

PrefRAG: Correcting Semantic Errors in Auto-Formalization for Logical Reasoning with Program Preference RAG

Yuyin Zhou, Yongmei Liu*

Department of Computer Science, Sun Yat-sen University, Guangzhou 510006, China
zhouyy89@mail2.sysu.edu.cn; ymliu@mail.sysu.edu.cn

Abstract

Recent advances in large language models (LLMs) have spurred interest in neuro-symbolic methods for logical reasoning based on auto-formalization, where LLMs first formalize problems into symbolic programs, for solvers to perform reasoning over. However, existing auto-formalization methods remain prone to both syntactic and semantic errors. Specifically, the absence of a program-level semantic verification mechanism leaves semantic errors largely unaddressed. In this paper, we propose a novel approach to semantic error correction via program preference retrieval-augmented generation (RAG). First, we conduct an in-depth analysis of semantic error patterns, and then automatically synthesize **SemanticPref**, a program preference dataset to model these patterns. Using the dataset as the knowledge base, we introduce **PrefRAG**, a general RAG framework for refinement in auto-formalization, which enables LLMs to detect and repair syntactic and semantic errors¹. Extensive evaluations across both in-distribution (ID) benchmarks (i.e., AR-LSAT and FOLIO) and out-of-distribution (OOD) datasets show that PrefRAG consistently outperforms strong baselines, achieving an average accuracy improvement of 2.39% on ID and 6.23% on OOD datasets.

1 Introduction

Recent advancements in large language models (LLMs) have demonstrated remarkable capabilities in various natural language processing (NLP) tasks (OpenAI et al., 2024; DeepSeek-AI et al., 2025). However, complex logical reasoning, which requires LLMs to make multiple steps of inferences over a set of premises, remains a significant challenge. In addition to prompting methods like

Chain-of-Thought (CoT) (Wei et al., 2022) and Tree-of-Thought (ToT) (Yao et al., 2023), as well as training approaches (Jiao et al., 2024; Feng et al., 2024), a neuro-symbolic paradigm based on auto-formalization is regarded as a promising alternative (Ye et al., 2023; Olausson et al., 2023). Migrated from mathematical reasoning, this method is a two-phase process: in auto-formalization phase, LLMs generate symbolic formulations for a reasoning problem, which is later executed by symbolic solvers in symbolic reasoning phase. This pipeline reserves the auto-formalization subtask by leveraging LLMs’ NLP capability, offloads the reasoning subtask from LLMs to symbolic solvers, and thereby guarantees the rigor of the reasoning process as long as the formalization is accurate.

However, auto-formalization with LLMs faces crucial challenges. Due to hallucinations, LLMs often produce syntactic and semantic errors in the formalization phase. For syntactic errors, which can be detected by error messages from solvers, various methods have been proposed. Logic-LM (Pan et al., 2023) introduces a self-refinement module into the pipeline to iteratively correct inexecutable programs with their error messages. LTRAG (Hu et al., 2025) enhances the framework with thought-guided RAG. Specifically, LTRAG builds its KBs by semi-automatically constructing examples that include annotated formulas and translation processes. Nevertheless, these methods heavily depend on the error messages from the solvers, which often provide limited feedback and are insufficient for LLMs to locate errors. For semantic errors, existing methods mainly compare models of a natural language (NL) sentence with those of its formalization. For example, (Raza and Milic-Frayling, 2025) proposes SSV (Semantic Self-Verification), where an LLM generates models for a NL sentence, and a SAT solver verifies whether the models satisfy the formalization; (Ryu et al., 2025) introduces CLOVER, which selects from multiple formaliza-

*Corresponding author.

¹<https://github.com/syusic/PrefRAG>

tions by generating models of formalizations with SAT solver and verifying whether the models satisfy the NL sentence with LLMs. However, these mechanisms only focus on constraint-level formulas, rather than program-level formalizations for the reasoning problems. Furthermore, a systematic analysis of semantic errors remains underexplored.

Toward this end, we present a general RAG framework that targets semantic errors in auto-formalization with a knowledge base (KB) of program preference pairs. **First**, to establish a rigorous foundation, we **systematically summarize a taxonomy of semantic error patterns**: declaration errors, missing information, inconsistent constraints and mismatched predicates. The error taxonomy is derived from primarily two sources: common LLM failure modes in auto-formalization (e.g., hallucinations and insufficient NL understanding), and general structures of Z3 programs. Hence, the classification is domain-independent. **Second**, to model these semantic error patterns, we **propose to reflect these semantic error patterns with preference pairs**. Leveraging rule-based operations implemented with regular expressions and LLMs, we **automatically synthesize SemanticPref, a preference dataset** with a rich and natural distribution of semantic errors. **Third**, with SemanticPref as the KB, we **adopt a Retrieval-Augmented Generation (RAG) paradigm to enhance the self-refinement module in auto-formalization, and propose PrefRAG for logical reasoning**. In contrast to prior methods that only refine the programs when parsing errors occur, our framework empowers an LLM to decide when to correct the program or to exit the refinement loop. We conduct a thorough evaluation of the framework not only on in-distribution (ID) benchmarks AR-LSAT (Zhong et al., 2022) and FOLIO (Han et al., 2024), but also on out-of-distribution (OOD) benchmarks: LogicalDeduction from Big-bench (Srivastava et al., 2023), and ProverQA (Qi et al., 2025). Our experimental results show that the framework achieves the state-of-the-art performance compared to the best baselines, with an average improvement in accuracy of 2.39% on ID datasets and 6.23% on OOD datasets. Further analysis reveals that our approach is significantly more effective at addressing syntactic and semantic errors.

To summarize, the key contributions of this paper are as follows:

A systematic taxonomy for semantic errors in auto-formalization for logical reasoning. Based

on a structural analysis, we categorize semantic errors into four distinct categories: declaration errors, missing information, inconsistent constraints and mismatched predicates. To the best of our knowledge, we are the first to fill this gap, laying a foundation for future improvement.

SemanticPref: an automatically generated preference dataset. We introduce the use of program preference pairs to model semantic errors. We define several rule-based operations to construct the preference pairs, synthesizing 2921 and 3091 samples of preference pairs on AR-LSAT and on FOLIO respectively.

PrefRAG: a RAG framework for correcting semantic errors in auto-formalization. Utilizing SemanticPref as the KB, we adopt a RAG paradigm for the self-refinement module. The evaluation shows that it achieves state-of-the-art performance on both ID and OOD datasets.

2 Related Work

Logical reasoning with LLMs. To enhance the reasoning capabilities of LLMs, existing approaches can be broadly categorized into three main types. The first category is **in-context learning methods**. These methods construct detailed prompting pipelines for reasoning. Chain-of-Thought (CoT) (Wei et al., 2022), Self-Consistency (Wang et al., 2023) and Tree-of-Thought (Yao et al., 2023) are the representatives. Following these methods, subsequent approaches extend the reasoning structures. Graph-of-Thought (Besta et al., 2024) utilizes a graph structure to represent pathways of thoughts and knowledge. Symb-CoT (Xu et al., 2024) and Logic-of-Thought (Liu et al., 2025) add symbolic languages to enhance the reasoning traces. Secondly, **fine-tuning methods** use logic-based techniques to automatically generate large-scale synthetic datasets, then pre-train or fine-tune LLMs to enhance their inherent logical reasoning capabilities (Morishita et al., 2024; Jiao et al., 2024). The third is **neuro-symbolic methods** based on search, or auto-formalization as presented in the Introduction. RAP (Hao et al., 2023) views the reasoning process as planning, and leverages Monte Carlo Tree Search (MCTS) to address the process. Aristotle (Xu et al., 2025) adopts a decompose-search-resolve framework to handle logical reasoning tasks.

Auto-formalization with LLMs. This neuro-symbolic paradigm stems from mathematical rea-

soning, and is later applied to logical reasoning. For mathematical reasoning, methodologies have advanced from few-shot prompting to RAG frameworks. CRAMF (Lu et al., 2025) designs an ontology of the KB to better represent the structure of mathematical concepts. DRIFT (Zhang et al., 2025) decomposes informal statements into subqueries to improve the subsequent premise retrieval. For logical reasoning, in addition to the prompting methods and the RAG framework presented in the Introduction, recent studies fine-tune LLMs to improve formalization. (Yang et al., 2024) presents a large-scale dataset of sentence-level NL-FOL pairs and a fine-tuned model LogicLLAMA. LogicPO (Viswanadha et al., 2025) introduces the first preference dataset for NL-FOL auto-formalization, but its preference labels are judged only by programs’ execution results, lacking an in-depth analysis of semantic errors.

3 SemanticPref: Reflect Semantic Error Patterns with Preference Pairs

Z3 Solver (de Moura and Bjørner, 2008) is a Satisfiability Modulo Theories (SMT) solver from Microsoft Research, which can efficiently solve constraint satisfaction (CSP), Boolean satisfiability (SAT) and most first-order logic (FOL) reasoning problems. Thus, we encode logical reasoning problems with the Z3 Python API for versatility.

We first conduct an analysis of Z3 Python programs’ general structure, and identify possible semantic error patterns. Then, we construct program preference pairs with regular expressions and LLMs to reflect possible semantic error patterns, proposing a dataset **SemanticPref**.

3.1 Program Error Patterns

A Z3 program for a reasoning problem mainly consists of three sections: Declaration, Constraints, and Query. In Declaration, sorts for entities and predicates are declared. In Constraints, explicit and implicit premises in the given context are listed. In Query, Python functions are defined to decide the answer with SAT checks. Semantic errors mostly appear in the first two sections, and those in Declaration remain unable to be located with existing semantic verifications.

The majority of semantic error patterns can be concluded as follows:

- **Declaration errors.** A program might fail to execute when using unsuitable sorts for enti-

ties. For example, if a Z3 function is declared with integer sort as the domain, and subsequent constraints involving universal quantifiers are asserted over this function, the solver may inevitably attempt to reason over the infinite integer domain, which consequently leads to execution failure.

- **Missing Information.** LLMs might omit some constraints, especially implicit constraints in formalization. Implicit constraints refer to the rules and knowledge not explicitly clarified in the context, such as how people are adjacent when they sit around a table. The omission of these knowledge would harm the accuracy of reasoning results.
- **Inconsistent Constraints.** Due to the issues of hallucinations and deficient reasoning abilities, LLMs may sometimes produce Z3 constraints that are not equivalent to the NL statements. These errors sometimes cannot be detected by the program parsing result.
- **Mismatched Predicates.** In some cases, LLMs might use predicate names that are misaligned with the declaration, but with equivalent semantics, due to hallucinations. This type of error might appear when the predicates have complicated semantics.

Note that these error patterns are concluded based on the structure of Z3 programs and LLM hallucinations, rather than specific data patterns of reasoning tasks. Therefore, we can claim that these error patterns do not rely on specific datasets.

3.2 Construction of Program Preference Pairs

We propose **SemanticPref**, a dataset that models semantic error patterns with program preference pairs. In this section, we will first give definitions of rule-based operations to construct programs corresponding to the semantic error patterns, and present the data construction pipeline.

The preference pairs can be formalized as $(NL, P_{pref}, P_{rejected})$, where NL is the NL context and question for the reasoning problem, and $P_{pref}, P_{rejected}$ are the preferred and rejected Z3 programs for the problem, respectively. Our data is built upon AR-LSAT and FOLIO.

We first define several rule-based operations to synthesize error programs given a reasoning problem NL and the corresponding correct program

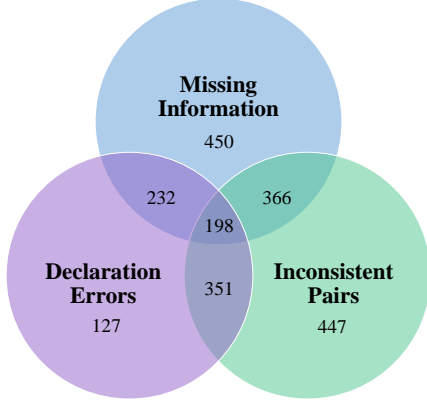


Figure 1: Distribution of rule-based error construction result on AR-LSAT. There are 1246 samples under Missing Information, 1346 under Declaration Errors, and 1362 under Inconsistent Pairs.

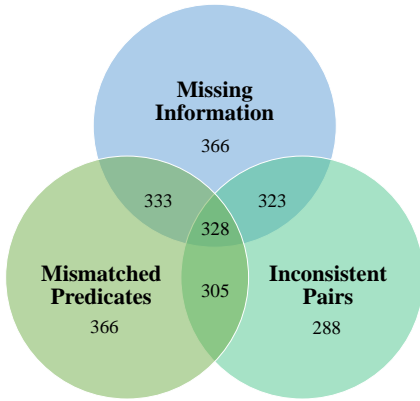


Figure 2: Distribution of rule-based error construction result on FOLIO. There are 1350 samples under Missing Information, 1332 for Mismatched Predicates, and 1244 for Inconsistent Pairs.

P_c . These operations are designed according to the semantic error patterns presented in Section 3.1.

op_DE. To generate preference pairs that reflect declaration errors, we replace integer sorts with enumeration sorts with numeral enumerated constants, and conversely, replace the enumeration sorts with integer sorts. This operation is implemented with regular expressions.

op_MI. For errors of missing information, several constraints from the correct program are randomly chosen to be deleted, forming an error program.

op_IC. This operation is designed to simulate the semantically inconsistent constraints that would likely occur in real scenarios. We adopt different implementations for AR-LSAT and FOLIO, as their textual patterns differ. For AR-LSAT, we first translate a given Z3 constraint C_{Z3} back into NL, and perturb the semantics with an LLM, with the result

denoted as C'_{NL} . Subsequently, we translate C'_{NL} back into the Z3 format, obtaining a new constraint C'_{Z3} . For FOLIO, we prompt the LLM to produce FOL premises that are nonequivalent to the annotated ones, then transform them into Z3 format to get C'_{Z3} . Finally, we replace the original formulas with the produced ones, and preserve the program if it produces errors.

op_MP. We utilize an LLM to produce semantically related but mismatched predicate names that differ from those in the Declaration. Then, we substitute the original names with the mismatched ones to obtain multiple error programs, and preserve the programs that produce errors when executed.

Leveraging the operations above, our pipeline of data construction consists of three steps: **Seed set collection** to prepare accurate programs for the subsequent steps, **Rule-based error construction**, and **Data augmentation with LLMs**.

Seed set collection. We take advantage of the translation KBs from LTRAG (Hu et al., 2025) for AR-LSAT and FOLIO. For AR-LSAT, we filter out programs that cannot produce correct answers. For FOLIO, we first transform them into Z3 programs, then conduct manual correction on incorrect programs. We eventually have 151 correct programs for AR-LSAT and 122 for FOLIO as the seed set.

Rule-based error construction. For each reasoning problem NL from the seed set, we use the correct program as the preferred program P_{pref} . Next, we execute different combinations of operations, synthesizing multiple error programs $P_{e1}, P_{e2}, \dots, P_{em}$. We choose different operations for programs from AR-LSAT and FOLIO, as their data complexities differ. For AR-LSAT, we choose op_DE, op_MI and op_IC. For FOLIO, op_MI, op_IC and op_MP are chosen. We finally match them together and have preference pairs $(NL, P_{pref}, P_{ei}), i = 1, 2, \dots, m$. We have 2,171 and 2,209 pairs for AR-LSAT and FOLIO respectively for this step, some of which contain multiple error patterns. The distribution are illustrated in Figure 1 and Figure 2.

Data augmentation with LLMs. The synthesized error programs in the preceding step might fail to cover all of the error patterns in real scenarios. Therefore, we prompt different LLMs, gpt-4o (OpenAI et al., 2024) and DeepSeek v3.2 (DeepSeek-AI et al., 2025) to produce Z3 programs on the train sets, and leverage an LLM-as-judge system to judge the preference and produce an explanation. The judge system is enhanced with

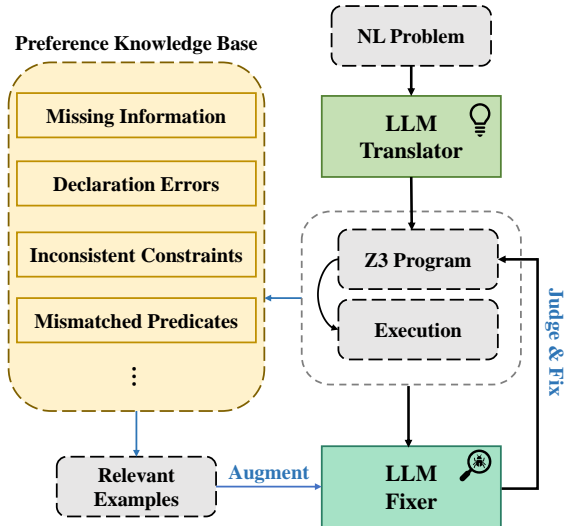


Figure 3: The overview of PrefRAG.

RAG, where preference pairs from the previous steps are used as the KB. We randomly select 750 samples from AR-LSAT’s train set and 882 from FOLIO’s to conduct the augmentation. Finally, the resulting total dataset contains 2,921 samples from AR-LSAT and 3,091 from FOLIO. An example of SemanticPref is shown in Appendix A.

It’s worth noting that directly using the LLM judge to assign preference labels may introduce inevitable noise, as the judge model itself might be affected by hallucinations. However, by integrating the RAG system, we effectively mitigate such noise, improving the reliability of the generated labels.

4 The PrefRAG Framework

With the preference dataset from Section 3.2, we propose **PrefRAG**, a domain-general framework for logical reasoning, adopting a straightforward RAG paradigm for the self-refinement module in auto-formalization. The foundational framework without RAG is adapted from Logic-LM (Pan et al., 2023), a translate-then-refine pipeline. The overall framework is illustrated in Figure 3.

Preference Knowledge Base Population. We have two KBs, for AR-LSAT and on FOLIO respectively. For each KB, the data is categorized by the error types each sample has. Note that if a sample covers multiple patterns of errors, it is assigned into each corresponding category. For samples that are constructed based on defined operations, we load them into a text template given in Appendix B along with the programs’ execution results, and list their errors briefly. For samples that are synthe-

sized with LLMs in data augmentation, we use the explanations that the LLM Judge produces as the illustration for errors.

Pipeline. As shown in Figure 3, for an input of a logical reasoning problem, we first leverage the LLM Translator to translate the NL context and query into a Z3 program. After the program is executed, the program along with its execution result are sent to the LLM Fixer. For samples from ID datasets, relevant samples would be retrieved from each category of the corresponding KB by embeddings similarities, then the union of them would be taken. For samples from OOD datasets, samples would be retrieved from both KBs of AR-LSAT and FOLIO. Subsequently, the fixer would decide whether the program needs to be fixed according to the input and retrieved samples. If so, the fixer would output the corrected program. The program will be iteratively corrected until a maximum number of attempts is reached or the fixer decides no more further correction is needed.

5 Evaluation

5.1 Experimental Setup

We select four benchmarks to evaluate the ID and OOD performance of PrefRAG. For ID evaluation, we adopt two datasets that offer authentic complexity: 1) **AR-LSAT** (Zhong et al., 2022), an analytical reasoning dataset collected from Law School Admission Test, containing 231 samples from the dev set; 2) **FOLIO** (Han et al., 2024), a human-annotated complex first-order logic (FOL) reasoning dataset, which contains 204 samples from the test set. For OOD evaluation, we select two challenging benchmarks with distinct data distributions: 3) **LogicalDeduction**, a synthetic logical reasoning task from Big-bench (Srivastava et al., 2023) about deducing the orders of objects, with 300 test samples; 4) **ProverQA** (Qi et al., 2025), a deductive reasoning benchmark generated by LLMs, and we evaluate 500 instances from the hard subset.

We compare PrefRAG against the following baselines: **Standard prompting**, where 2 in-context examples from training set are used in the prompt for a direct answer; **CoT prompting**, which adds reasoning traces to these examples to guide the reasoning process; **Logic-LM** (Pan et al., 2023), a representative auto-formalization paradigm with self-refinement; **LTRAG** (Hu et al., 2025), the current state-of-the-art RAG framework for auto-formalization in logical reasoning. For Logic-LM,

Model	AR-LSAT				FOLIO			
	gpt-4o	dpsk-ch	o3-mini	dpsk-rs	gpt-4o	dpsk-ch	o3-mini	dpsk-rs
Standard	36.36	34.20	91.34	87.88	67.64	68.14	75.98	80.88
CoT	35.93	80.52	<u>93.94</u>	90.04	72.06	82.84	76.47	81.37
Logic-LM	35.06	72.73	<u>91.34</u>	<u>90.17</u>	78.92	80.39	76.47	81.86
LTRAG	<u>56.71</u>	<u>84.42</u>	58.01	89.56	75.27	<u>82.42</u>	<u>81.32</u>	<u>84.62</u>
PrefRAG	61.47	88.75	95.24	95.24	<u>77.94</u>	82.84	84.8	85.78

Table 1: Accuracies on in-distribution benchmarks AR-LSAT and FOLIO (%).

Model	LogicalDeduction				ProverQA			
	gpt-4o	dpsk-ch	o3-mini	dpsk-rs	gpt-4o	dpsk-ch	o3-mini	dpsk-rs
Standard	74.67	70.00	100.00	<u>98.33</u>	42.80	38.60	58.00	61.00
CoT	88.88	100.00	<u>99.00</u>	100.00	48.40	63.40	<u>59.20</u>	70.60
Logic-LM	91.33	96.67	76.67	97.67	<u>61.40</u>	<u>66.00</u>	58.80	<u>75.40</u>
PrefRAG	99.00	<u>99.33</u>	100.00	100.00	78.00	76.80	73.00	77.00

Table 2: Accuracies on out-of-distribution benchmarks LogicalDeduction and ProverQA (%).

we follow its original setup on AR-LSAT, FOLIO, and LogicalDeduction. On ProverQA, which is not evaluated by Logic-LM, we apply the same configuration for FOLIO as they share a similar task setting. LTRAG is evaluated solely on the ID benchmarks (AR-LSAT and FOLIO), and not tested on the OOD datasets (LogicalDeduction and ProverQA) due to its lack of generalizability.

We conduct experiments using gpt-4o (OpenAI et al., 2024) and DeepSeek v3.2 (dpsk-ch) (DeepSeek-AI et al., 2025), as well as reasoning models o3-mini² and DeepSeek v3.2 reasoner (dpsk-rs). For reproducibility, we set the temperature to 0 for non-reasoning models, and reasoning models do not support temperature setting. We re-run some experiments of baselines under the same model versions to ensure a fair comparison. For Logic-LM, the number of refinement rounds is set to 3 for non-reasoning and 1 for reasoning models, since the gains with multiple rounds of refinement are marginal, as detailed in Appendix C.3.

5.2 Main Results

The main experimental results are shown in Table 1 (ID) and Table 2 (OOD). To facilitate comparison, we employ the same evaluation metric as Logic-LM and LTRAG: the accuracy computed with CoT backup. Specifically, if the final program produced

by LLMs fails to execute or produces no output, the final answer will be substituted with the one generated by the CoT baseline.

Overall, **PrefRAG consistently outperforms all baselines across both ID and OOD datasets.** On ID datasets AR-LSAT and FOLIO, PrefRAG surpasses the best baseline by an average of 2.39% across all models. The improvement is more pronounced on AR-LSAT (+3.87%) than on FOLIO (+0.92%). This demonstrates that our approach of modeling semantic errors via preference pairs is more effective than general auto-formalization method (Logic-LM) and RAG framework (LTRAG). On OOD datasets, PrefRAG achieves even larger gains, outperforming the best baseline by an average of +1.75% on LogicalDeduction and +10.70% on ProverQA. These results suggest that our method is strongly generalizable, while Logic-LM requires domain-specific prompts and LTRAG is limited to specific datasets of its KBs. Thus our method is more robust on unseen data distributions. The gains achieved on FOLIO and LogicalDeduction are smaller, as they are less challenging and offer less room for improvement.

We also observe that **reasoning models outperform non-reasoning models across most datasets.** For instance, on AR-LSAT, PrefRAG with o3-mini and dpsk-rs both reach 95.24% accuracy, substantially higher than those with non-reasoning models gpt-4o (61.48%) and dpsk-ch

²<https://openai.com/index/o3-mini-system-card/>

Dataset	Method	ER	EA	RA
AR-LSAT	Logic-LM	26.84	43.55	11.69
	LTRAG	69.26	50.00	34.63
	PrefRAG	74.46	72.09	53.68
FOLIO	Logic-LM	80.39	78.05	62.74
	LTRAG	99.45	75.14	74.73
	PrefRAG	98.04	78.50	76.96
Logical Deduction	Logic-LM	100.00	91.33	91.33
	PrefRAG	100.00	99.00	99.00
ProverQA	Logic-LM	41.60	78.85	32.80
	PrefRAG	98.40	78.46	77.20

Table 3: Executable rate (ER), execution accuracy (EA) and raw accuracy (RA) with gpt-4o (%).

Dataset	Method	ER	EA	RA
AR-LSAT	Logic-LM	9.96	69.57	6.93
	LTRAG	48.48	20.54	9.96
	PrefRAG	98.27	95.59	93.94
FOLIO	Logic-LM	0.00	0.00	0.00
	LTRAG	100.00	81.32	81.32
	PrefRAG	100.00	84.80	84.80
Logical Deduction	Logic-LM	37.67	38.94	14.67
	PrefRAG	100.00	100.00	73.00
ProverQA	Logic-LM	0.60	33.33	0.20
	PrefRAG	100.00	73.00	77.20

Table 4: Executable rate (ER), execution accuracy (EA) and raw accuracy (RA) with o3-mini (%).

(88.75%). This improvement is unsurprising, as auto-formalization relies on capabilities such as textual decomposition and summarization, and reasoning models excel in these areas due to their enhanced NL understanding and programming capabilities. A notable exception occurs on ProverQA, where reasoning models attain an average accuracy of 75.00%, slightly lower than the 77.40% accuracy by non-reasoning models. This might be attributed to the distinct data characteristics of ProverQA.

Several minor exceptions to the above trends are worth noting: on FOLIO, PrefRAG with gpt-4o (77.94%) slightly underperforms Logic-LM (78.92%); on LogicalDeduction, a similar pattern can be found with dpsk-ch, where CoT baseline (100.00%) marginally outperforms PrefRAG (99.33%). These can be explained with results in Section 5.3. On FOLIO, PrefRAG has a sig-

nificantly higher executable rate than Logic-LM with gpt-4o. Consequently, fewer samples fail to parse and fall back to backup answers, which leads to a seemingly lower main accuracy. However, the overall raw accuracy without backup strategies demonstrate that PrefRAG actually outperforms Logic-LM in auto-formalization task, reaching an improvement of 22.66%. The same explanation applies to LogicalDeduction: without backup strategies, PrefRAG with dpsk-ch outperforms Logic-LM by 2.66% in raw accuracy.

5.3 Evaluation of Auto-formalization

To evaluate the performance of auto-formalization of PrefRAG, we analyze the executable rate, execution accuracy and raw accuracy achieved by different frameworks. Results for gpt-4o and for o3-mini are reported in Table 3 and Table 4 respectively, and those for remaining models dpsk-ch and dpsk-rs are in Appendix C.1. **Executable rate (ER)** refers to the percentage of programs that are syntactically valid and can be executed by the solver. **Execution accuracy (EA)** denotes the percentage of programs that can produce correct answers out of the executable programs. **Raw accuracy (RA)**, defined as the product of ER and EA, is equal to the overall accuracy without backup strategies. These three metrics correspond to syntactic correctness, semantic correctness and overall correctness of auto-formalization, respectively.

Compared with Logic-LM and LTRAG, **our framework PrefRAG significantly produces fewer syntactic as well as semantic errors in auto-formalization**. As shown in Table 3, PrefRAG achieves higher execution rate than Logic-LM and LTRAG across all datasets, improving greatly over the best baselines on benchmarks such as AR-LSAT (74.46% vs. 69.26%) and ProverQA (98.40% vs. 41.60%). In terms of execution accuracy, PrefRAG outperforms both baselines, with an average gain of +7.46% over the best baseline across all datasets. This results in the highest raw accuracies by PrefRAG. The improvement indicates that our method is less prone to syntax errors for auto-formalization task, as we directly formalize the problem to Z3 programs, taking advantage of programming capability of LLMs. Instead, Logic-LM and LTRAG use an intermediate language that LLMs are less familiar with. In addition, by modeling semantic error patterns and using RAG paradigm, our method is more capable to identify and fix semantic errors during the process.

Dataset	Method	gpt-4o			dpsk-ch		
		ER	EA	RA	ER	EA	RA
AR-LSAT	LTRAG	69.26	50.00	34.63	63.64	85.03	54.11
	PrefRAG-A	68.40	72.15	49.35	77.49	89.94	69.70
	PrefRAG	74.46	72.09	53.68	81.82	91.01	74.46
FOLIO	LTRAG	99.45	75.14	74.73	100.00	82.42	82.42
	PrefRAG-A	97.55	74.37	72.55	97.55	85.93	83.82
	PrefRAG	98.04	78.50	76.96	97.06	83.34	80.88

Table 5: Comparison of the previous SOTA method LTRAG, PrefRAG, and PrefRAG-A with non-reasoning models gpt-4o (gpt-4o) and DeepSeek v3.2 (dpsk-ch). PrefRAG-A represents the RAG framework same as PrefRAG but the augmented data is removed from the KB.

We can also observe that **while o3-mini performs poorly with Logic-LM framework, both LTRAG and PrefRAG notably ease the issue**. Logic-LM with o3-mini fails to produce executable programs for FOLIO, and almost none for AR-LSAT (9.96%) and ProverQA (0.60%). Appendix D presents an output example of Logic-LM with o3-mini. Based on the observation of this example, we find a possible two-folded explanation. Firstly, o3-mini as a reasoning model may not be fine-tuned for instruction-following, and thus produce extra reasoning trace, making the program fail to parse. Secondly, the static in-context samples offer limited guidance for reasoning models, and may even interrupt the model’s reasoning process, as it is observed that o3-mini output programs not only for the given question but also for the in-context samples. In contrast, the model performs robustly under LTRAG and PrefRAG. For instance, with PrefRAG, it achieve an average raw accuracy of 82.34% across all datasets, surpassing the 76.71% attained by gpt-4o. We hypothesize that the improvement stems from the more standardized prompt format and, more importantly, the dynamic prompting strategy employed by both LTRAG and PrefRAG, which provides clear contextual guidance, enabling o3-mini to better understand and execute the auto-formalization task.

5.4 Ablation Studies

Since the LLM-augmented data in SemanticPref heavily depend on LLMs for generation and preference judgement, its reliability requires explicit verification. Thus, we conduct an targeted ablation study where this part of data is removed from the KB of PrefRAG, forming the ablated variant PrefRAG-A, and evaluate its auto-formalization

performance over the ID datasets. We focus on this specific ablation, as the comparison between baselines and our method has revealed the effectiveness of each module. On one hand, ablating the entire RAG module would reduce the framework to Logic-LM. On the other hand, comparing PrefRAG with LTRAG features the advantage of our preference KB.

The results are presented in Table 5, using gpt-4o and dpsk-ch. The experiments with reasoning models are provided in Appendix C.2. These comparative results lead to two key conclusions. First, **data augmentation of SemanticPref contributes notably, leading to an improvement in overall model performance**. When data augmentation is removed on AR-LSAT, the average raw accuracy across non-reasoning models decreases from 64.07% to 59.53%, and both the executable rate and execution accuracy show a declining trend. On FOLIO, despite dpsk-ch being an outlier where the ablated framework marginally surpasses the original one, the falling trend is consistently observed for other models. Second, **even when the augmented data is excluded, our method still remains competitive compared with the previous SOTA method LTRAG**. On AR-LSAT, the average raw accuracy of PrefRAG-A surpasses LTRAG by a substantial margin of 15.16%. On FOLIO, it slightly underperforms LTRAG by 0.39%. These ablation results further validate the effectiveness of our overall dataset SemanticPref and framework PrefRAG, and confirm that the LLM-augmented data plays a beneficial role.

5.5 Error Analysis

To gain a deeper understanding of the limitations of PrefRAG, we manually inspect a subset of erro-

neous cases from AR-LSAT using DeepSeek v3.2, and observe two recurring failure patterns.

Syntactic repair failure. While our framework can fix most of the syntactic errors, the model would occasionally fail to correct more complicate ones, such as the example shown below, which stems from a conflict between Z3’s operator overloading and Python’s interpretation. In this example, the expression `d != d1` in the initial program yields a Z3 symbolic expression rather than an immediately evaluatable Python Boolean. While it is intended to be passed to Z3 for satisfiability solving, it is evaluated by Python immediately at runtime, causing a Z3 exception. When the model tries to fix it in the refinement phase, the model fails to diagnose the error and makes undirected attempts of corrections, continuing to iterate even after a correct fix, resulting in failure.

```
...
sales_consecutive = Or([And(tour(d1)
↪ == Sales, tour(d2) == Sales,
↪ And([tour(d) != Sales for d in days
↪ if d != d1 and d != d2])) for (d1,
↪ d2) in consecutive_pairs])
...
```

Hallucinations under multi-faceted constraint semantics. When the constraint satisfaction problems involve multiple constraint dimensions, LLMs often fail to gain a complete understanding of the problems before formalizing them, leading to substantial hallucinations. Consider the example in Appendix C.4, which involves organizing a construction crew. The problem imposes restrictions on which task each worker can perform, as well as constraints on the overall project schedule. As our framework lacks an explicit module for parsing and organizing the structure of constraints, the model directly proceeds to the formalization phase and produces syntax and semantic errors. For instance, the model may omit an entire constraint dimension, such as the ordering constraints on task execution. It may also fail to correctly identify the solving goal, such as whether the problem requires the list of workers or the project schedule. We also observe that in such scenarios, LLMs tend to make more syntax errors than in simpler problems, such as omitting variable declarations.

In summary, these cases indicate that our framework is fundamentally constrained by LLM’s capacity to reason in complex scenarios. A possible

solution is to incorporate a pre-formalization processing module and external syntactic knowledge to mitigate these limitations.

5.6 Computational Cost Analysis

Our framework incurs higher overhead than the baseline method LTRAG, yet the overall cost remains manageable. We focus on this comparison because both are RAG methods and the other baselines rely on prompting without retrieval. LTRAG retrieves 1–3 in-context examples in both the translation and fix stages, with a maximum of three fix rounds. Each example in its translation KBs contains an NL context and the corresponding program, while each example in its fixer KBs consists of an NL context, the original formal program, an error analysis, and the repaired program. In contrast, our framework performs retrieval only in the fix stage, also with a maximum of three fix rounds, and each retrieved example contains an NL context, two formal programs, and a preference analysis, making its structure broadly comparable to that of LTRAG’s fixer examples. Concretely, on the AR-LSAT dataset, the average token length of LTRAG’s fixer KB examples is 1,089.34, while that of our examples is 1,939.46. Since the two frameworks share a largely similar architecture, our overhead remains within a constant factor of LTRAG’s and is therefore fully manageable.

6 Conclusion

In this paper, we propose PrefRAG, a RAG framework for correcting semantic errors in auto-formalization for logical reasoning. Building upon a systematic analysis of semantic error patterns, we propose to model these patterns with program preference pairs, and further construct a preference dataset SemanticPref, where the pairs are automatically synthesized with rule-based operations and LLM-assisted data augmentation. Extensive evaluations across ID (AR-LSAT and FOLIO) and OOD datasets demonstrate that PrefRAG outperforms strong baselines for logical reasoning, and further exhibits robust generalizability. A promising future direction is to leverage the SemanticPref dataset to fine-tune smaller models via preference alignment algorithms such as DPO (Rafailov et al., 2023). Such an approach would empower smaller models with the capability to detect and correct semantic errors, which could enable substantial reductions in inference cost and latency.

Limitations

We acknowledge two primary limitations.

First, **the overall generalizability of PrefRAG is bounded by the expressivity and solving capability of the underlying symbolic solver (e.g. Z3)**. For example, Z3 solver may fail to solve problems with high complexity due to timeout or memory exhaustion. Moreover, Z3 is designed for satisfiability modulo theories (SMT) solving and does not support reasoning paradigms beyond its scope, such as non-monotonic or modal reasoning. Admittedly, this limitation is shared by other auto-formalization-based approaches.

Second, **although our data construction pipeline is designed to be general and automatic, it still requires a seed set of correct programs of reasoning problems, which is often manually curated**. This initial human effort is necessary to start the rule-based and LLM-augmented synthesis process, if the pipeline is migrated to a different formal language. Furthermore, the scale and diversity of the final preference dataset are constrained by the quality of the seed set, thus affecting the performance of the RAG framework.

Despite these limitations, the required human effort for migrating PrefRAG remains manageable. The effort could be mitigated by using LLMs to generate candidate programs for human verification and correction, especially given that the seed set is relatively small in scale.

Acknowledgments

We acknowledge support from the Natural Science Foundation of China under Grant No. 62476295.

References

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michał Podstawski, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2024. [Graph of Thoughts: Solving Elaborate Problems with Large Language Models](#). In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2008. [Z3: an efficient SMT solver](#). In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang,

Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. [DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning](#). *Computing Research Repository*, arXiv:2501.12948.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, and 245 others. 2025. [Deepseek-v3.2: Pushing the frontier of open large language models](#). *Computing Research Repository*, arXiv:2512.02556.

Jiazhan Feng, Ruochen Xu, Junheng Hao, Hiteshi Sharma, Yelong Shen, Dongyan Zhao, and Weizhu Chen. 2024. [Language models can be deductive solvers](#). In *Findings of the Association for Computational Linguistics: NAACL 2024*.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhen-ting 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. 2024. [FOLIO: Natural language reasoning with first-order logic](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP 2024)*.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. [Reasoning with language model is planning with world model](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP 2023)*.

Ruikang Hu, Shaoyu Lin, Yeliang Xiu, and Yongmei Liu. 2025. [LTRAG: Enhancing autoformalization and self-refinement for logical reasoning with thought-guided RAG](#). In *Findings of the Association for Computational Linguistics: ACL 2025*.

Fangkai Jiao, Zhiyang Teng, Bosheng Ding, Zhengyuan Liu, Nancy Chen, and Shafiq Joty. 2024. [Exploring self-supervised logic-enhanced training for large language models](#). In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL 2024)*.

Tongxuan Liu, Wenjiang Xu, Weizhe Huang, Yuting Zeng, Jiaying Wang, Xingyu Wang, Hailong Yang, and Jing Li. 2025. [Logic-of-thought: Injecting logic into contexts for full reasoning in large language models](#). In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL 2025)*.

Wangyue Lu, Lun Du, Sirui Li, Ke Weng, Haozhe Sun, Hengyu Liu, Minghe Yu, Tiancheng Zhang, and

- Ge Yu. 2025. [Automated formalization via conceptual retrieval-augmented llms](#). *Computing Research Repository*, arxiv:2508.06931.
- Terufumi Morishita, Gaku Morio, Atsuki Yamaguchi, and Yasuhiro Sogawa. 2024. [Enhancing reasoning capabilities of llms via principled synthetic logic corpus](#). In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 73572–73604.
- 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 (EMNLP 2023)*.
- OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. [GPT-4 technical report](#). *Computing Research Repository*, arXiv:2303.08774.
- OpenAI, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, and 400 others. 2024. [GPT-4o system card](#). *Computing Research Repository*, arXiv:2410.21276.
- 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*.
- Chengwen Qi, Ren Ma, Bowen Li, He Du, Binyuan Hui, Jinwang Wu, Yuanjun Laili, and Conghui He. 2025. [Large language models meet symbolic provers for logical reasoning evaluation](#). In *The Thirteenth International Conference on Learning Representations (ICLR)*.
- Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. [Direct preference optimization: Your language model is secretly a reward model](#). In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Mohammad Raza and Natasa Milic-Frayling. 2025. [Instantiation-based formalization of logical reasoning tasks using language models and logical solvers](#). In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2025)*.
- Hyun Ryu, Gyeongman Kim, Hyemin S. Lee, and Eunho Yang. 2025. [Divide and translate: Compositional first-order logic translation and verification for complex logical reasoning](#). In *The Thirteenth International Conference on Learning Representations (ICLR 2025)*.
- 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](#). *Transactions on Machine Learning Research*.
- Koushik Viswanadha, Deepanway Ghosal, and Somak Aditya. 2025. [LOGICPO: efficient translation of nl-based logical problems to FOL using llms and preference optimization](#). *Computing Research Repository*, arXiv:2506.18383.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. [Self-consistency improves chain of thought reasoning in language models](#). In *The Eleventh International Conference on Learning Representations, (ICLR 2023)*.
- 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 *Advances in Neural Information Processing Systems (NeurIPS)*.
- Jundong Xu, Hao Fei, Meng Luo, Qian Liu, Liangming Pan, William Yang Wang, Preslav Nakov, Mong-Li Lee, and Wynne Hsu. 2025. [Aristotle: Mastering logical reasoning with a logic-complete decompose-search-resolve framework](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL 2025)*.
- Jundong Xu, Hao Fei, Liangming Pan, Qian Liu, Mong-Li Lee, and Wynne Hsu. 2024. [Faithful logical reasoning via symbolic chain-of-thought](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024)*.
- Yuan Yang, Siheng Xiong, Ali Payani, Ehsan Shareghi, and Faramarz Fekri. 2024. [Harnessing the power of large language models for natural language to first-order logic translation](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024)*.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. [Tree of Thoughts: Deliberate problem solving with large language models](#). In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. 2023. [SatLM: Satisfiability-aided language models using declarative prompting](#). In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Meiru Zhang, Philipp Borchert, Milan Gritta, and Gerasimos Lampouras. 2025. [DRIFT: decompose, retrieve, illustrate, then formalize theorems](#). In *The*

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*.

A An Example from SemanticPref

We present an example from SemanticPref for AR-LSAT from rule-based error construction. Examples from FOLIO are similar.

```
NL problem:
# Context:
At a benefit dinner, a community
↳ theater's seven sponsors—K, L, M,
↳ P, Q, V, and Z—will be seated at
↳ three tables—1, 2, and 3. Of the
↳ sponsors, only K, L, and M will
↳ receive honors, and only M, P,
↳ and Q will give a speech. The
↳ sponsors' seating assignments
↳ must conform to the following
↳ conditions: Each table has at
↳ least two sponsors seated at it,
↳ and each sponsor is seated at
↳ exactly one table. Any sponsor
↳ receiving honors is seated at
↳ table 1 or table 2. L is seated
↳ at the same table as V.
# Question:
Which one of the following is an
↳ acceptable assignment of
↳ sponsors to tables?
# Options:
A) Table 1: K, P; Table 2: M, Q;
↳ Table 3: L, V, Z
B) Table 1: K, Q, Z; Table 2: L, V;
↳ Table 3: M, P
C) Table 1: L, P; Table 2: K, M;
↳ Table 3: Q, V, Z
D) Table 1: L, Q, V; Table 2: K, M;
↳ Table 3: P, Z
E) Table 1: L, V, Z; Table 2: K, M,
↳ P; Table 3: Q

Ground truth: A

chosen_program:
from z3 import *
# Declarations
```

```
# At a benefit dinner, a community
↳ theater's seven sponsors—K, L, M,
↳ P, Q, V, and Z—will be seated at
↳ three tables—1, 2, and 3. Of the
↳ sponsors, only K, L, and M will
↳ receive honors, and only M, P,
↳ and Q will give a speech.
sponsors_sort, (K, L, M, P, Q, V, Z)
↳ = EnumSort('sponsors', ['K', 'L',
↳ 'M', 'P', 'Q', 'V', 'Z'])
tables_sort = IntSort()
sponsors = [K, L, M, P, Q, V, Z]
tables = [1, 2, 3]
seated = Function('seated',
↳ sponsors_sort, tables_sort)
# Pre_Conditions
pre_conditions = []
# Each table has at least two
↳ sponsors seated at it
pre_conditions.append(And([Sum(
[seated(s) == t for s in sponsors])
↳ >= 2 for t in tables]))
# Each sponsor is seated at exactly
↳ one table
s = Const('s', sponsors_sort)
pre_conditions.append(ForAll([s],
↳ Or([seated(s) == t for t in
↳ tables])))
# Any sponsor receiving honors is
↳ seated at table 1 or table 2
s = Const('s', sponsors_sort)
pre_conditions.append(ForAll([s],
↳ Implies(Or(s == K, s == L, s ==
↳ M), Or(seated(s) == 1, seated(s)
↳ == 2))))
# L is seated at the same table as V
pre_conditions.append(seated(L) ==
↳ seated(V))
s0 = Const('s0', sponsors_sort)
pre_conditions.append(ForAll([s0],
↳ And(1 <= seated(s0), seated(s0)
↳ <= 3)))
# Question: Which one of the
↳ following is an acceptable
↳ assignment of sponsors to tables?
def is_sat(option_constraints):
    solver = Solver()\n
    ↳ solver.add(pre_conditions)
    solver.add(option_constraints)
    return solver.check() == sat
# A) Table 1: K, P; Table 2: M, Q;
↳ Table 3: L, V, Z
```

```

if is_sat(And(seated(K) == 1,
→ seated(P) == 1, seated(M) == 2,
→ seated(Q) == 2, seated(L) == 3,
→ seated(V) == 3, seated(Z) == 3)):
→ print('(A)')
# B) Table 1: K, Q, Z; Table 2: L, V;
→ Table 3: M, P
if is_sat(And(seated(K) == 1,
→ seated(Q) == 1, seated(Z) == 1,
→ seated(L) == 2, seated(V) == 2,
→ seated(M) == 3, seated(P) == 3)):
→ print('(B)')
# C) Table 1: L, P; Table 2: K, M;
→ Table 3: Q, V, Z
if is_sat(And(seated(L) == 1,
→ seated(P) == 1, seated(K) == 2,
→ seated(M) == 2, seated(Q) == 3,
→ seated(V) == 3, seated(Z) == 3)):
→ print('(C)')
# D) Table 1: L, Q, V; Table 2: K, M;
→ Table 3: P, Z
if is_sat(And(seated(L) == 1,
→ seated(Q) == 1, seated(V) == 1,
→ seated(K) == 2, seated(M) == 2,
→ seated(P) == 3, seated(Z) == 3)):
→ print('(D)')
# E) Table 1: L, V, Z; Table 2: K, M,
→ P; Table 3: Q
if is_sat(And(seated(L) == 1,
→ seated(V) == 1, seated(Z) == 1,
→ seated(K) == 2, seated(M) == 2,
→ seated(P) == 2, seated(Q) == 3)):
→ print('(E)')

rejected program:
...

```

B Templates for Rule-based KB Samples

For samples in SemanticPref that are synthesized with rule-based operations, we use a text template to populate them into the KB, and explain their errors. We have two nearly identical templates which only differ in the sequence of preferred and rejected programs, and we randomly choose between them for KB population. One of the templates are shown as follows.

```

For the given story, we have two Z3
→ logic programs using python API.
[[CONTEXT]]
# Program 1:
[[CHOSEN_PROGRAM]]

```

```

Execution Result:
[[CHOSEN_EXEC]]
# Program 2:
[[REJECTED_PROGRAM]]
Execution Result:
[[REJECTED_EXEC]]
The first program is preferred than
→ the second one, because:
(these description are only included
→ when the sample has corresponding
→ error patterns)
1. the program 2 has missing
→ constraints from the original
→ text, while the other one has
→ them.
2. the program 2 contains semantic
→ errors in declaration, while the
→ other one is correct.
3. the program 2 uses undefined
→ predicates that's not in the z3
→ declaration, while the other one
→ is correct.
4. the program 2 contains constraints
→ that have inconsistent semantics
→ with the context, while the other
→ one is semantically correct.

```

C Additional Results and Discussion

C.1 Additional Results for Performance of Auto-formalization

The performance of auto-formalization with DeepSeek v3.2 and its reasoning mode are reported in Table 7 and Table 8 respectively. These results further confirm the effectiveness of PrefRAG in auto-formalization using different settings of DeepSeek v3.2, as PrefRAG consistently achieves higher executable rate, execution accuracy and raw accuracy over baselines on ID and OOD benchmarks.

C.2 Additional Results for Ablation Studies

Table 6 provides a comparison between the performance of baselines, PrefRAG and ablated PrefRAG with reasoning models. Same variant from Section 5.4 are adopted. As reasoning models perform unstably under different baseline settings, results of both Logic-LM and LTRAG are listed. A similar trend reported in Section 5.4 is observed from these results, that the variant PrefRAG-A consistently outperforms baselines but underperforms PrefRAG.

Dataset	Method	o3-mini			dpsk-rs		
		ER	EA	RA	ER	EA	RA
AR-LSAT	Logic-LM	9.96	69.57	6.93	74.49	89.14	66.67
	LTRAG	48.48	20.54	9.96	3.70	36.36	1.35
	PrefRAG-A	98.7	94.30	93.07	95.63	97.72	93.45
	PrefRAG	98.27	95.59	93.94	96.10	96.40	92.64
FOLIO	Logic-LM	0.00	0.00	0.00	94.12	81.25	76.47
	LTRAG	100.00	81.32	81.32	100.00	84.62	84.62
	PrefRAG-A	99.51	83.74	83.33	99.02	85.15	84.31
	PrefRAG	100.00	84.80	84.80	99.51	85.71	85.29

Table 6: Comparison of baselines, PrefRAG, and PrefRAG-A with reasoning models o3-mini and DeepSeek v3.2 reasoning mode. PrefRAG-A represents the RAG framework same as PrefRAG but augmented data is removed from the KB.

Dataset	Method	ER	EA	RA
AR-LSAT	Logic-LM	41.56	68.75	28.57
	LTRAG	63.64	85.03	54.11
	PrefRAG	81.82	91.01	74.46
FOLIO	Logic-LM	85.29	79.31	67.64
	LTRAG	100.00	82.42	82.42
	PrefRAG	97.06	83.34	80.88
Logical Deduction	Logic-LM	100.00	96.67	96.67
	PrefRAG	100.00	99.33	99.33
ProverQA	Logic-LM	80.60	65.26	52.60
	PrefRAG	98.60	77.08	76.00

Table 7: Executable rate (ER), execution accuracy (EA) and raw accuracy (RA) with DeepSeek v3.2 (%).

Dataset	Method	ER	EA	RA
AR-LSAT	Logic-LM	74.79	89.14	66.67
	LTRAG	4.33	40.00	1.73
	PrefRAG	96.10	96.40	92.64
FOLIO	Logic-LM	94.12	81.25	76.47
	LTRAG	100.00	84.62	84.62
	PrefRAG	99.51	85.71	85.29
Logical Deduction	Logic-LM	100.00	97.67	97.67
	PrefRAG	100.00	100.00	100.00
ProverQA	Logic-LM	98.60	76.06	75.00
	PrefRAG	100.00	77.00	77.00

Table 8: Executable rate (ER), execution accuracy (EA) and raw accuracy (RA) with DeepSeek v3.2 reasoning mode(%).

Therefore, data augmentation is proved to be a key component of the data construction.

C.3 Experiment on Refinement Rounds of Reasoning Models on Logic-LM

To test the effect of the number of refinement rounds on performance, we conduct a preliminary experiment on 20 examples from the AR-LSAT dev set using DeepSeek-Reasoner within the Logic-LM framework. As a result, after one round of refinement, the executable rate is 0.45 and raw accuracy is 0.35; after three rounds, while the executable rate increases to 0.55, the raw accuracy remains at 0.35, showing no improvement. Given the marginal gains and the substantial computational overhead, we ultimately adopted the single-round setting.

C.4 A Full Example from AR-LSAT for Error Analysis

A crew of up to five workers is to install a partition in at most three days. The crew completes five tasks in this order: framing, wallboarding, taping, sanding, priming. The crew is selected from the following list, which specifies exactly the tasks each person can do: George: taping Helena: sanding, priming Inga: framing, priming Kelly: framing, sanding Leanda: wallboarding, taping Maricita: sanding Olaf: wallboarding, priming The following conditions must apply: At least one task is done each day. Taping and priming are done on different days. Each crew member does at least one task during the installation, but no more than one task a day. Each task is done by exactly one worker, completed the day it is started and before the next task begins.

D An Example of Logic-LM's Output with o3-mini

We here list an output generated with the translation prompt of Logic-LM with o3-mini. We can observe that the model first repeated the six programs from the in-context examples, then generated the final program. The program did not follow the grammar as shown in the examples, such as using different notes for comments, and using natural descriptions like "set of children". This example shows that o3-mini performs poorly in instruction following task, and its reasoning process as well as its output are disturbed by the in-context samples.

```

////////////////////////////////////
// Problem 1
// Tuesday Meals for Vladimir and
↳ Wendy
...
////////////////////////////////////
// Problem 2
// Technician Machine Repairs
...
////////////////////////////////////
// Problem 6
// Assigning Lockers to Children

// Declarations
children = EnumSort([Fred, Juan,
↳ Marc, Paul, Nita, Rachel,
↳ Trisha])
lockers = EnumSort([1, 2, 3, 4, 5])
assigned = Function([children] ->
↳ [lockers])
// Additional helper
↳ functions/relations:
shared = Function([lockers] -> [set
↳ of children]) // the set of
↳ children assigned to a locker

// Constraints
// Each locker is assigned to either
↳ one or exactly two children, and
↳ every child is assigned to
↳ exactly one locker.
ForAll([l: lockers], Or(Count([c:
↳ children], assigned(c) == 1) ==
↳ 1, Count([c: children],
↳ assigned(c) == 1) == 2)) :::
↳ "Locker occupancy condition"

```

```

ForAll([c: children], Exists([l:
↳ lockers], assigned(c) == 1)) :::
↳ "Each child is assigned a locker"
// If a locker is shared (assigned to
↳ two children) then one must be a
↳ boy and one a girl.
boys = EnumSort([Fred, Juan, Marc,
↳ Paul])
girls = EnumSort([Nita, Rachel,
↳ Trisha])
ForAll([l: lockers],
↳ Implies(Count([c: children],
↳ assigned(c) == 1) == 2,
↳ Exists([b: boys, g: girls],
↳ assigned(b) == 1 And
↳ assigned(g) == 1))) :::
↳ "Shared locker gender mix"
// Specific gender constraints for
↳ sharing
assigned(Fred) == 3 ::: "Fred is
↳ assigned to locker 3"
(Count([c: children], And(c == Juan,
↳ Count([l: lockers], assigned(c)
↳ == 1) == 1)) = 1) ::: "Juan is
↳ assigned to some locker and must
↳ share (see below)"
// Juan must share a locker -> the
↳ locker Juan is in must contain 2
↳ children.
ForAll([l: lockers],
↳ Implies(assigned(Juan) == 1,
↳ Count([c: children], assigned(c)
↳ == 1) == 2)) ::: "Juan must share
↳ a locker"
// Rachel cannot share a locker ->
↳ her locker is solo.
ForAll([l: lockers],
↳ Implies(assigned(Rachel) == 1,
↳ Count([c: children], assigned(c)
↳ == 1) == 1)) ::: "Rachel does not
↳ share a locker"
// Nita's locker cannot be adjacent
↳ to Trisha's locker.
Not(Abs(assigned(Nita) -
↳ assigned(Trisha)) == 1) :::
↳ "Nita's locker not adjacent to
↳ Trisha's"
// Marc alone is assigned to locker
↳ 1.
assigned(Marc) == 1 ::: "Marc is
↳ assigned to locker 1 and is alone
↳ there"

```

```
// Additional note: When a locker is
→ shared it must be one girl and
→ one boy - already enforced.

// Extra given conditions for this
→ problem instance:
assigned(Trisha) == 3 ::: "Trisha is
→ assigned to locker 3"

// Options
Question ::: "If Trisha is assigned
→ to locker 3 and Marc alone is
→ assigned to locker 1, then which
→ one of the following must be
→ true?"
is_true(assigned(Juan) == 4) :::
→ "(A)"
is_true(assigned(Juan) == 5) :::
→ "(B)"
is_true(assigned(Paul) == 2) :::
→ "(C)"
is_true(assigned(Rachel) == 2) :::
→ "(D)"
is_true(assigned(Rachel) == 5) :::
→ "(E)"
```