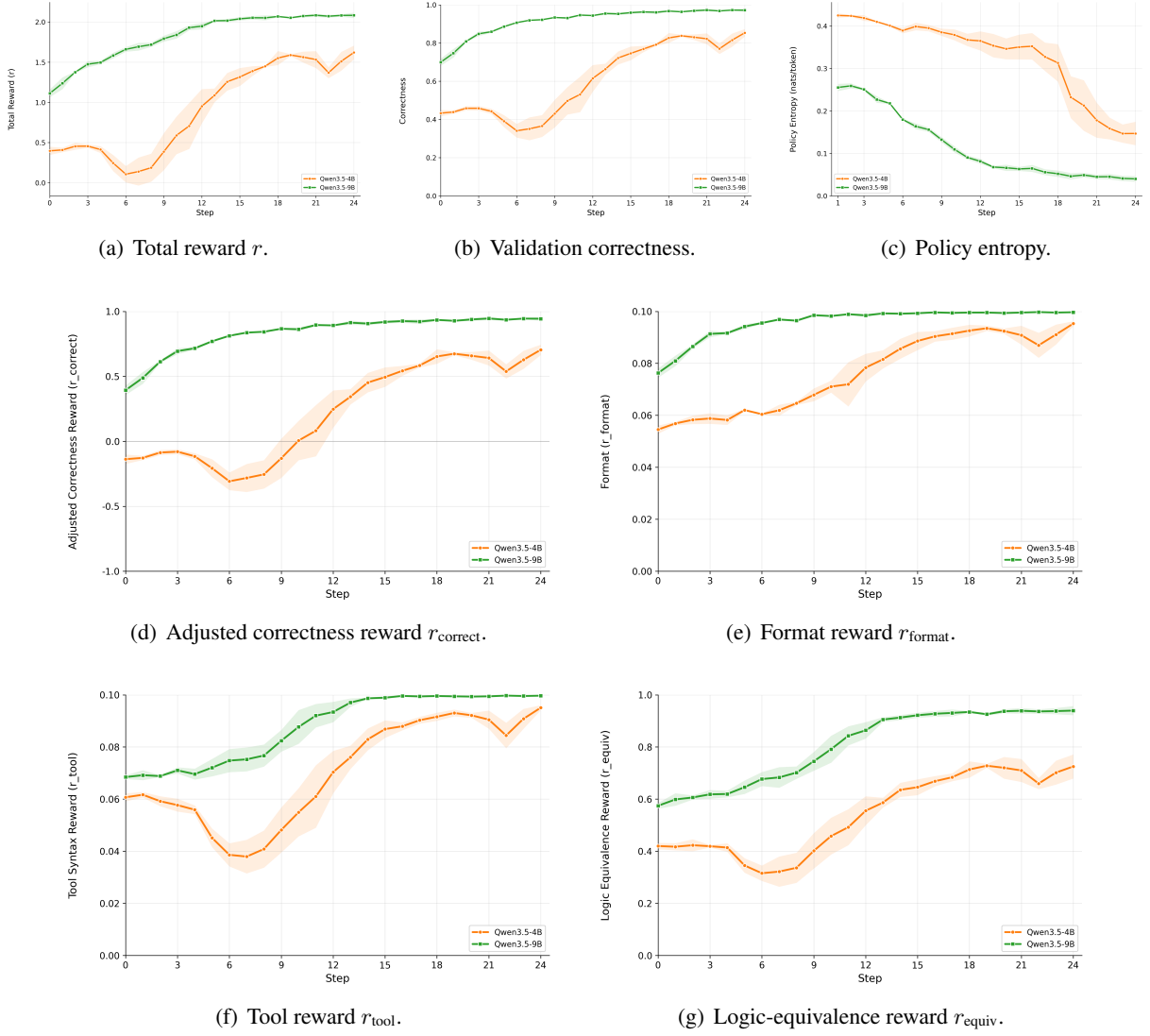


Figure 5: SAT-RL training trajectories on SAT-Bench for Qwen3.5-4B and Qwen3.5-9B. Curves show means over three runs; shading shows population standard deviation.