Symbolic Working Memory Enhances Language Models for Complex Rule Application

Siyuan Wang[1], **Zhongyu Wei**[2], **Yejin Choi**[3,4], **Xiang Ren**[1]
[1]University of Southern California, [2]Fudan University,
[3]University of Washington, [4]Allen Institute for Artificial Intelligence
siyuanwang1997@gmail.com

Abstract

Large Language Models (LLMs) have shown remarkable reasoning performance but struggle with multi-step deductive reasoning involving a series of rule application steps, especially when rules are presented non-sequentially. Our preliminary analysis shows that while LLMs excel in single-step rule application, their performance drops significantly in multi-step scenarios due to the challenge in rule grounding. It requires anchoring the applicable rule and supporting facts at each step, amidst multiple input rules, facts, and inferred facts. To address this, we propose augmenting LLMs with external working memory and introduce a neurosymbolic framework for rule application. The memory stores facts and rules in both natural language and symbolic forms, enabling precise tracking. Utilizing this memory, our framework iteratively performs symbolic rule grounding and LLM-based rule implementation. The former matches predicates and variables of symbolic rules and facts to ground applicable rules at each step. Experiments indicate our framework's effectiveness in rule application and its robustness across various steps and settings [1].

[Sequential Input]
Facts: Nicole's grandfather, Harold, accompanied her to the basketball match. **(F1)**
Beverly went car shopping with her husband Louis and her daughter Nicole. **(F2)**
Harold bought a new dress for his daughter Marie. **(F3)**
Rules: If B is A's daughter, and C is B's grandfather, then C is the father of A. **(R1)**
If B is the father of A, and C is the daughter of B, then C is the sister of A. **(R2)**

[Non-Sequential Input]
Facts: Harold bought a new dress for his daughter Marie. **(F3)**
Nicole's grandfather, Harold, accompanied her to the basketball match. **(F1)**
Beverly went car shopping with her husband Louis and her daughter Nicole. **(F2)**
Rules: If B is A's father, and C is B's daughter, then C is the sister of A. **(R2)**
If B is A's daughter, and C is B's grandfather, then C is the father of A. **(R1)**

[Query] How is Marie related to Beverly?
[Rule Application Order]: R1→ (F2+F1) ⟹ F4; R2 → (F4+F3) ⟹ Answer

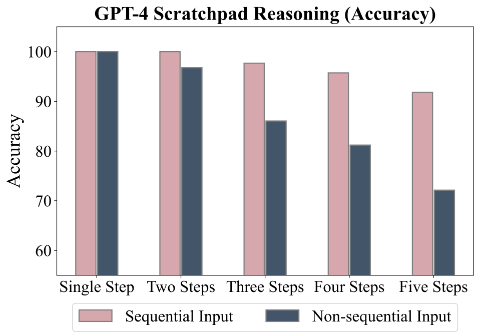


Figure 1: Performance of GPT-4 using scratchpad Chain-of-Thought (CoT) reasoning across various rule application steps on CLUTRR (Sinha et al., 2019), with an example of two-step rule application shown above.

1 Introduction

Large Language Models (LLMs) (OpenAI, 2023; Touvron et al., 2023; Team et al., 2023; Wei et al., 2022) have demonstrated impressive performance across diverse reasoning tasks. However, they still face challenges with multi-step deductive reasoning (Creswell et al., 2022; Ling et al., 2024; Lee and Hwang, 2024), where LLMs are provided with a set of facts and logical rules, and need to derive an answer to the query through a sequence of rule application steps. Specifically, each step of rule application requires applying a specific rule to its supporting facts to deduce new conclusions. Moreover, LLMs especially struggle when the surface patterns deviate from the sequential ordering of the rules (Chen et al., 2024; Berglund et al., 2023).

We conduct a preliminary analysis of LLM performance across various rule application steps, with rules sequentially and non-sequentially input in their application order. As shown in Figure 1, we observe three phenomena: (1) LLMs are effective at executing single-step rule application. (2) Their performance declines as the number of rule application steps increases. (3) Performance significantly worsens when rules are presented non-sequentially compared to sequentially, especially in long-term reasoning. Overall, LLMs excel in single-step rule application but face challenges in multi-step rule application, that requires tracking long-term facts and rules and determining appropriate rule and

[1] Code and data are available at https://github.com/SiyuanWangw/RuleApplication.

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 17583–17604
November 12-16, 2024 ©2024 Association for Computational Linguistics

facts for application at each step.

Each step of rule application typically consists of two processes: rule grounding and rule implementation. Rule grounding anchors the current applicable rule with supporting facts from the input, while rule implementation infers new facts based on the identified rule and facts. The before-mentioned challenges primarily arise from rule grounding using LLMs. Specifically, complex reasoning involves multiple input facts, rules, and intermediate inferred facts, making it difficult to accurately track long-term rule and facts (especially inferred ones) for each step using LLMs' internalized reasoning (Lanchantin et al., 2024). Additionally, as rules are often provided in a non-sequential order or include irrelevant ones, rule grounding requires referencing back and forth across all rules to identify the applicable one at each step, posing challenges for auto-regressive LLMs (Chen et al., 2024).

For precise tracking in multi-step rule application, we propose augmenting LLMs with an external working memory, inspired by humans' extensive use of memory for intelligence tasks (Hardman and Cowan, 2016). It explicitly stores an unlimited list of facts and rules, facilitating easy access during rule grounding, and the writing of new facts after intermediate rule implementation. Besides, it stores rules and facts in a non-ordered manner, minimizing the influence of the input order on LLMs reasoning. We implement this working memory to store rules and facts in both natural language and their symbolic forms (*i.e.*, in Prolog), thus supporting precise symbolic reference.

Building on working memory, we propose a neurosymbolic framework for rule application. This framework uses working memory for symbolic rule grounding and LLMs for rule implementation, leveraging LLMs' effectiveness in single-step rule application. This combination is more flexible than purely symbolic execution and more precise than fully LLM-driven methods. The workflow begins by writing all input facts and rules into working memory. It then proceeds with multiple steps of rule application, each involving symbolic rule grounding followed by LLM-based rule implementation. Specifically, symbolic rule grounding performs predicate and variable matching within the symbolic forms of facts and rules, checking for conflicts to determine the applicable rule with supporting facts. In rule implementation, LLMs infer new facts based on the grounded rule and facts, and the new inferred facts with their symbolic notations are written into the working memory. This cycle continues until the inferred facts solve the query or a maximum number of steps is reached.

We conduct experiment on four datasets involving multi-step rule application: CLUTRR and ProofWriter for logical reasoning, AR-LSAT for constraint satisfaction and Boxes for object state tracking. Results show that our framework outperforms CoT-based and symbolic-based baselines using GPT-4 and GPT-3.5, and exhibits robustness across various rule application steps and settings.

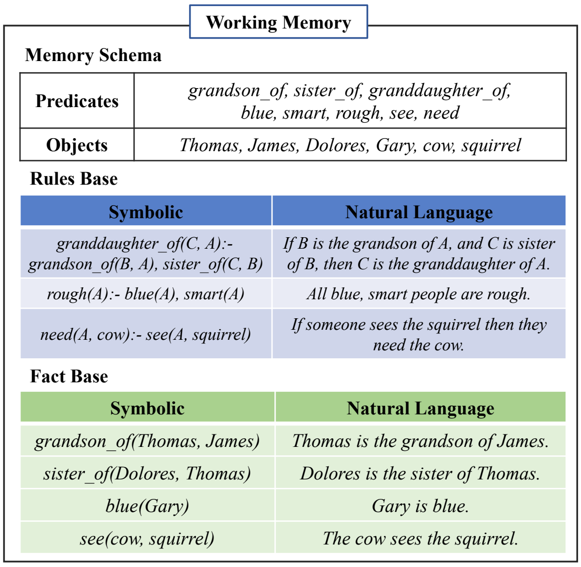
2 Preliminary

2.1 Problem Definition

We consider reasoning tasks involving deductive rule application in natural language, which take a context and a query as input. The context includes all necessary facts and rules for solving the query, though they may be non-sequentially provided in their application order and include irrelevant distractors. The model needs to apply specific rules to both the given and intermediate inferred facts to deduce new facts and ultimately output the answer.

2.2 External Working Memory

To enhance LLMs for precise long-term tracking in multi-step rule application, we introduce an external working memory to explicitly store rules and facts, as illustrated in Figure 2.



Working Memory

Memory Schema

Predicates	*grandson_of, sister_of, granddaughter_of, blue, smart, rough, see, need*
Objects	*Thomas, James, Dolores, Gary, cow, squirrel*

Rules Base

Symbolic	Natural Language
granddaughter_of(C, A):- grandson_of(B, A), sister_of(C, B)	*If B is the grandson of A, and C is sister of B, then C is the granddaughter of A.*
rough(A):- blue(A), smart(A)	*All blue, smart people are rough.*
need(A, cow):- see(A, squirrel)	*If someone sees the squirrel then they need the cow.*

Fact Base

Symbolic	Natural Language
grandson_of(Thomas, James)	*Thomas is the grandson of James.*
sister_of(Dolores, Thomas)	*Dolores is the sister of Thomas.*
blue(Gary)	*Gary is blue.*
see(cow, squirrel)	*The cow sees the squirrel.*

Figure 2: An illustration of the working memory.

Working Memory Composition The working memory consists of three components: a fact base, a rule base and a memory schema. The fact base stores a list of facts from the input context and intermediate reasoning, while the rule base saves a list of input rules. The facts and rules are stored in both natural language and their symbolic forms to support precise symbolic reference and verbalized utilization during multi-step rule application. The memory schema maintains a unified vocabulary of all involved predicates and objects in each instance, avoiding semantic duplication. For example, if "father_of" or "located_in" are in the schema, then "father-in-law_of" or "located_at" will not excluded. The symbolic facts and rules in the memory are constituted using these predicates and objects from the schema.

The working memory supports two operations: read and write. The read operation retrieves necessary facts and rules from the memory. The write operation involves adding new rules or facts to the memory, or updating existing facts. The decision to add or update facts depends on whether the context involves fact updating, such as an object's location changing over time. If new facts conflict with existing ones, updating occurs; otherwise, new facts are added. In contrast, for static information like the kinship relationship between individuals, new inferred facts will never conflict with existing ones, allowing them to be directly added.

Symbolic Formulation Facts and rules are symbolically represented using Prolog notations (Apt et al., 1997). Specifically, a fact is a predicate expression with several arguments, formatted as *predicate(arg1, arg2, ...)*, where *args* are specific objects. For example, the fact "*Dolores is the sister of Thomas.*" can be formulated as *"sister_of(Dolores, Thomas)*". A rule typically takes the form *conclusion:-premises*, interpreted as *If premises, then conclusion.* Both the conclusion and premises are composed of atomic facts, where *args* including both abstract variable symbols like *A, B, C* and specific objects. For example, "*If B is the grandson of A, and C is sister of B, then C is the granddaughter of A*" can be represented as *granddaughter_of(C, A):-grandson_of(B, A), sister_of(C, B)*. More examples are in Figure 2.

Memory Schema A key challenge in managing working memory is ensuring no duplication caused by different expressions conveying the same semantic meaning. This is essential for updating facts and identifying applicable rules based on supporting facts. To address this, we establish a memory schema for maintaining canonical predicates and objects. Symbolic facts and rules are formulated using predicates and objects from this schema.

The schema is dynamically constructed throughout the symbolic formulation process. Initially, the schema is empty. When formulating each fact or rule, the process first looks up whether the existing memory schema can accommodate the necessary predicates and objects to encode that piece of information. If it can, symbolic formulation is conducted directly based on the memory schema. If it cannot, new predicates or objects are created and added to the memory schema, and the symbolic formulation proceeds using these additions. The dynamic construction process of the memory schema can be viewed in Appendix A.

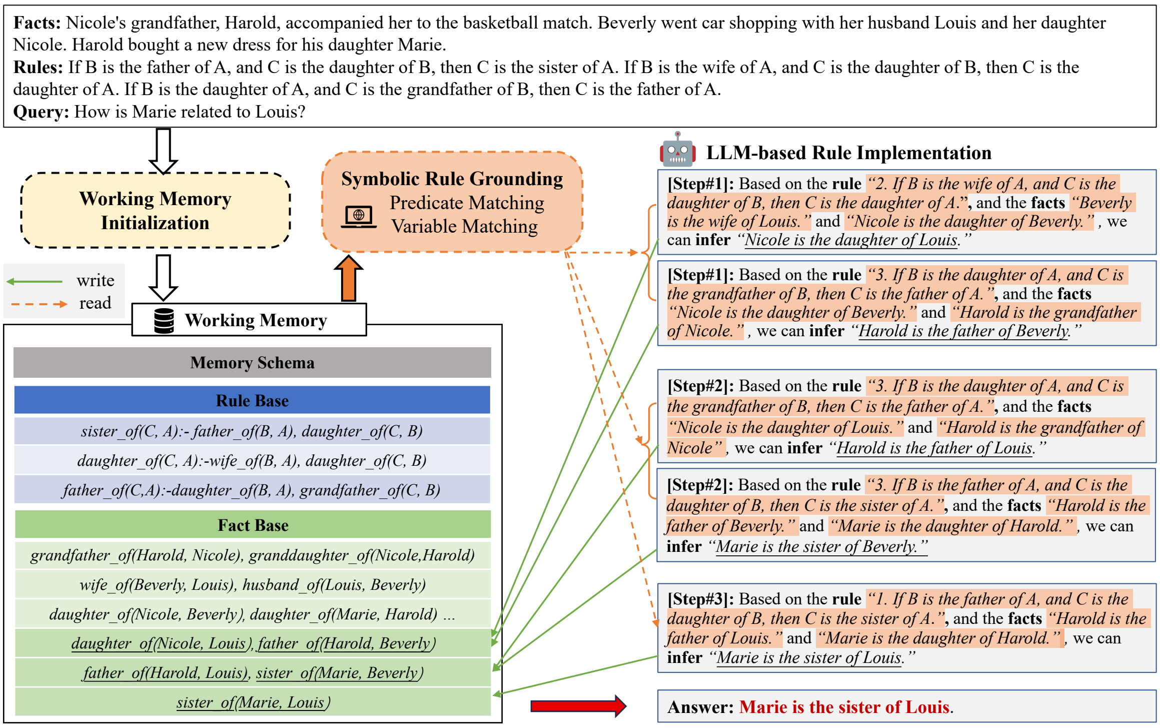
3 Framework

Complex reasoning often necessitates multi-step rule application amid non-sequential and irrelevant rules and fact. To address this, we propose a two-stage paradigm for each rule application step: rule grounding and rule implementation. Rule grounding anchors the applicable rules and supporting facts at each step. Rule implementation then infers new facts based on the grounded rules and facts.

Following this paradigm, we introduce a working memory-based neurosymbolic framework for rule application. It first initializes the working memory with all facts and rules from the input context. It then iteratively performs multi-step rule application, each step involving symbolic rule grounding based on symbolic formulations of facts and rules, followed by LLMs-based rule implementation. This process continues until the query is solved or a maximum number of steps is reached. The detailed workflow is shown in Figure 3.

3.1 Working Memory Initialization

To comprehensively initialize the working memory from the input context, we first decompose the context into multiple sentences. Then we prompt LLMs to list existing facts and rules for each sentence within the context. This involves extracting the natural language expressions and simultaneously parsing their symbolic formulations based on the memory schema. Both the natural language and symbolic representations of all facts and rules are then written into the working memory. Any new predicates and objects beyond the memory schema are also incorporated into the working memory. The detailed prompt can be found in Appendix D.



Facts: Nicole's grandfather, Harold, accompanied her to the basketball match. Beverly went car shopping with her husband Louis and her daughter Nicole. Harold bought a new dress for his daughter Marie.
Rules: If B is the father of A, and C is the daughter of B, then C is the sister of A. If B is the wife of A, and C is the daughter of B, then C is the daughter of A. If B is the daughter of A, and C is the grandfather of B, then C is the father of A.
Query: How is Marie related to Louis?

Working Memory Initialization
Symbolic Rule Grounding
Predicate Matching
Variable Matching
write
read
Working Memory
Memory Schema
Rule Base
sister_of(C, A):- father_of(B, A), daughter_of(C, B)
daughter_of(C, A):-wife_of(B, A), daughter_of(C, B)
father_of(C,A):-daughter_of(B, A), grandfather_of(C, B)
Fact Base
grandfather_of(Harold, Nicole), granddaughter_of(Nicole,Harold)
wife_of(Beverly, Louis), husband_of(Louis, Beverly)
daughter_of(Nicole, Beverly), daughter_of(Marie, Harold) ...
daughter_of(Nicole, Louis), father_of(Harold, Beverly)
father_of(Harold, Louis), sister_of(Marie, Beverly)
sister_of(Marie, Louis)
LLM-based Rule Implementation

[Step#1]: Based on the **rule** *"2. If B is the wife of A, and C is the daughter of B, then C is the daughter of A."*, and the **facts** *"Beverly is the wife of Louis."* and *"Nicole is the daughter of Beverly."* , we can **infer** *"Nicole is the daughter of Louis."*

[Step#1]: Based on the **rule** *"3. If B is the daughter of A, and C is the grandfather of B, then C is the father of A."*, and the **facts** *"Nicole is the daughter of Beverly."* and *"Harold is the grandfather of Nicole."* , we can **infer** *"Harold is the father of Beverly."*

[Step#2]: Based on the **rule** *"3. If B is the daughter of A, and C is the grandfather of B, then C is the father of A."*, and the **facts** *"Nicole is the daughter of Louis."* and *"Harold is the grandfather of Nicole"*, we can **infer** *"Harold is the father of Louis."*

[Step#2]: Based on the **rule** *"3. If B is the father of A, and C is the daughter of B, then C is the sister of A."*, and the **facts** *"Harold is the father of Beverly."* and *"Marie is the daughter of Harold."* , we can **infer** *"Marie is the sister of Beverly."*

[Step#3]: Based on the **rule** *"1. If B is the father of A, and C is the daughter of B, then C is the sister of A."*, and the **facts** *"Harold is the father of Louis."* and *"Marie is the daughter of Harold."*, we can **infer** *"Marie is the sister of Louis."*

Answer: **Marie is the sister of Louis.**

Figure 3: The workflow of our neurosymbolic rule application framework based on working memory. Details of the memory schema and natural language expressions of facts and rules are omitted in the memory for simplicity.

3.2 Symbolic Rule Grounding

At each step of rule application, we first ground the current applicable rules and corresponding supporting facts from the working memory. We adopt a symbolic predicate and variable matching strategy between facts and rules for precise grounding.

- **Predicate Matching** checks if the predicates of selected facts match those of the rule's premises. This exact string matching can be further relaxed using approximate string or model-based semantic matching to accommodate parsing inconsistencies for more flexible grounding.
- **Variable Matching** verifies whether the arguments of facts can instantiate the variables in rule premises without conflicts (*i.e.*, each variable is instantiated by the same argument), or can match the objects in rule premises.

Detailed examples are illustrated in Figure 4. We observe that the predicates of facts *F1* and *F2* do not match with rule *R*, while the arguments of *F2* and *F4* cannot instantiate the variable *B* in rule *R*. After this symbolic rule grounding, rule *R* is applicable to its supporting facts *F2* and *F3*.

Specifically, we adopt different rule grounding approaches for various tasks types. For tasks like logical reasoning, where **facts have no inherent chronological order and a single fact never involves updating**, we adopt exhaustive enumeration for rule grounding. We enumerate all combinations of facts for each rule according to the number of premise facts, and check all rules. We perform both predicate and variable matching, deeming a rule applicable if no conflicts arise with the corresponding facts. Notably, each set of supporting facts for the current step's applicable rules must include the newly inferred facts from the previous round to avoid repeating rule implementation. For particular constraint satisfaction tasks where all rules need to be satisfied with diverse constraint predicates, we only conduct variable matching to rank the most applicable rule at each step.

R: brother_of(C, A) :- sister_of(B, A), brother_of(C, B)
F1: **grandson_of**(John, James) **F2:** sister_of(Mary, John) **predicate unmatched**
F2: sister_of(Mary, John) **F3:** brother_of(James, Mary) **predicate matched**

R: brother_of(C, A) :- sister_of(**B**, A), brother_of(C, **B**)
F2: sister_of(Mary, John) **F3:** brother_of(James, Mary) **variable matched**
F2: sister_of(**Mary**, John) **F4:** brother_of(Clarence, **Timmy**) **variable unmatched**

Figure 4: Examples of predicate and variable matching.

For tasks like object state tracking, where **facts follow an inherent sequential order due to temporal operations**, causing single state facts to update over time, we perform rule grounding according to the chronological order of given operations. For the operational fact at each step, we identify the most applicable rule and relevant state facts based on both predicate matching and variable matching.

Most reasoning tasks that involve rule application can be categorized into two main types: static reasoning and dynamic operational decision-making. These tasks can be approached using above two rule grounding strategies: exhaustive enumeration and chronological grounding.

3.3 LLM-based Rule Implementation

LLMs are effective at single-step rule application. After symbolic rule grounding that identifies the applicable rules and corresponding supporting facts from the working memory at each step, we leverage LLMs to implement all applicable rules in parallel. Specifically, we input each rule with its supporting facts and prompt LLMs to infer possible new facts in both natural language and symbolic formulations. The inferred facts are then written into the working memory accordingly. During each step of rule implementation, we also determine whether newly inferred fact solves the query (for logical reasoning) or check for rule-facts conflicts (for constraint satisfaction).

Final Answer Prediction If a new fact resolves the query, the iteration ends and we utilize that fact for the final answer. For multi-choice constraint satisfaction, we select the option without conflict as the final answer (or reversely taking the option with conflict for negative questions). For object state tracking where iteration ends only after all operations, the query can be directly answered by looking up the query object's state from the working memory. If all inferred facts in each step cannot solve the query, the process will proceed to the next iteration. The cycle continues until the query is resolved or a maximum step count is reached. If the query remains unsolved, we employ a backup CoT method to output the final answer. Detailed prompts are provided in Appendix D.

4 Experiments

4.1 Setup

Datasets We conduct experiments on four reasoning datasets that involve multi-step of deductive rule application, including CLUTRR (Sinha et al., 2019), ProofWriter (Tafjord et al., 2020), AR-LSAT (Zhong et al., 2021) and Boxes (Kim and Schuster, 2023), detailed as follows:

- **CLUTRR and ProofWriter** are two logical reasoning datasets, involving the application of commonsense and predefined logical rules. For CLUTRR, we select 235 test instances requiring 2-6 steps of rule application. For ProofWriter, we select instances necessitating 3-5 of reasoning steps from the open-world assumption subset, totaling 300 instances with balanced labels.
- **AR-LSAT** is a constraint satisfaction dataset sourced from the Law School Admission Test, and requires applying all conditional rules to find satisfactory solutions. Since multiple instances in the original dataset share the same context, which may deviate the evaluation, we select all instances with unique contexts from both the development and test sets, resulting in 80 examples for our evaluation.
- **Boxes** requires reasoning about objects' states after multiple operations, where apply inferential rules for these operations can enhance reasoning. We collect all 135 instances involving 6-7 operations to ensure evaluation difficulty. As rules are not provided, we manually curate the corresponding rule for each operation.

Baseline We compare our framework with two types of baselines: CoT-based methods and symbolic-based methods. The CoT-based methods include: (1) Scratchpad-CoT (Nye et al., 2021; Wei et al., 2022) performs chain-of-thought reasoning in a scratchpad manner after the entire input; (2) Self-Consistency CoT (SC-CoT) (Wang et al., 2022b) samples three reasoning paths and takes the majority vote as the final predication. Specifically, we shuffle input order for the first three tasks and adopt different temperatures (*i.e.*, 0, 0.5, 1.0) for the last task for sampling; (3) Self-Notes (Lanchantin et al., 2024) prompts the model to generate multiple internal reasoning notes interleaving with the input. The symbolic-based methods include: (4) Logic-LM (Pan et al., 2023) utilizes LLMs to parses natural language problems into symbolic formulations and then performs deterministic inference with symbolic solvers, like Z3 theorem prover (De Moura and Bjørner, 2008); and (5) SymbCoT (Xu et al., 2024) fully utilizes LLMs to parse language facts and rules into symbolic expressions and solve problems step-by-step by CoT.

Our working memory-based neurosymbolic

[2] The results we report of Logic-LM on ProofWriter are lower than the performance stated in its paper. This is because we re-implement it on our sampled subset (reasoning depths 3-5), which is more challenging than the original *depth-5* subset that actually includes reasoning depths from 0 to 5.

Method	CLUTRR		ProofWriter		AR-LSAT		Boxes	
	GPT-4	GPT-3.5	GPT-4	GPT-3.5	GPT-4	GPT-3.5	GPT-4	GPT-3.5
CoT-base Methods								
Scratchpad-CoT	83.83%	57.02%	61.33%	49.67%	41.25%	30.00%	91.85%	15.60%
SC-CoT	85.53%	59.57%	62.00%	54.00%	45.00%	31.25%	93.33%	17.04%
Self-Notes	74.04%	55.74%	62.00%	52.67%	47.50%	23.75%	92.59%	18.52%
Symbolic-based Methods								
Logic-LM	/	/	62.33%	52.00%	50.00%	31.25%	/	/
SymbCoT	/	/	65.67%	51.33%	60.00%	21.25%	/	/
WM-Neurosymbolic	**92.34%**	**78.72%**	**77.33%**	**58.00%**	**70.00%**	**35.00%**	**100%**	**34.29%**

Table 1: Experimental results (accuracy %) of different methods using GPT-4 and GPT-3.5-turbo[2].

framework, WM-Neurosymbolic, is implemented based on two different backbone LLMs: GPT-4 (gpt-4-turbo-0409 for CLUTRR, ProofWriter and Boxes, gpt-4o for AR-LSAT) and GPT-3.5 (gpt-3.5-turbo-0125). This enables evaluation of its effectiveness with various abilities of symbolic semantic parsing and one-step rule application. We adopt one-shot prompting strategy for CoT-based baselines, while symbolic-based methods, which require better output format control in sub-procedures, use few-shot prompts with multiple examples. Similarly, WM-Neurosymbolic employs few-shot prompts, but we try to ensure all examples in each prompt belong to a single instance for a fair comparison. We also provide comparisons with multi-shot CoT-based methods in Appendix C.1, according to the maximum number of examples used by our framework in each dataset. More implementation details are available in Appendix B.

4.2 Overall Performance

The overall results are presented in Table 1. For symbolic-based methods, which may fail to return an answer caused by symbolic formulation errors, we use Scratchpad-CoT as a backup. We have the following observations:

(1) Our method significantly outperforms all baselines across all datasets, including the extremely challenging AR-LSAT dataset, demonstrating the superiority of our working memory-based neurosymbolic framework.

(2) Our framework is effective on top of different LLM backbones with varying abilities in symbolic parsing and one-step rule application. Specifically, GPT-3.5-based framework shows significant improvement on formally expressed problems (CLUTRR, Boxes) while GPT-4 excels at more naturalistic problems (ProofWriter, AR-LSAT). This suggests our framework are more effective as backbone LLMs advance.

(3) Compared to previous symbolic-based methods that perform both rule grounding and implementation either symbolically or by LLMs, our framework exhibits improvement, demonstrating flexibility and robustness by disentangling rule grounding and implementation, respectively symbolically and through LLMs.

4.3 Ablation Study

To investigate the effectiveness of different stages in our framework, we conduct an ablation study taking GPT-4 as the backbone on the CLUTRR and ProofWriter datasets[3]. We substitute decomposed-based memory initialization with scratchpad-CoT initialization, symbolic rule grounding with LLM-based grounding, and LLM-based rule implementation with symbolic implementation, respectively. Scratchpad-CoT initialization involves formulating all facts and rules within the entire context at once via scratchpad-CoT. LLM-based grounding prompts LLMs to iteratively determine the applicable rules with associated facts at each steps (similar to SELECTION-INFERENCE method (Creswell et al., 2022)). Symbolic implementation is a deterministic process defined by ourselves.

As shown in Table 2, all substitutions lead to significant performance drops, underscoring the effectiveness of our framework design. Compared to scratchpad-CoT initialization, the decomposed-based strategy simplifies fact and rule formulation by breaking down the context into individual sentences, achieving more comprehensive initialization and improved reasoning. LLM-based rule grounding even performs worse than the base-

[3] To save computational costs, we select instances from ProofWriter that require 5 reasoning steps for analysis.

Method	CLUTRR	ProofWriter
WM-Neurosymbolic	92.34%	74.67%
→ Scratchpad Initialization	86.81%	66.67%
→ LLM-based Grounding	82.98%	73.33%
→ Symbolic Implementation	90.64%	52.00%
Scratchpad-CoT	83.83%	53.33%

Table 2: Ablation study based on GPT-4. The arrows denote the replacement of corresponding stages in our framework with specified components.

line on CLUTRR, revealing LLMs' deficiency in determining rule application order and tracking long-term facts in multi-step reasoning. However, it shows only a slight drop on ProofWriter, because its reasoning involves a single object, reducing complexity for LLMs. Symbolic implementation causes a greater decline in ProofWriter than in CLUTRR, indicating that advanced LLMs are more robust at one-step rule application for more naturalistic, complex problems than symbolic solvers.

4.4 Effectiveness on Open-source LLMs

To showcase the effectiveness of our framework using affordable open-source LLMs, we implement it on Llama-3-8B-Instruct and compare the results with LLama-based CoT baselines on the CLUTRR and ProofWriter datasets. As shown in Table 3, our framework exhibits robust effectiveness on both closed-source and open-source models.

Method	CLUTRR LLama3-8B	ProofWriter LLama3-8B
Scratchpad-CoT	52.77%	50.33%
SC-CoT	54.47%	53.67%
Self-Notes	51.49%	52.33%
WM-Neurosymbolic	63.40%	58.67%

Table 3: Result on LLama-3-8B-Instruct.

5 Further Analysis

5.1 Varying Rule Application Steps

To evaluate the effectiveness of our framework across different steps of rule application, we report the performance of various GPT-4-based methods on the CLUTRR and ProofWriter datasets, which involves 2-6 steps and 3-5 steps. As shown in Figure 5, our framework consistently performs the best across all steps. As problem complexity increases with more steps, our advantage remains significant. Moreover, Self-Consistency CoT outperforms the baseline CoT on fewer steps, but this advantage diminishes with more steps due to the increased likelihood of generating discrepancies. This can be mitigated by executing more sampling.

5.2 Different Rule Settings

In real-world questions, rules are presented in various ways as follows. (1) Ordered Rules: rules are arranged in their application order. (2) Shuffled Rules: rules are provided in a random order. (3) Noisy Rules: rules are shuffled and include irrelevant ones. This setup closely aligns with real-world retrieved-based scenarios where logical rules are retrieved from external sources and may contain distractors. We discuss these three rules settings using the CLUTRR dataset (focusing on 5-6 rule application steps) and compare our framework to CoT-based baselines on GPT-4. Since self-consistency CoT involves shuffling input order, we do not report its performance. For noisy rules, we manually add two irrelevant rules to distract each instance.

Rule Settings	Ordered	Shuffled	Noisy
Scratchpad-CoT	66%	64%	58%
Self-Notes	68%	54%	50%
WM-Neurosymbolic	74%	74%	76%

Table 4: Performance on different rule settings.

Table 4 shows that CoT-based baselines are susceptible to perturbations from rule order and noise, especially the Self-Notes method. In contrast, our framework exhibits robust effectiveness across all rule settings, even with noisy distractors. Notably, our framework outperforms CoT-based baselines even in the ordered rule setting, underscoring its enhanced ability to precisely track facts at each step and iteratively perform multi-step rule application. Moreover, we implement our framework without rules provided in Appendix C.2 to simulate some realistic scenarios where rules are typically well-established commonsense principles derived from real-world observations but not explicitly input.

5.3 Symbolic Investigation

Symbolic-based methods inevitably lead to execution failures due to syntax or semantic errors during symbolic formulation, even performed by an LLM parser. To mitigate this, our framework decouples the symbolic rule application process into executing rule grounding symbolically and rule imple-

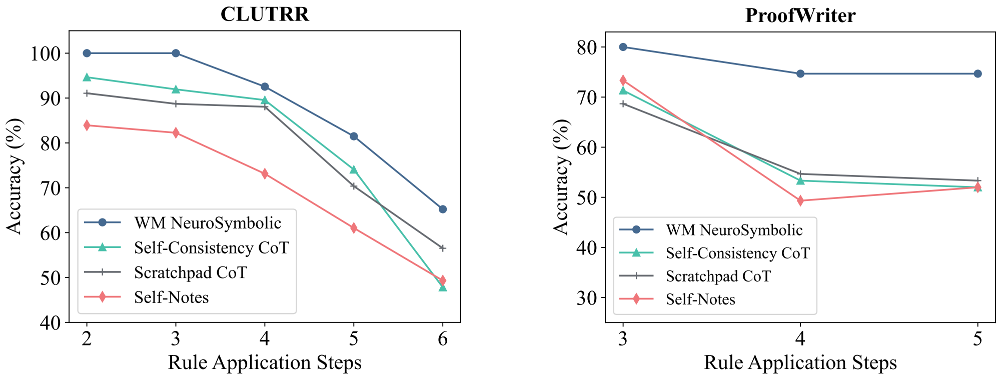


Figure 5: Performance across varying steps of rule application.

mentation based on LLMs. To illustrate our framework's flexibility and efficacy, we report its execution success rate and accuracy across all datasets. Specifically, the execution rate denotes the proportion of instances that can be directly solved by our neurosymbolic framework without backup, and accuracy is calculated for these executable instances.

Executable	GPT-4		GPT-3.5	
Statistics	Rate	Accuracy	Rate	Accuracy
CLUTRR	68.94%	100.00%	57.02%	97.76%
ProofWriter	67.00%	85.57%	67.67%	85.22%
AR-LSAT	56.25%	93.33%	12.50%	70.00%
Boxes	100.00%	100.00%	100.00%	34.29%

Table 5: Execution rate and accuracy statistics for our framework based on GPT-4 and GPT-3.5.

As depicted in Table 5, our framework successfully executes over 50% of instances for all datasets on both GPT-4 and GPT-3.5, except for the complex AR-LSAT dataset on GPT-3.5. Additionally, it achieves high accuracy on executable instances. In contrast, Logic-LM executes fewer than 10% of ProofWriter instances, with 33.75% and 8.75% of AR-LSAT instances executable based on GPT-4 and GPT-3.5, respectively.[4] This demonstrates the flexibility of our rule application framework, combining matching-based grounding with LLM-based implementation for a softer symbolic approach. While SymbCoT achieves 100% execution success, it shows limited accuracy, highlighting the precision of our framework by symbolic grounding.

5.4 Error Analysis

We further analyze the cases where our framework incorrectly answers and summarize the major error types. (1) Incomplete and inaccurate initialization of the working memory. This primarily occurs when each sentence describes multiple facts or contains coreference, or each instance has inconsistent expressions of predicates with the same meaning even using the memory schema. This issue can be mitigated by utilizing more in-context demonstrations, initializing by sliding every two sentences, or using softer string matching strategies. (2) Limited number of LLM-based rule implementation. Since there may be multiple applicable rules at each step, we adopt a pruning method to restrict the maximum numbers of rule implementation at each step to reduce computational costs, making it insufficient to answer some instances. This can be improved by running more rule implementation rounds at each step. (3) Inaccurate LLM-based rule implementation, especially for challenging tasks like AR-LSAT. This requires employing backbone LLMs with more advanced reasoning capabilities.

6 Related Work

LLMs with External Memory LLMs (Touvron et al., 2023; Abdin et al., 2024) have demonstrated remarkable performance across tasks, but struggle with complex reasoning that involves memorizing or grounding long-term information from context or interaction history. Beyond extending LLMs' context length (Lee et al., 2024; Lu et al., 2024), recent advancements augment LLMs with external memory. Park et al. (2023); Guo et al. (2023) equip LLMs agents with external memory modules to store and reference long-term dialogue history for better interaction. For knowledge-intensive tasks, Yue et al. (2024); Wang et al. (2024b) encode long-form context into memory for retrieval and utilization. However, previous working memory mainly stores natural language or parametric

[4] These figures are obtained from our re-implementation.

entries, making accurate referencing and updating challenging. Symbolic memory is further proposed to address this issue. ChatDB (Hu et al., 2023) uses databases as symbolic memory for precise information recording and processing, but is limited to product inventory. Statler (Yoneda et al., 2023) introduces symbolic world memory to maintain robot states for embodied reasoning. Our work leverages external memory to store both natural language and symbolic facts and rules, enabling more precise rule grounding for multi-step rule application.

Rule Application for Reasoning Rules are well-established principles abstracted from broad real-world observations (Wang et al., 2024a; Zhu et al., 2023), or predetermined constraints designed for specific situations (Qiu et al., 2023). They serve as a crucial basis for drawing inferences in complex contexts by applying them to known facts to derive new conclusions. For example, logical reasoning (Wang et al., 2021; Sun et al., 2023; Chen et al., 2023) involves applying rules to contextual facts to answer queries, with Olausson et al. (2023); Pan et al. (2023) operating in a symbolic manner. Constraint satisfaction (Wang et al., 2022a) applies rules to find solutions meeting all restrictions. Complex reasoning requires multi-step deductive rule application, each step involving rule grounding and rule implementation for more faithful reasoning (Sanyal et al., 2022; Creswell et al., 2022). We propose to iteratively perform these two processes in a neurosymbolic manner based on working memory.

7 Conclusion

In this paper, we augment LLMs with an external working memory and propose a neurosymbolic framework for multi-step rule application to enhance LLMs' reasoning capabilities. The memory stores facts and rules in both natural language and symbolic forms, facilitating accurate retrieval during rule application. After writing all input facts and rules into the working memory, the framework iteratively performs symbolic rule grounding based on predicate and variable matching, followed by LLM-based rule implementation. It effectively combines the strengths of both symbolic and LLM methods. Our experiments demonstrate the framework's superiority over CoT-based and symbolic-based baselines, and show its robustness across various rule application steps and settings. In the future, we will extend our framework to incorporate more backbone LLMs and datasets, especially on more complex and long-term reasoning tasks.

Limitations

Limitation on Experimented Datasets Due to computational costs, our work mainly experiments with four datasets, focusing on logical reasoning, constraint satisfaction and object state tracking tasks. Future work will include a broader range of tasks and datasets to further validate our framework's effectiveness.

Limitation on Backbone LLMs We build our framework upon GPT-4 and GPT-3.5 to demonstrate its effectiveness with various abilities of symbolic semantic parsing and one-step rule application. We will expand our scope to take more backbone LLMs, including open-source models.

Risk of Environmental Impact A significant risk associated with our framework is the potential increase in computational costs and environmental burden due to the extensive use of LLMs APIs. This impact can be mitigated by adopting advanced open-source models like Llama-3-70B that are more efficient with less environmental impact.

Acknowledgments

Xiang Ren's research is supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via the HIATUS Program contract #2022-22072200006, the Defense Advanced Research Projects Agency with award HR00112220046, and NSF IIS 2048211. We would like to thank all the collaborators in USC INK research lab for their constructive feedback on the work.

References

Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*.

Krzysztof R Apt et al. 1997. *From logic programming to Prolog*, volume 362. Prentice Hall London.

Lukas Berglund, Meg Tong, Max Kaufmann, Mikita Balesni, Asa Cooper Stickland, Tomasz Korbak, and Owain Evans. 2023. The reversal curse: Llms trained on" a is b" fail to learn" b is a". *arXiv preprint arXiv:2309.12288*.

Meiqi Chen, Yubo Ma, Kaitao Song, Yixin Cao, Yan Zhang, and Dongsheng Li. 2023. Learning to teach large language models logical reasoning. *arXiv preprint arXiv:2310.09158*.

Xinyun Chen, Ryan A Chi, Xuezhi Wang, and Denny Zhou. 2024. Premise order matters in reasoning with large language models. *arXiv preprint arXiv:2402.08939*.

Antonia Creswell, Murray Shanahan, and Irina Higgins. 2022. Selection-inference: Exploiting large language models for interpretable logical reasoning. *arXiv preprint arXiv:2205.09712*.

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

Jing Guo, Nan Li, Jianchuan Qi, Hang Yang, Ruiqiao Li, Yuzhen Feng, Si Zhang, and Ming Xu. 2023. Empowering working memory for large language model agents. *arXiv preprint arXiv:2312.17259*.

Kyle O Hardman and Nelson Cowan. 2016. Reasoning and memory: People make varied use of the information available in working memory. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 42(5):700.

Chenxu Hu, Jie Fu, Chenzhuang Du, Simian Luo, Junbo Zhao, and Hang Zhao. 2023. Chatdb: Augmenting llms with databases as their symbolic memory. *arXiv preprint arXiv:2306.03901*.

Najoung Kim and Sebastian Schuster. 2023. Entity tracking in language models. *arXiv preprint arXiv:2305.02363*.

Jack Lanchantin, Shubham Toshniwal, Jason Weston, Sainbayar Sukhbaatar, et al. 2024. Learning to reason and memorize with self-notes. *Advances in Neural Information Processing Systems*, 36.

Jinu Lee and Wonseok Hwang. 2024. Symba: Symbolic backward chaining for multi-step natural language reasoning. *arXiv preprint arXiv:2402.12806*.

Kuang-Huei Lee, Xinyun Chen, Hiroki Furuta, John Canny, and Ian Fischer. 2024. A human-inspired reading agent with gist memory of very long contexts. *arXiv preprint arXiv:2402.09727*.

Zhan Ling, Yunhao Fang, Xuanlin Li, Zhiao Huang, Mingu Lee, Roland Memisevic, and Hao Su. 2024. Deductive verification of chain-of-thought reasoning. *Advances in Neural Information Processing Systems*, 36.

Yi Lu, Xin Zhou, Wei He, Jun Zhao, Tao Ji, Tao Gui, Qi Zhang, and Xuanjing Huang. 2024. Longheads: Multi-head attention is secretly a long context processor. *arXiv preprint arXiv:2402.10685*.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. 2021. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*.

Theo Olausson, Alex Gu, Ben Lipkin, Cedegao Zhang, Armando Solar-Lezama, Joshua Tenenbaum, and Roger Levy. 2023. 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*.

OpenAI. 2023. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. 2023. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. *arXiv preprint arXiv:2305.12295*.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22.

Linlu Qiu, Liwei Jiang, Ximing Lu, Melanie Sclar, Valentina Pyatkin, Chandra Bhagavatula, Bailin Wang, Yoon Kim, Yejin Choi, Nouha Dziri, et al. 2023. Phenomenal yet puzzling: Testing inductive reasoning capabilities of language models with hypothesis refinement. *arXiv preprint arXiv:2310.08559*.

Soumya Sanyal, Harman Singh, and Xiang Ren. 2022. Fairr: Faithful and robust deductive reasoning over natural language. *arXiv preprint arXiv:2203.10261*.

Koustuv Sinha, Shagun Sodhani, Jin Dong, Joelle Pineau, and William L Hamilton. 2019. Clutrr: A diagnostic benchmark for inductive reasoning from text. *arXiv preprint arXiv:1908.06177*.

Hongda Sun, Weikai Xu, Wei Liu, Jian Luan, Bin Wang, Shuo Shang, Ji-Rong Wen, and Rui Yan. 2023. From indeterminacy to determinacy: Augmenting logical reasoning capabilities with large language models. *arXiv preprint arXiv:2310.18659*.

Oyvind Tafjord, Bhavana Dalvi Mishra, and Peter Clark. 2020. Proofwriter: Generating implications, proofs, and abductive statements over natural language. *arXiv preprint arXiv:2012.13048*.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Siyuan Wang, Zhongkun Liu, Wanjun Zhong, Ming Zhou, Zhongyu Wei, Zhumin Chen, and Nan Duan. 2022a. From lsat: The progress and challenges of complex reasoning. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:2201–2216.

Siyuan Wang, Zhongyu Wei, Yejin Choi, and Xiang Ren. 2024a. Can llms reason with rules? logic scaffolding for stress-testing and improving llms. *arXiv preprint arXiv:2402.11442*.

Siyuan Wang, Wanjun Zhong, Duyu Tang, Zhongyu Wei, Zhihao Fan, Daxin Jiang, Ming Zhou, and Nan Duan. 2021. Logic-driven context extension and data augmentation for logical reasoning of text. *arXiv preprint arXiv:2105.03659*.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. 2024b. Augmenting language models with long-term memory. *Advances in Neural Information Processing Systems*, 36.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022b. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

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

Jundong Xu, Hao Fei, Liangming Pan, Qian Liu, Mong-Li Lee, and Wynne Hsu. 2024. Faithful logical reasoning via symbolic chain-of-thought. *arXiv preprint arXiv:2405.18357*.

Takuma Yoneda, Jiading Fang, Peng Li, Huanyu Zhang, Tianchong Jiang, Shengjie Lin, Ben Picker, David Yunis, Hongyuan Mei, and Matthew R Walter. 2023. Statler: State-maintaining language models for embodied reasoning. *arXiv preprint arXiv:2306.17840*.

Xihang Yue, Linchao Zhu, and Yi Yang. 2024. Fragrel: Exploiting fragment-level relations in the external memory of large language models. *arXiv preprint arXiv:2406.03092*.

Wanjun Zhong, Siyuan Wang, Duyu Tang, Zenan Xu, Daya Guo, Jiahai Wang, Jian Yin, Ming Zhou, and Nan Duan. 2021. Ar-lsat: Investigating analytical reasoning of text. *arXiv preprint arXiv:2104.06598*.

Zhaocheng Zhu, Yuan Xue, Xinyun Chen, Denny Zhou, Jian Tang, Dale Schuurmans, and Hanjun Dai. 2023. Large language models can learn rules. *arXiv preprint arXiv:2310.07064*.

A Memory Schema Update

An example of the memory schema construction process is illustrated in Figure 6. Before each symbolic formulation, the process first looks up the memory schema to determine whether its maintained predicates and objects can cover the current fact or rule to be formulated. If it can, symbolic formulation is conducted directly based on the memory schema. If it cannot, new predicates or objects are created and added to the memory schema, and the symbolic formulation proceeds based on the updated memory schema. Then new formulated facts and rules are written into the working memory.

B Implementation Details

We implement our framework based on two different backbone LLMs: GPT-4 (gpt-4-turbo-0409 for CLUTRR, ProofWriter and Boxes, gpt-4o for AR-LSAT) and GPT-3.5 (gpt-3.5-turbo-0125), to test its effectiveness with different capabilities of symbolic semantic parsing and one-step rule application. For fair comparison, we re-implement all baseline methods using corresponding LLMs. All CoT-based baselines utilize the same in-context demonstrations. The generation temperature is set to 0.0 by default. The maximum number of steps in our framework is set to 4, 6, 8 for actual 2, 3-4, and 5-6 steps in CLUTRR and ProofWriter. For AR-LSAT, the maximum steps are set according to the number of rules, and for Boxes, they are set according to the number of operational facts.

C Further Experiments

C.1 Comparison with Multi-shot CoT-based Methods.

Since we implement WM-Neurosymbolic using few-shot prompts to better control output formats, we conduct additional experiments to illustrate our framework's effectiveness even when compared to CoT-based methods with multi-shot demonstrations. Specifically, we set the number of demonstrations in CoT-based methods for each dataset according to the maximum number of examples used by our framework: 2 for CLUTRR and AR-LSAT, and 3 for ProofWriter and Boxes. As shown in Table 6, using more examples in few-shot CoT prompting does not always lead to performance improvement. However, compared to both one-shot and multi-shot CoT-based methods, our framework consistently exhibits enhanced performance.

C.2 Rule Application without Rules Provided

To simulate realistic scenarios where rules are commonsense principles derived from real-world observations but not explicitly provided, we additionally experiment our framework on CLUTRR and Boxes datasets with rules not pre-defined. Here, our working memory only stores and updates facts. In each step, we select applicable facts (those with overlapping objects) from memory, and ask LLMs to self-generate applicable rules for rule implementation until the query is resolved. As shown in Table 7, compared to the Scratchpad-CoT baseline without provided rules, our framework on top of GPT-4 still shows improvement.

D Framework Prompts

Table 8, 9 and 10 show the example prompts for fact initialization, rule initialization, and LLM-based rule implementation in the CLUTRR dataset. Table 11, 12 and 13 show the example prompts for the ProofWriter dataset. Table 14, 15 and 16 show the example prompts for the AR-LSAT dataset. Table 17, 18 and 19 show the example prompts for the Boxes dataset.

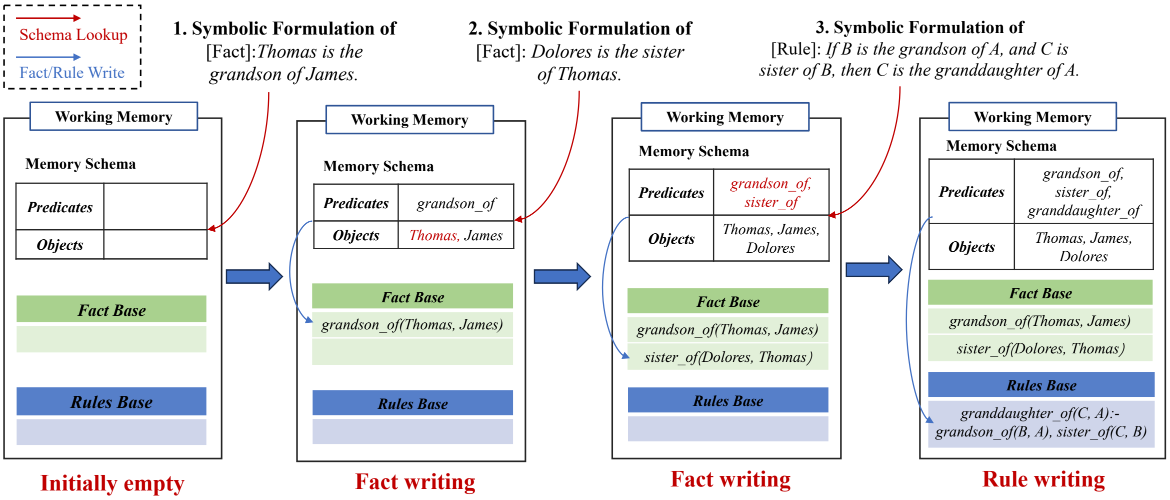

Schema Lookup
Fact/Rule Write
1. Symbolic Formulation of [Fact]:Thomas is the grandson of James.
2. Symbolic Formulation of [Fact]: Dolores is the sister of Thomas.
3. Symbolic Formulation of [Rule]: If B is the grandson of A, and C is sister of B, then C is the granddaughter of A.
Working Memory
Memory Schema
Predicates
Objects
Fact Base
Rules Base
Working Memory
Memory Schema
Predicates
grandson_of
Objects
Thomas, James
Fact Base
grandson_of(Thomas, James)
Rules Base
Working Memory
Memory Schema
Predicates
grandson_of, sister_of
Objects
Thomas, James, Dolores
Fact Base
grandson_of(Thomas, James)
sister_of(Dolores, Thomas)
Rules Base
Working Memory
Memory Schema
Predicates
grandson_of, sister_of, granddaughter_of
Objects
Thomas, James, Dolores
Fact Base
grandson_of(Thomas, James)
sister_of(Dolores, Thomas)
Rules Base
granddaughter_of(C, A):- grandson_of(B, A), sister_of(C, B)
Initially empty
Fact writing
Fact writing
Rule writing

Figure 6: An example construction process of our working memory schema alongside the memory initialization.

Method	CLUTRR		ProofWriter		AR-LSAT		Boxes	
	GPT-4	GPT-3.5	GPT-4	GPT-3.5	GPT-4	GPT-3.5	GPT-4	GPT-3.5
One-shot CoT-base Methods								
Scratchpad-CoT	83.83%	57.02%	61.33%	49.67%	41.25%	30.00%	91.85%	15.60%
SC-CoT	85.53%	59.57%	62.00%	54.00%	45.00%	31.25%	93.33%	17.04%
Self-Notes	74.04%	55.74%	62.00%	52.67%	47.50%	23.75%	92.59%	18.52%
Multi-shot CoT-base Methods								
Shot Number	2-shot		3-shot		2-shot		3-shot	
Scratchpad-CoT	86.38%	59.57%	64.33%	48.00%	52.50%	17.50%	97.04%	22.22%
SC-CoT	87.23%	60.85%	66.33%	48.33%	50.00%	18.75%	97.78%	24.44%
Self-Notes	72.76%	54.89%	61.67%	56.33%	53.75%	21.25%	97.04%	25.19%
WM-Neurosymbolic	**92.34%**	**78.72%**	**77.33%**	**58.00%**	**70.00%**	**35.00%**	**100%**	**34.29%**

Table 6: Comparison to multi-shot CoT-based methods.

Methods	CLUTRR	Boxes
Scratchpad-CoT	82.13%	89.63%
WM-Neurosymbolic	83.83%	96.30%

Table 7: Performance without rule provided.

Prompt for Fact Initialization (CLUTRR)

Please list all explicitly mentioned facts from the context.

Each fact should be presented on a separate line under the header "Facts:". Format each fact as "Person A is the Relationship of Person B." and follow it with its symbolic triplet formatted as "[predicate(A, B)]".

For each fact, also provide the corresponding reverse fact. For example, if the fact is "Person A is the Relationship of Person B," the reverse fact is "Person B is the Reverse_Relationship of Person A.

Please try to use the objects and predicates in the provided schema to describe symbolic facts. If the schema does not contain corresponding elements to describe the context, generate the symbolic fact directly from its natural language form. Note: Avoid using objects and predicates that do not exist in the given context when generating facts.

Examples:

Schema Objects: null

Schema Predicates: null

Context: Don's father, Joshua, and grandfather, James, went hiking during the first weekend of spring.

Facts:

- Joshua is the father of Don. [father_of(Joshua, Don)]
- Don is the son of Joshua. [son_of(Don, Joshua)]
- James is the grandfather of Don. [grandfather_of(James, Don)]
- Don is the grandson of James. [grandson_of(Don, James)]

——

Schema Objects: Joshua, Don, James

Schema Predicates: father_of, son_of, grandfather_of, grandson_of

Context: James took his daughter Lena out for dinner.

Facts:

- Lena is the daughter of James. [daughter_of(Lena, James)]
- James is the father of Lena. [father_of(James, Lena)]

Here's what you need to do.

Schema Objects: {objects}

Schema Predicates: {predicates}

Context: {context}

Facts:

Table 8: The prompt for fact initialization in CLUTRR.

Prompt for Rule Initialization (CLUTRR)

Please convert the following inference rule into a symbolic representation in Prolog without changing its wordings.

Ensure the conclusion and the premises are separated by ":-".

The predicates for each atom should be represented as relationships in lowercase.

Please try to use the objects and predicates in the provided schema to describe the symbolic rule. If the schema does not contain corresponding elements to describe the rule, generate the symbolic rule directly from its natural language form.

Examples:

Schema Objects: Joshua, Don, James

Schema Predicates: sister_of, brother_of

Rule: If B is the sister of A, and C is the brother of B, then C is the brother of A.

Symbolic Rule: brother_of(C, A) :- sister_of(B, A), brother_of(C, B).

——

Schema Objects: Joshua, Don, James

Schema Predicates: sister_of, father_of, brother_of

Rule: If B is the father of A, and C is the daughter of B, then C is the sister of A.

Symbolic Rule: sister_of(C, A) :- father_of(B, A), daughter_of(C, B).

Here's what you need to do.

Schema Objects: {objects}

Schema Predicates: {predicates}

Rule: {rule}

Symbolic Rule:

Table 9: The prompt for rule initialization in CLUTRR.

Prompt for Rule Implementation (CLUTRR)

System: You are an expert in determining kinship relationships. You will receive a query about the kinship between two individuals, and your task is to answer this query.

User: At each turn, you will be provided a list of identified supporting facts and an inference rule.
Please on a new line starting with "Rule Implementation:" to implement the rule based on the supporting facts to analyze and deduce new potential fact.
Then on a new line starting with "New fact:" to outline the new inferred fact in both natural language form and its corresponding symbolic format within "[" and "]".
Please try to use the objects and predicates in the provided schema to describe symbolic facts. If the schema does not contain corresponding elements, generate the symbolic fact directly from its natural language form.
Finally predict "Yes" or "No" to judge whether the new inferred fact can solve the query, in a new line starting with "Judgement:".

Examples:
Schema Objects: Joshua, Don, James
Schema Predicates: sister_of, brother_of
Query: How is Irvin related to Hugh?
Fact List: 3. Frances is the mother of Wesley. 6. Hugh is the son of Frances.
Rule: If B is the mother of A, and C is the son of B, then C is the brother of A.
Rule Implementation: According to the rule, since Frances is the mother of Wesley, and Hugh is the son of Frances, we can infer that Hugh is the brother of Wesley.
New fact: Hugh is the brother of Wesley. [brother_of(Hugh, Wesley)]
Judgement: No. Because the new fact does not state the relationship between Irvin and Hugh.

——

Schema Objects: Joshua, Leno, James
Schema Predicates: father_of, sister_of, daughter_of
Query: How is Joshua related to Lena?
Fact List: 1. James is the father of Joshua. 3. Leno is the daughter of James.
Rule: If B is the father of A, and C is the daughter of B, then C is the sister of A.
Rule Implementation: According to the rule, since James is the father of Joshua, and Lena is the daughter of James, we can infer that Lena is the sister of Joshua.
New fact: Lena is the sister of Joshua. [sister_of(Lena, Joshua)]
Judgement: Yes. Because the new fact states the relationship between Joshua and Lena.

Here's what you need to do.
Schema Objects: {objects}
Schema Predicates: {predicates}
Query: {query}
Fact List: {facts}
Rule: {rule}
Rule Implementation:

Table 10: The prompt for LLM-based rule implementation in CLUTRR.

Prompt for Fact Initialization (ProofWriter)

Please list the symbolic fact of the given context.
Format each symbolic fact in Prolog notation as "predicate(X, Y, ...)" where X, Y, ... are the arguments of the predicate. Avoid predicate nesting such as not(smart(X)), but using not_smart(X) instead.
Please try to use the objects and predicates in the provided schema to describe symbolic facts. If the schema is null or does not contain corresponding elements to describe the context, generate the symbolic fact directly from its natural language form.

Examples:
Schema Objects: David
Schema Predicates: kind
Context: Context: Bob is big.
Fact: big(Bob)

——

Schema Objects: bald eagle
Schema Predicates: needs
Context: The cow visits the bald eagle.
Fact: visits(cow, bald eagle)

——

Schema Objects: lion, squirrel
Schema Predicates: sees
Context: The lion does not see the squirrel.
Fact: not_see(lion, squirrel)

Here's what you need to do.
Schema Objects: {objects}
Schema Predicates: {predicates}
Context: {context}
Fact:

Table 11: The prompt for fact initialization in ProofWriter.

Prompt for Rule Initialization (ProofWriter)

Please convert the explicitly provided rule into their symbolic forms in Prolog without changing its original wordings.
Format each symbolic rule in Prolog notation with the conclusion and premises separated by ":-", and format each atom fact in the rule as "predicate(X, Y, ...)" where X, Y, ... are the arguments of the predicate. Avoid predicate nesting such as not(smart(X)), but using not_smart(X) instead.
Please try to use the objects and predicates in the provided schema to describe the symbolic rule. If the schema is null or does not contain corresponding elements to describe the rule, generate the symbolic rule directly from its natural language form. Note: Avoid using objects and predicates that do not exist in the provided rule when generating its symbolic form.

Examples:
Schema Objects: Bob
Schema Predicates: kind, smart
Rule: If something is kind and smart then it is nice.
Symbolic Rule: nice(X) :- kind(X), smart(X)

——

Schema Objects: bald eagle
Schema Predicates: needs, sees
Rule: If someone needs the tiger then the tiger sees the bald eagle.
Symbolic Rule: sees(tiger, bald eagle) :- needs(X, tiger)

——

Schema Objects: Bob
Schema Predicates: kind, big, furry
Rule: Kind, big people are not furry.
Symbolic Rule: not_furry(X) :- kind(X), big(X)

Here's what you need to do.
Schema Objects: {objects}
Schema Predicates: {predicates}
Rule: {rule}
Symbolic Rule:

Table 12: The prompt for rule initialization in ProofWriter.

Prompt for Rule Implementation (ProofWriter)

System: You are an expert in logical reasoning. You will receive a context including a list of facts and inference rules, and a specific query. Your task is to answer this query following the provided rule.

User: At each turn, you will be provided a list of identified supporting facts and an inference rule.
Please on a new line starting with "Rule Implementation:" to implement the rule based on the supporting facts to analyze and deduce new potential fact.
Then on a new line starting with "New fact:" to outline the new inferred fact in both natural language form and its corresponding symbolic format within "[" and "]".
Please try to use the objects and predicates in the provided schema to describe symbolic facts. If the schema is null or does not contain corresponding elements, generate the symbolic fact directly from its natural language form.
Finally predict "Yes" or "No" to judge whether the new inferred fact can solve the query, in a new line starting with "Judgement:".

Examples:
Schema Objects: Gary
Schema Predicates: big, not_green
Query: Is it true that Gary is not red?
Fact List: 3. Gary is big.
Rule: All big things are not green.
Rule Implementation: According to the rule, since Gary is big, we can infer that Gary is not green.
New fact: Gary is not green. [not_green(Gary)]
Judgement: No. Because the new fact does not state the relationship between Gary and red.

Schema Objects: Bob
Schema Predicates: furry, big, not_quiet
Query: Is it true that Bob is not quiet?
Fact List: 1. Bob is furry. 2. Bob is big.
Rule: If Bob is furry and Bob is big then Bob is not quiet.
Rule Implementation: According to the rule, since Bob is furry and Bob is big, we can infer that Bob is not quiet.
New fact: Bob is not quiet. [not_quiet(Bob)]
Judgement: Yes. Because the new fact states the relationship between Bob and quiet.

Here's what you need to do.
Schema Objects: {objects}
Schema Predicates: {predicates}
Query: {query}
Fact List: {facts}
Rule: {rule}
Rule Implementation:

Table 13: The prompt for LLM-based rule implementation in ProofWriter.

Prompt for Fact Initialization (AR-LSAT)

You will receive a context including a list of constraint rules, and a specific query with five candidate options (A, B, C, D, E).
Please list the symbolic forms of all established facts in the given query and option.
Format each symbolic fact in Prolog notation as "predicate(X, Y, ...)" where X, Y, ... are the arguments of the predicate.
Please try to use the objects and predicates in the provided schema to describe symbolic facts. If the schema is null or does not contain corresponding elements to describe the context, generate the symbolic fact directly from its natural language form. Please always use one predicate, i.e., assign.

Examples:
Schema Objects: Monday, Tuesday, Wednesday, morning
Schema Predicates: assign
Context: Of the eight students-George, Helen, Irving, Kyle, Lenore, Nina, Olivia, and Robert-in a seminar, exactly six will give individual oral reports during three consecutive days-Monday, Tuesday, and Wednesday. Exactly two reports will be given each day-one in the morning and one in the afternoon-according to the following conditions.
Query: If Kyle and Lenore do not give reports, then the morning reports on Monday, Tuesday, and Wednesday, respectively, could be given by
Option: A) Helen, George, and Nina
Facts:
- Helen gives report on Monday morning. [assign(Helen, Monday, morning)]
- George gives report on Tuesday morning. [assign(George, Tuesday, morning)]
- Nina gives report on Wednesday morning. [assign(Nina, Wednesday, morning)]

——

Schema Objects: Monday, Tuesday, Wednesday, morning
Schema Predicates: assign
Context: Each of seven candidates for the position of judge2̌014Hamadi, Jefferson, Kurtz, Li, McDonnell, Ortiz, and Perkins2̌014will be appointed to an open position on one of two courts2̌014the appellate court or the trial court. There are three open positions on the appellate court and six open positions on the trial court, but not all of them will be filled at this time. The judicial appointments will conform to the following conditions.
Query: Which one of the following is an acceptable set of appointments of candidates to courts?
Option: E) appellate: Li, Perkins; trial: Hamadi, Jefferson, Kurtz, McDonnell, Ortiz
Facts:
- The appellate court appoints Li and Perkins. [assign(appellate, Li, Perkins)]
- The trial court appoints Hamadi, Jefferson, Kurtz, McDonnell and Ortiz. [assign(trial, Hamadi, Jefferson, Kurtz, McDonnell, Ortiz)]

Here's what you need to do.
Schema Objects: {objects}
Schema Predicates: {predicates}
Context: {context}
Query: {query}
Option: {option}
Facts:

Table 14: The prompt for fact initialization in AR-LSAT.

Prompt for Rule Initialization (AR-LSAT)

Please list the symbolic forms of the given constraint rule in Prolog without changing its original wordings.
Format each symbolic rule in Prolog notation, representing it either as a conclusion or as a combination of a conclusion and premises, separated by ":-". Format each atom fact in the rule as "predicate(X, Y, ...)" where X, Y, ... are the arguments of the predicate. Avoid predicate nesting such as not(smart(X)), but using not_smart(X) instead. Avoid mathematic expression such as N =< 4, but using samller_than(N, 4).
Please try to use the objects and predicates in the provided schema to describe the symbolic rule. If the schema does not contain corresponding elements, generate the symbolic rule directly from its natural language form. Please always use one predicate, i.e., constraint.

Examples:
Schema Objects: Monday, Tuesday, Wednesday, morning, Kyle, Lenore, Helen, George, Nina
Schema Predicates: constraint
Context: Of the eight students-George, Helen, Irving, Kyle, Lenore, Nina, Olivia, and Robert-in a seminar, exactly six will give individual oral reports during three consecutive days-Monday, Tuesday, and Wednesday. Exactly two reports will be given each day-one in the morning and one in the afternoon-according to the following conditions.
Constraint Rule: Tuesday is the only day on which George can give a report.
Symbolic Rule:
- constraint(George, Tuesday)

——

Schema Objects: Monday, Tuesday, Wednesday, morning, Kyle, Lenore, Helen, George, Nina
Schema Predicates: constraint
Context: Of the eight students-George, Helen, Irving, Kyle, Lenore, Nina, Olivia, and Robert-in a seminar, exactly six will give individual oral reports during three consecutive days-Monday, Tuesday, and Wednesday. Exactly two reports will be given each day-one in the morning and one in the afternoon-according to the following conditions.
Constraint Rule: If Nina gives a report, then on the next day Helen and Irving must both give reports, unless Nina's report is given on Wednesday.
Symbolic Rule:
- constraint(Helen, Irving, Tuesday) :- constraint(Nina, Monday)
- constraint(Helen, Irving, Wednesday) :- constraint(Nina, Tuesday)

Here's what you need to do.
Schema Objects: {objects}
Schema Predicates: {predicates}
Context: {context}
Constraint Rule: {rule}
Symbolic Rule:

Table 15: The prompt for rule initialization in AR-LSAT.

Prompt for Rule Implementation (AR-LSAT)

System: You are an expert in logical reasoning. You will receive a context including background information followed by a list of constraint rules, and a specific query with five candidate options (A, B, C, D, E). Your task is to accurately select the answer that satisfies the provided rule.

User: At each turn, you will be provided a context background, a constraint rule and a list of relevant facts.
Please on a new line starting with "Rule Implementation:" to implement the rule based on the facts to analyze there is a conflict between them. If no conflict, proceed to deduce new potential facts.
Then predict "Yes" or "No" to judge whether there is a conflict between the rule and facts, in a new line starting with "Judgement:".
If the judgement is No, proceed on a new line starting with "New fact:" to outline the new inferred fact in both natural language form and its corresponding symbolic format as "predicate(X, Y, ...)" within "[" and "]".
Please try to use the objects and predicates in the provided schema to describe symbolic facts. If the schema does not contain corresponding elements, generate the symbolic fact directly from its natural language form. Please always use one predicate, i.e., assign.

Examples:
Schema Objects: Monday, Tuesday, Wednesday, morning, Kyle, Lenore, Helen, George, Nina, Irving, Robert
Schema Predicates: assign
Context: Of the eight students-George, Helen, Irving, Kyle, Lenore, Nina, Olivia, and Robert-in a seminar, exactly six will give individual oral reports during three consecutive days-Monday, Tuesday, and Wednesday. Exactly two reports will be given each day-one in the morning and one in the afternoon-according to the following conditions.
Rule: Tuesday is the only day on which George can give a report.
Query: If Kyle and Lenore do not give reports, then the morning reports on Monday, Tuesday, and Wednesday, respectively, could be given by
Fact List:
- B) Irving, Robert, and Helen
Rule Implementation: According to the rule and the fact Robert give report on Tuesday morning, there is no conflict and we can infer George give a report on Tuesday afternoon.
Judgement: No.
New fact: George give a report on Tuesday afternoon. [assign(George, Tuesday, afternoon)]
——
Schema Objects: Monday, Tuesday, Wednesday, morning, afternoon, Kyle, Lenore, Helen, George, Nina, Irving, Robert
Schema Predicates: assign
Context: Of the eight students-George, Helen, Irving, Kyle, Lenore, Nina, Olivia, and Robert-in a seminar, exactly six will give individual oral reports during three consecutive days-Monday, Tuesday, and Wednesday. Exactly two reports will be given each day-one in the morning and one in the afternoon-according to the following conditions.
Rule: Neither Olivia nor Robert can give an afternoon report.
Query: If Kyle and Lenore do not give reports, then the morning reports on Monday, Tuesday, and Wednesday, respectively, could be given by
Fact List:
- B) Irving, Robert, and Helen
- George give a report on Tuesday afternoon.
Rule Implementation: According to the rule, and the facts Irving, Robert, and Helen all give report on morning, there is a conflict that can not give a report on the morning.
Judgement: Yes.

Here's what you need to do.
Schema Objects: {objects}
Schema Predicates: {predicates}
Context: {context}
Rule: {rule}
Query: {query}
Fact List: {facts}
Rule Implementation:

Table 16: The prompt for LLM-based rule implementation in AR-LSAT.

Prompt for State Fact Initialization (Boxes)

Please list the symbolic form of the explicitly provided fact in the context.
Format the symbolic fact in Prolog notation as "predicate(X, Y, ...)" where X, Y, ... are the arguments of the predicate verb.
Please try to use the objects and predicates in the provided schema to describe symbolic facts. If the schema is null or does not contain corresponding elements to describe the context, generate the symbolic fact directly from its natural language form.

Examples:
Schema Objects: null
Schema Predicates: contains, move_from_to, remove_from, put_into
Context: Box 0 contains the rose.
Fact: contains(Box 0, the rose)

——

Schema Objects: Box 0, the rose
Schema Predicates: contains, move_from_to, remove_from, put_into
Context: Box 4 contains the bread and the radio and the tape.
Fact: contains(Box 4, the bread, the radio, the tape)

——

Schema Objects: Box 0, the rose
Schema Predicates: contains, move_from_to, remove_from, put_into
Context: Box 1 contains nothing.
Fact: contains(Box 1, nothing)

Here's what you need to do.
Schema Objects: {objects}
Schema Predicates: contains, move_from_to, remove_from, put_into
Context: {context}
Fact:

Table 17: The prompt for state fact initialization in Boxes.

Prompt for Operation Fact Initialization (Boxes)

Please list the symbolic form of the explicitly provided fact in the context.
Format the symbolic fact in Prolog notation as "predicate(X, Y, ...)" where X, Y, ... are the arguments of the predicate.
Please try to use the objects and predicates in the provided schema to describe symbolic facts. If the schema is null or does not contain corresponding elements to describe the context, generate the symbolic fact directly from its natural language form.

Examples:
Schema Objects: null
Schema Predicates: contains, move_from_to, remove_from, put_into
Context: Put the shoe into Box 0.
Fact: put_into(the shoe, Box 0)

——

Schema Objects: null
Schema Predicates: contains, move_from_to, remove_from, put_into
Context: Remove the radio and the tape from Box 4.
Fact: remove_from(the radio, the tape, Box 4)

——

Schema Objects: Box 0, the rose, the bread, the radio, the tape
Schema Predicates: contains, move_from_to, remove_from, put_into
Context: Move the contents of Box 3 to Box 1.
Fact: move_from_to(the contents, Box 3, Box 1)

Here's what you need to do.
Schema Objects: {objects}
Schema Predicates: contains, move_from_to, remove_from, put_into
Context: {context}
Fact:

Table 18: The prompt for operation fact initialization in Boxes.

Prompt for Rule Implementation (Boxes)

System: You are an expert in logical reasoning. You will receive a context including a list of state facts and operational facts, a list of rules and a specific query. Your task is to answer this query following the provided rule.

User: At each turn, you will be provided a list of state facts and an operational fact, and a logical rule.
Please on a new line starting with "Rule Implementation:" to implement the rule based on the facts to infer new state facts after the operation.
Then output "New facts:" in a new line, and each new inferred fact in both natural language form and its corresponding symbolic format on separate lines under the header "New facts:".
Each line must cover all contents about a distinct Box. For example, the first is about Box 1, then the second line should not describe Box 1.
Format each fact in natural language as "Box X contains Y." where X is the box number and Y are the specific items instead of general "contents" in the box. Format each symbolic fact in Prolog notation as "predicate(X, Y, ...)" where X, Y, ... are the arguments of the predicate, and the predicate should be "contains". Please try to use the objects and predicates in the provided schema to describe symbolic facts. If the schema does not contain corresponding elements, generate the symbolic fact directly from its natural language form.

Examples:
Schema Objects: Box 0, the rose, the bread, the radio, the tape
Schema Predicates: contains, move_from_to, remove_from, put_into
State Facts: Box 1 contains the rose. Box 2 contains the letter.
Operational Fact: Move the contents from Box 2 to Box 1.
Rule: If move the contents X from Box A to Box B, then X are not in Box A and X are in Box B.
Rule Implementation: Based on the rule, after the moving operation, we can infer that Box 1 contains the rose and the letter, and Box 2 contains nothing.
New facts:
Box 1 contains the rose and the letter. [contains(Box 1, the rose, the letter)]
Box 2 contains nothing. [contains(Box 2, nothing)]
——
Schema Objects: Box 0, Box 1, Box 2, the rose, the bread, the radio, the tape, the letter, the book, nothing
Schema Predicates: contains, move_from_to, remove_from, put_into
State Facts: Box 2 contains the letter and the book.
Operational Fact: Remove the letter from Box 2.
Rule: If remove the contents X from Box A, then X are not in Box A.
Rule Implementation: Based on the rule, after the removing operation, we can infer that Box 2 contains the book.
New facts:
Box 2 contains the book. [contains(Box 2, the book)]

Here's what you need to do.
Schema Objects: {objects}
Schema Predicates: {predicates}
State Facts: {state facts}
Operational Fact: {op facts}
Rule: {rule}
Rule Implementation:

Table 19: The prompt for LLM-based rule implementation in Boxes.