Rewarding Novel Deductions: Solver-guided Process Supervision for Logical Reasoning

Muhammad Asif Ali[1,2], Wenqing Wang[3], Huan Wang[3], Mohammad Raza[4,†]

[1]FORTE Lab, [2]Faculty of Science, Information Technology University, Lahore, Pakistan
[3]College of Informatics, Huazhong Agricultural University, Wuhan, China
[4]Qatar Computing Research Institute, Hamad Bin Khalifa University, Doha, Qatar

[†]**Corresponding author:** mraza@hbku.edu.qa

Abstract

Logical reasoning remains a major challenge for large language models (LLMs), particularly on structured problems that require precise constraint tracking, consistency preservation, and multi-step deduction. This challenge is especially acute for small-scale LLMs, which are more prone to producing inconsistent, redundant, or brittle reasoning trajectories. Existing approaches for improving logical reasoning largely optimize for final-answer correctness, providing only weak supervision over the intermediate reasoning process. In this work, we propose **SPRING**: (**S**olver-guided **P**rocess **R**ewards for Novel Log**I**cal Reaso**N**ing Step **G**eneration). **SPRING** uses an SMT solver as a training-time verifier of intermediate reasoning steps to provide process-level supervision. It introduces the notion of a *novel reasoning step*, namely, a step that is logically valid, consistent with the evolving reasoning state, and not already implied by previously accepted non-contradictory deductions. Based on this solver-based assessment, it designs process rewards that encourage novel inferential progress while penalizing contradictory and uninformative reasoning steps. Evaluation across three logical reasoning benchmarks: ZebraLogic, AR-LSAT, and Knights and Knaves, and four LLMs shows that **SPRING** consistently outperforms base LLMs, outcome-only reward baselines, and Logic-LM. On ZebraLogic, SPRING improves puzzle accuracy by up to 49.71 and 15.43 points over the base LLM and strongest outcome-only baseline, respectively. On AR-LSAT, it improves overall accuracy by up to 64.43 and 11.39 percentage points over the base LLM and strongest outcome-only baseline, respectively. On Knights and Knaves, **SPRING** achieves up to 93.14 puzzle accuracy and 96.05 person accuracy.

1 Introduction

Logical reasoning is a fundamental capability for intelligent systems because it requires deriving conclusions that remain consistent with a set of premises and constraints. Unlike open-ended generation tasks, logical reasoning problems such as relational puzzles, symbolic deduction, and structured constraint-satisfaction tasks require not only a correct final answer, but also a sequence of intermediate deductions that is logically sound and progressively informative. Although LLMs have shown strong performance on a wide range of reasoning benchmarks, they still struggle with tasks that require precise deduction, consistency tracking, and long-horizon constraint maintenance [Wei et al., 2022, Kojima et al., 2022, Mirzadeh et al., 2025]. Their reasoning chains are often redundant, unsupported by the premises, or internally contradictory, even when the final answer appears plausible [Lightman et al., 2024, Cobbe et al., 2021, Mirzadeh et al., 2025]. This limitation is especially pronounced for *small-scale* LLMs, particularly models in the 4B-and-below regime, which often require stronger or more structured supervision to maintain coherent multi-step reasoning trajectories [Li et al., 2025,

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Zhang et al., 2025]. While such models are attractive because of their lower computational cost and easier deployment, they are more prone to breakdowns in multi-step logical reasoning and often struggle to maintain coherent reasoning traces over long deduction chains [Wei et al., 2022, Li et al., 2025, Zhang et al., 2025].

Motivated by these limitations, a growing line of work augments LLM reasoning with symbolic solvers such as SAT, SMT, Z3, and theorem provers, which provide exact checks for satisfiability, implication, and consistency [Barrett and Tinelli, 2018, De Moura and Bjørner, 2008]. Recent neurosymbolic methods improve reliability by translating natural-language problems into executable formal representations and delegating inference or verification to the solver [Ye et al., 2023, Pan et al., 2023, Olausson et al., 2023, Hu et al., 2025]. More recent works have also begun exploring the use of formal solvers and theorem provers for process supervision during training, providing feedback on the correctness of intermediate reasoning steps [Kim and Yun, 2026]. However, logical correctness alone does not guarantee meaningful reasoning progress. An LLM may repeatedly restate previously established facts or generate trivial variations of existing deductions to produce logically valid but redundant steps that do not make any inferential progress. Effective process supervision must therefore go beyond verifying the validity of individual deductions to also assess whether they contribute new information to the evolving reasoning state and make genuine progress towards a solution.

To address this challenge, we introduce **SPRING** (**S**olver-guided **P**rocess **R**ewards for Novel Log**I**cal Reaso**N**ing Step **G**eneration), a reinforcement learning framework that uses symbolic solvers to explicitly reward inferential progress during logical reasoning. Central to **SPRING** is the notion of rewarding *novel reasoning steps*. A key question is how one distinguishes genuinely novel deductions from redundant ones, since both must be logically implied by the problem premises. We address this by distinguishing between deductions that follow from the premises and those that are already implied by previously established reasoning steps alone: a deduction is considered novel if it is logically entailed by the premises but not by the preceding deductions in isolation. This distinction provides a principled basis for rewarding reasoning that progressively extends what has already been established. During reinforcement learning, a symbolic solver evaluates intermediate deductions to provide fine-grained process feedback, rewarding novel, logically supported deductions while penalizing contradictions and discouraging redundant or uninformative steps.

We evaluate **SPRING** across three logical reasoning benchmarks, ZebraLogic, AR-LSAT, and Knights and Knaves, using four LLMs spanning different model sizes and reasoning capabilities. Experimental results show that **SPRING** consistently outperforms base LLMs, outcome-only reward baselines, and Logic-LM. On ZebraLogic, **SPRING** improves puzzle accuracy by up to 49.71 points over the base LLM and 15.43 points over the strongest outcome-only baseline. On AR-LSAT, **SPRING** improves overall accuracy by up to 64.43 and 11.39 points over the base LLM and strongest outcome-only baseline, respectively. On Knights and Knaves, **SPRING** achieves up to 93.14 puzzle accuracy and 96.05 person accuracy. Beyond final-task performance, our analysis shows that **SPRING** produces shorter and cleaner successful traces, fewer contradictions, and reasoning trajectories that are more consistent with the underlying logical constraints. We summarize the key contributions of this work as follows:

- We formulate logical reasoning training as a process-supervised reinforcement learning problem that explicitly rewards inferential progress, rather than relying solely on final-answer correctness or intermediate-step validity.
- We introduce the notion of novel reasoning steps and show how a symbolic solver can verify them during training, thereby providing explicit supervision for inferential progress, validity, and logical consistency.
- We propose **SPRING**, a solver-guided reward framework that improves both reasoning trajectories and final logical reasoning performance by rewarding novel deductions while penalizing contradictory and uninformative reasoning steps.
- We conduct comprehensive experiments across three logical reasoning benchmarks and four LLMs, showing that **SPRING** consistently outperforms base LLMs, outcome-only reward baselines, and Logic-LM. Our analyses further show that **SPRING** produces shorter, cleaner, and more solver-consistent reasoning trajectories.

2 Related Work

Solver-Augmented Logical Reasoning for LLMs. A prominent line of recent work improves logical reasoning in LLMs by coupling them with external symbolic solvers rather than relying on unconstrained natural-language reasoning alone. In these approaches, the LLM typically serves as a semantic parser or translator that maps natural-language premises into a formal representation, while a symbolic backend performs exact deductive inference or consistency checking. This paradigm has become especially important for structured logical reasoning tasks, where global constraints, exclusivity conditions, and combinatorial dependencies are difficult to maintain through free-form chain-of-thought alone. Benchmark development has reinforced this need: AR-LSAT, introduced in *Analytical Reasoning of Text*, remains a challenging dataset for structured logical deduction, while more recent puzzle-style benchmarks such as ZebraLogic expose substantial performance drops as deductive complexity increases [Zhong et al., 2022, Lin et al., 2025].

Among solver-integrated methods, SatLM formulates reasoning problems declaratively, using the LLM to generate a satisfiability specification that is then solved by an automated theorem prover or SAT-style backend [Ye et al., 2023]. Logic-LM further develops this paradigm by translating natural-language reasoning problems into symbolic programs, invoking a deterministic solver, and then using self-refinement to improve formalization quality; importantly, it reports gains on multiple logical reasoning benchmarks including AR-LSAT [Pan et al., 2023]. LINC follows a similar neurosymbolic design, but grounds reasoning in first-order logic and theorem proving, showing that explicit logical formalization can substantially improve deductive reliability over purely textual reasoning [Olausson et al., 2023]. More recently, LTRAG improves this general family of methods by enhancing autoformalization and self-refinement with retrieval-augmented exemplars, and reports further gains over Logic-LM and LINC on FOLIO and AR-LSAT [Hu et al., 2025]. Collectively, these works show that symbolic solvers can significantly augment LLMs on logical reasoning tasks when the model is able to produce executable formal representations.

At the same time, recent studies show that the benefit of solver augmentation depends not only on whether a solver is used, but also on *how* it is used and how well the LLM can interface with the solver's symbolic language. Lam et al. [2024] perform a controlled comparison across multiple symbolic tools, including Z3, Pyke, and Prover9, and show that executable translation quality varies substantially across tools and is strongly tied to downstream reasoning accuracy. This observation is important because it clarifies that many current solver-augmented pipelines are bottlenecked by formalization quality: if the LLM fails to generate a faithful executable representation, the solver cannot recover the correct reasoning process. Thus, much of the literature on solver-augmented logical reasoning has focused on using solvers as *final inference engines*, *answer verifiers*, or *self-refinement signals* over formalized problems. More recently, formal solvers and theorem provers have also been explored as sources of process supervision during training, providing feedback on the correctness of intermediate reasoning steps [Kim and Yun, 2026].

Distinction from Existing Solver-Augmented Methods. While formal verification can establish the logical correctness of intermediate reasoning steps, correctness alone does not ensure that a reasoning trajectory makes meaningful progress. A model may generate deductions that are logically valid but merely restate or reformulate information already established in earlier steps. **SPRING** addresses this limitation by explicitly incorporating *inferential novelty* into solver-guided process supervision rather than rewarding deductions solely for their logical validity. In this way, **SPRING** extends solver-based process supervision beyond correctness verification to explicitly reward the generation of new logical information.

3 Problem Formulation

Let x denote a logic problem instance consisting of (i) a set of natural-language premises and/or clues $\mathcal{C}$ and (ii) a set of domain constraints Γ_{dom} encoding the structural rules of the problem, such as exclusivity, uniqueness, admissible assignments, or variable ranges. We define the *problem theory* as $\Gamma_{\mathrm{base}} = \Gamma_{\mathrm{dom}} \cup \mathcal{C}$. Given an input problem x, our goal is to train a language model policy π_θ to generate a reasoning trajectory $\tau = (s_1, s_2, \ldots, s_T)$ followed by a final solution y, where each step s_t is *interleaved* and contains both a natural-language explanation and a formal logical statement executable in Z3. Concretely, each step is represented as $s_t = (u_t, z_t)$, where u_t is a free-form natural-language rationale and z_t is an optional formal statement in a constrained logical language. The full model output is therefore $o = (\tau, y) \sim \pi_\theta(\cdot \mid x)$. Our objective is not only to predict the

correct final solution, but to learn a policy that produces *reasoning trajectories* whose intermediate formal steps are both *valid* and *novel*. Intuitively, a valid step is one that is logically supported by the underlying problem theory, while a novel step contributes new information beyond previously generated non-contradictory steps and does not merely restate an existing premise or prior deduction. The definitions of these core concepts are presented in Section 3.1.

3.1 Core Definitions

We first formalize the reasoning-step properties used by our solver-guided reward design. Let Γ_{dom} denote the domain constraints of a logic problem, $\mathcal{C}$ the set of premises, and $\Gamma_{\mathrm{base}} = \Gamma_{\mathrm{dom}} \cup \mathcal{C}$ the resulting base theory. At reasoning step t, let $\mathcal{P}^{\mathrm{nc}}_{t-1}$ denote the set of previously accepted non-contradictory formal steps up to step $t-1$.

Definition 1 (Strong Implication)**.** *Let Φ be a background theory and let A and B be formulas. We say that A* strongly implies *B under Φ, denoted $A \Rightarrow_{\Phi} B$, if $\Phi \cup \{A, B\}$ is satisfiable and $\Phi \cup \{A, \neg B\}$ is unsatisfiable. Thus, B follows from A under Φ, while vacuous implication caused by inconsistency of $\Phi \cup \{A\}$ is excluded.*

Definition 2 (Valid Step)**.** *A generated formal step z_t is* valid *if it is logically implied by the base theory and remains consistent with it; that is, $\Gamma_{\mathrm{base}} \cup \{z_t\}$ is satisfiable and $\Gamma_{\mathrm{base}} \cup \{\neg z_t\}$ is unsatisfiable. Equivalently, z_t is a sound deduction from the premises and domain constraints.*

Definition 3 (Tautological Step)**.** *A generated formal step z_t is* tautological *with respect to the problem if it is implied by the domain constraints alone, i.e., if $\Gamma_{\mathrm{dom}} \cup \{\neg z_t\}$ is unsatisfiable. Such a step does not depend on the premises or any intermediate reasoning and therefore provides no problem-specific inferential progress.*

Definition 4 (Novel Step)**.** *A valid step z_t is* novel *if it is not tautological (Definition 3), not already implied by the previously accepted non-contradictory reasoning steps, and not equivalent to any original premise. Let*

$$P_{t-1} = \bigwedge_{z \in \mathcal{P}^{\mathrm{nc}}_{t-1}} z$$

denote the conjunction of all previously accepted non-contradictory formal reasoning steps, i.e., the accumulated reasoning state up to step $t-1$. Then z_t is novel if $P_{t-1} \not\Rightarrow_{\Gamma_{\mathrm{dom}}} z_t$ and $z_t \not\equiv C_i$ for all $C_i \in \mathcal{C}$, after tautological steps have been excluded according to Definition 3. Equivalently, novelty requires that both $\Gamma_{\mathrm{dom}} \cup \mathcal{P}^{\mathrm{nc}}_{t-1} \cup \{z_t\}$ and $\Gamma_{\mathrm{dom}} \cup \mathcal{P}^{\mathrm{nc}}_{t-1} \cup \{\neg z_t\}$ remain satisfiable.

Definition 5 (Contradictory Step)**.** *A generated formal step z_t is* contradictory *if adding it to the current reasoning state makes the theory unsatisfiable. Equivalently, if we define the current context as $\Gamma^{(t)}_{\mathrm{ctx}} = \Gamma_{\mathrm{base}} \cup \mathcal{P}^{\mathrm{nc}}_{t-1}$, then z_t is contradictory when*

$$\Gamma^{(t)}_{\mathrm{ctx}} \cup \{z_t\} \text{ is unsatisfiable.}$$

Thus, z_t conflicts with the premises, the domain constraints, or the previously accepted non-contradictory deductions.

Definition 6 (Consistency with Final Solution)**.** *Let F denote the final predicted solution and let $\mathcal{Z}$ denote the set of formal reasoning steps extracted from a trajectory. We say that $\mathcal{Z}$ is* consistent with *F if $\Gamma_{\mathrm{dom}} \cup \mathcal{Z} \cup \{F\}$ is satisfiable. Otherwise, the reasoning trajectory is inconsistent with the final solution.*

4 SPRING

Overview. The workflow of **SPRING** is illustrated in Figure 1. It is a reinforcement learning framework for improving logical reasoning in language models through symbolic verification. It targets structured logic problems, with ZebraPuzzles used as a representative case for visualization. The workflow proceeds as follows. Given a problem instance, the LLM generates: parsed syntactic premises, interleaved reasoning steps (natural-language and formal), and a final solution. The parsed syntactic premises are first used to construct the symbolic solver state, after which we check whether the resulting constraint system is satisfiable. If the solver state is SAT, it is then used to evaluate the generated formal reasoning steps and determine whether they are valid, novel, tautological, or contradictory. These solver-based judgments are converted into process rewards, encouraging reasoning trajectories that make genuine inferential progress while discouraging inconsistent, redundant, or otherwise uninformative steps.

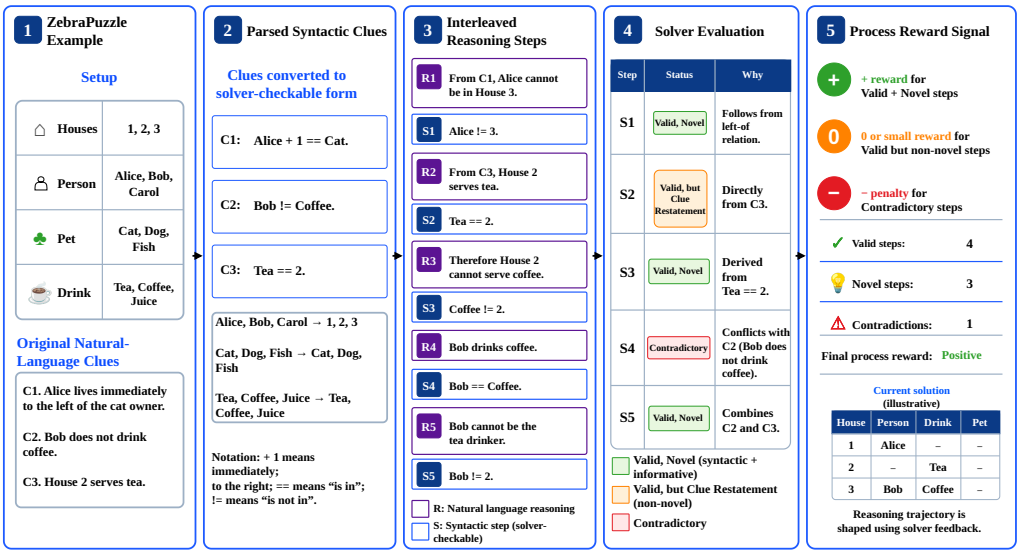

Figure 1: Illustration of **SPRING** using a ZebraPuzzle example. Natural-language clues are converted into solver-checkable syntactic clues, interleaved reasoning steps are classified by the solver, and the resulting signals are mapped to process rewards for training.

4.1 LLM Prompting and Output Structure

Given a logic problem instance x, we prompt the LLM to produce a structured output that serves as the interface between free-form language reasoning and symbolic verification. Specifically, the output contains three main components: (i) *Parsed Syntactic Premises*, which normalize the original natural-language premises into solver-checkable constraints, (ii) an *Interleaved Reasoning trajectory*, which alternates between natural-language explanations and formal deduction steps, and (iii) a *Final Solution*, which specifies the predicted assignment or answer. The prompts used in our experiments are given in Appendix E.

Formally, let π_θ denote the policy LLM parameterized by θ. Given an input problem instance x, the model generates a structured output:

$$o = \big(\hat{\mathcal{C}}, \tau, y\big) \sim \pi_\theta(\cdot \mid x),$$

where $\hat{\mathcal{C}} = (\hat{C}_1, \ldots, \hat{C}_m)$ denotes the *Parsed Syntactic Premises*, $\tau = (s_1, \ldots, s_T)$ is the reasoning trajectory with $s_t = (u_t, z_t)$, and y is the final predicted solution. We adopt an interleaved reasoning format because LLMs are better at natural-language deduction, while formal steps are better suited for precise symbolic verification. In this setting, u_t provides the explanation and z_t the solver-checkable deduction.

4.2 Solver Construction and Step Verification

Given the parsed syntactic premises $\hat{\mathcal{C}}$ generated by the policy LLM, we construct the symbolic solver by combining them with the domain constraints Γ_{dom}, yielding the base theory:

$$\Gamma_{\text{base}} = \Gamma_{\text{dom}} \cup \hat{\mathcal{C}}.$$

We first check whether Γ_{base} is satisfiable and whether it admits a unique solution. For ZebraLogic, we check that the constraints admit a unique satisfying assignment. For AR-LSAT, we instead check that exactly one answer option is correct, while allowing the underlying constraints to admit multiple satisfying assignments. These checks establish that the parsed constraints are solver-compatible and yield a unique answer under the generated formalization. If Γ_{base} is unsatisfiable or fails to yield a unique solution, the trajectory is treated as solver-incompatible and receives a reduced reward.

Symbolic Step Verification. Once the base solver state is established, we use it to verify each parsed formal reasoning step z_t. Let $\mathcal{P}^{\text{nc}}_{t-1}$ denote the set of previously accepted non-contradictory

steps, and define the current reasoning context as

$$\Gamma_{\mathrm{ctx}}^{(t)} = \Gamma_{\mathrm{base}} \cup \mathcal{P}_{t-1}^{\mathrm{nc}}.$$

For each generated step z_t, we extract the core solver signals that drive our reward design. First, we check whether z_t is *valid* according to Definition 2, i.e., whether it is entailed by Γ_{base} while remaining satisfiable with it. Second, we check whether z_t is *tautological* according to Definition 3, namely whether it follows from Γ_{dom} alone and therefore contributes no problem-specific information. Third, for a valid and non-tautological step, we test whether it is *novel* using Definition 4. Concretely, letting $P_{t-1} = \bigwedge_{z \in \mathcal{P}_{t-1}^{\mathrm{nc}}} z$, we check whether

$$P_{t-1} \not\models_{\Gamma_{\mathrm{dom}}} z_t,$$

and whether z_t is not equivalent to any original premise $C_i \in \mathcal{C}$. This step-level novelty signal is aggregated into the reward term r_{novel}, which measures the extent to which the trajectory introduces new inferential content beyond the previously accepted reasoning state. Fourth, we check whether z_t is *contradictory* according to Definition 5, i.e., whether

$$\Gamma_{\mathrm{ctx}}^{(t)} \cup \{z_t\}$$

is unsatisfiable. The corresponding contradiction signal is accumulated into r_{contra}, which penalizes trajectories that introduce logically inconsistent deductions. Finally, at the trajectory level, we check whether the generated reasoning remains *consistent with the final solution* using Definition 6, namely whether the formal reasoning steps and the final prediction can coexist in a satisfiable theory. This trajectory-level compatibility signal is mapped to $r_{\text{consistency}}$, which rewards reasoning traces whose intermediate deductions remain coherent with the final predicted solution.

These solver-derived signals, namely novelty, contradiction, and final-solution consistency, provide the fine-grained process supervision used by **SPRING**. Intuitively, they allow the model to distinguish between steps that make genuine inferential progress and those that are redundant, uninformative, or logically inconsistent.

4.3 Process-Level Reward

We define a structured reward that combines solver-derived process signals (Section 4.2) with task-level and formatting signals. Concretely, our reward uses the core symbolic signals of *novelty*, *contradiction*, and *final-solution consistency*, together with three auxiliary signals: (i) r_{format}, a binary reward that takes value 1 if the generated reasoning follows the required interleaved structure of natural-language and parsed formal steps, and 0 otherwise; (ii) r_{parsed}, a binary reward that takes value 1 if the LLM-generated output can be successfully parsed into the target symbolic representation, and 0 otherwise; and (iii) r_{accuracy}, which measures final accuracy. **Note**, for ZebraPuzzles, we instantiate r_{accuracy} as puzzle accuracy. We distinguish two cases using the satisfiability indicator $s_{\text{sat}} \in \{0, 1\}$ of the parsed premises system. If $s_{\text{sat}} = 0$, the reward is restricted to auxiliary quality signals and final-answer quality; if $s_{\text{sat}} = 1$, we additionally apply process-level rewards from symbolic step verification:

$$R = \begin{cases} r_{\mathrm{base}}, & s_{\mathrm{sat}} = 0, \\ r_{\mathrm{base}} + (0.5 + 0.5 r_{\mathrm{accuracy}}) r_{\mathrm{proc}}, & s_{\mathrm{sat}} = 1, \end{cases}$$

where

$$r_{\mathrm{base}} = 0.15 r_{\mathrm{parsed}} + 0.10 r_{\mathrm{format}} + 0.60 r_{\mathrm{accuracy}},$$
$$r_{\mathrm{proc}} = 0.40 r_{\mathrm{novel}} + 0.30 r_{\mathrm{consistency}} - 0.15 r_{\mathrm{contra}}.$$

This reward design preserves the same base accuracy reward regardless of s_{sat}, while adding process supervision based on solver-verified reasoning quality when available. As a result, trajectories with more novel and solution-consistent deductions, and fewer contradictions, receive higher reward, aligning reinforcement learning with both *outcome quality* and *reasoning quality*.

4.4 RL Training

We train **SPRING** with a group-based reinforcement learning objective, using GRPO for policy optimization and a DAPO-style distributed runtime for scalable sampling and training [Shao et al., 2024, Yu et al., 2025]. Given a logic problem x, the policy π_θ samples a group of candidate outputs

$\{o_i\}_{i=1}^G$, where each $o_i = (\tau_i, y_i)$ contains an interleaved reasoning trajectory τ_i and a final solution y_i. Each candidate receives a scalar reward $r_i = R(x, o_i)$ from our solver-guided verifier. The training objective is to maximize the expected reward:

$$\max_{\theta} \ \mathbb{E}_{x\sim\mathcal{D},\, o\sim\pi_\theta(\cdot|x)}\big[R(x,o)\big]. \tag{1}$$

Following GRPO, rewards are normalized within each sampled group. Let $\bar{r} = \frac{1}{G}\sum_{i=1}^G r_i$ and $\sigma_r = \sqrt{\frac{1}{G}\sum_{i=1}^G (r_i - \bar{r})^2}$. The group-relative advantage is then $A_i = \frac{r_i - \bar{r}}{\sigma_r + \epsilon}$, where $\epsilon > 0$ ensures numerical stability. Using the policy ratio $\rho_i(\theta) = \frac{\pi_\theta(o_i|x)}{\pi_{\theta_{\text{old}}}(o_i|x)}$, the clipped GRPO loss is:

$$\mathcal{L}_{\text{GRPO}}(\theta) = \mathbb{E}_x\left[\frac{1}{G}\sum_{i=1}^{G} \min\big(\rho_i(\theta)A_i,\ \text{clip}(\rho_i(\theta), 1-\delta, 1+\delta)A_i\big)\right] \tag{2}$$

This encourages the policy to prefer candidates with better solver-guided reasoning quality relative to others in the same group. To stabilize training, we regularize the policy toward a frozen reference policy π_{ref}:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{GRPO}}(\theta) - \beta\,\text{KL}\big(\pi_\theta(\cdot \mid x)\,\|\,\pi_{\text{ref}}(\cdot \mid x)\big), \tag{3}$$

where β controls the regularization strength. In this framework, DAPO provides the distributed runtime for large-scale sampling, reward computation, and optimization, while Z3 solver [De Moura and Bjørner, 2008] supplies fine-grained process feedback over intermediate deductions.

5 Experimentation

5.1 Experimental Settings

(a) Datasets.

For evaluation, we use three logical reasoning benchmarks. (i) **ZebraLogic** [Lin et al., 2025] contains 1,000 Zebra puzzles across four difficulty levels: Small, Medium, Large, and Extra-Large. We use a 25%/5%/70% train/validation/test split, corresponding to 250/50/700 instances. (ii) **AR-LSAT** [Zhong et al., 2022] consists of analytical reasoning problems across three categories: Ordering, Grouping, and Assignment. We use 300 training and 50 validation instances per category. For testing, we use the official 230-instance test split, comprising 112 Ordering, 49 Grouping, and 69 Assignment problems. (iii) **Knights and Knaves (KnK)** [Xie et al., 2025] contains 6,200 training and 700 test puzzles spanning 2–8 characters. We sample 300 puzzles from the training set for RL model training and hold out an additional 50 for validation. We evaluate on the complete 700-puzzle test set. Dataset statistics are summarized in Table 3.

(b) Large Models. We evaluate **SPRING** on Qwen3-1.7B (non-thinking), Qwen3-4B-Thinking, and Qwen3-8B [Yang et al., 2025], as well as Phi-4-Reasoning [Abdin et al., 2025]. The first two cover contrasting small-model configurations (i.e., thinking and non-thinking), while the latter two extend the evaluation to a larger model and a different model family.

(c) Baselines. We consider the following baselines. (i) **Base-NL** and **Base-INT** denote the pre-trained LLMs without task-specific reinforcement learning, using a natural-language prompt and an interleaved prompt, respectively. To isolate the contribution of solver-guided process supervision, we additionally compare against two outcome-only reward baselines. (ii) **OR-NL** uses the same natural-language prompt as Base-NL, but optimizes only for final-answer quality with reward $0.15 \cdot r_{\text{parsed}} + 0.6 \cdot r_{\text{accuracy}}$. (iii) **OR-INT** uses the same interleaved prompt as Base-INT, but likewise optimizes only for final-answer quality with reward $0.15 \cdot r_{\text{parsed}} + 0.6 \cdot r_{\text{accuracy}}$. Thus, OR-NL and OR-INT remove the process-level reward terms of **SPRING** and differ only in the prompting format. From existing research on logical reasoning with LLMs, we use Logic-LM [Pan et al., 2023] as a representative baseline for comparison.

(d) Experimental Settings. For RL training, we use the following settings. The GRPO group size n is set to 8, the sampling temperature during training is 0.8, and the inference temperature is fixed at 0.0. We set $\beta = 0.01$, and the maximum input and output lengths to 8×1024 tokens. All experiments are conducted using the VERL framework[1] on a cluster with $4\times$ NVIDIA H200 GPUs. We use the

[1] https://verl.readthedocs.io/

Table 1: **SPRING** performance across different LLMs on ZebraLogic, AR-LSAT, and Knights and Knaves. We report standard deviation for **SPRING** and its ablation. AR-LSAT Overall accuracy is the weighted mean of Ordering, Grouping, and Assignment accuracies.

LLM	System	ZebraLogic		AR-LSAT (Acc.)				Knights and Knaves	
		Puzzle Acc. ↑	Cell Acc. ↑	Order. ↑	Group. ↑	Assign. ↑	Overall ↑	Puzzle Acc. ↑	Person Acc. ↑
Qwen3-1.7B	Base-NL	11.14	35.04	16.07	24.48	0.00	13.04	53.28	64.43
	OR-NL	38.57	49.87	19.64	40.81	1.44	18.69	68.57	79.64
	Base-INT	31.85	41.45	23.21	24.48	14.49	20.86	18.71	55.61
	OR-INT	36.00	46.09	35.71	51.02	24.63	35.65	73.57	80.54
	Logic-LM	12.85	33.16	-	-	-	15.58	3.57	22.51
	SPRING (–N)	19.28 (±0.92)	59.62 (±0.81)	34.82	55.10	46.37	42.61 (±0.95)	78.57 (±0.97)	82.14 (±0.95)
	SPRING	**54.00 (±0.62)**	**59.87 (±0.65)**	**41.07**	**57.14**	**49.56**	**47.04 (±0.66)**	**85.28 (±0.90)**	**86.94 (±0.87)**
Qwen3-4B-Thinking	Base-NL	24.00	24.00	6.25	18.36	0.00	6.95	55.28	65.41
	OR-NL	67.28	68.30	25.00	65.30	10.14	29.13	74.28	84.66
	Base-INT	17.00	23.32	6.26	14.28	5.50	7.74	33.14	63.47
	OR-INT	66.14	72.72	61.60	79.59	65.21	66.52	76.57	87.58
	Logic-LM	21.42	38.12	-	-	-	20.78	7.14	35.48
	SPRING (–N)	61.42 (±0.84)	68.27 (±0.73)	63.39	77.55	66.66	67.39 (±0.87)	82.85 (±0.88)	90.13 (±0.85)
	SPRING	**73.71 (±0.51)**	**81.52 (±0.53)**	**69.64**	**81.63**	**69.56**	**72.17 (±0.76)**	**88.71 (±0.85)**	**94.21 (±0.79)**
Phi4-Reasoning	Base-NL	7.14	22.42	12.50	24.48	17.39	16.52	57.85	68.13
	OR-NL	47.14	51.27	21.42	40.81	30.43	28.25	72.85	85.57
	Base-INT	18.71	20.27	0.00	18.36	2.89	4.78	36.42	68.84
	OR-INT	49.00	53.54	23.21	46.93	26.08	29.12	81.42	88.13
	Logic-LM	20.0	36.75	-	-	-	18.61	9.28	40.12
	SPRING (–N)	53.71 (±1.15)	56.34 (±1.17)	22.32	42.85	24.63	27.39 (±1.04)	83.57 (±1.10)	88.51 (±0.95)
	SPRING	**55.71 (±1.10)**	**59.52 (±1.07)**	**26.78**	**51.02**	**34.78**	**34.34 (±1.03)**	**89.28 (±0.99)**	**94.57 (±0.92)**
Qwen3-8B	Base-NL	47.42	50.13	29.46	40.81	30.43	32.17	56.0	68.49
	OR-NL	65.67	64.85	51.78	61.22	56.52	55.21	74.28	80.76
	Base-INT	50.42	58.61	33.92	30.61	34.78	33.47	49.57	72.14
	OR-INT	70.85	75.13	65.17	77.08	63.76	67.28	77.14	82.85
	Logic-LM	22.85	40.56	-	-	-	25.97	8.57	55.20
	SPRING (–N)	69.42 (±0.80)	76.57 (±0.79)	51.78	75.51	50.72	56.52 (±0.72)	87.14 (±0.84)	91.24 (±0.81)
	SPRING	**77.85 (±0.78)**	**79.41 (±0.74)**	**71.42**	**85.71**	**69.56**	**73.91 (±0.69)**	**93.14 (±0.95)**	**96.05 (±0.90)**

Z3 solver for symbolic-step verification [De Moura and Bjørner, 2008]. Under this configuration, training Qwen3-4B-Thinking requires approximately 45 minutes per epoch. Unless stated otherwise, training is run for up to 25 epochs with early stopping using a patience of 3 epochs, and we report the checkpoint with the best validation performance. All experiments are repeated over 3 runs, and we report average scores. The prompt templates used in our experiments are provided in Appendix E.

(e) Evaluation Metrics. We evaluate model performance using task-specific metrics. For ZebraLogic, we report **Puzzle Acc.** and **Cell Acc.**; for AR-LSAT, we report **Acc.**; and for Knights and Knaves, we report **Puzzle Acc.** and **Person Acc.**. Further details and the mathematical definitions of these metrics are provided in Appendix B.2.

5.2 Main Results

The results in Table 1 show that **SPRING** consistently improves logical reasoning across four LLMs and three benchmarks. The gains hold across models of different sizes and reasoning capabilities, including Qwen3-1.7B, Qwen3-4B-Thinking, Phi4-Reasoning, and Qwen3-8B. In contrast, the base models exhibit substantial sensitivity to prompting style, while outcome-only reward learning improves performance but generally remains below **SPRING**. Logic-LM also performs substantially below **SPRING** across all evaluated model–benchmark combinations.

(a) ZebraLogic. **SPRING** achieves strong gains across all four LLMs, reaching Puzzle Acc scores of 54.00, 73.71, 55.71, and 77.85 for Qwen3-1.7B, Qwen3-4B-Thinking, Phi4-Reasoning, and Qwen3-8B, respectively. For Qwen3-1.7B, **SPRING** improves Puzzle Acc from 38.57 for the strongest outcome-only baseline (OR-NL) to 54.00, a gain of 15.43 points. For Qwen3-4B-Thinking, it reaches 73.71 Puzzle Acc and 81.52 Cell Acc, compared with 67.28 and 72.72 for the strongest outcome-only results on the respective metrics. The improvements remain evident for other LLMs with **SPRING** reaching 55.71/59.52 Puzzle/Cell Acc with Phi4-Reasoning and 77.85/79.41 with Qwen3-8B. Logic-LM performs considerably worse, with Puzzle Acc ranging from 12.85 to 22.85 across the four LLMs. These results indicate that solver-guided process optimization provides benefits beyond prompting, outcome-only optimization, and existing solver-augmented logical reasoning.

Table 2: Reasoning-trace quality across three reward settings for Qwen3-4B-Thinking. **Solved traces** are averaged over correctly solved cases, while **Failed traces** are averaged over incorrect cases.

System	Puzzle Acc. ↑	Solved traces				Failed traces			
		SAT-rate ↑	Steps ↓	Parsability ↑	Validity rate ↑	Steps ↓	Parsability ↑	Validity rate ↑	Contradictions / 100 Steps ↓
−N	61.42	0.94	35.20	0.92	0.98	49.10	0.76	0.82	6.77
−NC	66.14	0.81	34.82	0.69	0.95	11.23	0.61	0.91	2.52
SPRING	**73.71**	**0.96**	**22.37**	**0.98**	**0.99**	**9.39**	**0.99**	**0.99**	**0.46**

The novelty reward is particularly important on ZebraLogic. Removing it (**SPRING** (–N)) reduces Puzzle Acc from 54.00 to 19.28 for Qwen3-1.7B, from 73.71 to 61.42 for Qwen3-4B-Thinking, from 55.71 to 53.71 for Phi4-Reasoning, and from 77.85 to 69.42 for Qwen3-8B. The effect varies across models, but the consistent reduction in puzzle-level accuracy shows that rewarding novel deductions complements solver-validity and helps translate locally valid reasoning into complete puzzle solutions.

(b) AR-LSAT. SPRING also consistently improves performance on AR-LSAT, achieving overall accuracies of 47.04, 72.17, 34.34, and 73.91 for Qwen3-1.7B, Qwen3-4B-Thinking, Phi4-Reasoning, and Qwen3-8B, respectively. For Qwen3-1.7B, the strongest outcome-only baseline, OR-INT, achieves 35.65 overall accuracy, whereas **SPRING** reaches 47.04, with the largest improvement on Assignment (24.63 to 49.56). For Qwen3-4B-Thinking, **SPRING** improves the overall score from 66.52 for OR-INT to 72.17, while for Qwen3-8B it improves from 67.28 to 73.91. Phi4-Reasoning exhibits lower absolute AR-LSAT performance, but **SPRING** still improves over its strongest outcome-only result from 29.12 to 34.34. Removing the novelty reward reduces overall accuracy for all four LLMs, further supporting the contribution of novel-step supervision beyond solver-validity alone.

(c) Knights and Knaves. The same trend extends to Knights and Knaves, where **SPRING** achieves the strongest Puzzle Acc and Person Acc for all four LLMs. Puzzle Acc reaches 85.28, 88.71, 89.28, and 93.14 for Qwen3-1.7B, Qwen3-4B-Thinking, Phi4-Reasoning, and Qwen3-8B, respectively, with corresponding Person Acc scores of 86.94, 94.21, 94.57, and 96.05. Compared with the strongest outcome-only baseline for each model, this corresponds to Puzzle Acc gains of 11.71, 12.14, 7.86, and 16.00 points, respectively. Logic-LM performs particularly poorly at puzzle-level exact solving, achieving only 3.57–9.28 Puzzle Acc. Moreover, removing the novelty reward consistently reduces Puzzle Acc, from 85.28 to 78.57, 88.71 to 82.85, 89.28 to 83.57, and 93.14 to 87.14 across the four models. Together, these results demonstrate that the benefits of solver-guided process rewards and novelty supervision generalize across model families, model scales, and structurally different logical reasoning tasks.

Across three runs, **SPRING** exhibits low run-to-run variability, with standard deviations ranging from 0.51 to 1.10 percentage points. Even the largest reported standard deviation is small relative to the gains over the strongest outcome-only baselines, supporting the consistency of the observed improvements.

5.3 Ablation Analyses

To assess the contribution of the key reward components in **SPRING**, we conduct an ablation study for different model components on ZebraLogic benchmark. Specifically, (i) −N denotes the variant without the novel-step reward; (ii) −NC removes both the novel-step reward and the contradiction penalty; (iii) **SPRING**, i.e., the full model. We compare these variants to quantify how each component contributes to improved reasoning trajectories and final performance.

Table 2 compares the three reward settings from both an outcome-level and a process-level perspective. **Puzzle Acc.** measures end-task success. The columns under **Solved traces** summarize the quality of reasoning trajectories on correctly solved cases only, including **SAT-rate**, which measures solver consistency of successful traces, **Steps**, which measures reasoning efficiency, and **Parsability** (Parsed symbolic steps / Total symbolic steps) and **Validity rate** (Valid symbolic steps / Parsed symbolic steps), which quantify structural well-formedness and logical correctness of the generated steps. The columns under **Failed traces** characterize the behavior of unsuccessful trajectories, including how long the model continues reasoning after failure, how structurally usable those traces remain, and how often they introduce contradictions. Taken together, these fields allow us to evaluate not only whether a system solves the puzzle, but also how efficient, reliable, and solver-faithful its reasoning process is.

(a) Removing novel-step rewards (−N). Compared with **SPRING**, the (−N) setting achieves substantially lower Puzzle Acc. (61.42 vs. 73.71) and requires much longer solved traces (35.20 vs. 22.37 steps). Its failed traces are especially poor: when the model is incorrect, it continues generating for an average of 49.10 steps and accumulates 6.77 contradictions per 100 steps. In addition, both its Parsability and Validity Rate are consistently worse than those of **SPRING**, for solved as well as failed cases. This indicates that without the novel-step reward, reasoning becomes less efficient, less stable, and far more contradiction-prone.

(b) Removing both novelty and contradiction (−NC). The (−NC) setting yields lower performance compared to **SPRING**. Its solved traces have the lowest SAT-rate (0.81), the worst Parsability (0.69), and a noticeably lower Validity Rate (0.95) than both alternatives. Its failed traces are also substantially noisier than those of **SPRING**, with a contradiction rate of 2.52 per 100 steps. Thus, while (−NC) improves over the novelty-free baseline in terms of trace length and final accuracy, it still fails to maintain solver-faithful and structurally reliable reasoning trajectories.

A possible explanation for the observed ordering between **SPRING** (−NC) and **SPRING** (−N) is the asymmetric role of the reward components: the contradiction term penalizes undesirable steps, whereas the novelty term provides positive guidance toward useful inferential progress. Without the novelty reward, retaining the contradiction penalty may provide insufficient constructive guidance during optimization, potentially explaining why **SPRING** (−N) underperforms **SPRING** (−NC) in some settings. In the full model, novelty rewards may provide constructive guidance, with the contradiction penalty acting as a complementary regularizer.

The two ablations also exhibit different failure behavior. On ZebraLogic, **SPRING** (−N) produces longer failed traces than **SPRING** (−NC), averaging 49.10 versus 11.23 steps, and has a higher contradiction rate of 6.77 versus 2.52 per 100 steps. These observations suggest different failure patterns; the lower contradiction rate of **SPRING** (−NC) does not by itself establish better reasoning. Since these rates are normalized by trace length, longer traces alone cannot explain the difference.

(c) Impact of novel steps (SPRING). **SPRING** is the strongest system across both final performance and trace quality. It attains the highest Puzzle Acc. (73.71) and the highest SAT-rate on solved cases (0.96), indicating that its successful trajectories are most compatible with solver-based verification. It also reaches correct solutions with the shortest solved traces and the best Parsability and Validity Rate, showing that its reasoning is both more concise and more structurally reliable. The advantage is equally clear on failed cases: **SPRING** has the shortest failed traces (9.39 steps), the highest Parsability and Validity Rate, and by far the lowest contradiction rate (0.46 per 100 steps). Relative to both (−N) and (−NC), these results show that solver-guided process supervision does not merely improve whether the model gets the answer right; it also regularizes the entire reasoning trajectory toward shorter, cleaner, and more solver-faithful traces.

5.4 Further Analyses

To better understand the behavior of **SPRING**, we perform a set of additional analyses on the ZebraLogic benchmark, including: **(a)** Scaling Behavior of **SPRING** across different puzzle sizes; **(b)** Solver SAT rate across Epochs; **(c)** Error Analyses; and **(d)** Qualitative case study. Corresponding results, together with the core findings, are reported in Appendix C. These analyses complement the main results by revealing how **SPRING** behaves under increasing puzzle difficulty, how its solver consistency develops during training, and how its reasoning traces remain qualitatively cleaner and more stable than those of the baselines.

6 Conclusion and Future Work

We introduced **SPRING**, a solver-guided reinforcement learning framework that provides process-level supervision by rewarding novel, logically consistent deductions and penalizing contradictory and uninformative steps. Across three benchmarks—ZebraLogic, AR-LSAT, and Knights and Knaves—and four LLMs, **SPRING** consistently outperforms base LLMs, outcome-only reward baselines, and Logic-LM. Our analyses show shorter successful reasoning traces, fewer contradictions, and greater consistency with logical constraints. Future work will extend **SPRING** to broader reasoning domains, including code generation, using suitable formalizations and verification procedures for intermediate steps.

Acknowledgments

The authors gratefully acknowledge the computing support and resources provided by the Panther high-performance computing cluster at the Qatar Computing Research Institute (QCRI).

References

Marah Abdin et al. Phi-4-reasoning technical report. *arXiv preprint arXiv:2504.21318*, 2025.

Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of model checking*, pages 305–343. Springer, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

Ruikang Hu, Shaoyu Lin, Yeliang Xiu, and Yongmei Liu. Ltrag: Enhancing autoformalization and self-refinement for logical reasoning with thought-guided rag. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 2483–2493, 2025.

Minsu Kim and Se-Young Yun. Process-verified reinforcement learning for theorem proving via lean. In Carl Vondrick, Bharath Hariharan, Colin Raffel, Lerrel Pinto, Diyi Yang, and Aleksandra Faust, editors, *ICLR*. proceedings.iclr.cc, 2026. ISBN 9798331339678. URL `http://dblp.uni-trier.de/db/conf/iclr/iclr2026.html#KimY26`.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213, 2022.

Long Hei Matthew Lam, Ramya Keerthy Thatikonda, and Ehsan Shareghi. A closer look at tool-based logical reasoning with llms: The choice of tool matters. In *Proceedings of the 22nd Annual Workshop of the Australasian Language Technology Association*, pages 41–63, 2024.

Yuetai Li, Xiang Yue, Zhangchen Xu, Fengqing Jiang, Luyao Niu, Bill Yuchen Lin, Bhaskar Ramasubramanian, and Radha Poovendran. Small models struggle to learn from strong reasoners. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 25366–25394, 2025.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The twelfth international conference on learning representations*, 2024.

Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. Zebralogic: On the scaling limits of llms for logical reasoning. *arXiv preprint arXiv:2502.01100*, 2025.

Seyed Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Theo Olausson, Alex Gu, Ben Lipkin, Cedegao Zhang, Armando Solar-Lezama, Joshua Tenenbaum, and Roger Levy. Linc: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5153–5176, 2023.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3806–3824, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.248.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Chulin Xie, Yangsibo Huang, Chiyuan Zhang, Da Yu, Xinyun Chen, Bill Yuchen Lin, Bo Li, Badih Ghazi, and Ravi Kumar. On memorization of large language models in logical reasoning. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pages 2742–2785. The Asian Federation of Natural Language Processing and The Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.ijcnlp-long.148.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. Satlm: Satisfiability-aided language models using declarative prompting. *Advances in Neural Information Processing Systems*, 36:45548–45580, 2023.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Xuechen Zhang, Zijian Huang, Chenshun Ni, Ziyang Xiong, Jiasi Chen, and Samet Oymak. Making small language models efficient reasoners: Intervention, supervision, reinforcement. *arXiv preprint arXiv:2505.07961*, 2025.

Wanjun Zhong, Siyuan Wang, Duyu Tang, Zenan Xu, Daya Guo, Yining Chen, Jiahai Wang, Jian Yin, Ming Zhou, and Nan Duan. Analytical reasoning of text. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 2306–2319, 2022.

A Background

We consider three representative logical reasoning benchmarks in this work: ZebraLogic, AR-LSAT, and Knights and Knaves. These benchmarks require multi-step deduction under explicit logical constraints but differ substantially in structure and output format. ZebraLogic emphasizes exact assignment under global consistency constraints, AR-LSAT evaluates textual reasoning over ordering, grouping, and assignment scenarios, and Knights and Knaves requires truth-conditional reasoning over multiple characters. Together, these benchmarks provide complementary settings for evaluating logically valid and informative intermediate reasoning [Lin et al., 2025, Zhong et al., 2022].

A.1 ZebraPuzzles

ZebraPuzzles, also known as logic grid puzzles, are structured reasoning problems in which the goal is to determine a unique assignment of entities to attributes using a set of natural-language clues together with domain constraints. A typical puzzle specifies a fixed number of positions or houses and several attribute categories, such as person, pet, drink, or color. Each value in an attribute category must be assigned exactly once, and each position must receive exactly one value from each category. Solving the puzzle therefore requires satisfying both clue-derived constraints and global one-to-one assignment constraints. ZebraLogic formalizes this setting as a benchmark for studying logical reasoning under controllable complexity, with puzzles generated from underlying constraint satisfaction problems [Lin et al., 2025].

From the perspective of formal reasoning, ZebraPuzzles are particularly attractive because they admit a clean symbolic representation. Clues such as equality, inequality, adjacency, left–right relations, and exclusivity can be encoded as logical constraints, while the final solution corresponds to a complete satisfying assignment. This makes the benchmark especially suitable for our setting, where a symbolic solver can verify whether an intermediate deduction is valid, contradictory, or genuinely contributes new information to the evolving reasoning state. ZebraLogic further shows that performance drops sharply as puzzle complexity increases, indicating that these puzzles remain challenging even for strong modern LLMs [Lin et al., 2025].

Example. Consider a simple Zebra-style puzzle with three houses and three attribute categories:

$$\begin{aligned} \textit{Person} &= \{\text{Alice}, \text{Bob}, \text{Carol}\}, \\ \textit{Pet} &= \{\text{Cat}, \text{Dog}, \text{Fish}\}, \\ \textit{Drink} &= \{\text{Tea}, \text{Coffee}, \text{Juice}\}. \end{aligned}$$

Each house contains exactly one person, one pet, and one drink, and each attribute value is used exactly once.

Clues.

1. Alice lives immediately to the left of the cat owner.
2. Bob lives in the house where tea is served.
3. The person in House 2 drinks tea.
4. Carol owns the fish.
5. The dog owner drinks coffee.
6. Bob does not drink coffee.

To solve this puzzle, the model may proceed through the following intermediate reasoning steps.

Reasoning Steps.

1. From Clues 2 and 3, Bob must live in House 2.
2. From Clue 1, Alice cannot live in House 3, since there is no house to its right.
3. Since Bob already occupies House 2, Alice cannot live in House 2. Therefore, Alice must live in House 1, and Carol must live in House 3.
4. By Clue 1, the cat owner must live immediately to the right of Alice. Since Alice is in House 1, the cat owner must be in House 2. Hence Bob owns the cat.

5. From Clue 4, Carol owns the fish, so the remaining pet, the dog, must belong to Alice in House 1.
6. From Clue 5, the dog owner drinks coffee. Therefore, Alice drinks coffee.
7. House 2 already serves tea by Clue 3, so the remaining drink, juice, must belong to Carol in House 3.

Solution. This example also illustrates different types of reasoning steps. The deduction "Alice cannot live in House 3" is a *valid* intermediate step. The statement "Bob drinks coffee" is *contradictory*, since it violates Clues 2, 3, and 6. A *novel* reasoning step is a valid deduction that is not already implied by previously accepted steps, such as the deduction that Bob must own the cat after combining Clue 1 with the placement of Alice in House 1.

Complete Solution.

House	Person	Pet	Drink
1	Alice	Dog	Coffee
2	Bob	Cat	Tea
3	Carol	Fish	Juice

A.2 AR-LSAT

AR-LSAT is a benchmark derived from the analytical reasoning section of the Law School Admission Test. Each instance consists of a passage describing a structured scenario, a question, and multiple answer options. Solving the task requires identifying the entities involved, interpreting the governing rules, and reasoning over their consequences to determine the correct answer. The benchmark was introduced to study analytical reasoning over text using questions collected from LSAT exams from 1991 to 2016, and prior work shows that these problems are challenging for neural models because they require explicit reasoning over constraints rather than shallow lexical matching [Zhong et al., 2022].

A key feature of AR-LSAT is that its instances cover multiple reasoning types. In this work, we focus on the three major categories commonly used in prior studies: *ordering*, *grouping*, and *assignment*. Ordering problems require arranging entities subject to precedence or positional rules. Grouping problems require partitioning entities into valid subsets under compatibility constraints. Assignment problems require mapping entities to roles, slots, or resources while satisfying exclusivity and dependency constraints. This diversity makes AR-LSAT a useful complement to ZebraPuzzles: unlike ZebraPuzzles, which emphasize fully specified grid-style assignments, AR-LSAT is more text-centric and question-oriented, requiring the model to reason over partial consequences of a rule system [Zhong et al., 2022].

AR-LSAT is also well matched to solver-guided reasoning. Although the input is fully textual, the scenario can often be converted into a structured representation involving participants, positions, groups, or assignments together with logical constraints. Prior neurosymbolic work has shown that combining LLMs with symbolic solvers can substantially improve performance on AR-LSAT, highlighting the usefulness of explicit formalization for solving such problems [Pan et al., 2023].

Example. Consider a simple AR-LSAT-style ordering problem involving three presentations:

$$\{A, B, C\}.$$

Each presentation must be scheduled in exactly one of the three positions: first, second, and third.

Rules.

1. A must occur before B.
2. C cannot be first.
3. B cannot be third.

Question. Which of the following must be true?

1. A is first.
2. B is third.
3. C is second.
4. A is third.

Reasoning Steps. To solve this problem, the following intermediate reasoning steps can be derived.

1. Since C cannot be first, either A or B must be first.
2. Because A must occur before B, B cannot be first.
3. Therefore, A must be first.
4. Since B cannot be third, B must be second.
5. Consequently, C must be third.

Final Answer. The statements that must be true are:

$$A \text{ is first}, \quad B \text{ is second}, \quad C \text{ is third}.$$

Therefore, among the answer choices above, the correct option is:

$$\boxed{A \text{ is first}}$$

This example illustrates step-level reasoning in AR-LSAT. The deduction "A must be first" is a *valid* intermediate step. The statement "B is first" is *contradictory*, since it violates Rule 1. A *novel* reasoning step is a valid deduction that is not already implied by previously accepted steps, such as the inference that "B must be second" after combining Rules 1 and 3.

A.3 Knights and Knaves

Knights and Knaves (KnK) consists of logical reasoning puzzles in which each character is either a knight, who always tells the truth, or a knave, who always lies Xie et al. [2025]. Each character makes a statement about one or more characters, and the task is to determine the identity of every character while ensuring that all statements are logically consistent. The benchmark contains puzzles with 2–8 characters, requiring increasingly complex reasoning as the number of characters and dependencies among their statements grows.

Knights and Knaves is well suited to solver-guided reasoning because each character's identity can be represented as a Boolean variable and each statement can be expressed as a logical constraint. A knight's statement must evaluate to true, whereas a knave's statement must evaluate to false. The resulting constraints can therefore be checked directly by a symbolic solver.

Example. Consider a simple Knights and Knaves puzzle involving three characters: $\{A, B, C\}$.

Statements.

1. A says: "B is a knave."
2. B says: "A and C are of the same type."
3. C says: "A is a knight."

Reasoning Steps. To solve this puzzle, the following intermediate reasoning steps can be derived.

1. Suppose A is a knight. Then A's statement is true, so B is a knave.
2. Since B is a knave, B's statement is false, so A and C are of different types.
3. Since A is a knight, C must therefore be a knave.
4. However, C's statement that A is a knight is true, contradicting the fact that C is a knave.
5. Therefore, A must be a knave. A's statement is false, so B must be a knight.
6. Since B is a knight, A and C are of the same type. Hence C is also a knave.

Table 3: Data statistics for ZebraLogic, AR-LSAT, and Knights and Knaves.

Dataset	Category / Split	Train	Validation	Test	Notes
ZebraLogic	Small	80	12	224	1000 total puzzles
	Medium	67	12	196	
	Large	52	13	138	
	Extra-Large	51	13	142	
	Total	250	50	700	
AR-LSAT	Ordering	300	50	112	Official test split
	Grouping	300	50	49	
	Assignment	300	50	69	
	Total	900	150	230	
Knights & Knaves	Total	300	50	700	Official test split

Final Answer. A is a knave, B is a knight, and C is a knave.

This example illustrates step-level reasoning in Knights and Knaves. For example, under the assumption that A is a knight, the deduction that B is a knave is a valid intermediate step. However, this assumption eventually leads to a contradiction because C, inferred to be a knave, makes a true statement. The solver can detect such inconsistencies and verify the deductions leading to the final assignment.

B Additional Experimental Settings

B.1 Evaluation Benchmarks

Table 3 summarizes the data splits used for ZebraLogic, AR-LSAT, and Knights and Knaves. ZebraLogic contains 1000 puzzles in total, divided by difficulty into Small, Medium, Large, and Extra-Large categories, with 250/50/700 instances for training, validation, and testing, respectively. For AR-LSAT, we construct balanced training and validation sets with 300 and 50 instances, respectively, for each reasoning type: Ordering, Grouping, and Assignment. The test set follows the official split, containing 112 Ordering, 49 Grouping, and 69 Assignment questions, for a total of 230 evaluation instances. Knights and Knaves contains 6200 training and 700 test puzzles spanning 2–8 characters. We sample 300 instances from the training set for RL model training, 50 for validation and evaluate on the complete 700-instance test set. Overall, these benchmarks cover complementary logical reasoning settings, including difficulty-controlled constraint puzzles, distinct reasoning types, and truth-conditional reasoning over multiple characters.

B.2 Evaluation Metrics

(a) Puzzle Acc. (ZebraLogic; Knights and Knaves). Puzzle accuracy measures the fraction of puzzles whose complete predicted assignments match the ground truth. Let $\hat{Y}_i$ and Y_i denote the predicted and ground-truth solutions for puzzle i, respectively. For N test puzzles,

$$\text{Puzzle Acc} = \frac{1}{N}\sum_{i=1}^{N} 1\left[\hat{Y}_i = Y_i\right].$$

For ZebraLogic, an exact match requires all cell assignments to be correct; for Knights and Knaves, it requires all character identities to be correct. Neither benchmark awards partial credit under this metric.

(b) Cell Acc. (ZebraLogic) and Person Acc. (Knights & Knaves). These metrics measure the fraction of individual assignments predicted correctly: cell values in ZebraLogic and character identities in Knights and Knaves. Let M_i denote the number of assignments in puzzle i, and let Y_{ij} and $\hat{Y}_{ij}$ denote the ground-truth and predicted values of assignment j, respectively. Both metrics are

Table 4: **SPRING** performance aggregated by puzzle category using the predefined size groups. These results are reported using Qwen3-4B-Thinking.

Category	#Samples	Cell-Acc.	SAT-rate	#Novel-steps	Puzzle Acc.
Small	224	0.941	0.924	2.81	0.915
Medium	196	0.922	0.908	4.66	0.888
Large	138	0.882	0.884	4.92	0.783
Extra-Large	142	0.405	0.641	2.08	0.204

computed as

$$\text{Cell/Person Acc} = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M_i} 1\left[\hat{Y}_{ij} = Y_{ij}\right]}{\sum_{i=1}^{N} M_i}.$$

We report this metric as Cell Acc. for ZebraLogic and Person Acc. for Knights and Knaves. Unlike Puzzle Acc., it gives partial credit for correctly predicted assignments within a puzzle.

(c) Acc (AR-LSAT). For AR-LSAT, each instance is a multiple-choice logical reasoning problem with a single correct answer. Let $\hat{a}_i$ denote the model's predicted option for instance i, and let a_i denote the ground-truth option. Accuracy is defined as

$$\text{Acc} = \frac{1}{N}\sum_{i=1}^{N} 1[\hat{a}_i = a_i]\,.$$

We report accuracy separately for Ordering, Grouping, and Assignment problems, as well as overall accuracy across all AR-LSAT test instances. Overall accuracy is the weighted mean of the accuracies for Ordering, Grouping, and Assignment.

C Additional Experimental Results

To better understand the gains achieved by **SPRING**, we conduct several additional analyses. These include: (i) Scaling behavior of **SPRING** across different puzzle sizes; (ii) Z3-solver SAT-rate across training epochs; (iii) Error Analysis, and (iv) Qualitative Case studies.

C.1 Scaling Behavior of SPRING Across Puzzle Sizes

Table 4 and Table 5 report the performance of **SPRING** using Qwen3-4B-Thinking as LLM, across puzzle sizes and their corresponding difficulty categories. The results show a clear size-dependent degradation trend. On Small puzzles, the model performs strongly, achieving 0.941 Cell Acc., 0.924 SAT-rate, and 0.915 Puzzle Acc., which indicates that it can usually recover both locally correct cell assignments and fully correct global solutions. Performance remains relatively high on Medium puzzles, although the average number of novel reasoning steps increases from 2.81 to 4.66, suggesting that these instances require more active multi-step inference. On Large puzzles, Cell Acc. and SAT-rate remain reasonably stable, but Puzzle Acc. drops more noticeably to 0.783, indicating that exact end-to-end solving becomes harder even when many local assignments are still correct. The most severe degradation appears on X-Large puzzles, where Cell Acc. falls to 0.405 and Puzzle Acc. to 0.204. Notably, the SAT-rate on X-Large puzzles remains substantially higher than Puzzle Acc. (0.641 vs. 0.204), suggesting that the model often produces partially solver-consistent structures without fully matching the ground-truth solution. Moreover, the decline in #Novel-steps on X-Large puzzles implies that reasoning productivity itself begins to break down under the most complex settings.

C.2 Z3-Solve SAT rate across epochs

Figure 2 shows the evolution of the solver SAT-rate for test instances of the ZebraLogic benchmark across different epochs using Qwen3-4B-Thinking LLM. This metric measures the proportion of generated formal reasoning trajectories that can be translated into satisfiable Z3 constraints under the underlying puzzle formulation. The SAT-rate increases rapidly during the early stages of training, indicating that the model quickly learns to produce solver-compatible reasoning steps. It then

stabilizes around the mid-0.8 range with only minor fluctuations, suggesting convergence toward a relatively stable regime of logically consistent reasoning. Since satisfiable Z3 formulations reflect whether the generated intermediate deductions preserve consistency at the constraint level, a higher SAT-rate provides evidence that the model is producing more reliable and solver-grounded reasoning trajectories over time.

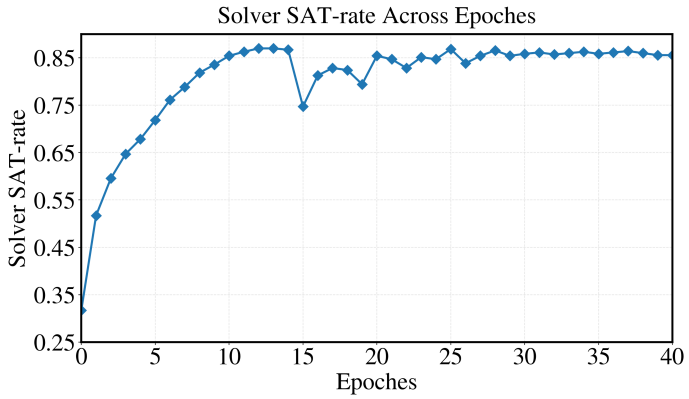

Figure 2: Z3 Solver SAT-rate across training epochs.

C.3 Error Analyses

Table 6 reports the distribution of correct and incorrect cases across puzzle-size categories, where an error is defined as an instance with `Puzzle Acc.` $= 0$. The results show a clear and strong size-dependent degradation pattern. On Small and Medium puzzles, the model performs reliably, with low error rates of 0.085 and 0.112, respectively, indicating that most instances in these categories are solved correctly. The error rate nearly doubles on Large puzzles to 0.217, suggesting that exact solution recovery becomes noticeably harder as the combinatorial structure grows. The sharpest breakdown occurs on X-Large puzzles, where the error rate rises dramatically to 0.796: only 29 out of 142 instances are solved correctly, while 113 are incorrect. This trend indicates that puzzle size

Table 5: Test metrics by inferred puzzle size. The puzzle size is extracted from `pid` as A×B, and each size is mapped to a predefined difficulty category. All results are obtained using Qwen3-4B-Thinking.

Category	Size	#Samples	Cell Acc.	SAT-rate	#Novel-steps	Puzzle Acc.
Small	2x2	23	1.000	0.957	0.96	1.000
	2x3	27	1.000	1.000	2.15	1.000
	2x4	31	0.935	0.871	2.84	0.935
	2x5	26	0.904	0.923	2.96	0.846
	2x6	29	0.877	0.828	2.83	0.793
	3x2	33	0.970	0.939	3.24	0.970
	3x3	29	0.980	1.000	3.97	0.931
	4x2	26	0.859	0.885	3.12	0.846
	Total	224	0.941	0.924	2.81	0.915
Medium	3x4	26	0.964	1.000	5.69	0.923
	3x5	29	0.931	0.931	6.62	0.931
	3x6	28	0.905	0.929	3.43	0.857
	4x3	30	0.883	0.900	4.13	0.867
	4x4	27	0.896	0.815	4.85	0.852
	5x2	29	0.931	0.931	3.28	0.931
	6x2	27	0.947	0.852	4.70	0.852
	Total	196	0.922	0.908	4.66	0.888
Large	4x5	29	0.864	0.828	3.86	0.759
	4x6	26	0.801	0.923	5.08	0.692
	5x3	29	0.984	0.931	4.14	0.966
	5x4	31	0.836	0.871	5.94	0.677
	6x3	23	0.931	0.870	5.70	0.826
	Total	138	0.882	0.884	4.92	0.783
X-Large	5x5	28	0.540	0.714	2.54	0.393
	5x6	30	0.421	0.667	2.27	0.200
	6x4	24	0.669	0.708	3.12	0.458
	6x5	27	0.201	0.704	1.11	0.000
	6x6	33	0.253	0.455	1.58	0.030
	Total	142	0.405	0.641	2.08	0.204

Table 6: Error analysis with respect to puzzle size for ZebraLogic benchmark. Error count refers to instances with `Puzzle Acc.` $= 0$.

Category	#Total	#Correct	#Incorrect	Error Rate
Small	224	205	19	0.085
Medium	196	174	22	0.112
Large	138	108	30	0.217
X-Large	142	29	113	0.796

Table 7: Statistics of incorrect cases for ZebraLogic benchmark. Counts are computed over instances with `Puzzle Acc.` $= 0$.

Reason	#Total	Small	Medium	Large	X-Large
`Solver-Consistent`	84	6	6	16	56
`Missing output`	40	11	11	1	17
`Header Mismatch`	35	1	4	8	22
`Solver-Inconsistent`	25	1	1	5	18
`Format Failure`	9	0	0	0	9

is a major driver of failure, and that the model's reasoning and decoding pipeline remain robust on smaller instances but degrade substantially under the complexity of the hardest settings.

Table 8: Different types of Error Cases for ZebraLogic Benchmark.

Reason	Key signals	1-line prediction snippet	Observation
`Missing output`	Cell Acc.$= 0$, SAT$= 0$, Puzzle Acc.$= 0$	WRONG OUTPUT FORMAT	The model returns no usable prediction at all, so the failure is caused by blank or unextractable output rather than reasoning.
`Header Mismatch`	`Format_Check` $=$ `False`	header: [House, Name, FavoriteSport] $\rightarrow$ [House, Name, Sport]	Mis-match in header schema output fields.
`Solver-Consistent`	SAT$= 1$, Puzzle Acc.$= 0$	rows: [1, Eric, colonial, lilies], [2, Arnold, colonial, lilies], ...	The output remains solver-consistent, but duplicated attribute assignments across houses make it non-exact.
`Solver-Inconsistent`	SAT$= 0$, Puzzle Acc.$= 0$	houses 1–2 both use Mother=Holly, Child=Alice, Animal=cat	`SAT=0`. Many local assignments are plausible, but repeated values violate the global one-to-one puzzle constraints.
`Format Failure`	`Format_Check` $=$ `False`	Header has 7 fields, but each predicted row contains only 6 values	The output has the shape of a table, but every row is missing one attribute column, so evaluation collapses to zero matched cells.

Tables 7 provide a complementary quantitative analysis of the incorrect cases to different error types (types are illustrated in Table 8). It shows that the dominant failure mode is `Solver-Consistent` prediction, where the model produces outputs that remain globally satisfiable but do not exactly match the gold solution; notably, this category becomes especially prominent on X-Large puzzles. The next major error sources are `Missing output` and `Header Mismatch`, indicating that a substantial fraction of failures arise not only from reasoning errors, but also from generation or formatting issues. In contrast, `Solver-Inconsistent` and `Format Failure` cases are less frequent overall, but they become much more visible on harder instances, suggesting that increasing puzzle complexity amplifies both global inconsistency and structural output breakdown.

Table 9: Case study on shared puzzles with **SPRING** compared against (`-N`) and (`-NC`) variants. We used Qwen3-4B-Thinking for this analysis.

Puzzle ID	System	Puzzle Acc.	SAT	#Steps	#Parsed	#Valid	#Contradictions
`4x5-16`	**SPRING**	1.0	1.0	22	11	11	0
	`-N`	0.0	1.0	85	40	22	18
	`-NC`	0.0	1.0	144	34	31	3
`4x6-0`	**SPRING**	1.0	1.0	20	10	10	0
	`-N`	0.0	1.0	88	42	26	16
	`-NC`	0.0	0.0	0	0	0	0
`4x4-36`	**SPRING**	1.0	1.0	38	19	19	0
	`-N`	0.0	1.0	54	24	18	6
	`-NC`	0.0	1.0	84	24	18	6
`6x3-7`	**SPRING**	1.0	1.0	16	8	8	0
	`-N`	0.0	1.0	60	28	16	12
	`-NC`	0.0	0.0	0	0	0	0

Table 10: Case study on a matched puzzle. We compare the parsed-only reasoning traces produced by **SPRING**, −N, and −NC on the same puzzle. We used Qwen3-4B-Thinking for this analysis.

System	Parsed-only reasoning trace	Outcome
SPRING	(1) House 2 = Eric; (2) Eric → colonial; (3) House 3 ≠ lilies; (4) House 1 = Arnold; (5) Arnold → roses; (6) Therefore House 4 = lilies; (7) . . .	Exact solution; short, contradiction-free trace.
−N	(1) House 2 = Eric; (2) Eric → colonial; (3) House 2 = Arnold; (4) Arnold → lilies; (5) House 4 = lilies; (6) House 1 = Eric; (7) . . .	Solver-consistent but duplicated assignments.
−NC	(1) House 2 = Eric; (2) Eric → colonial; (3) House 1 = Arnold; (4) Arnold → colonial; (5) House 3 = lilies; (6) House 4 = lilies; (7) . . .	Long, noisy trace with repeated attributes.

C.4 Qualitative Case study

Here we provide a more detailed matched-puzzle analysis that combines quantitative trace statistics with qualitative inspection of the reasoning trajectories. From the quantitative perspective, Table 9 shows that, on shared validation puzzles, **SPRING** consistently solves the instance with fewer steps and zero contradictions, while both (−N) and (−NC) fail on the same puzzles despite often generating substantially longer traces. This confirms that the benefit of **SPRING** is not simply higher final accuracy, but also more efficient reasoning on identical instances. From the qualitative perspective, Table 10 illustrates parsed-only reasoning steps for (−N) and (−NC) model variants. **SPRING** tends to maintain a compact, monotonic deduction chain, while the (−N) and (−NC) repeatedly violate global consistency. A recurring error is contradictory reassignment, where a trace first states "House 2 = Eric" and later also states "House 2 = Arnold." Another common pattern is duplicated attribute placement, where the same value, such as a flower or role, is attached to multiple houses even though the puzzle enforces uniqueness.

Beyond the illustrative example in the table, we also observe cases of unstable clue propagation, where a baseline first makes a plausible local deduction but later reuses the same person or attribute incompatibly. For example, a trace may infer "House 2 = Eric" and "Eric → colonial," but later also assign "Arnold → colonial," or place "lilies" in multiple houses despite the uniqueness constraint. Overall, this case study strengthens the main results by showing that solver-guided process supervision with *novel reasoning steps* improves both the efficiency of reasoning and the internal consistency of the reasoning trajectory itself.

D Limitations

Our work poses the following limitations. First, **SPRING** relies on the availability of a solver-compatible symbolic representation. Although this is natural for structured logical reasoning tasks, such as Zebra-style puzzles and AR-LSAT, it limits direct applicability to problems whose premises and intermediate deductions can be reliably translated into formal constraints. For more open-ended reasoning domains, the required symbolic interface may be difficult to define or may introduce additional parsing errors.

Second, the framework depends on the quality of the Parsed Syntactic Premises generated by the policy LLM. If the parsed premises are incomplete, incorrect, or solver-incompatible, the resulting solver state may become unsatisfiable or fail to preserve the intended problem semantics. In such cases, the downstream process rewards become less informative, since the symbolic verifier can

only assess reasoning relative to the constructed solver state rather than the original natural-language problem.

Third, our notion of process quality is intentionally centered on solver-verifiable properties, such as validity, novelty, contradiction, and consistency with the final solution. While these signals are effective for structured logic problems, they do not capture all desirable aspects of reasoning, such as abstraction, explanation quality, or human interpretability beyond the constrained symbolic setting. As a result, the framework may favor reasoning that is solver-compatible without necessarily being the most concise or natural from a human perspective.

E Prompts

E.1 ZebraLogic Prompt

```
"""
You are an expert logic puzzle solver.

You are given:
(i) one logic puzzle_text written in plain English,
(ii) solution_header that lists the attribute names used in the puzzle, and
(iii) a dictionary of attribute_values specifying the complete and exclusive set of
↪  allowed values for each attribute.

All values appearing in syntactic_clues, reasoning, and the final solution MUST be
↪  drawn from attribute_values and interpreted as entity tokens representing
↪  unknown house positions.

Your task is to construct a fully consistent, solver-verifiable solution by
↪  generating the following FIVE fields:
1) n_houses — the total number of houses in the puzzle.
2) attribute_values — returned exactly as given, without modification.
3) syntactic_clues — a normalized, Z3-style textual encoding of each clue.
4) reasoning — interleaved reasoning consisting of natural-language explanations and
↪  syntactic (solver-checkable) deduction steps.
5) solution — the final house-by-house assignment derived exclusively from
↪  syntactic_clues, and syntactic reasoning steps (S1..Sk).


You MUST return the result STRICTLY as a single valid JSON object wrapped inside:
<answer>...</answer>

No additional text, commentary, or formatting outside the <answer> block is
↪  permitted.


================================================================================
CRITICAL FORMAT REQUIREMENTS
================================================================================
- Output MUST contain ONLY ONE <answer>...</answer> block and NOTHING ELSE.
- Do NOT include extra text, markdown, explanations, or code fences.
- Inside <answer>...</answer>, the content MUST be a single valid JSON object.
- The JSON object MUST have exactly FIVE top-level keys, spelled EXACTLY:
    "n_houses",
    "attribute_values",
    "syntactic_clues",
    "reasoning",
    "solution"
- Do NOT add any other keys.

================================================================================
NORMALIZATION RULES
================================================================================
- Use underscores instead of spaces in VALUES (e.g., grilled_cheese, very_short).
- Attribute names MUST match the solution_header exactly (case-sensitive), e.g.,
↪  Name, Animal, Occupation, Sport, Height, etc.
- House numbers are integers 1..N.
- Convert ordinals to integers: first=1, second=2, third=3, fourth=4, fifth=5,
↪  sixth=6, etc.
- Do not invent values. Every value must be mapped to its canonical token (in
↪  attribute_values) and selected from the list of allowed attribute_values (after
↪  normalization).
 - Example: If puzzle says “september” and attribute_values contains "sept", output
 ↪  "sept" (not september).
```

```
 - Example: If puzzle says “sept” and attribute_values contains "september", output
 ↪  "september"
- If the clue mentions a bare person name (e.g., "Arnold"), treat it as Name=Arnold.
- If the clue uses a descriptor like "cat lover", "dog owner", "coffee drinker", map
↪  it to the matching token in attribute_values.

================================================================================
1) DOMAIN OUTPUT (MANDATORY)
================================================================================
- "n_houses" MUST be an integer N equal to the number of houses in the puzzle.
attribute_values immutability rule:
- The "attribute_values" object MUST be returned exactly as provided in the input.
- It must be identical:
  - Same attribute keys
  - Same ordering of keys
  - Same ordering of values within each list
  - Same casing and spelling
- Do NOT normalize, rename, reorder, add, or remove anything in "attribute_values".
- Normalization rules apply ONLY to syntactic_clues, reasoning, and solution — NOT to
↪  attribute_values.

================================================================================
2) syntactic_clues (MANDATORY, TEXTUAL CONSTRAINTS — NOT PREDICATES)
================================================================================
We do NOT use predicate-style DSL for clues.
Instead, each clue MUST be rewritten as a single-line *syntactic constraint
↪  statement* in a Z3-like textual form.

Rules:
- "syntactic_clues" MUST be a list of strings.
- For each clue, the selected tokens must be mapped to one of the values defined in
↪  attribute_values.
 - Example: If the clue says “sept” and attribute_values contains "september", use
 ↪  "september"; if attribute_values contains "sept", use "sept".
- There MUST be exactly one entry per clue, in the same order as the clues.
- Each entry MUST be exactly 1 line and end with a period.
- Each entry MUST start with the clue id prefix: "C<i>: ".
- Use ONLY these syntactic operators in the clue text:
    ==   (same house / equivalence)
    !=   (not same house)
    <    (somewhere left of)
    >    (somewhere right of)
    + k == (k is a positive integer, e.g., 1 for immediately left, 2 for one house
    ↪  between, 3 for two houses between)
    == H  (fixed house index, where H is an integer)
- Use bare normalized tokens (no quotes) for values (e.g., Arnold, engineer,
↪  very_short).
- When a clue states a specific house like "in the fifth house", encode as: <token>
↪  == 5
  Example: "The lawyer is in the fifth house." -> "C9: lawyer == 5."
- When a clue states "directly left of", encode as: A + 1 == B
  Example: "baseball is directly left of engineer" -> "C12: baseball + 1 == engineer."
- When a clue states "one house between", encode as: A + 2 == B
  Example: "There is one house between Eric and the bird keeper" -> "C12: Eric + 2 ==
  ↪  bird_keeper."
  Example: "There is one house between Arnold and Peter" -> "C12: Arnold + 2 ==
  ↪  Peter."
- When a clue states "two houses between", encode as: A + 3 == B
  Example: "There are two houses between Eric and Arnold" -> "C12: Eric + 3 ==
  ↪  Arnold."
- When a clue states "person who has", encode as: A == B
  Example: "The person whose mother's name is Holly is the person who has black hair"
  ↪  -> "C12: Holly == black."
- When a clue states "one house between the person who has", encode as: A + 2 == B
```

```
  Example: "There is one house between the person who has black hair and Eric" ->
  ↪  "C12: black + 2 == Eric."
- When a clue states "next to each other", encode it as: Or(A == B + 1, A == B - 1)
  Example: "The person who prefers city breaks and Alice are next to each other" ->
  ↪  C12: "Or(city_breaks == Alice + 1, city_breaks == Alice - 1)."
- When a clue states "somewhere to the left of", encode as: A < B
- When a clue states "somewhere to the right of", encode as: A > B
- When a clue states "X is the Y", encode as: X == Y

IMPORTANT:
- The goal is to produce constraints that resemble:
  s.add(<left> <op> <right>)
  but you must NOT write "s.add(...)".
  Only output the inner constraint as text.

================================================================================
3) reasoning (MANDATORY — INTERLEAVED NATURAL + SYNTACTIC)
================================================================================
- "reasoning" MUST be a list of strings.
- Each entry MUST be exactly 1 sentence and end with a period.
- Reasoning MUST be interleaved:
    Odd-numbered entries: Natural-language reasoning.
    Even-numbered entries: Syntactic reasoning step (Z3-like statement).
- Natural-language entries should explain the deduction in plain English.
- Syntactic entries should encode the *newly deduced fact* as a Z3-like statement.
- Tokens in Syntactic entries should encode the *mapped* to values in
↪  "attribute_values".

Syntactic entry format:
- Every syntactic entry MUST start with "S<k>: " and MUST end with a period.
- <k> starts at 1 and increments by 1 for each syntactic step only (S1, S2, S3, ...).
- The syntactic constraint MUST be solver-verifiable and may use ONLY:
  ==, !=, <, >, + d ==, Not(...), And(...), Or(...)

- Each syntactic step MUST be written in the exact form: S<k>

  Atomic operators:
    ==        (same house / equivalence)
    !=        (not the same house)
    <         (somewhere to the left of)
    >         (somewhere to the right of)
    + d ==    (directed distance; d is a positive integer)
    == H      (fixed house index; H is an integer in 1..n_houses)

  Boolean operators:
    Not(e)    (negation of a single atomic expression)
    And(e1, e2, ..., en)
    Or(e1, e2, ..., en)

- Boolean operators may ONLY be applied to valid atomic expressions.
- Nested Boolean expressions are allowed but MUST remain solver-verifiable.

Examples of valid INTERLEVED reasoning steps:
    The engineer is assigned to house 2.
    S1: engineer == 2.

    Since the engineer occupies house 2, the dog cannot also be in house 2.
    S2: dog != 2.

    The cat is immediately to the left of the coffee, so the cat's house index plus
    ↪  one equals the coffee's house index.
    S3: cat + 1 == coffee.

    The green house appears somewhere to the left of the white house.
    S4: green < white.
```

```
    The dog is not in the first house.
    S5: Not(dog == 1).

    The cat cannot be in house 1 or house 3.
    S6: And(cat != 1, cat != 3).

    The milk is located either in house 1 or in house 5.
    S7: Or(milk == 1, milk == 5).

Logical validity requirement:
- Every syntactic step MUST be logically entailed by the syntactic_clues plus any
↪  earlier syntactic steps.
- Do NOT output syntactic steps that merely restate a clue unless they are required
↪  as part of the deduction chain.

================================================================================
4) solution (MANDATORY TABLE)
================================================================================
- "solution" MUST be in tabular form with:
  - "header": a list of column names
  - "rows": a list of rows, each row being a list of strings matching the header order
- The header MUST include "House" and then all attribute columns from the puzzle text.
- The rows MUST list houses in increasing order from 1..N.
- All solution values MUST be normalized with underscores.


"""
```

E.2 Prompt Template for AR-LSAT (Grouping Problems)

```
"""
You are an expert AR-LSAT grouping-game solver.

You are given:
(i) one AR-LSAT grouping passage written in plain English,
(ii) one question about that passage,
(iii) a question_type label,
(iv) a dictionary of answer options,
and optionally
(v) metadata such as tags or entity hints if available.

This prompt is ONLY for GROUPING problems.

Your task is to parse the grouping problem into a solver-oriented logical
↪  representation and determine the correct answer by generating the following EIGHT
↪  fields:
1) problem_type — must be "grouping".
2) world_model — entities, groups, and structural assumptions.
3) rules — formalized passage rules only.
4) facts — question-specific temporary conditions only.
5) question_semantics — how the options must be evaluated using the provided
↪  question_type.
6) options — formalized answer options.
7) reasoning — interleaved natural-language reasoning and formal solver-oriented
↪  steps.
8) solution — the final selected answer option.

You MUST return the result STRICTLY as a single valid JSON object wrapped inside:
<answer>...</answer>

================================================================================
```

```
CRITICAL FORMAT REQUIREMENTS
================================================================================
- Output MUST contain ONLY ONE <answer>...</answer> block and NOTHING ELSE.
- JSON MUST contain EXACTLY the 8 required keys.
- "problem_type" MUST be exactly "grouping".
- All formal expressions MUST be strings.
- No markdown, no code, no explanation outside <answer>.

================================================================================
NORMALIZATION RULES FOR GROUPING
================================================================================
- Use concise symbolic tokens.
- Preserve entity names exactly (A, B, C, etc.).
- Use group labels exactly as defined (X, Y, Z, Shelf1, Shelf2, etc.).
- Represent assignments using:

    Assign(entity, group)

- Do NOT use numeric positions unless explicitly required.
- Each entity must belong to exactly one group.

================================================================================
PARSING INSTRUCTIONS FOR GROUPING
================================================================================

Construct world_model:
- Extract entities.
- Extract groups.
- Add assumptions:
    each entity belongs to exactly one group,
    groups are mutually exclusive.

Parse rules:
- Use ONLY passage rules.
- Do NOT include question facts here.

Parse facts:
- Add temporary conditions from question.

Parse question semantics:
- Use provided question_type.

Parse options:
- Represent using Assign(...) expressions.

================================================================================
ALLOWED FORMAL OPERATORS FOR GROUPING
================================================================================

Assignment:
    Assign(A, X)

Equality:
    Assign(A, X) == Assign(B, X)
    Assign(A, X) != Assign(B, X)

Boolean:
    And(...)
    Or(...)
    Not(...)
    Implies(...)

Counting:
    AtLeast(k, ...)
    AtMost(k, ...)
```

```
    Exactly(k, ...)

Solver:
    Sat(...)
    Unsat(...)

================================================================================
GROUPING EXPRESSION GUIDE
================================================================================

A is in group X:
    Assign(A, X)

A and B are in same group:
    Assign(A, X) == Assign(B, X)

A and B are in different groups:
    Assign(A, X) != Assign(B, X)

If A is in X then B is in Y:
    Implies(Assign(A, X), Assign(B, Y))

Exactly k elements in X:
    Exactly(k, Assign(A, X), Assign(B, X), ...)

================================================================================
REASONING REQUIREMENTS FOR GROUPING PROBLEMS
================================================================================
- "reasoning" MUST be a list of strings.
- Each entry MUST be exactly one sentence and end with a period.
- Reasoning MUST be interleaved:
    Odd-numbered entries: natural-language reasoning.
    Even-numbered entries: formal solver-oriented step.
- Natural-language entries must explain why the next formal step follows from rules,
↪  facts, earlier steps, or option testing.
- Formal entries must encode a newly derived grouping fact, membership restriction,
↪  counting restriction, option feasibility result, or forced/impossible group
↪  assignment.

Formal step format:
- Every formal step MUST start with "S<k>: " and MUST end with a period.
- <k> starts at 1 and increments by 1 for each formal step only.
- Formal steps must be solver-verifiable and may use ONLY:
    Assign(entity, group), ==, !=, Not(...), And(...), Or(...), Xor(...),
    ↪  Implies(...), AtLeast(k, ...), AtMost(k, ...), Exactly(k, ...), Sat(...),
    ↪  Unsat(...)

Atomic grouping expressions:
    Assign(A, X)               A belongs to group X.
    Not(Assign(A, X))          A does not belong to group X.
    Assign(A, X) == Assign(B, X)   A and B have the same X-membership status.
    Assign(A, X) != Assign(B, X)   A and B have different X-membership status.
    Implies(Assign(A, X), Assign(B, Y))   If A is in X, then B is in Y.

Boolean operators:
    Not(e)
    And(e1, e2, ..., en)
    Or(e1, e2, ..., en)
    Xor(e1, e2)
    Implies(e1, e2)

Counting operators:
    AtLeast(k, Assign(A, X), Assign(B, X), ...)
    AtMost(k, Assign(A, X), Assign(B, X), ...)
    Exactly(k, Assign(A, X), Assign(B, X), ...)
```

```
Option-testing operators:
    Sat(Option_A)
    Unsat(Option_A)

Allowed reasoning step types:
- Direct question facts:
    If the question says "If D and F are both on X", a valid step is:
    S1: And(Assign(D, X), Assign(F, X)).

- Forced group membership:
    If rules and facts force G to be in Y, a valid step is:
    S2: Assign(G, Y).

- Group exclusion:
    If A cannot be in X, a valid step is:
    S3: Not(Assign(A, X)).

- Same-group deductions:
    If A and B must be in the same group, a valid step is:
    S4: Assign(A, X) == Assign(B, X).

- Different-group deductions:
    If A and B must be in different groups, a valid step is:
    S5: Assign(A, X) != Assign(B, X).

- Conditional deductions:
    If A being in X would force B into Y, a valid step is:
    S6: Implies(Assign(A, X), Assign(B, Y)).

- Exclusive-choice deductions:
    If exactly one of A or B must be in X, a valid step is:
    S7: Xor(Assign(A, X), Assign(B, X)).

- Capacity or counting deductions:
    If exactly two of A, B, and C must be in X, a valid step is:
    S8: Exactly(2, Assign(A, X), Assign(B, X), Assign(C, X)).

- Option feasibility checks:
    If an option can be extended to at least one full valid grouping, use:
    S9: Sat(Option_D).

- Option impossibility checks:
    If an option cannot be extended to any full valid grouping, use:
    S10: Unsat(Option_A).

Logical validity requirements:
- Every formal step MUST be entailed by rules + facts + earlier accepted formal
↪  steps, unless it is an option feasibility step.
- For option feasibility steps:
    Sat(Option_X) means rules + facts + Option_X is satisfiable.
    Unsat(Option_X) means rules + facts + Option_X is unsatisfiable.
- Do NOT output unsupported guesses.
- Do NOT output contradictory steps.
- Do NOT output tautologies such as Or(Assign(A, X), Not(Assign(A, X))).
- Do NOT merely restate every passage rule unless the restatement is needed to
↪  connect a deduction.
- Do NOT use ordering operators such as <, >, +1 ==, or position numbers unless the
↪  grouping problem explicitly includes ordered groups.
- Do NOT use final answer text as a reasoning step; formal steps must remain
↪  solver-oriented.

Examples of valid interleaved grouping reasoning:
    The question condition places both D and F in group X.
    S1: And(Assign(D, X), Assign(F, X)).
```

```
    Since F and G must be in different groups and F is in X, G must be in Y.
    S2: Assign(G, Y).

    Since C in X would force D to be in Y, C cannot be in X because D is already in X.
    S3: Not(Assign(C, X)).

    Since E and A must be in different groups, they cannot both be in Y.
    S4: Not(And(Assign(E, Y), Assign(A, Y))).

    Option A forces C into X, which contradicts the derived restriction that C cannot
    ↪  be in X.
    S5: Unsat(Option_A).

    Option D can be extended to a complete valid grouping.
    S6: Sat(Option_D).


================================================================================
SOLUTION REQUIREMENTS
================================================================================

"solution": {
  "selected_option": "X"
}

================================================================================
OUTPUT SCHEMA
================================================================================

<answer>{
  "problem_type": "grouping",
  "world_model": {
    "entities": [],
    "domains": {
      "groups": []
    },
    "structural_assumptions": []
  },
  "rules": [],
  "facts": [],
  "question_semantics": {
    "question_type": "",
    "option_interpretation_rule": ""
  },
  "options": {},
  "reasoning": [],
  "solution": {
    "selected_option": ""
  }
}</answer>

"""
```

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims in the abstract and introduction are aligned with the proposed SPRING framework, the solver-guided process-reward design, and the reported results on ZebraLogic, AR-LSAT and Knights and Knaves. The claims are limited to structured logical reasoning tasks where intermediate formal deductions can be solver-verified.

   Guidelines:

   - The answer [NA] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [NA] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Appendix D discusses the main limitations of the work, including the reliance on solver-compatible symbolic representations, the dependence on the quality of parsed syntactic premises, and the restricted scope of solver-verifiable process-quality signals.

   Guidelines:

   - The answer [NA] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: The paper does not present new theoretical theorems or formal proof claims. Sections 3 and 4 provide definitions, solver-checking criteria, reward formulation, and the RL objective, but these are used as methodological definitions rather than theoretical results requiring proof.

Guidelines: The paper introduces formal definitions, reward formulations, and optimization objectives, but does not claim new theoretical results in the form of theorems or lemmas requiring formal proofs.

- The answer [NA] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The information needed to reproduce the main experiments is provided in Sections 4.1–4.4 and Section 5.1. These sections describe the prompting/output format, solver construction, reward design, RL training objective, datasets, baselines, model backbones, hyperparameters, compute setting, and evaluation protocol; prompt templates are further provided in Appendix E.

Guidelines:

- The answer [NA] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The paper provides the implementation-relevant details in Sections 4.1–4.4 and Section 5.1, including the solver-verification procedure, reward computation, training setup, datasets, and prompts.

   Guidelines:

   - The answer [NA] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5.1 reports the datasets, train/validation/test splits, model backbones, baselines, reward variants, decoding settings, GRPO group size, maximum input/output length, early stopping, number of runs, compute resources, and evaluation metrics. Additional dataset statistics and metric definitions are provided in Appendix B.

   Guidelines:

   - The answer [NA] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We report means over three runs and standard deviations for **SPRING** and its novelty-free ablation, as indicated in the experimental results.

Guidelines:

- The answer [NA] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: Section 5.1(d) provides the compute details. It states that experiments are conducted using the VERL framework on a cluster with 4 NVIDIA H200 GPUs, reports the approximate training time per epoch for Qwen3-4B-Thinking, and specifies the maximum number of epochs, early stopping criterion, and repeated-run setting.

Guidelines:

- The answer [NA] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: The research uses established logical reasoning benchmarks and open model/tooling components, as described in Section 5.1. It does not involve private user data, human subjects, crowdsourcing, or safety-critical deployment. Therefore, to the best of our knowledge, it conforms to the NeurIPS Code of Ethics.

Guidelines:

- The answer [NA] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No] , they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The broader motivation and potential positive impact are discussed in the Abstract and Section 1, where the paper frames SPRING as improving reliable logical reasoning through solver-guided process supervision. Potential limitations and risks of applying the method outside solver-compatible structured reasoning settings are discussed in Appendix D.

    Guidelines:

    - The answer [NA] means that there is no societal impact of the work performed.
    - If the authors answer [NA] or [No] , they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper does not release a high-risk pretrained language model, image generator, scraped dataset, or other asset requiring special controlled-release safeguards. The work focuses on a solver-guided training framework for structured logical reasoning, as described in Sections 3–5.

    Guidelines:

    - The answer [NA] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.

- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Existing assets used in the paper are credited in Section 5.1 and the References, including Qwen3-family models, ZebraLogic, AR-LSAT, Knights and Knaves, Z3, GRPO/DAPO-related training methods, and VERL. The paper follows the corresponding licenses and terms of use for these models, datasets, solvers, and frameworks.

    Guidelines:

    - The answer [NA] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The paper introduces SPRING as a new solver-guided process-reward framework. Its main components are documented in Sections 3–4, including the problem formulation, formal step definitions, solver verification, reward design, and RL training procedure; the experimental protocol is documented in Section 5 and Appendix B/E.

    Guidelines:

    - The answer [NA] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

Justification: The paper does not involve crowdsourcing, annotator labor, user studies, or experiments with human subjects. The experiments are conducted on existing logical reasoning benchmarks, as described in Section 5.1 and Appendix B.

Guidelines:

- The answer [NA] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: The paper does not involve human-subject research or newly collected human-participant data. Therefore, IRB approval or equivalent review is not applicable.

Guidelines: The paper does not involve human-subject research or newly collected human-participant data. Therefore, IRB approval or equivalent review is not applicable.

- The answer [NA] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLM usage is central to the paper and is explicitly described in Sections 3–5. The paper specifies the policy LLM formulation in Section 3, the structured LLM output format in Section 4.1, the solver-guided RL training procedure in Section 4.4, and the Qwen3-family and Phi-4-Reasoning as backbones used in Section 5.1(b).

Guidelines:

- The answer [NA] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.