Knowledge-Based Systems 327 (2025) 114140

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

LoRP: LLM-based Logical Reasoning via Prolog

Zhengkun Di[a], Chaoli Zhang[b,*], Hongtao Lv[a,c,*], Lizhen Cui[a,c], Lei Liu[a,d]

[a] School of Software, Shandong University, No. 1500 Shunhua Road, Jinan, 250000, Shandong, China
[b] School of Computer Science and Technology (School of Artificial Intelligence), Zhejiang Normal University, No. 688 Yingbin Road, Jinhua, 321000, Zhejiang, China
[c] Joint SDU-NTU Centre for Artificial Intelligence Research (C-FAIR), Shandong University, No. 1500 Shunhua Road, Jinan, 250000, Shandong, China
[d] Shandong Research Institute of Industrial Technology, No. 1768 Xinluo Road, Jinan, 250000, Shandong, China

ARTICLE INFO

Keywords:
Large language model
Logical reasoning enhancement
Prolog

ABSTRACT

Enhancing the logical reasoning capabilities of large language models (LLMs) is crucial for advancing LLMs's applications in complex problem-solving contexts. Neurosymbolic programming-based approaches have demonstrated significant advantages in logical reasoning. Prolog is a high-level declarative programming language based on formal logic, well-suited for handling complex deductive reasoning tasks. However, its strict syntactic structure imposes inherent limitations on expressiveness, making it difficult to represent certain common logical constructs found in natural language. Since first-order logic (FOL) is the most fundamental formal language for logical semantic representation, we take it as a reference for analysis and find that even some of its most basic structures cannot be directly expressed in Prolog. To address this, we propose a systematic translation mechanism from FOL to Prolog, thereby extending Prolog's expressiveness to support richer logical representations. Building on this foundation, we propose LoRP (LLM-based Logical Reasoning via Prolog), a novel framework that utilizes LLMs to convert natural language queries into Prolog code, and delegates reasoning to the external SWI-Prolog interpreter. This hybrid architecture combines the formal rigor of symbolic logic with the flexibility of LLMs, enabling precise, interpretable, and verifiable reasoning. Empirical evaluations demonstrate that LoRP significantly improves LLMs' reasoning performance, particularly as inference depth increases. It also exhibits strong generalization and stability across various model architectures. These findings highlight the potential of symbolic-neural integration as a promising direction for advancing the logical reasoning capabilities of LLMs.

1. Introduction

Large Language Models (LLMs), such as GPT-4 [1] and DeepSeek-V3 [2], have exhibited substantial proficiency in understanding and generating natural language (NL), achieving cutting-edge performance across a broad spectrum of natural language processing (NLP) tasks, including text generation, summarization, and question answering. Trained on extensive corpora using sophisticated neural architectures, LLMs effectively capture complex linguistic patterns and subtle nuances inherent in human language.

Despite their notable successes in various language-related tasks, LLMs continue to exhibit significant limitations in logical reasoning [3]. Logical reasoning necessitates a systematic approach to deriving conclusions from explicit premises, requiring an understanding of formal rules, structured argumentation, and rigorous logic that extends beyond the statistical correlations embedded in NL. Although contemporary LLMs perform reasonably well in simple, few-step reasoning tasks but often struggle with those requiring multi-step logical reasoning [4]. In scenarios involving multi-step reasoning or complex logical structures, LLMs frequently rely on surface-level pattern matching without fully comprehending the underlying logical principles [5], resulting in inconsistencies and deviations from logical reasoning processes [6]. These deficiencies lead to suboptimal performance in tasks such as NL understanding and commonsense reasoning [7], structured decision-making (e.g., planning and summarization) [8], and maintaining coherence in dialogue systems [9]. Therefore, enhancing the logical reasoning capabilities of LLMs, particularly their multi-step reasoning abilities, is crucial.

The primary limitations of LLMs in logical reasoning arise from their reliance on NL as the sole medium for reasoning. NL is inherently flexible and dynamic, often lacking explicit logical structures, making it challenging for LLMs to detect the underlying logical relationships. While this issue may be less apparent in simple, few-step reasoning, it becomes particularly pronounced in multi-step reasoning. Implicit logical structures are frequently overlooked or misinterpreted, and errors accumulate with the increase in reasoning steps, leading to significant declines in accuracy and consistency. Thus, the core objective of our research

* Corresponding authors.
E-mail addresses: 202200400002@mail.sdu.edu.cn (Z. Di), chaolizcl@zjnu.edu.cn (C. Zhang), lht@sdu.edu.cn (H. Lv), clz@sdu.edu.cn (L. Cui), l.liu@sdu.edu.cn (L. Liu).

https://doi.org/10.1016/j.knosys.2025.114140
Received 5 April 2025; Received in revised form 9 July 2025; Accepted 19 July 2025
Available online 20 July 2025

is to enhance the multi-step reasoning capabilities of LLMs. Chain-of-thought (CoT) [10] guides the model to generate explicit intermediate reasoning steps, thereby simulating the human-like step-by-step reasoning process, with the goal of improving interpretability and accuracy in complex problem solving. Self-Consistency (SC) [11] further builds upon CoT by sampling multiple reasoning paths and selecting the most frequent answer to enhance answer stability. While both of them alleviate reasoning performance issues to some extent, they do not fundamentally address the inherent limitation of LLMs being confined to NL-based reasoning frameworks.

To address these limitations, we turn to symbolic logic languages that support more structured and interpretable reasoning. Among them, Prolog [12] stands out as a high-level declarative programming language designed for logic-based inference. It represents knowledge through facts and rules and, with its strict syntactic structure and built-in backtracking mechanism, enables efficient symbolic reasoning. These features make it particularly suitable for tasks such as logical inference, knowledge representation, and rule-based problem solving. However, Prolog's rigid syntax and limited expressive power pose challenges for directly representing certain logical structures inherent in NL. For example, Prolog cannot directly express the existential quantifier commonly found in NL. A sentence like *There exists a bird that is black* cannot be directly encoded in Prolog, which only supports universally applicable facts or rules. Similarly, Prolog has fundamental limitations in expressing classical negation. A statement such as *The tiger is not a bird* cannot be directly asserted as a fact in Prolog. These limitations significantly reduce the expressiveness of Prolog in capturing the logical structures of NL, and in turn, constrain its effectiveness as a reasoning engine for NL-derived logical representations.

To overcome the expressive limitations of Prolog, we design a systematic mechanism that translates first-order logic (FOL) into Prolog. Since FOL is the most fundamental and widely used formal language for logical representation, we extend Prolog's expressiveness to fully cover FOL, thereby establishing a formal foundation for encoding NL semantics. This approach enhances Prolog's expressiveness in representing NL, enabling it to accommodate more complex and diverse logical structures. Based on the extended Prolog, we propose our framework, **LoRP: LLM-based Logical Reasoning via Prolog**, as depicted in Fig. 1. The core idea of LoRP is to translate deductive reasoning tasks from NL into Prolog, mapping semantics from the flexible NL domain to the rigorous Prolog semantic space, and delegating reasoning tasks to an external interpreter. LoRP comprises four main steps: translation, validation, proving, and voting. In the **translation step**, the NL input is translated into the extended Prolog through a translator. In the **validation step**, the syntax validator assesses the translated Prolog code for completeness and correctness, ensuring that the code adheres to Prolog's syntactic requirements before proceeding to the next step. **In the proving step**, the validated Prolog code is executed using an external interpreter, SWI-Prolog, to derive reasonings. In the **voting step**, to enhance the robustness and reliability of the framework, a three-vote majority mechanism is implemented. Each query is independently processed three rounds, and the most frequent result is selected as the final output.

The advantages of LoRP lie in three key aspects. First, by explicitly recognizing and addressing the inherent limitations of LLMs in deductive reasoning—unlike existing frameworks such as Logic-of-Thought [13] and LLM-ARC [14]—LoRP translates reasoning tasks into Prolog and delegates them to an external interpreter, thereby enhancing the stability and accuracy of the reasoning process. Second, by extending Prolog's expressiveness, it enables more faithful representation of logical structures inherent in NL, ultimately improving generalizability across tasks and domains. Third, reasoning in LoRP is executed through Prolog's formal logic rules, resulting in transparent and traceable

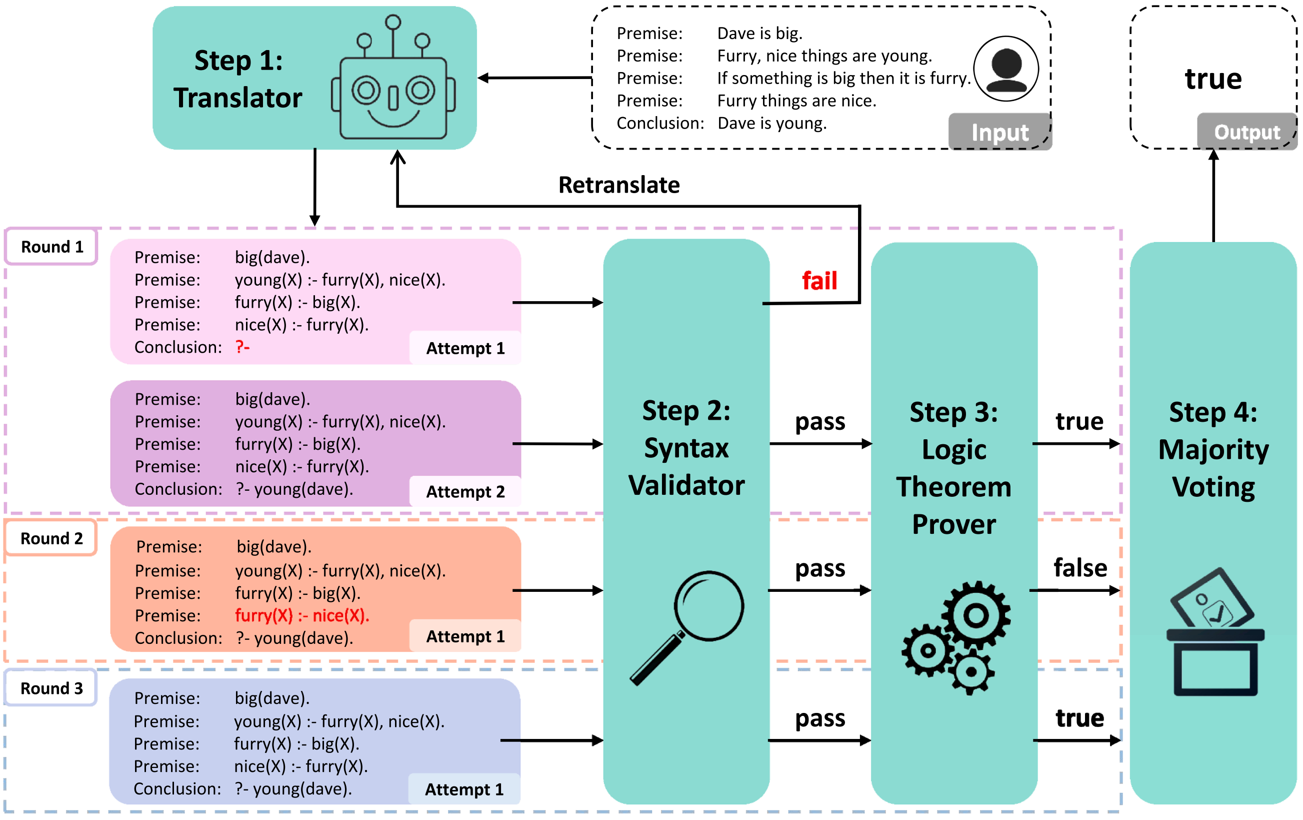

Step 1: Translator
Premise: Dave is big.
Premise: Furry, nice things are young.
Premise: If something is big then it is furry.
Premise: Furry things are nice.
Conclusion: Dave is young.
Input
true
Output
Retranslate
Round 1
Premise: big(dave).
Premise: young(X) :- furry(X), nice(X).
Premise: furry(X) :- big(X).
Premise: nice(X) :- furry(X).
Conclusion: ?-
Attempt 1
Premise: big(dave).
Premise: young(X) :- furry(X), nice(X).
Premise: furry(X) :- big(X).
Premise: nice(X) :- furry(X).
Conclusion: ?- young(dave).
Attempt 2
Round 2
Premise: big(dave).
Premise: young(X) :- furry(X), nice(X).
Premise: furry(X) :- big(X).
Premise: furry(X) :- nice(X).
Conclusion: ?- young(dave).
Attempt 1
Round 3
Premise: big(dave).
Premise: young(X) :- furry(X), nice(X).
Premise: furry(X) :- big(X).
Premise: nice(X) :- furry(X).
Conclusion: ?- young(dave).
Attempt 1
Step 2: Syntax Validator
fail
pass
pass
pass
Step 3: Logic Theorem Prover
true
false
true
Step 4: Majority Voting

Fig. 1. This figure illustrates the complete LoRP framework, encompassing translation step, validation step, proving step, and voting step. The translation step leverages an LLM-based translator to convert NL input into Prolog representations. The validation step ensures the correctness of the translated Prolog code by employing a syntax validator. In the proving step, SWI-Prolog is used to execute logical queries, and if it encounters errors, an LLM-based supplementary reasoning mechanism takes over. Finally, the voting step aggregates multiple independent results to enhance stability and determine the most reliable final answer.

inference chains with clearly structured intermediate steps. This design enables fully symbolic, verifiable, and interpretable reasoning, and helps prevent cumulative errors in multi-step inference.

Our contributions are threefold:

1. To enhance the logical reasoning capabilities of LLMs, we first expand the expressiveness of Prolog by designing a systematic translation mechanism from FOL to Prolog. This mechanism enables Prolog to represent a broader range of logical structures found in NL. Building upon this foundation, we propose LoRP, a framework that integrates the formal rigor of Prolog with the flexibility of LLMs. LoRP translates deductive reasoning tasks from NL into Prolog, delegates reasoning to an external interpreter, and thereby addresses key limitations of LLMs in handling complex multi-step reasoning.
2. Through a comparative analysis of four reasoning strategies across five datasets based on seven LLMs, LoRP demonstrates significant advantages, showcasing its superior logical reasoning capabilities as well as strong generalization and stability across different LLM architectures and reasoning tasks.
3. By conducting a comprehensive error analysis of LoRP and CoT, we identify distinct error patterns, providing insights into future avenues for integrating external tools to augment LLMs, and suggesting a potential complementary relationship between the two approaches.

2. Related Work

Several primary approaches have been proposed to enhance the logical reasoning capabilities of LLMs. These can be broadly categorized into three types based on the role of the LLM. First, some methods rely on prompt engineering to enable LLMs to perform reasoning tasks independently. Second, others integrate LLMs with alternative reasoning paradigms to improve performance. Third, external tools can be incorporated to enhance the accuracy and stability of reasoning processes.

2.1. Prompt Engineering

Prompt engineering is a classical approach aimed at maximizing the inherent reasoning capabilities of LLMs through carefully crafted prompts, such as CoT [10], CoT-SC [11] and Struct-X [15]. CoT enables LLMs to reason step by step, enhancing the coherence and consistency of the reasoning process. CoT-SC builds on this by introducing multiple sampling, which improves the stability of LLM reasoning. Struct-X introduces a five-stage framework-read, model, fill, reflect, and reason-designed to enhance LLM reasoning over structured inputs by constructing graph-based representations and filtering irrelevant information. It integrates an auxiliary prompt generation module that formulates task-specific prompts to effectively steer the model's reasoning process.

Despite their effectiveness, prompt engineering methods remain fundamentally constrained by the internal generation process of LLMs. They lack symbolic grounding and offer limited control over intermediate reasoning steps, often leading to instability, hallucinations, or failure on structurally complex tasks. Moreover, the reasoning steps are not formally verifiable, making error tracing and correction difficult. In contrast, LoRP addresses these limitations by explicitly transforming natural language inputs into Prolog and outsourcing logical inference to an external interpreter. This design ensures symbolic soundness, verifiability, and robustness. The use of Prolog allows for clear and deterministic inference chains, enhancing consistency and enabling rigorous handling of complex logical constructs.

2.2. Integrating LLMs with Alternative Reasoning Modes

Another mainstream approach is to combine LLMs with other forms of reasoning, leveraging the reasoning power of LLMs to enhance the stability, accuracy, and overall effectiveness of the reasoning process. Entailer [16] integrates LLMs with backward reasoning, starting from the problem and progressively searching for premises and intermediate steps that support the goal, eventually backtracking to known information. This approach makes reasoning more targeted and helps avoid unnecessary branches. Reasoning via Planning (RAP) [17], by combining LLMs with world modeling and planning algorithms, conducts strategic exploration during reasoning, balancing exploration and exploitation to find the optimal reasoning path. Logic-of-thought (LoT) [13] embeds logical structures and explicit logical cues into the input context, enabling the model to better grasp the nature of the task and thereby improve the accuracy of its reasoning. Graph of Thought (GoT) [18] improves logical reasoning in LLMs by structuring reasoning as a graph, where nodes represent logical steps and edges capture their relationships. This structure allows for more dynamic and transparent reasoning, enhancing performance in multi-step reasoning tasks. HGAJ [19] introduces an adaptive heterogeneous graph reasoning framework for modeling relational dependencies in interconnected systems. It constructs dynamic, type-aware graph structures that capture both semantic and structural relationships among entities, and performs reasoning through learned graph representations. This approach enables more effective relational inference in domains characterized by complex and heterogeneous interactions.

Although these methods improve reasoning performance to some extent by enriching intermediate representations, the execution of reasoning remains reliant on LLMs. This dependency makes them prone to inconsistency and hallucination, especially in multi-step inference. In contrast, LoRP decouples translation and reasoning: the LLM handles the semantic translation from NL to Prolog, while formal inference is delegated to an external Prolog interpreter. This separation improves robustness, interpretability, and logical consistency without modifying or fine-tuning the LLM.

2.3. Incorporation of External Tools

Integrating external tools is a common approach in enhancing LLM reasoning. This can include frameworks such as [20], which enhance reasoning and decision-making by incorporating external knowledge sources like knowledge graphs, and frameworks such as Logical Inference via Neurosymbolic Computation (LINC) [21] and Large Language Model with an Automated Reasoning Critic (LM-ARC) [14]. LINC improves the stability and accuracy of reasoning by converting NL into symbolic representations and using external provers, while LM-ARC introduces Automated Reasoning Critic (ARC) to identify reasoning errors and provide specific feedback, thereby enhancing the reasoning quality and logical rigor of LLMs. Logic-LM [8] also adopts a hybrid approach by combining LLMs with external symbolic solvers, such as Prover9 and Z3. It supports multiple types of symbolic reasoning-including deductive and constraint solving-by selecting the appropriate formalism based on the task. Rather than committing to a single symbolic language, Logic-LM employs different reasoning backends for different symbolic tasks. Unlike the frameworks mentioned above, LoRP translates NL into Prolog and performs reasoning using SWI-Prolog. The straightforward nature of Prolog's syntax makes it particularly well-suited for deductive reasoning. LoRP adopts a similar strategy by combining LLMs with an external symbolic interpreter. However, it differs from prior works by extending the expressiveness of Prolog to better align with the logical structures found in NL. Moreover, LoRP introduces a supplementary reasoning mechanism to handle challenging or incomplete inputs. These design choices contribute to its stronger generalizability across a wide range of reasoning tasks.

Several recent frameworks also incorporate Prolog into their reasoning pipelines, but pursue different reasoning goals and system designs compared to LoRP. For example, PROPER [22] is developed for mathematical reasoning tasks such as GSM8K, focusing on extracting quantitative relationships from natural language and expressing them in Prolog for arithmetic computation. Unlike LoRP, which emphasizes symbolic logical inference, PROPER does not address the representation of logical structure or multi-step deductive reasoning. ProSLM [23], by contrast, targets domain-specific knowledge-based question answering (KBQA). It integrates Prolog with structured knowledge rules to assist LLMs in generating accurate responses within specialized domains. In contrast to LoRP, which focuses on general-purpose logical reasoning, ProSLM is applied to task-specific datasets and emphasizes domain-dependent knowledge retrieval. THOUGHT-LIKE-PRO [24] also incorporates Prolog, but with a different objective and usage paradigm. Instead of translating natural language into executable logic programs, it prompts an LLM to generate Prolog code comprising rules, facts, and queries. These programs are executed and verified by a Prolog engine, and the valid reasoning traces are converted into natural language chain-of-thoughts to fine-tune the model. The key distinction between THOUGHT-LIKE-PRO and LoRP lies in their use of Prolog: LoRP employs Prolog and its interpreter as integral components in an end-to-end system that performs symbolic inference at runtime, whereas THOUGHT-LIKE-PRO uses Prolog solely to construct logically consistent training data; inference is still carried out by the LLM.

3. Preliminary

This section provides an overview of the key concepts underpinning our approach, including the Closed World Assumption (CWA), an overview of Prolog, the reasoning interpreter adopted in our framework, and essential FOL inference rules. These foundational principles establish the theoretical groundwork for translating FOL representations into Prolog, thereby enabling structured and rigorous reasoning within LLMs.

3.1. Closed-World and Open-World Assumptions

In the field of knowledge representation and logical reasoning, the Closed World Assumption (CWA) [25] is one of the fundamental principles used to interpret the truth value of facts. Under the CWA, anything that is not known to be true is assumed to be false. This assumption is typically adopted in logic programming languages including **Prolog**, enabling efficient reasoning by treating unknown information as false.

In contrast, the Open World Assumption (OWA) [26] holds that the absence of knowledge does not imply falsity, and statements that cannot be proven true are treated as “unknown” rather than “false”. OWA is prevalent in domains characterized by incomplete or evolving knowledge, such as ontology-based reasoning and the Semantic Web, where preserving epistemic uncertainty is essential for sound inference.

3.2. Prolog

Prolog is a logic programming language widely used for knowledge representation and symbolic reasoning. The following are commonly used elements in Prolog programs:

- $:-$ (implication): The expression A :- B. denotes that A is true if B is true.
- \+ (negation as failure): The expression $\backslash + P(X)$ means that $P(X)$ cannot be proven true.
- ; (disjunction): The expression $A; B$ means either A is true or B is true.
- , (conjunction): The expression A, B means A and B must both be true.

Prolog offers the following key advantages. First, it is based on first-order and predicate logic, enabling efficient and precise representation of complex rules and relationships. For instance, *If someone is quiet, then they are clever* is translated as *clever(X) :- quiet(X)*. This concise and accurate logical representation, combined with the interpreter, ensures consistency and precision in multi-step reasoning. Second, as a declarative language, Prolog allows developers to define logical relationships through simple facts and rules, with the interpreter automatically handling the reasoning process. This significantly reduces the development complexity compared to traditional procedural languages like C or Java [27]. Third, Prolog incorporates a backtracking mechanism. Prolog and its interpreters (e.g., SWI-Prolog) employ a depth-first search (DFS) strategy [28], querying rules sequentially as they appear in the code. When a reasoning path fails, the interpreter backtracks to explore alternative paths, effectively examining multiple potential solutions. This mechanism mitigates a common problem faced by LLMs—getting stuck in incorrect paths and consequently missing the correct reasoning sequence.

While Prolog’s rigorous syntax facilitates reliable reasoning, it also imposes constraints on its expressive capacity. For example, the rule $P(X)$:- $\backslash + Q(X)$ denotes that $P(X)$ holds when $Q(X)$ cannot be proven true. However, if we intend to express the opposite—that P(X) does not hold when $Q(X)$ is true—an intuitive formulation in Prolog would be $\backslash + P(X)$:- $Q(X)$. This, however, is syntactically invalid in standard Prolog. As a result, Prolog lacks direct support for representing negative conclusions derived from positive premises.

3.3. SWI-Prolog

Among the available Prolog interpreters, SWI-Prolog [29] was selected for LoRP due to several key advantages. First, unlike alternatives such as SICStus Prolog or GNU Prolog, SWI-Prolog is open-source and freely accessible, making it widely available. Second, it also features built-in multithreading capabilities, allowing for parallel execution that significantly improves efficiency in experimental settings. Third, SWI-Prolog provides seamless integration with modern programming languages, including Python, Java, and C, facilitating its use in contemporary software development contexts.

3.4. FOL Inference Rules

In this section, we introduce several theorems that are essential for the subsequent FOL to Prolog translation mechanism. The formal conditions and usage constraints of these rules are detailed in standard texts on mathematical logic [30].

Modus ponens (MP). Modus Ponens [31] allows deducing a proposition B from a conditional statement $A \Rightarrow B$ and the assertion A, and is formally expressed as: $\frac{A, A \Rightarrow B}{B}$.

Universal instantiation (UI). Universal Instantiation [32] allows inferring a specific instance $P(a)$ from a universally quantified statement $\forall x\, P(x)$, and is formally expressed as: $\frac{\forall x\, P(x)}{P(a)}$.

Universal generalization (UG). Universal Generalization [32] allows generalizing from a specific instance $P(a)$ to a universally quantified statement $\forall x\, P(x)$, and is formally expressed as: $\frac{P(a)}{\forall x\, P(x)}$.

4. Extending the Expressiveness of Prolog

As discussed in Section 3.2, Prolog’s strict syntax imposes inherent limitations on its expressiveness, making it difficult to represent certain logical elements found in NL. FOL, as the most fundamental formal language for representing logical semantics, serves as the foundation for our analysis. Despite its relative simplicity, some FOL elements still cannot be directly expressed in Prolog. To address this gap, we categorize the types of logical elements in FOL that Prolog fails to represent and propose a systematic mechanism for translating FOL into Prolog, thereby extending Prolog’s expressiveness. These logical elements are classified into two categories based on their compatibility with Prolog: perfectly

Table 1
A systematic classification of FOL elements based on their expressibility in Prolog. The table summarizes representative translation strategies and clearly distinguishes between elements that are perfectly expressible and those that require approximation due to Prolog's inherent limitations. It also identifies key limitations in representing complex logical structures.

Prolog expressibility	FOL element	Translation strategy	Limitations
Perfectly expressible	Equivalence ($\Leftrightarrow$)	Translated as mutual rules: P :- Q. and Q :- P.	
	Existential quantifier ($\exists$)	Skolemization into constant: $P(c)$.	
	Rule Queries ($? - \forall x\ B_1(x) \wedge B_2(x) \wedge \cdots \wedge B_n(x) \rightarrow H(x)$)	Insert premises as facts, query conclusion	
Imperfectly expressible	Negation ($\neg$)	Approximated via $not\_P(X)$:- \+$P(X)$	May trigger infinite mutual recursion in the absence of explicit facts
	Disjunction ($\vee$)	Approximated as $A \wedge B$ in rules	Lacks uncertainty modeling
	Exclusive Disjunction ($\oplus$)	Encoded with mutually negated rules	Fails to enforce exclusivity and completeness under closed-world assumption

expressible elements, whose original semantics can be faithfully preserved through transformation into Prolog; and imperfectly expressible elements, which cannot be faithfully represented due to Prolog's intrinsic limitations and thus require approximations. Table 1 summarizes the transformation strategies for both perfectly and imperfectly expressible elements in FOL, and simultaneously lists the limitations encountered when mapping the latter into Prolog.

4.1. Perfectly Expressible Elements

Equivalence, existential quantifiers, and rule queries are categorized as perfectly expressible elements.

Equivalence ($\Leftrightarrow$):

In Prolog, under the CWA, $P(X) \Leftrightarrow Q(X)$ can be translated as mutual implications:

$$P(X) \text{ :- } Q(X).\ Q(X) \text{ :- } P(X). \tag{1}$$

Specifically, this mutual implication expresses a logical equivalence between $P(X)$ and $Q(X)$, meaning that $P(X)$ is true if and only if $Q(X)$ is true. Due to the CWA in Prolog, propositions that cannot be proven are considered false, so there is no need to introduce additional explicit negation rules. The system will automatically infer negation from failure to prove. Therefore, this form of mutual implication fully captures the equivalence relation as expressed in FOL.

Existential quantifier ($\exists$):

Variables bound by $\exists$ are replaced by Skolem constants [33]. For a FOL formula of the form $\exists X, P(X)$, we apply Skolemization to introduce a new Skolem constant c, equivalently transforming it into a quantifier-free fact $P(c)$. The transformation preserves satisfiability, and the formal proof is provided in [33]. Thus, we do not repeat the proof here. The transformation can be expressed as:

$$\exists X, P(X) \Leftrightarrow P(c). \tag{2}$$

Rule queries ($? - \forall x\ B_1(x) \wedge B_2(x) \wedge \cdots \wedge B_n(x) \rightarrow H(x)$):

We introduce a fictitious constant e that does not appear in the original problem, split into fact $B_1(e) \wedge B_2(e) \wedge \cdots \wedge B_n(e)$ and query $? - H(e)$, with standardized variables. So we aim to prove the following transformation preserves satisfiability:

$$\begin{aligned} &? - \forall x\ B_1(x) \wedge B_2(x) \wedge \cdots \wedge B_n(x) \rightarrow H(x) \\ &\Leftrightarrow B_1(e) \wedge B_2(e) \wedge \cdots \wedge B_n(e), \quad ? - H(e). \end{aligned} \tag{3}$$

Proof. We now provide a proof from both the sufficiency and necessity perspectives.

- **Sufficiency:**
  Given:

$$\forall x\ B_1(x) \wedge B_2(x) \wedge \cdots \wedge B_n(x) \rightarrow H(x). \tag{4}$$

By Universal Instantiation, we obtain:

$$\begin{aligned} &\big(\forall x\ B_1(x) \wedge B_2(x) \wedge \cdots \wedge B_n(x) \rightarrow H(x)\big) \\ &\Rightarrow \big(B_1(a) \wedge B_2(a) \wedge \cdots \wedge B_n(a) \rightarrow H(a)\big). \end{aligned} \tag{5}$$

By Modus Ponens, from $B_1(a) \wedge B_2(a) \wedge \cdots \wedge B_n(a)$, we can derive $H(a)$.
Thus, after inserting $B_1(a) \wedge B_2(a) \wedge \cdots \wedge B_n(a)$ into the database, querying $? - H(a)$ will succeed. This completes the sufficiency proof.

- **Necessity:**
  (a) Since a is arbitrarily chosen and not specified by the original problem, a is an arbitrary constant.
  (b) a is not derived from any particular assumption.
  Assume that after inserting $B_1(a) \wedge B_2(a) \wedge \cdots \wedge B_n(a)$ into the database, the query $? - H(a)$ succeeds, i.e.,

$$B_1(a) \wedge B_2(a) \wedge \cdots \wedge B_n(a) \rightarrow H(a). \tag{6}$$

By Universal Generalization, we conclude:

$$\begin{aligned} &\big(B_1(a) \wedge B_2(a) \wedge \cdots \wedge B_n(a) \rightarrow H(a)\big) \\ &\Rightarrow \big(\forall x\ B_1(x) \wedge B_2(x) \wedge \cdots \wedge B_n(x) \rightarrow H(x)\big). \end{aligned} \tag{7}$$

Hence:

$$\forall x\ B_1(x) \wedge B_2(x) \wedge \cdots \wedge B_n(x) \rightarrow H(x). \tag{8}$$

$\square$

4.2. Imperfectly Expressible Elements

Negation, disjunction, and exclusive disjunction are categorized as imperfectly expressible elements.

Negation ($\neg$):

In FOL, $\neg$ is a fundamental operator used to express negation. It allows for the direct assertion of negated facts—e.g., $\neg animal(apple)$ denotes that *apple is not an animal* as a definitive truth in the model. Moreover, FOL also permits rules with negated conclusions, such as $haswings(X) \rightarrow \neg mammal(X)$, expressing that *if X has wings, then X is not a mammal*. In contrast, Prolog does not support either of these forms. First, negated facts cannot be directly represented. One cannot write a fact like \+$animal(apple)$. in the knowledge base. The operator \+ does not assert negation in the classical sense, but rather indicates that a proposition cannot be proven true under the current knowledge base—a mechanism known as negation as failure. Second, negation is not allowed in the head of a rule. A rule such as \+$animal(X)$:- $hasWings(X)$. is syntactically invalid in standard Prolog. Rule heads must be positive atomic formulas; \+ is only allowed in rule bodies.

Accordingly, we use $not\_P(X)$ to denote the negation of $P(X)$, whether it appears as a fact or as the head of a rule. However, $not\_P(X)$

$$\begin{tikzcd}
\text{not\_P(alice)} \arrow[d, "\text{not\_P(X) :- \textbackslash+ P(X).}"'] & \text{\textbackslash+ not\_P(alice)} \arrow[l] \\
\text{\textbackslash+ P(alice)} \arrow[r] & \text{P(alice)} \arrow[u, "\text{P(X) :- \textbackslash+ not\_P(X).}"']
\end{tikzcd}$$

Fig. 2. Illustration of infinite mutual recursion caused by negation approximation in Prolog. When neither $P(alice)$ nor $not\_P(alice)$ is explicitly defined, evaluating either predicate triggers a loop: $not\_P(alice)$ depends on $\backslash{+}P(alice)$, which in turn calls $P(alice)$, which again invokes $\backslash{+}not\_P(alice)$, and so on. This cycle arises because the two predicates are defined in terms of each other without a base case to terminate recursion.

is essentially a user-defined predicate and has no intrinsic logical connection to $P(X)$. To establish the relationship between the two, we explicitly define the following rules: $not\_P(X)$:- $\backslash + P(X)$. and $P(X)$:- $\backslash +$ $not\_P(X)$. This approximation performs reliably in certain scenarios. In particular, when the knowledge base explicitly contains either $P(X)$ or $not\_P(X)$ as a fact, it yields the expected behavior: one predicate succeeds while the other fails, thereby simulating classical negation to a reasonable extent.

However, this approach has a notable limitation: when both rules are defined without any supporting facts, querying either predicate (e.g., $not\_P(alice)$) results in infinite mutual recursion. As shown in Fig. 2, evaluating $not\_P(alice)$ triggers $\backslash{+}P(alice)$, which in turn requires evaluating $P(alice)$. That evaluation depends on $\backslash{+}not\_P(alice)$, returning the reasoning process to the original query. As a result, Prolog repeatedly alternates between the two predicates, entering a non-terminating loop rather than yielding a definitive truth value or failure.

Disjunction ($\vee$):

In FOL, $A(x) \vee B(x)$ denotes that at least one of the predicates holds, without requiring explicit knowledge of which one. In contrast, Prolog's disjunction (;) serves as a branching mechanism within rule bodies, as shown below:

$$C(X) \text{ :- } A(X); B(X). \tag{9}$$

In Prolog semantics, this means: "If $A(x)$ holds, then $C(x)$ holds; otherwise, if $B(x)$ holds, then $C(x)$ holds." This process is fully grounded on known facts, and cannot express the uncertainty captured by $A(x) \vee B(x)$ in FOL. Prolog adheres to the CWA, where unknown facts are assumed false. As a result, logical disjunctions like $A(x) \vee B(x)$ cannot be directly expressed as standalone facts or statements in Prolog.

Therefore, when Prolog's disjunction (;) cannot represent $\vee$ in FOL, we opt to relax $\vee$ into conjunction ($\wedge$). For example, $C \Rightarrow A \vee B$ is simplified as $C \Rightarrow A \wedge B$, and finally translated into A :- C, B :- C.

Exclusive disjunction ($\oplus$):

$\oplus$ captures the idea that "exactly one of two propositions holds", functioning as a mutual exclusivity constraint. However, standard Prolog, based on Horn clauses and unidirectional implications, lacks the ability to directly represent such mutually exclusive and exhaustive relationships. Therefore, in our translation step, $\oplus$ is approximated using mutually inverse rules as follows:

$$P(X) \leftarrow \neg Q(X).\ Q(X) \leftarrow \neg P(X). \tag{10}$$

Therefore, $P(X) \oplus Q(X)$ is translated into Prolog as a pair of mutually exclusive rules:

$$P(X) \text{ :- } \backslash + Q(X).\ Q(X) \text{ :- } \backslash + P(X). \tag{11}$$

This approximation captures the mutual exclusivity aspect of $\oplus$: if one predicate succeeds, the other must fail. In practice, when the knowledge base contains at least one of the two predicates as a fact, this encoding yields the expected behavior and allows Prolog to simulate exclusive disjunction to a reasonable extent.

However, it fails to enforce the exhaustiveness constraint – that exactly one of the propositions must hold. Under the CWA, if neither $P(X)$ nor $Q(X)$ is defined in the knowledge base, Prolog treats both as false, thereby violating the intended semantics of $P(X) \oplus Q(X)$.

5. LoRP: LLM-based Logical Reasoning via Prolog

In this section, we provide a detailed introduction to LoRP. As illustrated in Fig. 1, LoRP follows a systematic four-step reasoning process: **translation**, **validation**, **proving** and **majority voting**.

5.1. Translation Step

In the translation step, we build upon the expressiveness extensions to Prolog introduced in Section 4 to establish mapping rules between NL and Prolog logical structures. These rules are conveyed to the LLM through carefully designed prompts, which guide the model in performing accurate logical translation from NL to Prolog. To minimize common errors made by the translator during experiments, we summarized key logical structures and manually created corresponding translation examples to guide the LLM in recognizing logical patterns within the NL input, for example:

Example 5.1
NL: All rough things are kind.
Prolog: kind(X) :- rough(X).

Meanwhile, the translator processes sentences sequentially. If it detects the presence of existential quantifiers or rule queries in the problem, it translates them according to the aforementioned conversion mechanism, for example:

Example 5.2
NL:
  Premise: There is a strong man.
  …
  Conclusion: If something is big, then it is strong.
Prolog:
  strong(c).
  big(e).
  ?- strong(e)

5.2. Validation Step

The mapping from NL to Prolog may introduce syntactic and semantic errors, which can adversely affect the overall quality of reasoning. To address this, we introduce a syntax validator in LoRP. The syntax validator, also implemented by the LLM, identifies incomplete or erroneous Prolog statements and provides diagnostic feedback to pinpoint errors, allowing the translator to regenerate Prolog code when necessary. The syntax validation process is divided into two parts: one for query statements and the other for premise statements. Errors in a query trigger the regeneration of only the query, while errors in a premise trigger regeneration of only the premise, thereby avoiding unnecessary computational cost. Crucially, the syntax validator is designed to detect only **syntactic errors**, while ignoring **semantic errors**. Given the current limitations of LLMs in text comprehension and logical reasoning, self-validation of extracted logic would be both unreliable and a futile use of computational resources. Consequently, as illustrated in Fig. 1, despite the logical inaccuracies, the second round of translation passes validation due to its syntactic conformity to Prolog. Conversely, in the first round, the incomplete translation prompts the validator to identify the issue and initiate re-translation.

For the syntax validator, we limit the maximum number of retranslations to three. This means that if the Prolog code fails to pass the syntax validator three rounds consecutively, the third retranslated version is sent directly to the interpreter without further validation. This approach prevents the following issues. First, in some cases, the translator may receive error messages from the validator but fail to make meaningful changes, leading to repeated retranslations that are consistently blocked by the syntax validator. Second, the validator may sometimes imposes requirements that go beyond the standard Prolog syntax, resulting in valid Prolog code being incorrectly rejected.

5.3. *Proving Step*

In the proving step, the Prolog code that passes the validation process is provided to the SWI-Prolog interpreter to derive the result. However, during reasoning, the presence of redundant facts and rules may cause the interpreter to fall into infinite recursion or face other efficiency issues, preventing it from identifying the correct reasoning path, for example:

Example 5.3
eats(X, bear) :- needs(X, cow), sees(X, tiger).
sees(X, tiger) :- eats(X, bear).

If the two aforementioned rules are at the top of the list, Prolog prioritizes them: when attempting to validate *sees(tiger, tiger)*, it first tries *sees(tiger, tiger) :- eats(tiger, bear)*. This, in turn, requires validating *eats(tiger, bear)*, which triggers *sees(tiger, tiger)* again, resulting in an infinite loop. SWI-Prolog lacks a mechanism to detect or prevent such endless recursion; consequently, even if a valid reasoning path exists that does not involve recursion, SWI-Prolog is unable to find it once an infinite loop is entered. When SWI-Prolog encounters issues like infinite recursion, it may not generate an error but instead continue indefinitely. Thus, relying solely on whether the interpreter reports an error to determine if it is functioning correctly is inadequate. To address this limitation, we implemented a timeout mechanism for the interpreter, defining a maximum time limit for completing proofs. If this limit is exceeded, the proof attempt is also considered a failure by the interpreter.

Consequently, there are scenarios where SWI-Prolog fails to identify the correct reasoning path, even when one exists. To address this, we designed a supplementary reasoning mechanism: if SWI-Prolog encounters an error, the LLM is used to reason over the translated Prolog code to obtain the final answer. While LLM reasoning is not as rigorous or precise as the interpreter's, reasoning based on translated Prolog is clearly more effective than reasoning directly from NL. Therefore, the final mechanism for the proving step is illustrated in Fig. 3: when SWI-Prolog executes successfully, its result is deemed authoritative; in case of an error, the LLM takes over to reason through the Prolog code and provides the final output.

5.4. *Voting Step*

In the translation step, even a single-character error can lead to an incorrect final reasoning result. For example, miswriting “Scmidt” as “Schmidt” may cause the entire reasoning process to fail. Such issues are difficult to eliminate entirely, even with a subsequent validation step.

Therefore, in the voting step, we design a three-vote majority mechanism based on reasoning results. Specifically, for each input question, the model is prompted three times independently with the same instruction to generate three versions of Prolog code. After the proving step, three reasoning results are obtained. These results are then compared, and the most frequently occurring outcome is selected as the final prediction. This approach reduces the impact of minor semantic variations or translation errors on the final output, thereby improving the overall stability and reliability of the reasoning process. To enhance computational efficiency and reduce cost, we further introduce an early termination mechanism: if the reasoning results from the first two rounds are consistent, the third round is skipped, as the majority outcome can already be determined.

6. Experiment

In this section, we provide a detailed overview of our experimental setup, including descriptions of the four baseline approaches, the models employed, and the dataset used. Furthermore, we conduct a comprehensive analysis of the experimental results, covering accuracy, reasoning depth, recall rate, precision rate, and failure modes.

6.1. *Experimental Setup*

6.1.1. *LLMs*

We utilized GPT-3.5-turbo [34], GPT-4 [1], o1, DeepSeek-V3 [2], DeepSeek-R1 [35], Gemini 1.5 Pro [36] and LLaMA 3 [37] in our experiments, using a decoding temperature of $T = 0.5$ for all trials. **GPT-3.5-turbo**, an optimized version of GPT-3, balances efficiency and cost, making it well-suited for general NL processing tasks. In comparison, **GPT-4** demonstrates enhanced abilities in complex reasoning, conversational consistency, and multi-step problem solving, rendering it ideal for more demanding tasks. Recently, **o1**, a successor model with further improvements in reasoning, code generation, and tool-use integration, has been introduced, representing a significant step toward more capable and efficient LLMs. Additionally, **DeepSeek-V3** and **DeepSeek-R1** are newly released LLMs that emphasize advanced reasoning and mathematical problem-solving abilities, with R1 particularly optimized for high-fidelity instruction following and code understanding. The GPT and DeepSeek series were chosen due to their state-of-the-art capabilities among LLMs, particularly in text processing, reasoning, and related applications [2,35,38,39].

6.1.2. *Baselines*

We compare LoRP against four baselines: Naive, CoT, CoT-SC and LINC. These baselines were selected to represent a diverse range of reasoning methodologies, highlighting the distinctive strengths of LoRP: For the **Naive** baseline, the LLM is prompted to generate answers (“true” or “false”) directly in response to the input question. This baseline is chosen to serve as a reference for evaluating the simplest reasoning strategy, without any structured reasoning steps. Comparing LoRP to the Naive baseline helps illustrate the effectiveness of incorporating logical processes to enhance reasoning accuracy. For the **CoT** baseline, the LLM is required to perform step-by-step reasoning to provide justifications for its answers. This baseline is selected to assess the impact of

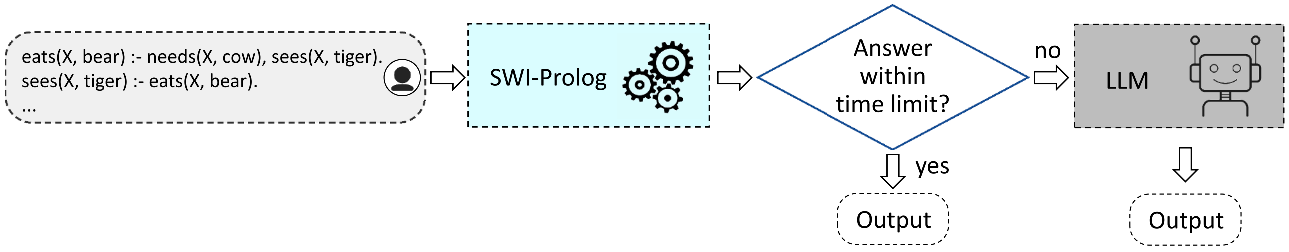

eats(X, bear) :- needs(X, cow), sees(X, tiger).
sees(X, tiger) :- eats(X, bear).
...
SWI-Prolog
Answer within time limit?
no
yes
LLM
Output
Output

Fig. 3. This figure illustrates the final proving step, incorporating the supplementary reasoning mechanism. Once the validated Prolog code reaches SWI-Prolog, a timer starts. If SWI-Prolog fails to produce an answer within the specified time limit, the supplementary reasoning mechanism is activated, and the LLM is invoked to perform the reasoning.

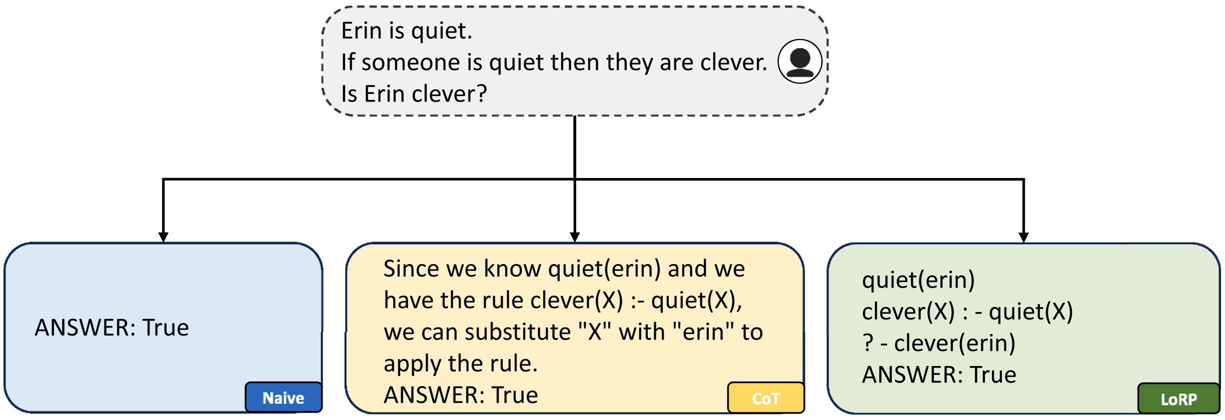

Erin is quiet.
If someone is quiet then they are clever.
Is Erin clever?
ANSWER: True
Naive
Since we know quiet(erin) and we have the rule clever(X) :- quiet(X), we can substitute "X" with "erin" to apply the rule.
ANSWER: True
CoT
quiet(erin)
clever(X) : - quiet(X)
? - clever(erin)
ANSWER: True
LoRP

Fig. 4. This figure illustrates how baseline methods and LoRP perform logical reasoning. It presents the reasoning process employed in CoT and the Prolog-based representation used in LoRP, demonstrating their distinct strategies for logical inference.

explicit reasoning traces on LLM performance. By comparing with CoT, we can demonstrate how LoRP's external logical interpreter can overcome the limitations of internally generated reasoning processes. For the **CoT-SC** baseline, we extend the CoT approach by generating three independent samples and selecting the most frequent answer as the final output, thereby reducing the inherent instability in the LLM's reasoning process. This baseline is chosen to explore whether stability improvements gained from Self-Consistency mechanisms could match or exceed the advantages provided by LoRP's integration of Prolog-based deductive reasoning. Fig. 4 illustrates the reasoning processes of the baselines and LoRP. CoT-SC is omitted for conciseness in the figure, as it merely adds a voting mechanism to CoT while retaining an essentially identical reasoning process. For the **LINC** baseline, we adopt the framework proposed by Olausson et al. [21]. LINC first parses NL premises and conclusion into FOL expressions using an LLM, and then invokes a symbolic prover—specifically, Prover9—to determine entailment. However, because LINC operates under the OWA, it returns "unknown" when the conclusion cannot be formally derived from the knowledge base. To ensure complete outputs and fair comparison with our closed-world system, we augment LINC with a fallback mechanism: when Prover9 returns "unknown", the same LLM is re-prompted to answer the original question using a CoT strategy.

6.1.3. *Datasets*

Our experiments utilize five established datasets: Proof-Writer [40], PARARULE-Plus [41], RuleTaker [42], PrOntoQA [43] and FOLIO [44]. **PARARULE-Plus** is an extended version of the original PARARULES dataset, which aims to evaluate the model's robustness in understanding and reasoning over linguistically diverse expressions of logic. For our evaluation, we selected the most challenging test set, characterized by a reasoning depth of five. Of the 300 samples in this test set, 84 were identified as containing errors, leaving 216 samples for the actual evaluation. **RuleTaker** assesses the model's ability to reason with NL rules, comprising five scenarios with reasoning depths of 0, 1, 2, 3, and 5. For our evaluation, we randomly selected 60 problems from each depth, resulting in a total of 300 problems used as the evaluation dataset. **PrOntoQA** is a dataset specifically designed to assess a model's ability to perform complex logical reasoning and multi-step deduction. For evaluation purposes, we used an evaluation set consisting of 500 problems. **ProofWriter** is designed to evaluate a model's ability to construct formal proofs and perform logical inference over natural language. The dataset provides both closed-world and open-world variants. As our framework is built on Prolog, which operates under the CWA and treats unprovable statements as false, the "unknown" labels in the open-world setting are semantically incompatible. We therefore restrict our evaluation to examples labeled as "true" or "false" in the closed-world version. To facilitate large-scale LLM experiments, we randomly sample 300 instances from the training set and exclude two mislabeled cases, resulting in 298 examples for evaluation. **FOLIO** is one of the largest benchmark datasets for FOL reasoning, covering a wide range of logical inference tasks, including simple fact derivation, complex quantification (existential and universal quantifiers), conditional reasoning, negation, exclusive disjunction (XOR), and compositional reasoning. Following [21], we use the validation set of FOLIO as the test set. After removing the erroneous examples identified in their work as well as additional errors found by us, a total of 127 examples remain for evaluation.

6.1.4. *Prompt Design*

For the prompts provided to the LLM, we manually set fewer than ten translation examples for the translator. The prompts for supplementary reasoning using the LLM were also manually configured, including fewer than three examples. These examples encompassed detailed reasoning processes and step-by-step justifications to guide the model effectively. The overall structure of the prompts remained consistent across all datasets, only a small portion of the examples were adapted to address dataset-specific error patterns. Therefore, we present a representative prompt in the Appendix, designed for the ProofWriter dataset, to illustrate the typical form and content.

6.2. *Results & Discussion*

6.2.1. *Accuracy*

The performance of LoRP and various baselines across five datasets is shown in Table 2.

For GPT-3.5-turbo, LoRP achieves the best performance on ProofWriter, RuleTaker, and PARARULE-Plus, although it is slightly outperformed by CoT-SC on PrOntoQA and FOLIO. This suggests that CoT-SC's Self-Consistency mechanism still offers advantages in handling certain complex reasoning tasks. Nevertheless, LoRP significantly outperforms Naive and vanilla CoT, validating the effectiveness of its external deductive reasoning framework in structured reasoning scenarios. In cases where LoRP underperforms compared to the baselines on certain datasets, the primary reason lies in the limited ability of GPT-3.5-turbo to accurately recognize logical structures in NL and translate them into correct Prolog code, resulting in lower translation quality. A detailed explanation will be provided in the following discussion.

For both GPT-4 and o1, all methods show substantial improvements over GPT-3.5-turbo, reflecting the benefits of enhanced model reasoning capabilities. Notably, LoRP consistently achieves the highest accuracy across all datasets, including near-perfect scores on benchmarks such as PrOntoQA and PARARULE-Plus. It significantly outperforms CoT-SC and maintains a clear margin over both Naive, CoT and LINC, underscoring the importance of structured reasoning even for advanced models. While o1 exhibits even stronger performance than GPT-4—particularly

Table 2
Comparison of LoRP and baseline models on different datasets with GPT-3.5-turbo, GPT-4, o1, Gemini 1.5 Pro, LLaMA 3, Deepseek-V3 and Deepseek-R1, each result represents the average of three independent evaluations. Since PARARULE-Plus contains multiple sub-questions for each main question, we calculate the percentage of correct answers based on the total number of sub-questions.

Model	Method \ Dataset	PrOntoQA	ProofWriter	RuleTaker	PARARULE-Plus	FOLIO
GPT-3.5-turbo	Naive	54.4	30.8	65.0	30.8	82.6
	CoT	74.2	59.7	67.3	67.5	83.5
	CoT-SC	**81.2**	58.1	70.3	69.8	**84.8**
	LINC	65.4	60.0	54.7	67.5	44.2
	LoRP	72.7	**65.7**	**83.1**	**89.1**	77.1
GPT-4	Naive	94.2	34.1	76.4	52.2	83.5
	CoT	98.6	57.8	79.6	81.5	83.7
	CoT-SC	99.2	58.1	79.3	82.9	**84.8**
	LINC	65.0	62.7	56.0	81.5	59.3
	LoRP	**100**	**90.9**	**94.9**	**99.8**	83.7
o1	Naive	99.8	86.6	83.0	90.8	84.8
	CoT	100	86.3	85.0	88.2	79.1
	CoT-SC	100	86.3	85.3	90.2	84.8
	LINC	73.2	60.3	63.6	88.2	51.6
	LoRP	**100**	**99.0**	**100**	**99.8**	**85.7**
DeepSeek-V3	Naive	90.6	42.7	86.3	66.2	82.5
	CoT	98.3	73.2	90.0	90.4	83.7
	CoT-SC	99.0	75.4	90.6	92.5	83.7
	LINC	73.4	61.2	60.7	90.4	43.9
	LoRP	**100**	**93.6**	**99.1**	**99.8**	**89.6**
DeepSeek-R1	Naive	95.6	86.0	82.6	95.8	82.5
	CoT	96.6	86.3	89.7	87.4	79.7
	CoT-SC	97.4	86.0	90.6	88.3	81.4
	LINC	77.6	56.3	63.3	87.4	50.4
	LoRP	**100**	**99.3**	**100**	**99.8**	**91.8**
Gemini 1.5 Pro	Naive	74.8	63.4	81.7	84.2	84.9
	CoT	99.5	84.8	82.0	90.8	82.5
	CoT-SC	99.6	85.0	83.1	92.3	82.5
	LINC	63.6	54.7	55.2	90.8	52.8
	LoRP	**100**	**90.5**	**98.3**	**99.8**	**87.2**
LLaMA 3	Naive	87.2	47.3	75.6	62.0	83.7
	CoT	100	66.5	76.1	83.7	80.4
	CoT-SC	100	66.7	76.5	86.8	81.2
	LINC	74.2	56.3	56.7	83.7	48.6
	LoRP	**100**	**87.3**	**87.7**	**99.8**	**84.6**

on complex reasoning tasks like ProofWriter and FOLIO—LoRP continues to deliver state-of-the-art results, confirming its strong generalizability and adaptability across different model architectures.

For DeepSeek-V3 and DeepSeek-R1, although the Naive and CoT methods provide reasonably strong baselines, integrating LoRP leads to substantial further improvements. LoRP achieves near-perfect or perfect accuracy on PrOntoQA and PARARULE-Plus, and performs strongly on ProofWriter and RuleTaker. On the challenging FOLIO dataset, LoRP achieves 89.6 % with DeepSeek-V3 and 91.8 % with DeepSeek-R1, significantly surpassing all other baselines. These results underscore LoRP's adaptability and superior symbolic reasoning capabilities on non-OpenAI models.

For both Gemini 1.5 Pro and LLaMA 3, LoRP consistently outperforms all baseline methods across all datasets. On Gemini 1.5 Pro, it achieves perfect accuracy on PrOntoQA and PARARULE-Plus, and yields notable improvements on more challenging datasets such as ProofWriter and FOLIO. For LLaMA 3, where baseline methods exhibit substantial limitations in symbolic reasoning, LoRP delivers consistent performance gains-most notably 87.3 % on ProofWriter and 99.8 % on PARARULE-Plus. These findings further validate LoRP's robustness and generalizability in enhancing logical reasoning across diverse LLM architectures, particularly beyond the OpenAI model family.

On the PARARULE-Plus dataset, every example contains at least one intermediate fact that is not explicitly stated, making it unprovable under LINC's OWA. As a result, LINC returns "unknown" for all queries. Consequently, its final accuracy on this dataset is entirely determined by the fallback CoT responses. This behavior is specific to PARARULE-Plus and does not occur in other datasets. Across the remaining datasets, LINC consistently underperforms compared to other baselines. Although it occasionally matches or slightly outperforms vanilla CoT under GPT-3.5-turbo, it often performs worse than even the Naive baseline. Crucially, its low accuracy is not due to frequent fallback usage-in most cases, the symbolic theorem prover does attempt a direct "true/false" judgment, but that judgment is incorrect. This exposes a core limitation in LINC's inference pipeline and highlights the advantage of LoRP's closed-world deductive reasoning in tasks requiring precise truth-value classification. Additionally, the performance gap can be partly attributed to differences in representational complexity. LINC relies on FOL, which demands high formal precision and abstraction, making it difficult for LLMs to generate correct logical forms from natural language. In contrast, LoRP employs Prolog-a logic programming language with a more concrete and syntactically constrained structure-enabling LLMs to produce accurate and executable logic more reliably.

Notably, LoRP underperforms the other three baselines—Naive, CoT, and CoT-SC—on the FOLIO dataset when using GPT-3.5-turbo, an anomaly not observed with any other model or dataset. Further analysis indicates that this issue arises from GPT-3.5-turbo's limited ability to handle complex logical quantifiers and its insufficient understanding of the FOL translation mechanism defined in our framework. For instance, the NL statement "There are three types of wild turkeys: Eastern wild turkey, Osceola wild turkey and Gould's wild turkey" can be expressed in FOL as: $\forall x(wildturkey(x) \rightarrow eastern(x) \vee osceola(x) \vee goulds(x))$.

According to our translation mechanism, the correct Prolog translation should be:

$$eastern(X) \text{ :- } wildturkey(X).$$
$$osceola(X) \text{ :- } wildturkey(X).$$
$$goulds(X) \text{ :- } wildturkey(X). \quad (12)$$

However, GPT-3.5-turbo frequently produces an incorrect Prolog translation such as:

$$wild\_turkey(X) \text{ :- } eastern(X); osceola(X); goulds(X). \quad (13)$$

The lack of understanding of the translation mechanism further degrades the already limited translation quality of GPT-3.5-turbo, ultimately resulting in LoRP performing worse than the three baselines on the FOLIO dataset.

In conclusion, LoRP demonstrates outstanding competitiveness and often leading performance across all datasets and models, especially in complex logical reasoning tasks. From GPT-3.5-turbo, GPT-4, and o1 to the DeepSeek series, as models' reasoning capabilities improve, LoRP's advantages become even more pronounced. At the same time, the persistent underperformance of Naive approaches further emphasizes the critical role of structured reasoning frameworks in logical reasoning. These results collectively validate the effectiveness and necessity of combining external deductive reasoning frameworks like LoRP with state-of-the-art LLMs to achieve robust and accurate logical reasoning, providing strong evidence for advancing reasoning-augmented LLM systems.

6.2.2. Reasoning Depth

Fig. 5 presents the accuracy across different reasoning depths for the RuleTaker dataset using GPT-3.5-turbo, GPT-4, o1, DeepSeek-V3 and DeepSeek-R1. The x-axis represents reasoning depth (0 to 5), and the y-axis represents accuracy. The five methods compared are Naive, CoT, CoT-SC, and LoRP. The shaded areas represent the half-width of the 95 % confidence intervals, indicating the stability in performance for each method.

For GPT-3.5-turbo, as shown in Fig. 5(a), the accuracy of all three baselines decreases as the depth of reasoning increases, particularly at higher reasoning depths. Although CoT and CoT-SC exhibit higher overall accuracy compared to LoRP, they display a marked decline in accuracy at deeper reasoning levels. Conversely, LoRP method maintains greater accuracy in deep reasoning tasks, particularly at depths 4 and 5, highlighting its capability to effectively handle multi-step reasoning problems. Due to the instability in translation quality generated by GPT-3.5-turbo, all methods exhibit relatively high variability across different reasoning depths.

As shown in Fig. 5(b) and (c), all three baseline methods exhibit a declining trend in accuracy as reasoning depth increases, both on GPT-4 and o1. Compared to GPT-3.5-turbo, GPT-4 leads to substantial improvements in both accuracy and stability across all methods, with performance differences becoming more pronounced at deeper reasoning levels. CoT and CoT-SC outperform the Naive method at moderate to high depths, while LoRP consistently achieves the highest accuracy across all depths, especially at depths 4 and 5, highlighting its advantage in handling complex reasoning. On o1, overall performance is further improved relative to GPT-4, particularly on challenging reasoning datasets such as ProofWriter and FOLIO. LoRP continues to outperform all baselines across reasoning depths, and shows a remarkable advantage at depth 0, achieving perfect accuracy (100 %) with a confidence interval of zero, indicating complete reliability.

For the DeepSeek series models, as shown in Fig. 5(d) and (e), the trend of decreasing accuracy with increasing reasoning depth remains evident, with DeepSeek-R1 performing slightly better than DeepSeek-V3. Compared to the baselines, LoRP once again demonstrates higher accuracy and greater stability, achieving perfect accuracy (100 %) at depth 0. Although the performance gap between LoRP and the baselines is less pronounced than in the previous three models, it still maintains a clear advantage.

Comparing GPT-3.5-turbo and GPT-4, the performance gap between CoT and CoT-SC is less pronounced in the latter, reflecting the limitations of Self-Consistency methods in improving logical reasoning for GPT-4. LoRP's performance on GPT-4 is significantly higher than on GPT-3.5-turbo, indicating that LoRP is influenced by the underlying performance of the language model and is capable of achieving better results with more advanced models. The reasons for this will be further analyzed in Section 6.3.

Comparing o1 with the GPT series and DeepSeek-R1 with DeepSeek-V3, it becomes evident that for models specifically designed for reasoning, such as o1 and DeepSeek-R1, the enhancement effects of CoT and CoT-SC on reasoning ability are significantly weaker compared to their impact on general-purpose models like the GPT series and DeepSeek-V3. In contrast, LoRP consistently improves the reasoning performance across all models, demonstrating strong adaptability and generalizability across different model architectures.

At the same time, LoRP demonstrates exceptional stability across all models except GPT-3.5-turbo. The variability observed in GPT-3.5-turbo may stem from its inherent limitations, such as inconsistencies in response generation and weaker reasoning capabilities. In contrast, more advanced models, including GPT-4, DeepSeek-V3, and DeepSeek-R1, provide a more stable foundation for LoRP, enabling it to maintain high accuracy and consistent performance across different reasoning depths. This further highlights LoRP's robustness and its potential to enhance logical reasoning in more advanced language models.

6.2.3. Recall and Precision

We evaluated LoRP's performance across the five datasets, focusing on recall and precision metrics. Given the instability of GPT-3.5-turbo and the limited significance of its results, we focused our analysis solely on GPT-4.

Table 3 presents the performance of GPT-4 on the five datasets—PrOntoQA, ProofWriter, RuleTaker, PARARULE-Plus and FOLIO—using recall and precision metrics. For PrOntoQA and PARARULE-Plus, the model achieves perfect recall and precision, indicating that it successfully predicted nearly all true positives without any false positives or false negatives. This suggests that GPT-4 is particularly effective on these datasets, likely due to the consistency and structure of the questions.

For ProofWriter, RuleTaker and FOLIO, recall and precision are slightly below 1, indicating that the model faced some challenges on these datasets, due to the increased complexity or ambiguity of the reasoning tasks involved. The relatively balanced recall and precision rates suggest that LoRP's reasoning and judgment abilities for positive and negative cases are well-matched. Although these metrics are slightly below optimal, they remain relatively high, reflecting LoRP's overall robust performance in logical reasoning tasks.

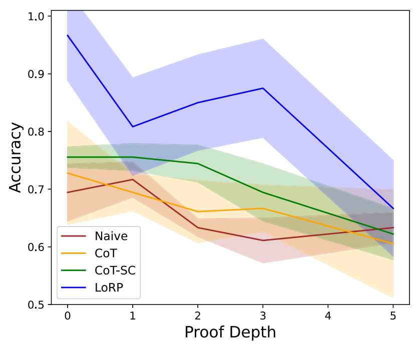
6.3. Failure Mode

In this subsection, we present and analyze the failure modes of LoRP and CoT. We identify the failure modes of CoT as C1 and C2, and those of LoRP as L1, L2 and L3. We begin by examining the failure modes of CoT:

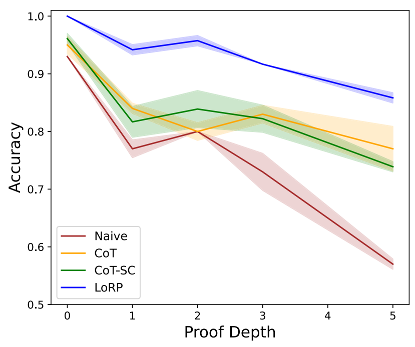
6.3.1. C1: CoT results in flawed logical reasoning

In the reasoning steps of CoT, logical errors frequently occur, such as swapping premises and conclusions or neglecting to apply negation. An example of this is as follows:

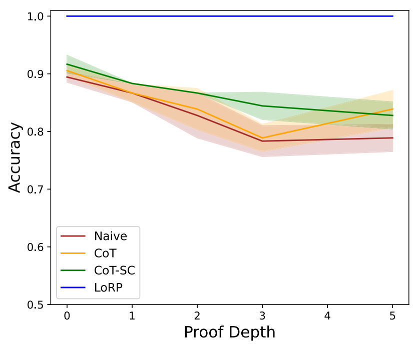
Example 6.1
It's said that 'All young, smart things are green', 'If Fiona was young, she would be green.' But we have no information indicating that Fiona is green. In these circumstances, we might infer that 'Fiona is not young' because she doesn't fulfill all the conditions required for a 'smart, young thing'.



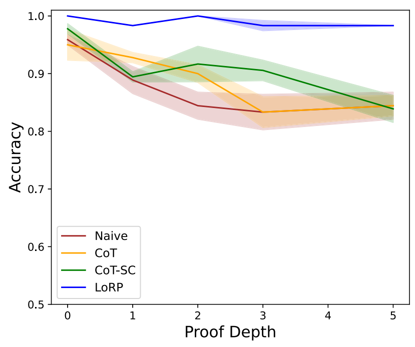
(a) GPT-3.5-turbo



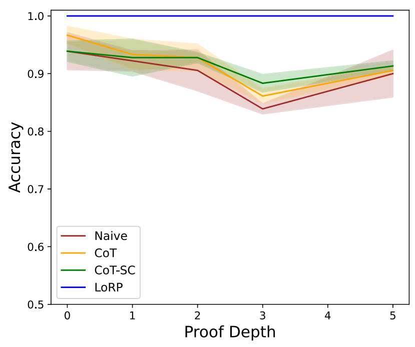
(b) GPT-4



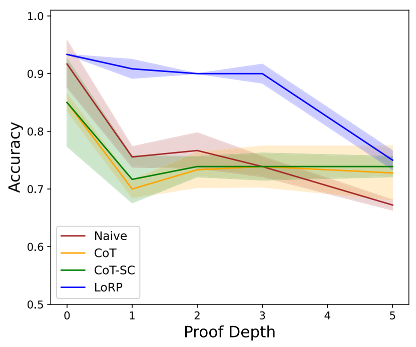
(c) o1



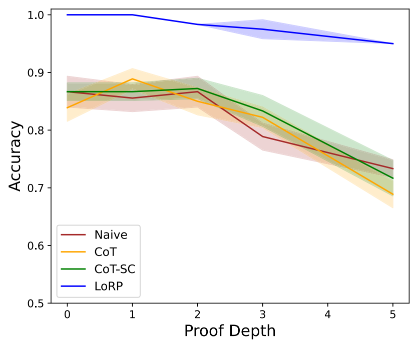
(d) DeepSeek-V3



(e) DeepSeek-R1



(f) LLaMA 3



(g) Gemini 1.5 Pro

Fig. 5. The accuracy of RuleTaker across different reasoning depths using various models, averaged over three runs, with shaded areas representing the 95 % confidence intervals. LINC is excluded from the figure due to its consistently low accuracy and large confidence intervals across all depths, which make it uninformative for analyzing the relationship between reasoning depth and model performance.

Table 3
Recall, and precision for LoRP based on GPT-4 across five datasets. LoRP achieves perfect scores (1.0) on PrOntoQA and PARARULE-Plus, demonstrating strong reliability. On ProofWriter, RuleTaker, and FOLIO, recall ranges from 0.86 to 0.93, and precision from 0.87 to 0.97, indicating high accuracy with minor variations across datasets.

	PrOntoQA	ProofWriter	RuleTaker	PARARULE-Plus	FOLIO
Recall	1	0.93	0.93	1	0.86
Precision	1	0.97	0.95	1	0.87

Table 4
Ablation study on the LoRP framework by selectively disabling individual components, including the validator, SWI-Prolog, and voting mechanism. Accuracy is reported on GPT-4 and GPT-3.5-turbo.

Configuration	Validator	SWI-Prolog	Voting mechanism	GPT-4 Acc. (%)	GPT-3.5-turbo Acc. (%)
Full model	✓	✓	✓	94.9	83.1
w/o Validator	✗	✓	✓	94.6	80.9
w/o SWI-Prolog	✓	✗	✓	88.3	79.7
w/o Voting Mechanism	✓	✓	✗	92.3	81.6

In this line of reasoning, CoT concludes that *Fiona is not young* based on the statement *All young, smart things are green* and the inability to prove that *Fiona is green*. This is a classic error, where from "$A \rightarrow B$" and "$\neg B$", it incorrectly concludes "$\neg A$".

6.3.2. C2: Failure to identify the correct reasoning path

Example 6.2
Premise: The mouse is big.
Premise: If someone is big, then they like the dog.
…
Conclusion: The mouse likes the dog.
CoT: …Since there is no direct rule or statement that can clarify this contradiction, based on the specific statements and rules given, we must assume the mouse does not like the dog. The final result is: False.

It is straightforward to conclude from *The mouse is big* and *If someone is big, then they like the dog* that *The mouse likes the dog*. However, CoT fails to identify this correct reasoning path, even after multiple attempts.

The following are the failure modes observed in LoRP:

6.3.3. L1: Semantic or logical translation errors

Example 6.3
NL: Young things are blue.
Prolog: young(X) :- blue(X).

Shown above is an example of one of the most common logical errors made by translators during the translation process: inverting cause and effect. In NL, "A leads to B" is incorrectly translated as "B leads to A." Such errors contribute to the disparity in LoRP's performance between GPT-4 and GPT-3.5-turbo. GPT-4 demonstrates a significantly better ability to accurately recognize and deconstruct NL, enabling it to produce more logically complete and accurate Prolog translations.

6.3.4. L2: Syntactic translation errors

Example 6.4
NL: Big things are furry.
Prolog: furry(X) :-

Although LoRP incorporates a syntax validator, there are still instances where the syntax validator fails to identify a syntax error or exceeds the maximum number of retranslations, particularly when the generated query is null or the statement is incomplete.

6.3.5. L3: Errors in the proving step

When the interpreter encounters issues, such as infinite recursion as previously mentioned, the LLM responsible for supplementary reasoning may still produce incorrect results. Although Prolog makes the previously implicit logic explicit to the LLM, it may still face challenges similar to those seen in CoT, albeit with a reduced likelihood. This is also a significant reason why LoRP's correctness is influenced by the performance of the LLM. The better the LLM performs as a supplementary reasoning module, the higher the overall correctness of LoRP.

Table 5
Ablation study of FOL-to-Prolog translation mechanism on GPT-4 and GPT-3.5-turbo.

FOL-to-Prolog translation mechanism	GPT-4 Acc. (%)	GPT-3.5-turbo Acc. (%)
✓	83.7	77.1
✗	65.1	62.3

6.4. Ablation Study

We perform ablation studies to evaluate the contribution of individual components in the LoRP. Specifically, we assess the roles of three key modules-Validator, SWI-Prolog executor, and Voting Mechanism-as well as the effect of the structured FOL-to-Prolog translation mechanism. These experiments allow us to quantify the importance of each part in supporting accurate and stable logical reasoning.

6.4.1. Contribution of LoRP Modules

Table 4 shows that removing the SWI-Prolog executor leads to the most significant drop in accuracy, −6.6 % on GPT-4 and −3.4 % on GPT-3.5-turbo, confirming the importance of symbolic reasoning. Disabling the voting mechanism also results in a performance decline, indicating its role in enhancing inference stability. Although the accuracy drop from removing the Validator is relatively small, it still demonstrates its utility in maintaining syntactic correctness.

6.4.2. Impact of FOL-to-prolog Translation Mechanism

As shown in Table 5, disabling this component results in a substantial accuracy decrease of 18.6 % on GPT-4 and 14.8 % on GPT-3.5-turbo. These findings highlight the critical importance of structured logical translation in enabling effective reasoning within the LoRP framework.

6.5. Efficiency Analysis of SWI-prolog Integration

We measure the average execution time of SWI-Prolog reasoning per query, excluding all cases involving the supplementary reasoning mechanism. Empirically, the Prolog engine responds with minimal latency—averaging just 0.02 milliseconds per query on our test set. Given that this cost is orders of magnitude lower than the latency of LLM inference, it introduces virtually no additional time overhead. This confirms that integrating an external symbolic reasoning module like SWI-Prolog is both computationally efficient and practically feasible.

7. Conclusion

In conclusion, this study has introduced LoRP, a novel framework that leverages Prolog to enhance the logical reasoning capabilities of LLMs. By integrating the formal deductive reasoning strengths of Prolog with the adaptability of LLMs, LoRP provides a robust solution to overcome the inherent limitations of NL-based reasoning, particularly in handling complex multi-step reasoning tasks. Our empirical evaluations demonstrate that LoRP excels in accuracy and reliability, particularly in tasks requiring deep logical reasoning, showing a clear advantage over the three compared reasoning strategies: Naive, CoT, and CoT-SC. Meanwhile, our FOL-to-Prolog translation mechanism has been validated on the FOLIO dataset, demonstrating its effectiveness and successfully extending Prolog's expressiveness in representing NL. Additionally, the error analysis provided insights into the unique failure modes of both CoT and LoRP, underscoring the importance of integrating external formal reasoning tools to enhance the logical rigor of LLMs. Future research can focus on improving Prolog and SWI-Prolog to enable the representation and derivation of more complex logical structures, thereby further enhancing the capabilities of LoRP. Furthermore, extending LoRP to other forms of logical formalism could broaden its applicability across diverse fields of artificial intelligence and complex decision-making systems.

CRediT authorship contribution statement

Zhengkun Di: Writing – original draft, Visualization, Software, Methodology; **Chaoli Zhang:** Writing – review & editing, Resources; **Hongtao Lv:** Writing – review & editing, Resources; **Lizhen Cui:** Writing – review & editing, Supervision, Conceptualization; **Lei Liu:** Writing – review & editing, Supervision, Conceptualization.

Data availability

Data will be made available on request.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported in part by China NSF grant No. 62220106004 and No. 62302267, in part by Zhejiang Provincial Natural Science Foundation of China under Grant No. LQN25F020020, in part by Zhejiang Jinhua Science and Technology Project No. 2024-4-019, in part by Shandong Provincial Natural Science Foundation No. ZR2023QF083, in part by Taishan Scholars Program.

Appendix

The following examples illustrate representative prompts used in LoRP. These are drawn from the ProofWriter dataset to demonstrate the typical form and structure used across experiments.

Prompts for the Translator
You are an expert in converting NL into Prolog
code. For each natural language sentence,
generate the corresponding Prolog rule according to its
logical semantics. Below are a few examples:
Rough things are quiet. ⇒ rough(X) :- quiet(X).
If someone is nice then they are rough.
⇒ rough(X) :- nice(X).
If someone is rough then they see the rabbit.
⇒ sees(X, rabbit) :- rough(X).
Green things are nice. ⇒ nice(X) :- green(X).
All rough things are kind. ⇒ kind(X) :- rough(X).
If Bob is cold then Bob is quiet.
⇒ quiet(bob) :- cold(bob).
Now, convert the following logical reasoning problem into
Prolog code. Important: The query must start with '?-'.

Prompts for the Interpreter
You are a Prolog reasoning interpreter. Follow
the structure of the example below to perform
step-by-step reasoning on the next query:
cold(charlie).
cold(dave).
cold(fiona).
cold(gary).
furry(dave).
furry(gary).
nice(fiona).
quiet(fiona).
green(X) :- furry(X).
green(X) :- quiet(X).
quiet(X) :- nice(X).
quiet(X) :- cold(X), green(X).
quiet(X) :- cold(X).
rough(X) :- furry(X).
furry(X) :- quiet(X), green(X).
big(charlie) :- furry(charlie).
nice(X) :- big(X), furry(X).
Query:
?- + nice(charlie).
Reasoning process:
1. Facts and rule definitions:
- Facts: cold(charlie), furry(charlie), etc.
- Rules: nice(X) is true under certain conditions, e.g.,
big(charlie) :- furry(charlie) and nice(X) :- big(X), furry(X).

2. Deriving nice(charlie):
- According to the rule big(charlie) :- furry(charlie),
since furry(charlie) is true, big(charlie) is also true.
- Then, based on nice(X) :- big(X), furry(X), since
both big(charlie) and furry(charlie) are true, nice(charlie) is true.

3. Query ?- + nice(charlie):
- + nice(charlie) means "cannot prove nice(charlie) is true."
- Since nice(charlie) has been proven true, + nice(charlie) is false.

Therefore, the result of ?- + nice(charlie) should
be false, as nice(charlie) has been derived as true.
Now, please determine the result based on the following
Prolog code and query, explain your reasoning step by step.
Ensure that the final line of your response is exactly
in the format "the result is: true" or "the result is:
false". Do not include any additional text after this line.

References

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F.L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al., GPT-4 technical report, arXiv preprint arXiv:2303.08774 (2023).

[2] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, et al., Deepseek-V3 technical report, arXiv preprint arXiv:2412.19437 (2024).

[3] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, et al., A survey on evaluation of large language models, ACM Trans. Intell. Syst. Technol. 15 (3) (2024) 1–45.

[4] K.S. Kalyan, A survey of GPT-3 family large language models including chatGPT and GPT-4, Natural Lang. Process. J. (2023) 100048.

[5] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, J. Gao, Large language models: a survey, arXiv preprint arXiv:2402.06196 (2024).

[6] M. Parmar, N. Patel, N. Varshney, M. Nakamura, M. Luo, S. Mashetty, A. Mitra, C. Baral, LogicBench: towards systematic evaluation of logical reasoning ability of large language models, in: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2024, pp. 13679–13707.

[7] T. Zheng, J. Bai, Y. Wang, T. Fang, Y. Guo, Y. Yim, Y. Song, CLR-Fact: evaluating the complex logical reasoning capability of large language models over factual knowledge, arXiv preprint arXiv:2407.20564 (2024).

[8] L. Pan, A. Albalak, X. Wang, W.Y. Wang, LOGIC-LM: empowering large language models with symbolic solvers for faithful logical reasoning, arXiv preprint arXiv:2305.12295 (2023).

[9] F. Xu, Q. Lin, J. Han, T. Zhao, J. Liu, E. Cambria, Are large language models really good logical reasoners? a comprehensive evaluation and beyond, arXiv e-prints (2023) arXiv–2306.

[10] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q.V. Le, D. Zhou, et al., Chain-of-Thought prompting elicits reasoning in large language models, Adv. Neural Inf. Process. Syst. 35 (2022) 24824–24837.

[11] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, D. Zhou, Self-consistency improves chain of thought reasoning in language models, arXiv preprint arXiv:2203.11171 (2022).

[12] P.M. Nugues, An Introduction to Prolog, Springer, 2006.

[13] T. Liu, W. Xu, W. Huang, X. Wang, J. Wang, H. Yang, J. Li, Logic-of-Thought: injecting logic into contexts for full reasoning in large language models, arXiv preprint arXiv:2409.17539 (2024).

[14] A. Kalyanpur, K. Saravanakumar, V. Barres, J. Chu-Carroll, D. Melville, D. Ferrucci, LLM-ARC: enhancing llms with an automated reasoning critic, arXiv preprint arXiv:2406.17663 (2024).

[15] X. Tan, H. Wang, X. Qiu, L. Cheng, Y. Cheng, W. Chu, Y. Xu, Y. Qi, STRUCT-X: enhancing the reasoning capabilities of large language models in structured data scenarios, in: Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1, 2025, pp. 2584–2595.

[16] O. Tafjord, B.D. Mishra, P. Clark, Entailer: answering questions with faithful and truthful chains of reasoning, arXiv preprint arXiv:2210.12217 (2022).

[17] S. Hao, Y. Gu, H. Ma, J.J. Hong, Z. Wang, D.Z. Wang, Z. Hu, Reasoning with language model is planning with world model, arXiv preprint arXiv:2305.14992 (2023).

[18] B. Lei, C. Liao, C. Ding, et al., Boosting logical reasoning in large language models through a new framework: the graph of thought, arXiv preprint arXiv:2308.08614 (2023).

[19] B. Li, H. Wang, X. Tan, Q. Li, J. Chen, X. Qiu, Adaptive heterogeneous graph reasoning for relational understanding in interconnected systems, J. Supercomput. 81(1) (2025) 112.

[20] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, X. Wu, Unifying large language models and knowledge graphs: a roadmap, IEEE Trans. Knowl. Data Eng. (2024).

[21] T.X. Olausson, A. Gu, B. Lipkin, C.E. Zhang, A. Solar-Lezama, J.B. Tenenbaum, R. Levy, LINC: a neurosymbolic approach for logical reasoning by combining language models with first-order logic provers, arXiv preprint arXiv:2310.15164 (2023).

[22] X. Yang, B. Chen, Y.-C. Tam, Arithmetic reasoning with LLM: prolog generation & permutation, arXiv preprint arXiv:2405.17893 (2024).

[23] P. Vakharia, A. Kufeldt, M. Meyers, I. Lane, L.H. Gilpin, Proslm: a prolog synergized language model for explainable domain specific knowledge based question answering, in: International Conference on Neural-Symbolic Learning and Reasoning, Springer, 2024, pp. 291–304.

[24] X. Tan, Y. Deng, X. Qiu, W. Xu, C. Qu, W. Chu, Y. Xu, Y. Qi, THOUGHT-LIKE-PRO: enhancing reasoning of large language models through self-driven prolog-based chain-of-thought, arXiv preprint arXiv:2407.14562 (2024).

[25] R. Reiter, On closed world data bases, in: Readings in Artificial Intelligence, Elsevier, 1981, pp. 119–140.

[26] N. Drummond, R. Shearer, The open world assumption, in: eSI Workshop: The Closed World of Databases Meets the Open World of the Semantic Web, volume 15, 2006, p. 1.

[27] P. Körner, M. Leuschel, J. Barbosa, V.S. Costa, V. Dahl, M.V. Hermenegildo, J.F. Morales, J. Wielemaker, D. Diaz, S. Abreu, et al., Fifty years of prolog and beyond, Theory Pract. Logic Program. 22(6) (2022) 776–858.

[28] D.S. Warren, Introduction to prolog, in: Prolog: The Next 50 Years, Springer, 2023, pp. 3–19.

[29] J. Wielemaker, An overview of the SWI-prolog programming environment, WLPE 3 (2003) 1–16.

[30] H.-D. Ebbinghaus, J. Flum, W. Thomas, A.S. Ferebee, Mathematical Logic, volume 2, Springer, 1994.

[31] A.N. Whitehead, B. Russell, 1913. Principia Mathematica, Cambridge 2 (1910) 1925–1927.

[32] G. Gentzen, Investigations into logical deduction, Am. Philos. Q. 1(4) (1964) 288–306.

[33] T. Skolem, Logisch-kombinatorische uńtersuchuńgeń uber die erfullbarkeit oder beweisbarkeit mathematischer satze nebst einem theoreme uber dichte mengen, Videnskąppselskąpets skrifter, I. Mątemątisk-nąturvedenskąbelig klasse (4) (1920) 252–263.

[34] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al., Training language models to follow instructions with human feedback, Adv. Neural Inf. Process. Syst. 35 (2022) 27730–27744.

[35] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, et al., [deepseek-r1]: incentivizing reasoning capability in llms via reinforcement learning, arXiv preprint arXiv:2501.12948 (2025).

[36] G. Team, P. Georgiev, V.I. Lei, R. Burnell, L. Bai, A. Gulati, G. Tanzer, D. Vincent, Z. Pan, S. Wang, et al., Gemini 1.5: unlocking multimodal understanding across millions of tokens of context, arXiv preprint arXiv:2403.05530 (2024).

[37] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, et al., The LLaMA 3 herd of models, arXiv preprint arXiv:2407.21783 (2024).

[38] A. Borji, M. Mohammadian, Battle of the wordsmiths: comparing chatgpt, gpt-4, claude, and bard, GPT-4, Claude, and Bard (June 12, 2023) (2023).

[39] J.W. Rae, S. Borgeaud, T. Cai, K. Millican, J. Hoffmann, F. Song, J. Aslanides, S. Henderson, R. Ring, S. Young, et al., Scaling language models: methods, analysis & insights from training gopher, arXiv preprint arXiv:2112.11446 (2021).

[40] O. Tafjord, B.D. Mishra, P. Clark, ProofWriter: generating implications, proofs, and abductive statements over natural language, arXiv preprint arXiv:2012.13048 (2020).

[41] Q. Bao, A.Y. Peng, T. Hartill, N. Tan, Z. Deng, M. Witbrock, J. Liu, Multi-Step deductive reasoning over natural language: an empirical study on out-of-distribution generalisation, arXiv preprint arXiv:2207.14000 (2022).

[42] P. Clark, O. Tafjord, K. Richardson, Transformers as soft reasoners over language, arXiv preprint arXiv:2002.05867 (2020).

[43] A. Saparov, H. He, Language models are greedy reasoners: a systematic formal analysis of chain-of-thought, arXiv preprint arXiv:2210.01240 (2022).

[44] S. Han, H. Schoelkopf, Y. Zhao, Z. Qi, M. Riddell, W. Zhou, J. Coady, D. Peng, Y. Qiao, L. Benson, et al., FOLIO: natural language reasoning with first-order logic, arXiv preprint arXiv:2209.00840 (2022).