Logic-Thinker: Teaching Large Language Models to Think more Logically.

Chengyao Wen, Qiang Cheng, Shaofei Wang, Zhizhen Liu, Deng Zhao, Lei Liang*

Ant Group
{wenchengyao.wcy, chengqiang.cq, wangshaofei.wsf,
zhizhen.lzz, zhaodeng.zd, leywar.liang}@antgroup.com

Abstract

Recent Large Reasoning Models (LRMs) have demonstrated the ability to generate long chains of thought (LongCoT) before arriving at a final conclusion. Despite remarkable breakthroughs in complex reasoning capabilities, LongCoT still faces challenges such as redundancy and logical incoherence. To address these issues, we aim to equip large language models (LLMs) with rigorous and concise logical reasoning capabilities. In this work, we propose Logic-Thinker, a neural-symbolic reasoning framework that employs symbolic solvers to precisely solve problems and transforms their internal solving processes into concise and rigorous chains of thought, referred to as ThinkerCoT. Our experimental results demonstrate that Logic-Thinker achieves state-of-the-art performance in logical reasoning problems. Additionally, LLMs fine-tuned with ThinkerCoT outperform models distilled from QwQ32B on logic reasoning tasks, achieving an overall accuracy improvement of 3.6% while reducing token output by 73%-91%. Furthermore, ThinkerCoT enhances the comprehensive reasoning capabilities of LLMs, as evidenced by performance improvements on reasoning benchmarks such as GPQA and AIME.

1 Introduction

Large language models (LLMs), such as the GPT series(OpenAI, 2023) and the Qwen series(Yang et al., 2024), have demonstrated remarkable capabilities across a wide range of natural language understanding tasks(Chang et al., 2024; Kasneci et al., 2023; Zhu et al., 2024). However, their performance in complex reasoning problems has remained suboptimal. To address this challenge, a new type of model has emerged in recent years: Large Reasoning Models (LRMs). By generating a long chain of thought (CoT)(Wei et al., 2022) before reaching the final conclusion, these models have achieved significant breakthroughs in many challenging reasoning tasks(Patel et al., 2024a; Rein et al., 2023; White et al., 2025). This inference-time scaling paradigm was popularized by OpenAI-o1(OpenAI, 2024), DeepSeek-R1(DeepSeek-AI, 2025), and QwQ(Team, 2025), gaining significant traction in the industry. Currently, a large number of studies are following this paradigm in an effort to enhance the upper limits of reasoning capabilities.

However, recent studies have highlighted several issues with this LongCoT reasoning paradigm: **1) Redundant Content**, including question restatements, verbose explanations and repetitive narratives(Munkhbat et al., 2025), results in a large amount of unnecessary token completion. **2) Overthinking**, specifically manifested as the repeated verification of some straightforward and direct questions(Chen et al., 2025; Qu et al., 2025). **3) Incoherent Reasoning**, when dealing with complex reasoning problems, LRMs may superficially jump between approaches, leading to shallow, fragmented reasoning, rather than a deep, coherent analysis(Wang et al., 2025).

To address these issues, many studies have proposed the idea of compressing LongCoT while retaining its critical reasoning steps. Through supervised fine-tuning (SFT), the refined CoT enables LLMs to acquire the capability if concise reasoning. Experiments have demonstrated that such methods can effectively reduce CoT redundancy while maintaining reasoning accuracy. Rather than post-processing LongCoT, some research has delved deeper into the fundamental causes of these issues, finding that LLMs allocate a large portion of their output to text coherence rather than core reasoning advancement(Su et al., 2025; Luo et al., 2025). On the other hand, the LongCoT capability of LRMs originates from the exploration of the response space during the Reinforcement Learning (RL)(Sutton and Barto, 1998) stage, primarily

* Corresponding author.

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 12955–12969
November 4-9, 2025

aiming at obtaining accuracy at the solution level, while the consistency of the intermediate reasoning process has been overlooked(Fatemi et al., 2025). Based on these conclusions, we propose that the redundancy, overthinking, and incoherent reasoning issues of LLMs are primarily caused by the inability of rigorous logic reasoning. Such rigorous logic reasoning ability is particularly critical in many logical reasoning tasks, including deduction, induction and hypothesis testing. Therefore, we focus on logical reasoning problems and aim to translate the inherent precise reasoning paths of neuro-symbolic solvers into concise and logically rigorous CoTs, thereby equipping LLMs with robust and concise reasoning capabilities.

In this work, We introduce Logic-Thinker, a neuro-symbolic reasoning framework that integrates both logic problem-solving and reasoning CoT generation capabilities. Logic-Thinker consists of three core modules: a Formulator for converting logical reasoning problems into symbolic expressions, a LogicSolver for executing symbolic expressions to solve problems, and a CoT-Generator for transforming the solver's internal execution process into CoTs, referred to as Thinker-CoT. We categorize common logical reasoning problems into two types: FOL (First-Order Logic) and CSP (Constraint Satisfaction Problems), and implement them respectively in the three modules.

The rest of this paper is organized as follows. section 2 presents related work. We present preliminaries in section 3. Our approach is discussed in section 4. We conduct comprehensive experiments to evaluate the effectiveness of out proposed methods in section 5 and we conclude the paper in Section 6.

In summary, we make the following contributions in this paper.

- We propose Logic-Thinker, a novel neuro-symbolic reasoning framework consists the capability of logic reasoning and CoT generation. Our framework achieves state-of-the-art (SOTA) performance on logical reasoning tasks.
- We utilize Logic-Thinker to generate Long-CoT data on several logical reasoning datasets, our SFT experiments demonstrate that ThinkerCoT significantly reduces the redundancy of LongCoT while further enhancing logical reasoning capabilities.
- We conducted an extension experiment to demonstrate that this rigorous logical reasoning capability can enhance the overall reasoning capabilities of LLMs.

2 Related Work

2.1 Concise Reasoning with SFT

SFT is a straightforward way to help models learn how to follow the instructions. There exist serval methods that fine-tune the models to achieve concise and accurate reasoning. Token Budget-Aware LLM Reasoning(Han et al., 2024b) first produces the target output by prompting the model with a CoT prompt that includes the optimized token budget, then they train the model with SFT to produce answers that adhere to the token budget. To eliminate redundant information in the reasoning chain, C3OT(Kang et al., 2024) employs GPT-4(OpenAI, 2023) as a compressor, preserving key information throughout the reasoning process. The model is then fine-tuned to learn the relationship between long and short CoTs. These works focus on controlling the length of CoT generated by large LLMs while maintaining the accuracy of reasoning. However, we take a different approach by relying on solvers to generate absolutely rigorous reasoning processes, rather than relying on LLMs for generation.

2.2 Neuro-Symbolic Reasoning

Neuro-symbolic approaches aim to enhance the logical reasoning capabilities of LLMs by integrating them with symbolic systems. Meta-Reasoning(Wang et al., 2024) proposed symbolically decomposing natural language questions, and Symbol-LLM(Xu et al., 2024a) transformed them into symbolic representations to be solved using solvers. In LINC (Olausson et al., 2023), an LLM is used to generate statements in First Order Logic (FOL), which are given to a FOL solver. To mitigate formalization errors, they generate K formalizations and make use of Kway majority voting to decide on the correct response. Logic-LM (Pan et al., 2023) is able to handle a broader range of problem types by supporting multiple symbolic formalizations and solvers, Moreover, a self-refinement module is introduced, which utilizes error messages from the symbolic solver to modify the symbolic formalization. However, the information loss during the formalization process may lead to reasoning failures, furthermore, most of these

work only focus on the accuracy of the final result, while the execution processes of the solvers have either been overlooked or not effectively utilized.

3 Preliminary

3.1 First-Order Logic Problem

First-Order Logic (FOL) problem is a form of logical deduction problem which extends propositional logic by allowing quantification over individual variables, enabling the expression of more complex statements and concepts. It is widely used in computer science, artificial intelligence, mathematics, and philosophy for formal reasoning and knowledge representation.

3.2 Constraint Satisfaction Problems

A Constraint satisfaction problem (CSP) seeks solution by assigning values to a set of variables while satisfying a number of constraints or conditions. Formally, Given (X, D, C) where $X = \{X_1, X_2, ..., X_n\}$ is the set of variables, $D = \{D_1, D_2, ..., D_n\}$ represents their respective domains of values, $C = \{C_1, C_2, ..., C_m\}$ is the set of constraints restricting variable assignments. For every constraint $C_j \in C$ is a pair of $\langle T_j, R_j \rangle$ where $T_j \subset X$ and R_j defines the relationships that T_j should satisfy.

4 Logic-Thinker

Symbolic solvers possess an inherent capability for rigorous reasoning: 1) The symbolic system ensures that each step's conclusion is deterministic rather than probabilistic and is verifiable. 2) Solvers follow a systematic search methodology, exploring the solution space exhaustively to ensure that no potential reasoning paths are overlooked. To enable LLMs to acquire such rigorous reasoning capabilities, we propose Logic-Thinker, which enables the transformation of the symbolic solver's internal execution process into concise and logically rigorous CoT, named ThinkerCoT. As shown in Figure 1, Logic-Thinker consists of three modules: A **Formulator** that formalizes natural language problems into symbolic expressions, a **LogicSolver** that executes symbolic expressions and derives conclusions, and a **CoT Generator** that produce ThinkerCoT based on reasoning processes.

4.1 Formulator

In this module, we leverage the in-context learning (ICL) capability of LLMs to translate natural language logic reasoning problems into symbolic representations. We implement a FOL Formulator and a CSP Formulator for these two categories of problems respectively.

FOL Formulator

Some previous studies use First-Order Logic to formalize problems and utilized certain open source FOL-solvers to address such problems. However, their accuracy remains suboptimal especially in complex problems, primarily due to information loss during the formalization process. After carefully analysis of the bad cases, we found the primary information losses lies in semantic information loss of Predicates. As shown in Figure 2, even when the problem is correctly formalized into FOL expressions, the solver is unable to deduce due to the loss of the *subClassOf* relationship between Gentleman and Man.

To address this issue, we combine certain semantic properties of OpenSPG(Liang et al., 2024; Yi et al., 2024), such as *subClassOf*, with FOL to create a semantic-enhanced FOL expression, refereed to as SeFOL. After the formalization of FOL expressions, we introduce a Semantic Extraction step to collect all the *predicates* and *functions* as **P**, we then prompt LLMs to extract semantic relationships in **P**, ruseults in **S**, which will be passed to the LogicSolver module along with the FOL expressions for reasoning. We provide an example of formalization prompt in Appendix A.1

CSP Formulator

XCSP^3(Boussemart et al., 2024) is a standardized format for representing and exchanging constraint satisfaction problems and related optimization problems. As the third iteration of the XML CSP (XCSP) series, XCSP^3 provides a simple, readable, and parsable XML-based format that allows researchers and developers to share models of various types of constraint problems. To solve CSP problems, we extended XCSP^3 to support the representation of question and option choices. An example of prompt is provided in Appendix A.2

4.2 LogicSolver

The LogicSolver module is designed for solving formalized problems and outputting the execution process to the downstream CoT Generator module. In addition, LogicSolver provides a syntax checker to verify the syntax correctness of the formalized problem. If there are syntax errors, it will provide

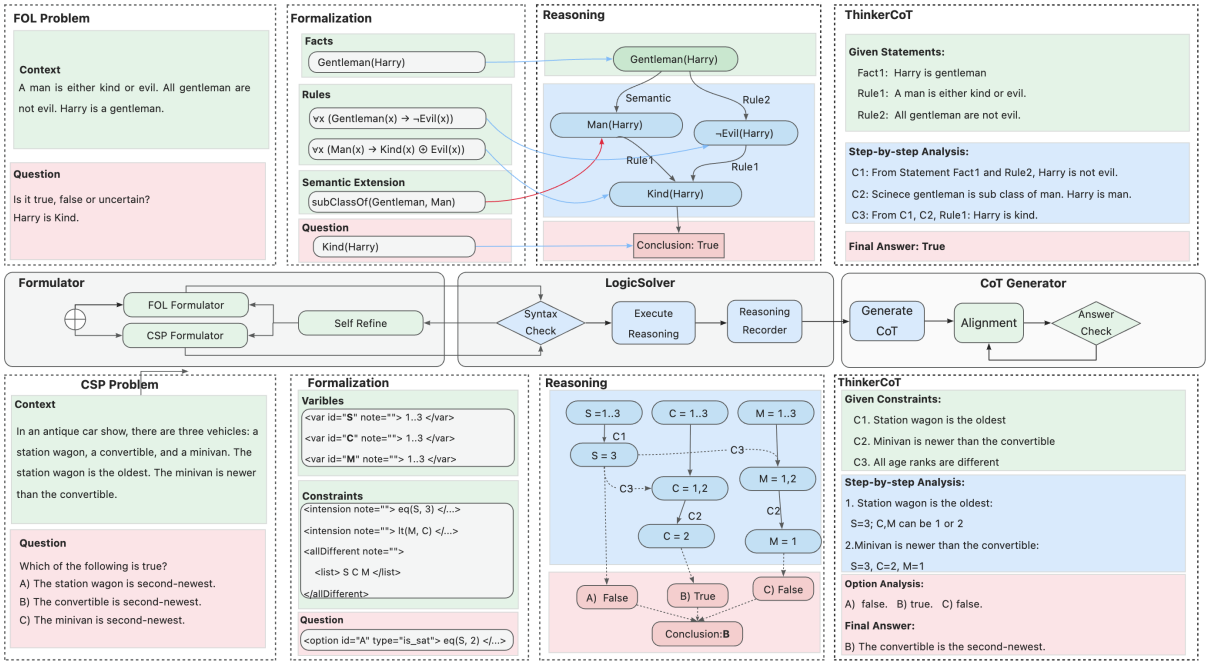


Figure 1: System Overview of Logic-Thinker. The framework consists of three modules: **(1)** Formulator formalize natural language problems into symbolic expressions. **(2)** LogicSolver conducts reasoning. **(3)** CoT-Generator produces the concise and logically rigorous CoT.

the error details to the upstream Formulator module for re-formalization.

FOL Solver

To support the semantic features of SPG, we introduced the logical reasoning engine *Thinker* of OpenSGP as the core engine of FOLSolver, and built a SeFOL-Parser for syntax parsing. *Thinker* is built on the SPO framework and incorporates forward, backward, and hybrid reasoning engines. The hybrid reasoning engine supports capabilities such as hypothesis-based reasoning and contraposition inference.

CSP Solver

A constraint satisfaction problem is to find assignments for all variables that satisfy all constraints. Common algorithms include backtracking, constraint propagation, and local search variants. For implementation, we introduce the Chocosolver(Prud'homme and Fages, 2022) library, a Java-based tool designed for Constraint Programming as the core engine of CSP Solver.

4.3 CoT Generator

The execution process of LogicSolver includes all possible reasoning paths, of which the truly effective reasoning path is a part. To ensure the conciseness of the output CoT, the CoT Generator module is designed for extracting the key reasoning paths and converting them to CoT. In addition, a style alignment procedure is employed to ensures the diversity of language styles.

FOL CoT Generator

Fol Solver utilizes a tree structure to record execution processes. However, redundant nodes can arise from noth duplicate recordings and excessive granularity, in addition, cyclic or contrapositive rules might introduce reasoning loops. To tackle this, we transform the tree into a directed acyclic graph (DAG) and implement optimizations such as merging duplicate nodes, eliminating negation nodes, and removing intermediate results. The DAG is composed of: conclusion nodes (terminals), fact/assumption nodes (starts), and deduction nodes (intermediates). We maintain a store of statement templates where each node is mapped to a natural language (NL) statement template based on its type. A CoT is generated through a traversal of the DAG, as shown in Algorithm 1. And a more detailed and complete example is shown in Figure 4 of Appendix B.

CSP CoT Generator

CSP Solver employs Implications to record CSP reasoning processes. Implications is a list structure that records the detailed changes of each variable

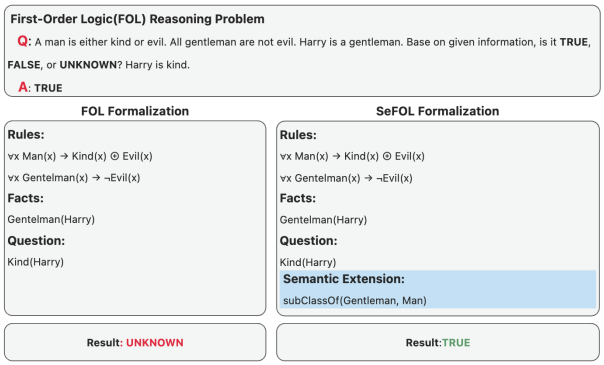


Figure 2: An example of reasoning failure caused by information loss. In the left part, the formalization of FOL is accurate, but *Gentleman* and *Man* are treated as two independent predicates in solvers, resulting an incorrect reasoning outcome. To address this, we incorporated semantic information extraction during the formalization process and supported these features in **FoL Solver**, ultimately achieving the correct reasoning result.

under the influence of different constraints. Since this information is overly detailed, in order to generate a concise and accurate COT, we first convert the Implications into a graph G, nodes in G represent variables and their corresponding values, while edges indicate a change in the value of a variable x from v_1 to v_2 under the influence of a constraint c. Then use breadth-first search (BFS) to extract the reasoning process as shown in Algorithm 2. And a more detailed and complete example is shown in Figure 5 of Appendix C.

5 Experiment

To comprehensively evaluate the effectiveness of Logic-Thinker, we designed experiments from two perspectives:

1. Logical reasoning capabilities of Logic-Thinker.
2. The effectiveness of ThinkerCoT.

We first introduce the datasets of logical reasoning problems used in the two experiments, followed by a detailed description of the two experiments, including their settings, baselines, experimental results, and analysis. Then, we conducted an experiment to verify whether applying ThinkerCoT to real-world SFT tasks could enhance the comprehensive reasoning capabilities of LLMs. Finally, we conduct a detailed case study to analyze the effectiveness of ThinkerCoT compared to LongCoT.

Algorithm 1 FOL CoT Generate

Input: $\mathrm{G} \leftarrow$ `reasoning DAG`, $\mathrm{T} \leftarrow$ `Template`
Output: $\mathrm{C} \leftarrow$ `CoT`

$\mathrm{S} \leftarrow \mathrm{Stack}()$
$\mathrm{S.push}(\texttt{getConclusion}(\mathrm{G}))$
while $\mathrm{S} \neq \emptyset$ **do**
  $\mathrm{cur} \leftarrow \mathrm{S.pop}()$
  if $\mathrm{cur.visited}$ **then**
    $\mathrm{C.remove}(\mathrm{cur})$
  $\mathrm{cur.visited} \leftarrow \mathrm{true}$
  $\mathrm{C.append}(\texttt{makeStatement}(\mathrm{G}, \mathrm{cur}, \mathrm{T}))$
  $\mathrm{childList} \leftarrow \mathrm{getChild}(\mathrm{G}, \mathrm{cur})$
  if $\mathrm{childList} \neq \emptyset$ **then**
    $\mathrm{S.push}(\mathrm{childList})$
return $\mathrm{reverse}(\mathrm{C})$

Algorithm 2 CSP CoT Generate

Input: $\mathrm{G} \leftarrow (\mathbf{V}, \mathbf{E})$ ▷ `reasoning graph`
Output: $\mathrm{C} \leftarrow \mathrm{CoT}$

$\mathrm{X_{pre}} \leftarrow \{\mathrm{v} \mid \mathrm{v} \in \mathrm{V}, \texttt{inDegree}(\mathrm{v}) = 0\}$
$\mathrm{X_{cur}} \leftarrow \emptyset$
while True **do**
  for $\mathrm{c} \in \texttt{getOutEdge}(\mathrm{G}, \mathrm{X_{pre}})$ **do**
    $\Delta\mathrm{X} \leftarrow \texttt{getOutNeighbor}(\mathrm{G}, \mathrm{X_{pre}}, \mathrm{c})$
    $\mathrm{X_{cur}} \leftarrow \texttt{update}(\mathrm{X_{pre}}, \Delta\mathrm{X})$
    $\mathrm{C} \leftarrow \texttt{append}\ (\mathrm{C}, \langle \mathrm{X_{pre}}, \mathrm{c}, \mathrm{X_{cur}} \rangle)$
    $\mathrm{X_{pre}} \leftarrow \mathrm{X_{cur}}$
  if $\mathrm{X_{pre}} == \mathrm{X_{cur}}$ **then**
    break
return C

5.1 Datasets

Consistent with related work, we selected five logical reasoning datasets for our experiments. PrOntoQA(Saparov and He, 2022), ProofWriter(Tafjord et al., 2021), FOLIO(Han et al., 2024a), LogicalDeduction(Srivastava et al., 2023) and AR-LSAT(Zhong et al., 2022).

5.2 Reasoning Performance of Logic-Thinker

Experiment Setup

We divided the five datasets into two categories: FOL and CSP, and solved them using Logic-Thinker. In the formalization stage, we employed a few-shot approach for each dataset to guide the LLM in converting the problem into symbolic expression. The details of each dataset are shown in Table 1

For the baselines, we compared our method

DataSet	Size		Type	Formalization
	Train	Valid		
PrOntoQA	300	200	FOL	SeFOL
ProofWriter	3000	600		
FOLIO	1204	204		
LogicalDeduction	1200	300	CSP	XCSP[3]
AR-LSAT	1585	231		

Table 1: The sizes of Train and Valid split, problem type and formaliation corresponding to each logical reasoning dataset. The original PrOntoQA dataset consists of a single split of 500 samples. We used the first 300 entries as the Train split, while the remaining entries were used as the Valid split for experiments.

with three state-of-the-art (SOTA) neural-symbolic reasoning systems: Logic-LM(Pan et al., 2023), LINC(Olausson et al., 2023), and SymbolCoT(Xu et al., 2024b). Logic-LM and LINC formalize the problem using a LLM and combine it with an external solver for reasoning. SymbolCoT, on the other hand, utilizes symbolic CoT to guide the LLM reasoning directly. To ensure a fair comparison, similar to these works, we used GPT-4 for the formalization process in our experiments. Considering that all five datasets are multiple-choice questions, we use accuracy as the evaluation metric. For reproducibility, we set the temperature to 0 and select the response with the highest probability from LLMs. Additionally, to validate the gains of neuro-symbolic methods, we also report the metrics of problem reasoning. based on GPT-4 directly and GPT-4 with CoT prompt.

Results and Analysis

We report the accuracy results of Logic-Thinker and other baselines on the valid split of each dataset in Table 2. The key observations are as follows.

Overall, Logic-Thinker achieves the second-best performance on FOLIO and outperforms other baselines on all other four datasets. In terms of average accuracy, Logic-Thinker reaches the state-of-the-art (SOTA) performance and demonstrates a significant improvement (over 5% accuracy gain) compared to the second-best baseline.

Compared to directly using GPT-4 for reasoning or employing the CoT-prompting method, Logic-Thinker achieves significant accuracy improvements across five datasets, with average accuracy increasing by 24.47% and 15.68%, respectively. Our experimental results effectively demonstrate that Logic-Thinker can significantly enhance the problem-solving capabilities of LLMs on logical reasoning tasks.

Compared to other symbolic solver framework, Logic-Thinker outperforms Logic-LM and LINC across all five datasets. For the FOLIO dataset, using SeFOL as the symbolic expression, Logic-Thinker mitigates the problem of information loss through Semantic Extension achieving improvements of 1.18% and 7.60% over Logic-LM and LINC, respectively.

Compared to SymbolCoT, Logic-Thinker achieves 100% accuracy on both the ProntoQA and ProofWriter datasets, demonstrating the advantages of symbolic reasoning over LLMs in accurate reasoning. However, we observe that SymbolCoT outperforms Logic-Thinker on FOLIO by 2.23%. This is because FOLIO contains a large number of complex semantic relationships in addition to logical relations. Our approach still has some issues with information loss, which we will further discuss in the Limitation section.

5.3 Effectiveness of ThinkerCoT

Experiment Setup

We utilize Logic-Thinker to perform reasoning on the training split of each dataset, generating ThinkerCoT for correctly answered questions. In the style alignment stage, we employed Qwen2.5-72b-Instruct for style rewriting. The temperature was set to 0.7 to ensure more diverse language expressions. Additionally, we incorporated a result validation process to ensure that the reasoning results after style rewriting remained correct. For the baseline, we selected QwQ32B(Team, 2025) to generate LongCoT for the same set of questions. To ensure a fair comparison, we only retained the questions for which both methods provided correct answers as the training data. We chose Qwen2.5-7b-Instruct as the base model for SFT and conducted experiments using full fine-tuning. For training hyperparameters, we set the learning rate to 2e-5, warm-up ratio to 0.06, and max tokenslength to 8192. Both ThinkerCoT and LongCoT were used to fine-tune the model under the same settings for 5 epochs. For evaluation, we used the fine-tuned models to perform reasoning on the valid split of each dataset. We recorded the accuracy of the reasoning results and the average output tokens for both models across the five datasets to assess the effectiveness of ThinkerCoT.

Type	Method	FOL			CSP		Avg.
		PrOntoQA	ProofWriter	FOLIO	LogicalDeduction	AR-LSAT	
Neuro-Symbolic	Logic-Thinker	**100.00**	**100.00**	80.10	**99.33**	**46.75**	**85.24**
	Logic-LM	83.20	79.66	78.92	87.63	43.04	74.49
	LINC	-	98.30	72.50	-	-	-
	SymbolCoT	99.60	82.50	**83.33**	93.00	43.91	80.47
Natural Language	GPT-4	77.40	52.67	69.11	71.33	33.33	60.77
	GPT - CoT	98.79	68.11	70.58	72.25	35.06	69.56

Table 2: Comparison of the accuracy results of Logic-Thinker and other baselines on five logic reasoning datasets. We report the results of Logic-LM, LINC and SymbolCoT in their respective papers. The results of Logic-LM are those with the LLM fall back in case of recurring non-executable errors. To provide a more intuitive overall comparison, we additionally report the average accuracy of each method across five datasets. The best result per dataset are put in bold.

Results and Analysis

We report the result of the accuracy and average output tokens of ThinkerCoT and other baselines on the valid split of each dataset in Table 2. The key observations are as follows.

Firstly, compared to the base model, the model fine-tuned with ThinkerCoT demonstrates a significant improvement in logical reasoning, achieving an average accuracy improvement of 12.96%. In particular, the fine-tuned model reaches 100% accuracy in the PrOntoQA data set and exhibits substantial improvements in the Proofwriter and LD datasets, with accuracy improvements of 23.5% and 28.33%, respectively. These three datasets are based on code synthesis tasks, which primarily require logical reasoning skills rather than semantic understanding. The observed results therefore corroborate our hypothesis that ThinkerCoT effectively enhances the model's capacity for rigorous logical reasoning.

Secondly, for comparison with LongCoT, ThinkerCoT achieves a 3.6% overall accuracy improvement while significantly reducing the consumption of output tokens, requiring only 9% to 27% of the output tokens used by LongCoT. This result validates the conciseness and logical rigor of ThinkerCoT.

Ablation Study

We conducted an ablation study to evaluate the impact of style alignment process in CoT Generator. Based on the setup of experiment in 5.3, we removed the style alignment process and used it as a control group compared to the original ThinkerCoT. The results are shown in Figure 3. We observed that by introducing style alignment, the output tokens further decreases by 25.4%-47.7%, and there was

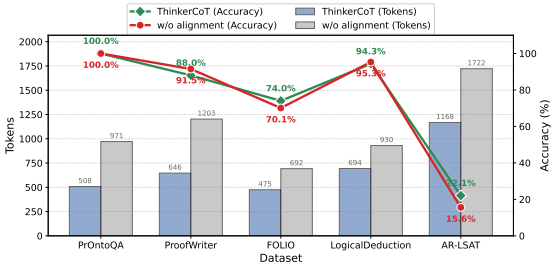


Figure 3: Comparison of the average output tokens and accuracy of ThinkerCoT and ThinkerCoT without the style alignment process.

a 6.5% improvement in accuracy on the AR-LSAT, while the other data remained stable.

5.4 How does ThinkerCoT impact LLM's Comprehensive Reasoning Capabilities?

We designed an experiment to verify whether Logic-Thinker and ThinkerCoT can enhance the comprehensive reasoning abilities of LLMs in real-world SFT stage of training a reasoning model.

We selected several public benchmarks from the domains of science, mathematics, and logic to evaluate the comprehensive reasoning capabilities of the model. For the science domain, we used GPQA(Rein et al., 2024) as the benchmark. For mathematics, we selected multiple benchmarks of varying difficulty levels, including GSM8K(Cobbe et al., 2021), MATH(Hendrycks et al., 2021), OlympiadBench(He et al., 2024), LiveBench-Math(White et al., 2025) and AIME24. For logical reasoning, we chose Multi-LogiEval(Patel et al., 2024b) and LiveBench-Reasoning(White et al., 2025) as evaluation benchmarks.

For the baseline configuration, we randomly sampled 10,000 instances from the NaturalRea-

Size	Method	FOL						CSP				Avg.	
		PrOntoQA		ProofWriter		FOLIO		LD		AR-LSAT			
		Acc	Tokens	Acc	Tokens	Acc	Tokens	Acc	Tokens	Acc	Tokens	Acc	Tokens
7B	Base Model	98.60	0.34K	64.50	0.53K	64.22	0.30K	66.00	0.45K	20.35	0.71K	62.73	0.46K
	LongCoT	88.18	3.51K	85.50	5.62K	72.55	5.14K	**95.67**	2.52K	18.61	6.84K	72.10	4.73K
	ThinkerCoT	**100.00**	0.51K	**88.00**	0.65K	**74.02**	0.47K	94.33	0.69K	**22.08**	1.17K	**75.69**	0.70K

Table 3: Comparison of ThinkerCoT and LongCoT across different logic reasoning datasets. The best accuracy result per dataset are put in bold. ThinkerCoT outperforms LongCoT with an overall accuracy improvement of 3.6% while reducing token output by 73%-91%

soning dataset (Yuan et al., 2025), which consists of reasoning chain-of-thought (CoT) data, and performed supervised fine-tuning (SFT) on the Qwen2.5-7B-Instruct model (denoted as $\mathrm{SFT}_{\mathrm{NR}}$). In parallel, we generated 5,000 ThinkerCoT and 5,000 QwQ32B LongCoT samples from five distinct logical reasoning datasets. These generated samples were then combined with the aforementioned NaturalReasoning examples, and further SFT was conducted on Qwen2.5-7B-Instruct. The resulting fine-tuned models are referred to as $\mathrm{SFT}_{\mathrm{NR+ThinkerCoT}}$ and $\mathrm{SFT}_{\mathrm{NR+LongCoT}}$ respectively.

As shown in Table 4, $\mathrm{SFT}_{\mathrm{NR+ThinkerCoT}}$ consistently outperformed the baseline models across all benchmarks in the science and logic domains. Specifically, on the Multi-LogiEval dataset, $\mathrm{SFT}_{\mathrm{NR+ThinkerCoT}}$ achieved 37.73% and 11.41% improvements over $\mathrm{SFT}_{\mathrm{NR}}$ and $\mathrm{SFT}_{\mathrm{NR+LongCoT}}$, respectively. In the mathematics domain, $\mathrm{SFT}_{\mathrm{NR+ThinkerCoT}}$ attained top performance on three datasets and secured second place on the remaining two, with marginal score differences ($< 1\%$) from the best results. Furthermore, an analysis based on difficulty levels revealed that ThinkerCoT significantly enhances the model's capability to solve more challenging mathematical problems. These findings underscore that rigorous logical reasoning contributes substantially to improving comprehensive reasoning abilities.

We further compared the average output tokens of $\mathrm{SFT}_{\mathrm{NR+ThinkerCoT}}$ and $\mathrm{SFT}_{\mathrm{NR+LongCoT}}$ across various benchmarks, with the results presented in Table 5. It can be observed that, compared to LongCoT, the model fine-tuned with ThinkerCoT significantly reduces the issue of redundant reasoning in logical-domain benchmarks, while also exhibiting varying degrees of reduction in mathematical and scientific-domain benchmarks. These findings support our argument that rigorous logical reasoning capabilities can mitigate redundancy in comprehensive reasoning tasks while enhancing reasoning effectiveness.

Domain	Benchmark	$\mathrm{SFT}_{\mathrm{NR}}$	$\mathrm{SFT}_{\mathrm{NR+ThinkerCoT}}$	$\mathrm{SFT}_{\mathrm{NR+LongCoT}}$
Science	GPQA	9.09	**14.90**	6.19
Mathmatic	GSM8K	81.80	82.26	**82.41**
	MATH	**59.36**	58.80	55.98
	OlympiadBench	26.96	**27.70**	27.56
	Livebench-Math	17.42	**21.19**	18.54
	AIME24	4.17	**4.58**	4.12
Logic	Multi-LogiEval	32.09	**69.82**	58.41
	Livebench-Reasoning	30.67	**34.00**	16.67

Table 4: The accuracy comparison of different models across various benchmarks. The best accuracy per benchmark are put in bold.

Domain	Benchmark	$\mathrm{SFT}_{\mathrm{NR+ThinkerCoT}}$	$\mathrm{SFT}_{\mathrm{NR+LongCoT}}$	Tokens Reduction
Science	GPQA	0.96k	7.86k	87.79%
Mathmatic	GSM8K	0.30k	0.50k	40.00%
	MATH	0.69k	0.82k	15.85%
	OlympiadBench	0.98k	1.23k	20.33%
	Livebench-Math	0.92k	3.67k	74.93%
	AIME24	1.25k	1.42k	11.97%
Logic	Multi-LogiEval	0.46k	2.93k	84.30%
	Livebench-Reasoning	1.23k	13.90k	91.15%

Table 5: Comparison of the average output tokens of different models across various benchmarks.

5.5 Case Study

In this section, we conducted a case study to explore the effectiveness of ThinkerCoT compared to LongCoT. We selected a question from PrOntoQA dataset and presented the details of ThinkerCoT and LongCoT, as shown in Figure 6 of Appendix D. ThinkerCoT arrived at the correct conclusion through concise and rigorous reasoning. In contrast, the reasoning errors in LongCoT can be attributed to the following: 1. Correctly deducing "Sam is a rompus" but failing to proceed further, prematurely altering the reasoning path. 2. Hallucination issues, positing "vumpus is zumpus" as a premise, which was not mentioned in the context. Despite utilizing a substantial number of tokens for reflection and verification afterward, it ultimately reasoned incorrectly. These observations further confirm the

issue of logical incoherence of LongCoT while also demonstrating the rigorous reasoning capability of ThinkerCoT.

6 Conclusion

We introduce a novel neuro-symbolic logic reasoning framework, named Logic-Thinker, which can transform the reasoning process of solvers into a rigorous and concise CoT, referred to as ThinkerCOT. The proposed framework consists of three modules: Formulator, LogicSolver and CoT-Generator. We categorize logical reasoning problems into two types and implement them respectively in three modules. The experimental results demonstrated that Logic-Thinker outperforms other state-of-the-art frameworks in logical reasoning capability. Additionally, ThinkerCoT achieves a 3.6% improvement in accuracy while reducing token output by 73%-91% compaed to LongCoT. Furthermore, the logical reasoning data synthesized through ThinkerCoT can effectively enhance the overall reasoning ability of LLMs, such as in mathematics and scientific reasoning, which demonstrates the significant value of our methods in practical LLM training tasks.

Limitations

Our current work has some limitations. First, our formalization process still suffers from a certain degree of information loss, which affects the accuracy on datasets like FOLIO and AR-LSAT, making it less than ideal. In the future, we plan to leverage more SPG semantic features to improve the quality of the formalization process.

Furthermore, we have not yet achieved a unified formalization process for FOL and CSP. Our long-term goal is to achieve a unified symbolic representation for two types of problems and employ a single unified solver along with a CoT generation pipeline. This would enable the framework to provide a unified solution for logical reasoning tasks.

References

Frederic Boussemart, Christophe Lecoutre, Gilles Audemard, and Cédric Piette. 2024. Xcsp3: An integrated format for benchmarking combinatorial constrained problems. *Preprint*, arXiv:1611.03398.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. 2024. A survey on evaluation of large language models. *ACM Trans. Intell. Syst. Technol.*, 15(3).

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. 2025. Do not think that much for 2+3=? on the overthinking of o1-like llms. *Preprint*, arXiv:2412.21187.

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

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Mehdi Fatemi, Banafsheh Rafiee, Mingjie Tang, and Kartik Talamadupula. 2025. Concise reasoning via reinforcement learning. *Preprint*, arXiv:2504.05185.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, Lucy Sun, Alexander Wardle-Solano, Hannah Szabó, Ekaterina Zubova, Matthew Burtell, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, and 16 others. 2024a. FOLIO: Natural language reasoning with first-order logic. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 22017–22031, Miami, Florida, USA. Association for Computational Linguistics.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2024b. Token-budget-aware llm reasoning. *arXiv preprint arXiv:2412.18547*.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. 2024. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *Preprint*, arXiv:2402.14008.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Yu Kang, Xianghui Sun, Liangyu Chen, and Wei Zou. 2024. C3ot: Generating shorter chain-of-thought without compromising effectiveness. *CoRR*, abs/2412.11664.

Enkelejda Kasneci, Kathrin Seßler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, George Louis Groh, Stephan Günnemann, Eyke Hüllermeier, Stephan Krusche, Gitta Kutyniok, Tilman Michaeli, Claudia Nerdel, Jürgen

Pfeffer, Oleksandra Poquet, Michael Sailer, Albrecht Schmidt, Tina Seidel, and 4 others. 2023. Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and Individual Differences*.

Lei Liang, Mengshu Sun, Zhengke Gui, Zhongshu Zhu, Zhouyu Jiang, Ling Zhong, Peilong Zhao, Zhongpu Bo, Jin Yang, and 1 others. 2024. Kag: Boosting llms in professional domains via knowledge augmented generation. *arXiv preprint arXiv:2409.13731*.

Haotian Luo, Li Shen, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Naiqiang Tan, Xiaochun Cao, and Dacheng Tao. 2025. O1-pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning. *Preprint*, arXiv:2501.12570.

Tergel Munkhbat, Namgyu Ho, Seo Hyun Kim, Yongjin Yang, Yujin Kim, and Se-Young Yun. 2025. Self-training elicits concise reasoning in large language models. *Preprint*, arXiv:2502.20122.

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*, pages 5153–5176, Singapore. Association for Computational Linguistics.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

OpenAI. 2024. Learning to reason with llms.

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

Bhrij Patel, Souradip Chakraborty, Wesley A. Suttle, Mengdi Wang, Amrit Singh Bedi, and Dinesh Manocha. 2024a. Aime: Ai system optimization via multiple llm evaluators. *Preprint*, arXiv:2410.03131.

Nisarg Patel, Mohith Kulkarni, Mihir Parmar, Aashna Budhiraja, Mutsumi Nakamura, Neeraj Varshney, and Chitta Baral. 2024b. Multi-logieval: Towards evaluating multi-step logical reasoning ability of large language models. *ArXiv*, abs/2406.17169.

Charles Prud'homme and Jean-Guillaume Fages. 2022. Choco-solver: A java library for constraint programming. *Journal of Open Source Software*, 7(78):4708.

Xiaoye Qu, Yafu Li, Zhaochen Su, Weigao Sun, Jianhao Yan, Dongrui Liu, Ganqu Cui, Daizong Liu, Shuxian Liang, Junxian He, and 1 others. 2025. A survey of efficient reasoning for large reasoning models: Language, multimodality, and beyond. *arXiv preprint arXiv:2503.21614*.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. Gpqa: A graduate-level google-proof qa benchmark. *Preprint*, arXiv:2311.12022.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

Abulhair Saparov and He He. 2022. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. *ArXiv*, abs/2210.01240.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, Agnieszka Kluska, Aitor Lewkowycz, Akshat Agarwal, Alethea Power, Alex Ray, Alex Warstadt, Alexander W. Kocurek, Ali Safaya, Ali Tazarv, and 431 others. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Trans. Mach. Learn. Res.*, 2023.

DiJia Su, Hanlin Zhu, Yingchen Xu, Jiantao Jiao, Yuandong Tian, and Qinqing Zheng. 2025. Token assorted: Mixing latent and text tokens for improved language model reasoning. *CoRR*, abs/2502.03275.

R Sutton and A Barto. 1998. *Reinforcement Learning:An Introduction*. Reinforcement Learning:An Introduction.

Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. 2021. ProofWriter: Generating implications, proofs, and abductive statements over natural language. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3621–3634, Online. Association for Computational Linguistics.

Qwen Team. 2025. Qwq-32b: Embracing the power of reinforcement learning.

Yiming Wang, Zhuosheng Zhang, Pei Zhang, Baosong Yang, and Rui Wang. 2024. Meta-reasoning: Semantics-symbol deconstruction for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 622–643, Bangkok, Thailand. Association for Computational Linguistics.

Yue Wang, Qiuzhi Liu, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Linfeng Song, Dian Yu, Juntao Li, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. 2025. Thoughts are all over the place: On the underthinking of o1-like llms. *Preprint*, arXiv:2501.18585.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh-Agrawal, Sandeep Singh Sandha, Siddartha Venkat Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. 2025. Livebench: A challenging, contamination-free LLM benchmark. In *The Thirteenth International Conference on Learning Representations*.

Fangzhi Xu, Zhiyong Wu, Qiushi Sun, Siyu Ren, Fei Yuan, Shuai Yuan, Qika Lin, Yu Qiao, and Jun Liu. 2024a. Symbol-LLM: Towards foundational symbol-centric interface for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13091–13116, Bangkok, Thailand. Association for Computational Linguistics.

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

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 23 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Peng Yi, Lei Liang, Da Zhang, Yong Chen, Jinye Zhu, Xiangyu Liu, Kun Tang, Jialin Chen, Hao Lin, Leijie Qiu, and Jun Zhou. 2024. Kgfabric: A scalable knowledge graph warehouse for enterprise data interconnection. *Proc. VLDB Endow.*, 17(12):3841–3854.

Weizhe Yuan, Jane Yu, Song Jiang, Karthik Padthe, Yang Li, Dong Wang, Ilia Kulikov, Kyunghyun Cho, Yuandong Tian, Jason E. Weston, and Xian Li. 2025. Naturalreasoning: Reasoning in the wild with 2.8m challenging questions. *CoRR*, abs/2502.13124.

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

Tong Zhu, Xiaoye Qu, Daize Dong, Jiacheng Ruan, Jingqi Tong, Conghui He, and Yu Cheng. 2024. LLaMA-MoE: Building mixture-of-experts from LLaMA with continual pre-training. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15913–15923, Miami, Florida, USA. Association for Computational Linguistics.

A Prompt Examples

In this section, we provide examples of prompts used by the formulator module of Logic-Thinker. The prompt consists of five components: a task description, context, question, options, and a domain-specific symbolic program. For simplicity, we present only one example for each type of logic reasoning problems in the following sections.

A.1 Prompt for FOL Formalization

Task Description: You are a logic expert specializing in translating natural language problems into First-Order Logic (FOL) expressions. Given a logical reasoning problem consisting of a premises and conclusion, perform the following steps:
Translation Steps:
1. Summarize the predicates from the text (both premises and conclusion)
2. Define the individual constants from the text and symbolize them.
3. Based on the outputs from steps 1 and 2, combine them with logical symbols to form first-order predicate expressions. Each sentence in the text should correspond to one expression. You can use the following logical symbols:
(... more context here ...)
4. Based on the outputs from steps 1, summarize the hierarchical relationships within predicates and describe them using subClassOf.
Output Format
(... more context here ...)
Input:
Premises: [BG] There are four seasons in a year: Spring, Summer, Fall, and Winter. All students who want to have a long vacation love summer the most. Emma's favorite season is summer. Mia's favorite season is not the same as Emma's. James wants to have a long vacation.
Conclusion: James's favorite season is summer.
Output:
Define Predicates:
Season(x) ::: x is a season.
(... more context here ...)
Define Constants:
Emma ::: represent the student of Emma
(... more context here ...)
Translate Premises:
Emma's favorite season is summer. ::: Favorite(Emma, Summer)
(... more context here ...)
Translate Conclusion:
James's favorite season is summer. ::: Favorite(James, Summer)
Hierarchical Relationships:
subClassOf(Spring, Season)
(... more context here ...)

A.2 Prompt for CSP Formalization

Task Description: Given a constraint satisfaction problem that includes a context and a question in natural language. The task is to describe the problem using the XCSP3 extension modeling language, consisting three parts: Variables, Constraints, and Options. You can use the following operators:
Arithmetic Opposite: neg(x) ::: -x
Logical Not: not(x) ::: ¬x
(... more context here ...)
Context: On Tuesday Vladimir and Wendy each eat exactly four separate meals: breakfast, lunch, dinner, and a snack. (... more context here ...) Wendy eats an omelet for lunch.
Question: Vladimir must eat which one of the following foods?
Option:
(A) fish
(B) hot cakes
(C) macaroni
(D) omelet
(E) poached eggs
Program:
(... more context here ...)

B CoT Generation for FOL Problem

As shown in Figure 4(a), a concrete problem example is presented from the FOLIO dataset. We first perform formalization, abstracting the content and the question into symbolic representations. The content is divided into Facts and Rules, as illustrated in Figure 4(b). Subsequently, we employ Fol Solver to deduce the problem and record the complete solution tree, as shown in Figure 4(c).

Next, we convert the tree into a graph structure and apply several optimizations, including intermediate result removal(indicated by the red dashed box in Figure 4(c)) and duplicate node merging(marked by the green dashed box in Figure 4(c)). As a result, we obtain a simplified directed acyclic graph (DAG), shown in Figure 4(d), where the assumption node serves as the starting node, the conclusion node as the end node, and the remaining nodes represent deduction nodes.

Finally, we apply Algorithm 1 to traverse the graph and generate the reasoning statement for each type of node based on the Chain-of-Thought (CoT) template. This process results in the complete CoT output, as shown in Figure 4(e).

C CoT Generation for CSP Problem

As shown in Figure 5, we selected a specific problem from the AR-LSAT dataset. First, we formalized the Context and Question into the XCSP^3 format using the formulator. The content is divided into Variables, Constraints, and Options, as illustrated in Figure 5(Formalization). Then, we used CSPSolver to solve the problem, obtained the intermediate reasoning process, and converted it into a graph, as shown in Figure 5(ReasoningGraph). Finally, we traversed ReasoningGraph following the process outlined in Algorithm 2 to generate the CoT. The traversal process, marked by colored edges, is shown in Figure 5(ReasoningGraph). The final CoT is presented in Figure 5(CoT).

D Comparison of ThinkerCoT and LongCoT

We select a specific problem from the test split of PrOntoQA dataset. We utilize the model fine-tuned on ThinekrCoT and LongCoT to predict the problem respectively. The completions of the two models are shown in Figure 6.

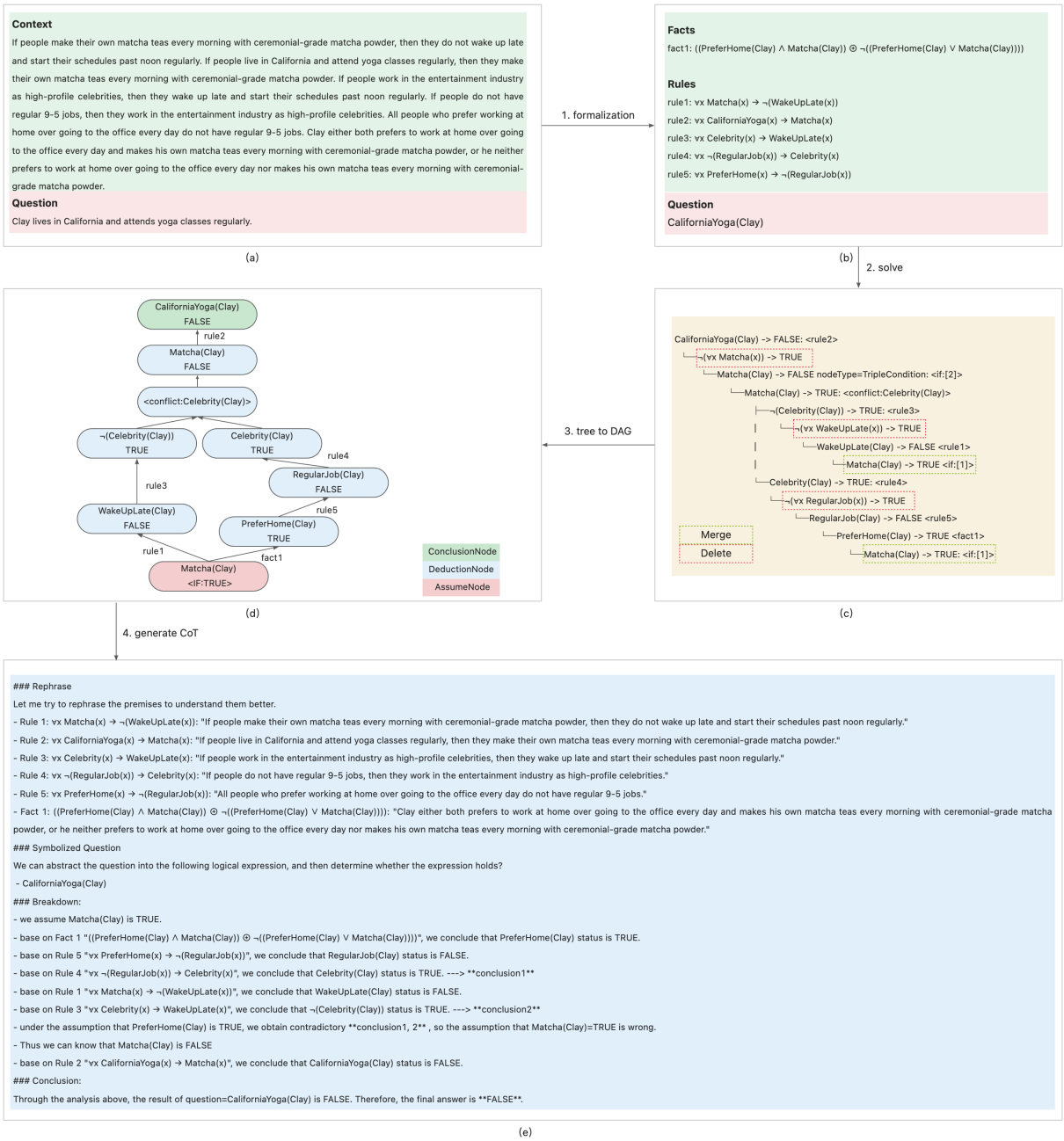


Figure 4: A complete example of FOL CoT generation which consists of five parts: (a) the original question (b) the symbolic representation after formalizing (c) the complete record tree of SeFolSolver (d) the simplified DAG (e) the final complete CoT.

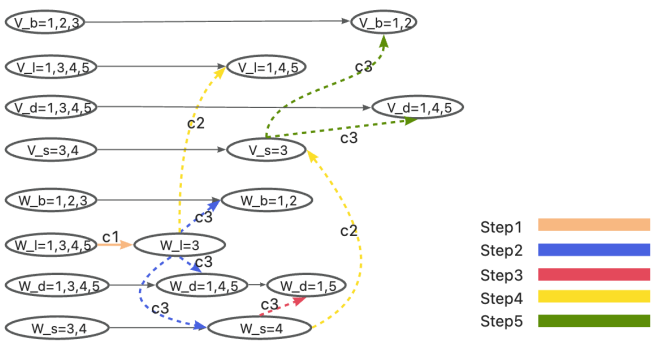
Problem

Context: On Tuesday Vladimir and Wendy each eat exactly four separate meals: breakfast, lunch, dinner, and a snack. The following is all that is known about what they eat during that day: At no meal does Vladimir eat the same kind of food as Wendy. Neither of them eats the same kind of food more than once during the day. For breakfast, each eats exactly one of the following: hot cakes, poached eggs, or omelet. For lunch, each eats exactly one of the following: fish, hot cakes, macaroni, or omelet. For dinner, each eats exactly one of the following: fish, hot cakes, macaroni, or omelet. For a snack, each eats exactly one of the following: fish or omelet. Wendy eats an omelet for lunch.
Question: Vladimir must eat which one of the following foods?
(A) fish (B) hot cakes (C) macaroni (D) omelet (E) poached eggs

Formalization

```
<instance format="XCSP3" type="CSP">
  <variables>
    <var id="V_b" note="Vladimir's breakfast:1=hot cakes, 2=poached eggs, 3=omelet"> 1 2 3 </var>
      ... ...
    <var id="V_s" note="Vladimir's snack:3=omelet, 4=fish"> 3 4 </var>
    <var id="W_b" note="Wendy's breakfast:1=hot cakes, 2=poached eggs, 3=omelet"> 1 2 3 </var>
      ... ...
    <var id="W_s" note="Wendy's snack:3=omelet, 4=fish"> 3 4 </var>
  </variables>
  <constraints>
    <intension note="Wendy eats an omelet for lunch"> eq(W_l, 3)</intension>
    <intension note="At no meal does Vladimir eat the same kind of food as Wendy"> ne(V_b, W_b) </intension>
    ... ...
    <allDifferent note="Neither of them eats the same kind of food more than once during the day">
      <list> V_b V_l V_d V_s </list>
    </allDifferent>
    ... ...
  </constraints>
  <options>
    <question> Vladimir must eat which one of the following foods? </question>
    ... ...
    <option id="D" type="must_sat" note="Vladimir must eat omelet">
      or(eq(V_b, 3), eq(V_l, 3), eq(V_d, 3), eq(V_s, 3))
    </option>
    ... ...
  </options>
</instance>
```

ReasoningGraph



CoT

Let's rephrase the constraint for better understanding:
- c1: Wendy eats an omelet for lunch
- c2: At no meal does Vladimir eat the same kind of food as Wendy
- c3: Neither of them eats the same kind of food more than once during the day
Breakdown:
To figure this out, we analyze the given constraints step-by-step:
Define variables:
- V_b: {1,2,3} , Vladimir's breakfast:1=hot cakes, 2=poached eggs, 3=omelet
... ...
- V_s: {3,4} , Vladimir's snack:3=omelet, 4=fish
- W_b: {1,2,3} , Wendy's breakfast:1=hot cakes, 2=poached eggs, 3=omelet
... ...
- W_s: {3,4} , Wendy's snack:3=omelet, 4=fish
Let's begin to analyse according to constraints:
1. Base on c1, we conclude that: ...,W_l:3,...
2. Base on c3, we conclude that: ...,W_b:{1,2},W_d:{1,4,5},W_s:{4},...
3. Base on c3, we conclude that: ...,W_d:{1,5},...
4. Base on c2: we conclude that: ...,V_l:{1,4,5},V_s:{3},...
5. Base on c3: we conclude that: ...,V_b:{1,2},V_d:{1,4,5},...
Option Analysis:
Option A("Vladimir must eat fish")
... ...

Figure 5: An example of CSP problem, which selected from the AR-LSAT dataset showcases the results of three stages in the LogicThinker pipeline: formalization, reasoning, and CoT generation.The ReasoningGraph highlights the process of generating the CoT.

Problem

Problem: Zumpuses are dull. Every vumpus is not transparent. Vumpuses are numpuses. Numpuses are blue. Numpuses are wumpuses. Wumpuses are liquid. Each wumpus is a tumpus. Tumpuses are not spicy. Tumpuses are rompuses. Each rompus is not dull. Rompuses are yumpuses. Every yumpus is floral. Every yumpus is an impus. Impuses are hot. Each impus is a jompus. Every jompus is large. Jompuses are dumpuses. Sam is a vumpus. Is the following statement true or false? Sam is not dull.
Answer: True.

ThinkerCoT

To solve this problem, let's break down the given premises and follow the logical steps to determine if Sam is not dull.
Initial Setup:
We have a series of statements about different types of creatures and their properties. We need to determine if Sam, who is a vumpus, is not dull.

Premises:
1. Zumpuses are dull. **2.** Every vumpus is not transparent. **3.** Vumpuses are numpuses. **4.** Numpuses are blue. **5.** Numpuses are wumpuses. **6.** Wumpuses are liquid. **7.** Each wumpus is a tumpus. **8.** Tumpuses are not spicy. **9.** Tumpuses are rompuses. **10.** Each rompus is not dull. **11.** Rompuses are yumpuses. **12.**. Every yumpus is floral. **13.** Every yumpus is an impus. **14.** Impuses are hot. **15.** Each impus is a jompus. **16.** Every jompus is large. **17.** Jompuses are dumpuses. **18.** Sam is a vumpus.

Step-by-Step Analysis:
1. Sam is a vumpus (Premise 18).
2. Vumpuses are numpuses (Premise 3): - Since Sam is a vumpus, Sam is also a numpus.
3. Numpuses are wumpuses (Premise 5): - Since Sam is a numpus, Sam is also a wumpus.
4. Each wumpus is a tumpus (Premise 7): - Since Sam is a wumpus, Sam is also a tumpus.
5. Tumpuses are rompuses (Premise 9): - Since Sam is a tumpus, Sam is also a rompus.
6. Each rompus is not dull (Premise 10): - Since Sam is a rompus, Sam is not dull.

Conclusion:
By following the chain of logical implications, we can conclude that Sam, being a vumpus, is also a numpus, wumpus, tumpus, and rompus. Since every rompus is not dull, Sam is not dull.

Final Answer:
True

LongCoT

<think>
Okay, let's see. I need to figure out if the statement "Sam is not dull" is true or false based on the given information. Let me start by listing out all the information provided and then see how it connects to Sam.
First, the given statements:
1. Zumpuses are dull. 2. Every vumpus is not transparent. 3. Vumpuses are numpuses. 4. Numpuses are blue. 5. Numpuses are wumpuses. 6. Wumpuses are liquid. 7. Each wumpus is a tumpus. 8. Tumpuses are not spicy. 9. Tumpuses are rompuses. 10. Each rompus is not dull. 11. Rompuses are yumpuses. 12. Every yumpus isfloral. 13. Every yumpus is animpus. 14. Impuses are hot. 15. Eachimpus is a jompus. 16. Every jompus is large. 17. Jompuses are dumpuses. 18. Sam is a vumpus.

Now, the question is whether "Sam is not dull" is true or false. Let's break this down step by step.

Starting with Sam: From statement 18, Sam is a vumpus. Then, looking at the other statements, we can see that vumpuses are numpuses (statement 3). So Sam is a vumpus, which is a numpus.
Now, numpuses are wumpuses (statement 5). So Sam is also a wumpus. Then, wumpuses are liquid (statement 6), so Sam is liquid. Each wumpus is a tumpus (statement 7), so Sam is a tumpus.
Tumpuses are rompuses (statement 9). **Therefore, Sam is a rompus. Now, looking at statement 10: Tumpuses are not spicy. Since Sam is a tumpus, that means Sam is not spicy. But that might not be directly relevant to the question about being dull.**

Next,rompuses are yumpuses (statement 11). So Sam is a yumpus. Then, every yumpus is floral (statement 12) and every yumpus is an impus (statement 13). Therefore, Sam is both florally and an impus. Since impuses are hot (statement 14), Sam must be hot. Also, eachimpus is a jompus (statement 15), so Sam is a jompus. And every jompus is large (statement 16), so Sam is large. Finally, jompuses are dumpuses (statement 17), so Sam is a dumpus.

Now, the key point here is whether Sam is dull. The first statement says "Zumpuses are dull." But does that mean all zumpuses are dull, or just that zumpuses can be dull? The wording is a bit ambiguous. However, in the given information, the first statement is "Zumpuses are dull." Since zumpuses are a type of vumpus (from statement 3, since vumpuses are numpuses, and numpuses are wumpuses, etc.), but wait, actually, the first statement says "Zumpuses are dull." Wait, but in the chain, Sam is a vumpus, which is a numpus, which is a wumpus, which is a tumpus, which is a rompus, which is a yumpus. So Sam is a yumpus. But yumpus is a type of vumpus? Wait, let me check the statements again.

Wait, statement 3 says "Vumpuses are numpuses." So vumpus is a subset of numpus. Then statement 5 says "Numpuses are wumpuses," so numpus is a subset of wumpus. So vumpus is a wumpus. Then statement 11 says "Rompuses are yumpuses," so rompus is a subset of yumpus. Therefore, yumpus is a type ofrompus, which is a type of wumpus, which is a type of numpus, which is a type of vumpus. So Sam is a vumpus, which is a numpus, which is a wumpus, which is a tumpus, which is arompus, which is a yumpus. So Sam is a yumpus.

Now, the first statement says "Zumpuses are dull." Since Sam is a vumpus, which is a numpus, which is a wumpus, which is a tumpus, which is a rompus, which is a yumpus. So Sam is a yumpus. But the first statement says "Zumpuses are dull." Since zumpuses are a type of vumpus, and vumpus is a type of numpus, etc., but the first statement is a general statement about zumpuses. However, the problem is that the first statement might not be saying that all zumpuses are dull, but just that zumpuses are dull in general. **Since Sam is a vumpus, which is a zumpuse,** then according to the first statement, zumpuses are dull, so Sam would be dull. But wait, the question is whether Sam is not dull. So if the first statement is that zumpuses are dull, then Sam, being a zumpus, would be dull, making the statement "Sam is not dull" false. But hold on, maybe there's a contradiction or another rule that could override this?

Wait, let me check again. The first statement is "Zumpuses are dull." Since zumpuses are a type of vumpus, and vumpus is a type of numpus, etc., so all the way down to Sam being a yumpus, which is a type of vumpus. Therefore, Sam is a zumpus, so according to the first statement, zumpuses are dull, so Sam must be dull. Therefore, the statement "Sam is not dull" would be false. But wait, maybe there's another rule that could make Sam not dull?

.....
Therefore, I think the answer is false.
</think>

Figure 6: An example of ThinkerCoT and LongCoT in PrOntoQA, where ThinkerCoT provides the correct answer, whereas LongCoT does not. The specific reasons for LongCoT's errors are highlighted in **bold red**, which include: 1. Correctly deducing "Sam is a rompus" but failing to proceed further, prematurely altering the reasoning path. 2. Hallucination issues, positing "vumpus is zumpus" as a premise, which was not mentioned in the context. Despite utilizing a substantial number of tokens for reflection and verification afterward, it ultimately reasoned incorrectly.