FOL-TRACES: Verified First-Order Logic Reasoning Traces at Scale

Isabelle Lee
USC
lee.isabelle.g@gmail.com

Sarah Liaw
Harvard University

Dani Yogatama
USC

Abstract

Reasoning in language models is difficult to evaluate: natural-language traces are unverifiable, symbolic datasets are too small, and most benchmarks conflate heuristics with inference. We present FOL-TRACES, the first large-scale dataset of programmatically verified reasoning traces, enabling rigorous evaluation of structured logical inference.[1] We also propose two challenging and comprehensive diagnostic tasks—*masked operation prediction* and *step completion*—that directly probe syntactic awareness and process fidelity. FOL-TRACES serves as a scalable testbed for rigorously studying how models perform structured logical inference. Systematic experiments with 5 reasoning LLMs show that the dataset remains challenging: models only reach around 45.7% accuracy on masked operation prediction and around 27% on two-step completion.

1 Introduction

To establish whether large language models (LLMs) produce correct reasoning, they must be evaluated in domains that provide both structural clarity and verifiable inference. Existing approaches fall short of this requirement. Natural-language chains of thought, while human-interpretable, are poorly defined (Bender and Koller, 2020), unverifiable, and often unfaithful to the model's underlying computation (Tyen et al., 2024; Min et al., 2022). Symbolic datasets such as math problem sets, algebraic puzzles, and theorem-proving corpora provide formal supervision, but they are typically too small or under-annotated for large-scale, systematic reasoning analysis. Widely used benchmarks further conflate heuristic shortcuts with inference, making it difficult to determine whether success reflects systematic reasoning or pattern exploitation.

Recent advances in reasoning-augmented language models—such as OpenAI's O1 (OpenAI et al., 2024), DeepSeek's R1 (Guo et al., 2025), and the Qwen reasoning model (Bai et al., 2023)—have intensified interest in structured problem-solving within LLMs. These models have shown strong performance on complex tasks, including AIME mathematical challenges (ValsAI, 2025) and coding benchmarks (Chen et al., 2021; Jimenez et al., 2024), often leveraging large-scale reinforcement learning and external verifiers. A key strategy is to train on extended sequences that incorporate chains-of-thought (CoT) or reasoning traces (Wei et al., 2023). In addition to recent advancements in post-training, emerging research explores how reasoning mechanisms might be integrated into pretraining itself (Geiping et al., 2025; Tack et al., 2025; Ruan et al., 2025; Dong et al., 2025; Hatamizadeh et al., 2025). Rather than treating reasoning as a byproduct of scale, these approaches suggest it can instead be structured through targeted development. A deeper understanding of how reasoning arises during training could enable us to induce it more deliberately.

1.1 Background on interpretability and Chain-of-Thought (CoT) reasoning

To better understand reasoning in LLMs, a plethora of recent work have turned to mechanistic explanations. Lindsey et al. (2025) analyze model circuitry to trace internal reasoning patterns, though their findings are limited to carefully controlled settings. Elhage et al. (2022), and Wang et al. (2022) identify components like induction heads and algorithmic circuits, offering insights into how models support

[1]The dataset is available at https://huggingface.co/datasets/fol-traces/fol-traces, and the generation and analysis code is available at https://github.com/iglee/fol-traces

Findings of the Association for Computational Linguistics: EACL 2026, pages 2181–2203
March 24-29, 2026 ©2026 Association for Computational Linguistics

multi-step reasoning.

However, other recent findings suggest that CoT traces may align poorly with actual inference dynamics. While CoT prompting has improved performance on complex reasoning tasks (Wei et al., 2023; Kojima et al., 2023), several studies demonstrate that its explanations can be unfaithful to the model's true computational process (Turpin et al., 2023; Wei Jie et al., 2024). Techniques such as self-consistency sampling (Wang et al., 2023), contrastive CoT (Chia et al., 2023), and faithful reasoning frameworks (Lyu et al., 2023) attempt to bridge this gap. Nonetheless, the central challenge remains: how to reliably relate surface-level reasoning traces to the internal mechanisms that actually drive model behavior.

These challenges expose a key gap in reasoning datasets. Natural language CoT resources are large but unverifiable due to linguistic ambiguity. Formal corpora such as theorem-proving datasets (e.g., Lean (Yang et al., 2023), Metamath (Yu et al., 2024)) offer symbolic guarantees yet lack detailed annotations or remain too small for modern pretraining. Mathematical datasets like MATH (Hendrycks et al., 2021) and GSM8K (Cobbe et al., 2021) provide intermediate scale but verify only final answers, not reasoning steps. No existing dataset combines programmatic step-level verification with pretraining-scale size. FOL-TRACES fills this gap, as shown in Table 1.

Dataset	Scale	Verified Steps	Domain
MATH	2.93K	✗ (answers only)	Math
GSM8K	8K	✗ (answers only)	Math
Lean	<100K	✓	Formal proofs
FOL-TRACES	7.4M	✓	First-order logic

Table 1: FOL-TRACES is the only dataset combining pretraining scale with step-level verification. Scale refers to the number of examples.

1.2 Formal Reasoning Presents an Opportunity

Formal reasoning provides a clear lens for examining how language models reason. Unlike natural language reasoning—which is often ambiguous and hard to verify—formal logic offers interpretable, verifiable structure, yet purely symbolic tasks like arithmetic can be too limited and detached from language. First-order logic (FOL) strikes a useful balance: it mirrors natural-language structure while remaining algorithmically grounded and semi-decidable (Boolos et al., 2007)—valid formulas can be confirmed by an algorithm, though invalid ones cannot always be disproved. Leveraging this property, we introduce FOL-TRACES, the first large-scale dataset with programmatically verified reasoning traces (7.4M examples, 3.5B tokens). Unlike unverifiable natural language CoT or theorem-proving corpora—which are too small for pretraining—FOL-TRACES uniquely combines symbolic guarantees with scale, enabling systematic analysis of how models acquire, represent, and generate logical reasoning with step-by-step verification.

Our Contributions. Building on this motivation, our contributions are threefold:

1. We introduce FOL-TRACES, the first large-scale dataset with programmatically verified reasoning traces (7.4M examples, 3.5B tokens)—addressing the critical gap between unverifiable natural language CoT and small-scale symbolic datasets.

2. We propose two diagnostic tasks—*masked operation prediction* and *step completion*—that probe models' syntactic awareness and process-level reasoning fidelity.

3. We benchmark both pretrained and reasoning-oriented LLMs, revealing systematic differences in how models acquire and internalize logical structure.

Scope. FOL-TRACES is intended as a diagnostic complement to natural language reasoning benchmarks rather than a replacement for them. By operating with programmatically verified FOL steps, it isolates core logical reasoning inference behavior while deliberately abstracting away linguistic ambiguity. FOL *does* represent a subset of natural language, but our dataset surgically, intentionally focuses on verifiable structured inference and does not aim to capture the full richness of open-ended natural language reasoning.

2 FOL-TRACES dataset

Most reasoning datasets face a trade-off: natural language CoT is unverifiable, while symbolic logic corpora are too small for large-scale training. FOL-TRACES combines programmatic verification with

pretraining-scale size, enabling systematic study of logical reasoning. The generation pipeline shown in Figure 1 (1) generates symbolic FOL formulas and (2) instantiates them with natural language predicates and variables via LLMs.

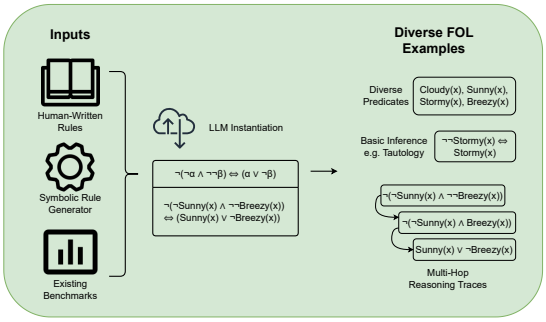


Figure 1: FOL Dataset Generation Overview.

We leverage existing benchmarks such as LogicBench (Parmar et al., 2024) and FOLIO (Han et al., 2024) as sources of human-annotated logical forms and prompts. In addition to the collected inference rules from existing datasets, we programmatically generate 1.5 million unique rules and corresponding step-by-step simplification traces. Then, we employ LLMs to instantiate them, resulting in a diverse and extensive collection of FOL expressions. We describe the programmatic unique rule generation process in §2.2.1. Then, we detail the various properties and the complexity dynamics of the rule sets and resulting datasets in §3.

2.1 Dataset Structure and Annotations

The dataset contains pretraining-scale training splits, as well as comprehensively labeled validation and test splits. Each example is grounded in an underlying first-order symbolic expression annotated with a unique rule ID. Subsequently, each rule is annotated in detail with the following fields.

Rule (`rule`). This field contains the first-order logic expression template to be instantiated. Each rule consists of at least one first-order logic formula. For curated rules, we use basic inference rules from first-order logic. For randomly generated rules, we programmatically generate a formula and then simplify it step by step to create a chain of first-order logic expressions that together form a rule.

Expressions (`exprs`). For randomly generated rules, each formula is simplified using basic first-order logic inference rules. Each simplification step is recorded as an expression and added to the list in this field.

Complexity by step (`complexity_by_step`). The circuit complexity corresponding to each expression in `exprs`. Here, we define the circuit complexity of a Boolean or first-order logical formula φ with

- Atoms(φ): the set of atomic propositions in φ (i.e., statements with Boolean values TRUE/FALSE),
- Sub(φ): the set of *immediate subformulas* of φ, i.e., those that occur exactly one level below φ in its syntactic tree. For example, if $\varphi = (\alpha \wedge \neg\beta) \vee \gamma$, then $\text{Sub}(\varphi) = \{\alpha \wedge \neg\beta, \gamma\}$.
- Each logical connective (e.g., $\neg, \wedge, \vee$) corresponds to a logic gate in a Boolean circuit.

Then, we define the circuit complexity as the circuit size $C(\varphi)$ needed to compute φ. Formally, we define as:

$$C(\varphi) = \begin{cases} 1, & \text{if } \varphi \text{ is atomic} \\ 1 + \sum_{i=1}^{k} C(\varphi_i), & \text{if } \varphi = f(\varphi_1, \ldots, \varphi_k) \end{cases} \quad (1)$$

where f is a logical operator (e.g., $\neg, \wedge, \vee$), and $\varphi_1, \ldots, \varphi_k \in \text{Sub}(\varphi)$. Generally, the circuit complexities decrease for subsequent expressions in `exprs`, as they simplify.

Elimination complexity (`elimination_complexity`). Each expression in `exprs` is a simplification of the one before it. The elimination complexity at index i is the program complexity of simplifying `exprs[`i`]` into `exprs[`i+1`]`.

Program complexity (`program_complexity`). This represents the total program complexity involved in randomly generating a first-order logic expression and subsequently, recursively simplifying it. In some cases, the expression is fully simplified. However, if the simplification requires too large a recursive jump, the program may stop prematurely. A detailed description of the full algorithm is provided in §2.2.1.

Original depth (`original_depth`). The original depth is the depth of the initial expression, i.e., `exprs[0]`.

Finally, Table 2 presents summary statistics for the dataset splits. Compared to other datasets, our reasoning CoT traces are substantially larger by a wide margin. Moreover, all of these existing datasets lack detailed annotations for assessing correctness or for quantifying different types of reasoning failures.

Dataset	Split	# Examples	# Tokens
FOL-TRACES	Train	7.40M	3.51B
	Dev	5K	3.05M
	Test	8.64K	5.25M
	Curated Train	8.80M	786.27M

Table 2: Number of examples and tokens in each dataset split. FOL-TRACES splits are generated from synthetic rules produced by the SymPy program. The curated training set was generated from human-annotated benchmarks augmented with LLM prompting.

2.2 Dataset Generation

We constructed the dataset through a hybrid approach combining human-curated rules, automated symbolic rule generation, and LLM prompting. The detailed breakdown of the inference rules as well as human curated rules are shown in §C, §E, §D, and §F. Our process began with the collection of existing benchmarks for logical reasoning, which provided both direct training material and a basis for generating generalized logical proofs via LLMs.

To expand the diversity of logical forms, we developed templates representing canonical inference rules in first-order logic, including De Morgan's laws, tautologies, and implications. These templates were instantiated with meaningful predicates using LLMs. For example, the general form

$$\neg(\alpha \vee \beta) \rightarrow \neg\alpha \wedge \neg\beta \tag{2}$$

was instantiated as

$$\neg(\text{Sunny}(x) \vee \text{Breezy}(x)) \rightarrow \neg\text{Sunny}(x) \wedge \neg\text{Breezy}(x) \tag{3}$$

where $\text{Sunny}(x)$ and $\text{Breezy}(x)$ are predicates describing environmental properties. Such instantiations were generated by prompting GPT-4o with the abstract logical form and a set of domain-relevant predicate candidates. Additionally, we used symbolic computation tools such as SymPy to synthesize first-order logic expressions, ensuring both syntactic correctness and logical validity. This multi-stage generation process enabled the creation of a high-quality, diverse dataset designed to evaluate and train models on robust logical reasoning tasks.

Algorithm 1 Generate a Random Formula

```
function RandomFormula(Σ, d, m, n)
  if d = 0 then
    return random choice from Σ, n+1
  end if
  op ← rand{∧, ∨, ¬, →}
  if op = ¬ then
    sub, n ← RandomFormula(Σ, d−1, m, n)
    return ¬sub, n+1
  else
    L, n ← RandomFormula(Σ, d−1, m, n)
    R, n ← RandomFormula(Σ, d−1, m, n)
    return op(L, R), n+1
  end if
end function
```

Algorithm 2 High-Level Subexpression Simplification

```
function Simplify(F, d)
  if depth(F) < threshold then
    simplify F directly
  else
    recursively simplify sub-parts
  end if
  if F ∈ {¬, ∧, ∨, →} then
    apply logical rules (e.g., De Morgan, tautology)
  end if
  return simplified F
end function
```

2.2.1 Generate rules with SymPy

To generate verifiably correct first-order logic expressions, we utilize SymPy. The generation process follows the algorithm outlined in Algorithm 1, which recursively constructs logical formulas based on a random selection of operators and symbols.

The depth and complexity of the formula are controlled by the parameters provided. Once a formula is generated, a high-level simplification procedure is applied (described in Algorithm 2) to reduce the formula to its simplest form by applying standard logical inference rules such as De Morgan's laws and eliminating tautologies. The full pseudocode for these processes is provided in §A.1.

Finally, we generate chains of reasoning traces by sequentially linking the simplification steps. This process is akin to CoT reasoning, where each intermediate reasoning step builds upon the previous one, forming a structured sequence that leads

to the final conclusion. However, unlike traditional reasoning traces, our approach ensures that each step is programmatically verified to be correct, leveraging SymPy's symbolic computation capabilities. By chaining these simplifications, we capture a stepwise progression of logically sound transformations, which can be useful for tasks that require a detailed, interpretable sequence of logical reasoning.

3 Dataset Statistics

We begin by examining the distribution of rules in our dataset, which were generated to provide a diverse set of templates for constructing complexity-annotated examples used in pretraining. A summary of this distribution is presented in Figure 2.

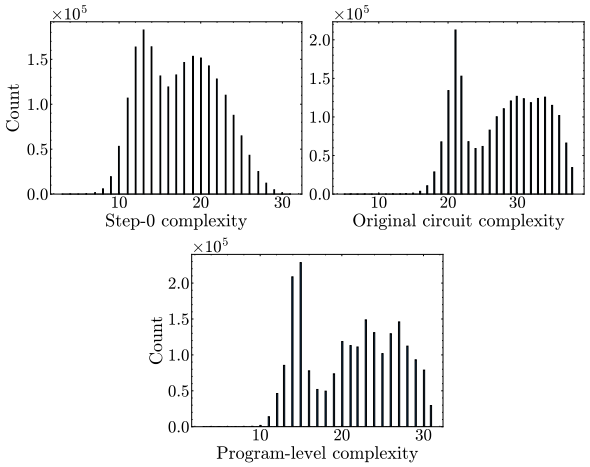


Figure 2: Rule-level complexity distributions.

Figure 2 illustrates the distribution of the circuit complexity $C(\varphi)$, as defined in Equation 1, at step 0. The distribution appears to be a bimodal normal distribution with the mean around 13 and 19. Then, the original circuit complexity in Figure 2 describes $C(\varphi)$+maximum depth of circuit+ number of unique variables. Notably, the distribution appears tri-modal, with a concentration of low-complexity rules around 20, a higher-complexity group near 30, and an overall mean complexity around 35. Then, as the rule simplifies, the total program complexity to generate the rule chains propagates as a trimodal distribution, with mean at 15, 23, and 27.

Next, we examine how rewrite rules evolve throughout the simplification process, as illustrated in Figure 3. The top left panel shows that the number of rewrite steps per rule follows an approximately normal distribution with a long tail, indicating that most rules simplify quickly while a few require many steps. The top right panel depicts elimination complexity at step 0, which is typically low but occasionally high, reflecting rare difficult simplifications. The bottom panel shows that circuit complexity generally increases with rewrite depth, suggesting that longer rewrite sequences correspond to more intricate transformations.

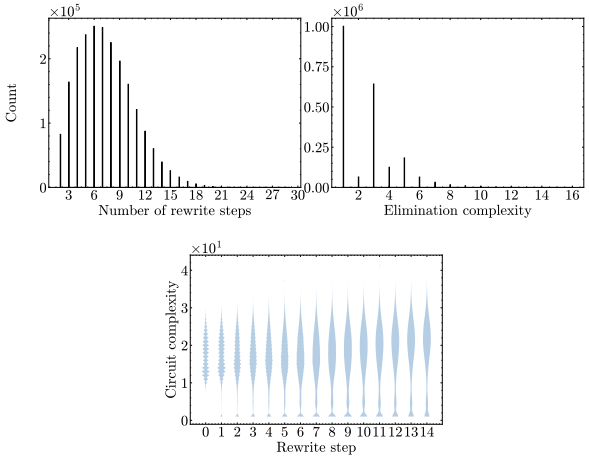


Figure 3: Rule-level dynamics. Top left: distribution of rewrite steps per rule. Top right: elimination complexity at step 0. Bottom: circuit complexity across rewrite steps. Together, these illustrate how complexity evolves and simplifies during the rewrite process.

Furthermore, we analyze how reasoning trace complexity changes across elimination steps (Figure 4). Complexity generally decreases, with a sharp drop near the end, and final expressions show greater variation than initial ones. Figure 9 summarizes token counts per training example.

4 Experiments

We conduct two stages of experiments to assess both the stability of our training setup and the reasoning competence it supports. First, we perform pretraining sanity checks to verify that the FOL-TRACES corpus enables stable optimization and coherent FOL representations in a small LM. We then test this initial pretraining on a one-step TRUE / FALSE evaluation as a minimal reasoning benchmark, as proxies for more complicated FOL tasks we introduce with current reasoning-oriented models. Second, we evaluate reasoning-oriented models on two structured diag-

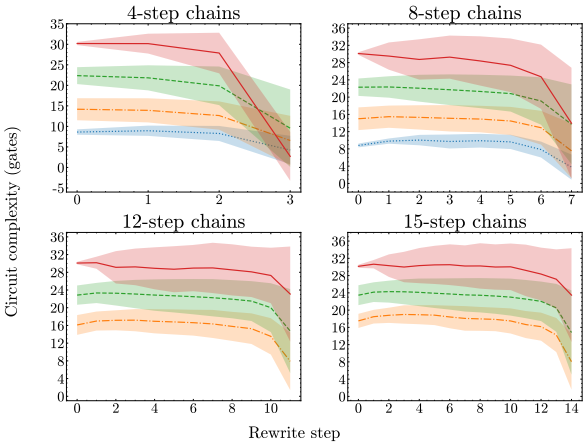


Figure 4: Circuit-complexity evolution by chain length. For each fixed-length bucket, we group traces by their starting gate complexity: *blue solid* = $[0-9]$, *orange dashed* = $[10-19]$, *green dash-dotted* = $[20-29]$, *red dotted* = ≥ 30 gates. Shaded bands show $\pm 1\sigma$ around the mean.

nostic tasks—(1) masked operation prediction and (2) step completions—that probe syntactic awareness and process fidelity, allowing systematic comparison across model families and scales.

4.1 Pretraining Sanity Checks

As a preliminary step, we pretrain Karpathy (2022)'s implementation of the 125M-parameter GPT-2-small architecture (12 layers, 12 heads per layer) from scratch on the FOL-TRACES corpus to validate that the dataset supports stable optimization and coherent FOL representations prior to downstream reasoning evaluations. The model uses an embedding size of 768, a block size of 1024 tokens, and a micro-batch size of 12 with 40 gradient accumulation steps to simulate a larger effective batch. Training employs the AdamW optimizer with a learning rate of 1×10^{-4}, weight decay of 0.1, gradient clipping at 1.0, and no dropout. The learning rate follows a warm-up over 400 iterations and decays to a minimum of 1×10^{-5} across 20,000 iterations. Training was conducted on two NVIDIA RTX A6000 GPUs for approximately 40 hours.

We conduct two complementary studies—one-step TRUE / FALSE evaluation as a basic reasoning task and representation-level probing—to assess how the 125M-parameter GPT-2 variant internalizes FOL.

One-Step TRUE / FALSE Evaluation As a quick diagnostic, we analyze one-step reasoning with definitive truth values, focusing on the simplest strictly evaluable forms of reasoning. For each formula–interpretation pair $(\varphi, \mathcal{I})$, we prompt the model with "<bos> $\varphi \Leftrightarrow$ __" and count a prediction correct when the next token matches the gold TRUE or FALSE. Although the full 50,304 vocabulary is available, the model rapidly collapses its output distribution onto a binary subspace, indicating early acquisition of the truth-evaluation task.

To analyze difficulty, we partition the evaluation set by original complexity into three buckets: low, medium, and high. Figure 11 traces Top-1 accuracy across training tokens, and Table 3 reports the tokens required to reach 70% accuracy on the unseen split as well as the final accuracies. Syntax is learned rapidly: after 5×10^8 tokens (5% of training), accuracy improves from 50% to 60%, indicating that the model collapses its output space to a binary decision.

Bucket	Tokens to 70%	Final (seen)	Final (unseen)
0–21	3.2 B	0.76	0.76
22–32	1.1 B	0.79	0.87
≥ 33	1.1 B	0.89	0.92

Table 3: Learning dynamics on one-step TRUE/FALSE. 'Tokens to 70%' refers to the UNSEEN split.

Probing Geometric Sensitivity to Syntactic Validity LLMs are typically evaluated on task accuracy, which overlooks whether their hidden states encode syntactic well-formedness. We recast our TRUE / FALSE corpus into a binary validity benchmark, creating minimal corruptions by deleting one token from fifty TRUE formulas to disrupt global coherence. We compare our pretrained NanoGPT with BERT-base, and compute the cosine similarities between the predicted embeddings and the valid labels. We quantify the Spearman's ρ bootstrapped over 1,000 samples. NanoGPT's embeddings showed a much stronger alignment with syntactic validity ($\rho \approx 0.68$) than BERT's ($\rho \approx 0.33$), likely because GPT's autoregressive training enforces a kind of "syntactic checksum" that quickly detects missing or malformed tokens, whereas BERT's bidirectional masking spreads error signals across the sequence, weakening its sensitivity to such structural breaks. Overall, this com-

pact FOL corpus and embedding-level RSA reveal that a small language model can encode syntactic structure geometrically when trained with FOL-TRACES.

Model	Representation	ρ	95% CI
NanoGPT (20 k)	final BOS stream	**0.682**	[0.666, 0.695]
BERT-base	final `[CLS]`	0.327	[0.308, 0.345]

Table 4: RSA between embedding similarities and validity labels.

4.2 Structured Reasoning Diagnostic Tasks

We introduce and perform two diagnostic tasks—*masked operation prediction* and *step completion*—that directly probe syntactic awareness and process fidelity. These tasks are designed to isolate core reasoning behaviors: the masked operation prediction task targets atomic-level agreement in first-order logic syntax, while the step completion task evaluates the model's ability to extend reasoning chains consistent with first-order logic. Both tasks are automatically evaluable via programmatic verifiers, enabling scalable and objective assessment. Experiments are conducted using five reasoning models from two families, Phi-3 (Abdin et al., 2024) and Qwen-2.5 (Bai et al., 2023), covering a range of model sizes.

4.2.1 Task 1: Masked Operation Prediction

For a targeted evaluation of distinct aspects of compositional reasoning, we isolate specific dimensions of the model's compositional understanding through targeted input masking. As illustrated in Figure 5, Task 1 employs three masking schemes to separate reasoning components: component masks (blue) obscure subexpressions or entities, operator masks (green) conceal logical or relational connectors, and predicate masks (red) hide function or action tokens. This setup enables controlled analysis of the model's internal representations, revealing how each subtask contributes to the overall reasoning process. The masked operation prediction tasks probe distinct facets of compositional reasoning: component accuracy measures the model's ability to correctly identify and process complete subexpressions within logical expressions; operator accuracy evaluates whether the model correctly applies logical or relational operators between components; and predicate accuracy assesses correctness in identifying function names. Together, these metrics provide a holistic view of the model's systematic reasoning capabilities.

$$\neg\text{Sunny}(x) \wedge \neg\text{Breezy}(x)$$

Figure 5: Masked regions for Task 1 for an example FOL phrase, showing *component* (blue), *operator* (green), and *predicate* (red) areas used in the respective subtasks.

Across all evaluated models, performance varies considerably by both model family and size. Within Phi-3 models, accuracy remains relatively low across all metrics, with *phi-3-medium-128k* performing particularly poorly and *phi-3.5-mini* showing only modest gains. In contrast, the Qwen-2.5 series demonstrates a clear scaling trend: larger models consistently outperform smaller ones, culminating in *qwen2.5-32b*, which achieves the highest scores in every category: 75% component accuracy, 73% operator accuracy, and 80% predicate accuracy. Despite these improvements, even the strongest models show limited robustness and generalization across compositional and operational reasoning tasks. Notably, models explicitly trained for reasoning, such as *phi-3.5-mini*, still underperform substantially compared to larger general-purpose models. This suggests that current reasoning-oriented training approaches alone may be insufficient for achieving strong systematic reasoning performance, highlighting the need for more integrated reasoning and compositional generalization strategies.

Model	**Comp. Acc**	**Oper. Acc**	**Pred. Acc**	**Overall**
phi-3.5-mini	26.00%	32.00%	19.00%	25.67%
phi-3-medium-128k	1.00%	22.00%	17.00%	13.33%
qwen2.5-7b	64.00%	25.00%	62.00%	50.33%
qwen2.5-14b	72.00%	42.00%	75.00%	63.00%
qwen2.5-32b	**75.00%**	**73.00%**	**80.00%**	**76.00%**
Overall	47.60%	38.80%	50.60%	45.67%

Table 5: Model accuracies across components, operators, and predicates prediction tasks.

4.2.2 Task 2: Step Completion

For the *step completion* task, we ask the models to complete the last couple steps given a FOL chain. Using SymPy as an automatic verifier, we programmatically check whether the generated reasoning

steps are logically equivalent to the gold steps. Table 6 summarizes the results, showing that the Qwen2.5 models outperform Phi-3. However, even the best-performing Qwen2.5-32B model struggles to surpass 50% accuracy, indicating that this step completion task remains highly challenging for large reasoning models.

Model	1-Step	2-Step	2-Step Chain
phi-3.5-mini	0.10	0.00	0.00
phi-3-medium-128k	0.10	0.08	0.20
qwen2.5-7b	0.54	0.22	0.44
qwen2.5-14b	0.60	0.26	0.48
qwen2.5-32b	**0.76**	**0.40**	**0.48**
Overall	0.43	0.16	0.27

Table 6: Model accuracies across models for last 1- and 2-step completion tasks

A more detailed break down for 2-steps completion is shown in Figure 6. We see that all the models perform on the last step prediction the better than the two step prediction.

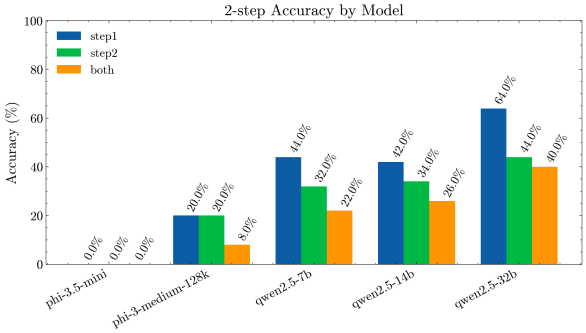
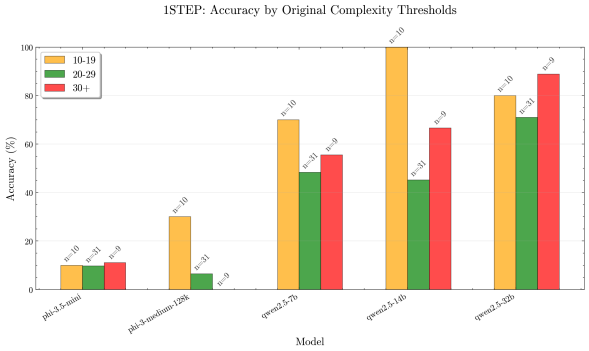
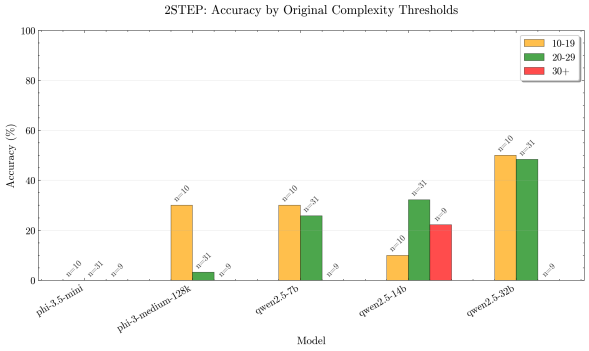


Figure 6: The last 1- and 2-step accuracies by model.

We further stratify the results based on the original complexities of the underlying symbolic FOL rules, as shown in Figure 7. For both the last 1- and 2-step completion tasks, LLMs achieve higher accuracy on low-complexity rules (yellow), while their performance declines markedly as rule complexity increases. As illustrated in Figure 7a, several reasoning LLMs are able to correctly generate low-complexity rules; however, their accuracy drops substantially for medium (green)- and high (red)-complexity examples. Similarly, Figure 7b shows that recent reasoning LLMs perform poorly even on relatively simple cases—with the best-performing model, Qwen2.5-32B, attaining only around 50% accuracy for low- and medium-complexity examples. Most models fail to generate



(a) Last 1-step completion accuracy by original complexity thresholds.



(b) Last 2-step completion accuracy by original complexity thresholds.

Figure 7: Accuracy across the two diagnostic tasks. Bars indicate performance over three complexity ranges (10–19, 20–29, 30+).

correct outputs for high-complexity cases (with the exception of Qwen2.5-14B, which reaches approximately 20% accuracy), underscoring that this task—and derivative n-step completion tasks—would remain highly challenging for current LLMs.

Figure 8 shows the error breakdown by model. Most errors stem from step 1 (blue), step 2 (yellow), or both (red). Additional failures include formatting issues (grey, where the generated FOL expression was malformed and unparseable by SymPy) and chain-only errors, where the model captures general, overarching logical relations of the chain but introduces structural mistakes such as subexpression swaps or misplaced predicates. An example of chain-only errors are shown in Table 14.

Our analyses indicate that reasoning LLMs may not be reasoning as rigorously as expected. They still perform poorly on both tasks 1 and 2, with error patterns revealing fundamental misunderstandings of basic logical syntax.

Figure 8: Error-type breakdown for the 2-step completion task by model. Bars show the percentage of 2-step samples falling into each error category: *Both correct*, *Step1 only*, *Step2 only*, *Both wrong*, *Chain only*, and *Malformed*.

5 Conclusion

We introduce FOL-TRACES, the first large-scale dataset of programmatically verified reasoning traces. In contrast to unverifiable natural language CoT corpora and limited theorem-proving datasets, FOL-TRACES provides 7.4M verified examples with fine-grained complexity annotations, addressing a critical gap in reasoning evaluation and training. To assess its utility, we conduct pretraining sanity checks confirming stable optimization and coherent logical representations, followed by two structured reasoning diagnostics: *masked operation prediction* and *step completion*. Qwen2.5-32B attains 75% accuracy on component prediction, while reasoning-tuned models such as Phi-3.5-mini perform markedly worse; on two-step completions, all models remain below 50%. These results expose persistent limitations in compositional and process-level reasoning, establishing FOL-TRACES as a scalable and verifiable benchmark for advancing genuine reasoning competence beyond surface-level pattern matching.

6 Limitations and Future Work

While FOL-TRACES offers a transparent and verifiable foundation for studying structured reasoning in language models, several limitations remain. The dataset focuses on first-order logic, which, though expressive and formally grounded, captures only part of the reasoning diversity found in naturalistic contexts. Its symbolic generation process, while rigorously validated, diverges from the distributional properties of typical pretraining corpora, potentially limiting transferability to open-domain reasoning. Moreover, our evaluations target diagnostic reasoning tasks rather than full downstream applications, leaving generalization and scaling behavior for future study. Despite these constraints, FOL-TRACES provides a controlled platform for probing symbolic competence and advancing more interpretable and transparent reasoning systems.

7 Risks

From a risk perspective, FOL-TRACES would pose limited direct societal or ethical concerns, as it consists entirely of synthetic, abstract logical inferencing data and contains no personal or sensitive content. However, the use of LLMs for instantiation process may introduce subtle biases from the underlying models and their internet-scale data. Additionally, we contend that if our symbolic reasoning scope is not properly acknowledged in future work, strong performance on our dataset could be misinterpreted as evidence of broadly reliable or human-like reasoning if results are extrapolated beyond.

8 Use of Generative AI

We disclose generative AI usage in the preparation of this work. We used LLMs primarily as a writing assistant for correcting grammar and phrasing, as well as some related work suggestions. Additionally, LLMs were used for aiding in the dataset generation process as described in Section 2.2. We also evaluated current reasoning LLMs as described in Section 4.

References

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Qin Cai, Vishrav Chaudhary, Dong Chen, Dongdong Chen, and 110 others. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *Preprint*, arXiv:2404.14219.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, and 29 others. 2023. Qwen technical report. *Preprint*, arXiv:2309.16609.

Emily M. Bender and Alexander Koller. 2020. Climbing towards NLU: On meaning, form, and understanding in the age of data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5185–5198, Online. Association for Computational Linguistics.

George S. Boolos, John P. Burgess, and Richard C. Jeffrey. 2007. *Computability and Logic*, 5 edition. Cambridge University Press.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Yew Ken Chia, Guizhen Chen, Luu Anh Tuan, Soujanya Poria, and Lidong Bing. 2023. Contrastive chain-of-thought prompting. *Preprint*, arXiv:2311.09277.

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. *Preprint*, arXiv:2110.14168.

Qingxiu Dong, Li Dong, Yao Tang, Tianzhu Ye, Yutao Sun, Zhifang Sui, and Furu Wei. 2025. Reinforcement pre-training. *Preprint*, arXiv:2506.08007.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. 2022. Toy models of superposition. *Preprint*, arXiv:2209.10652.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. 2025. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *Preprint*, arXiv:2502.05171.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

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. 2024. 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.

Ali Hatamizadeh, Syeda Nahida Akter, Shrimai Prabhumoye, Jan Kautz, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, and Yejin Choi. 2025. Rlp: Reinforcement as a pretraining objective. *Preprint*, arXiv:2510.01265.

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*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

Andrej Karpathy. 2022. NanoGPT. `https://github.com/karpathy/nanoGPT`.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Large language models are zero-shot reasoners. *Preprint*, arXiv:2205.11916.

Jack Lindsey, Wes Gurnee, Emmanuel Ameisen, Brian Chen, Adam Pearce, Nicholas L. Turner, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, and 8 others. 2025. On the biology of a large language model. *Transformer Circuits Thread*.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. 2023. Faithful chain-of-thought reasoning. In *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 305–329, Nusa Dua, Bali. Association for Computational Linguistics.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11048–11064, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, and 244 others. 2024. Openai o1 system card. *Preprint*, arXiv:2412.16720.

Mihir Parmar, Nisarg Patel, Neeraj Varshney, Mutsumi Nakamura, Man Luo, Santosh Mashetty, Arindam Mitra, and Chitta Baral. 2024. LogicBench: Towards systematic evaluation of logical reasoning ability of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13679–13707, Bangkok, Thailand. Association for Computational Linguistics.

Yangjun Ruan, Neil Band, Chris J. Maddison, and Tatsunori Hashimoto. 2025. Reasoning to learn from latent thoughts. *Preprint*, arXiv:2503.18866.

Jihoon Tack, Jack Lanchantin, Jane Yu, Andrew Cohen, Ilia Kulikov, Janice Lan, Shibo Hao, Yuandong Tian, Jason Weston, and Xian Li. 2025. Llm pretraining with continuous concepts. *Preprint*, arXiv:2502.08524.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. 2023. Language models don't always say what they think: unfaithful explanations in chain-of-thought prompting. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Gladys Tyen, Hassan Mansoor, Victor Carbune, Peter Chen, and Tony Mak. 2024. LLMs cannot find reasoning errors, but can correct them given the error location. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 13894–13908, Bangkok, Thailand. Association for Computational Linguistics.

ValsAI. 2025. Aime benchmark. Accessed: 2025-05-03.

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. 2022. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. *Preprint*, arXiv:2211.00593.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. *Preprint*, arXiv:2203.11171.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Yeo Wei Jie, Ranjan Satapathy, Rick Goh, and Erik Cambria. 2024. How interpretable are reasoning explanations from prompting large language models? In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 2148–2164, Mexico City, Mexico. Association for Computational Linguistics.

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. 2023. LeanDojo: Theorem proving with retrieval-augmented language models. In *Neural Information Processing Systems (NeurIPS)*.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2024. Metamath: Bootstrap your own mathematical questions for large language models. *Preprint*, arXiv:2309.12284.

A Appendix

A.1 Full Simplification Algorithm

Algorithm 3 Get Depth of Logical Expression

```
function GETDEPTH(expr)
    if expr is And, Or, or Implies then
        return 1 + max(GetDepth of each subexpression)
    else if expr is Not then
        return 1+ GetDepth of inner expression
    else
        return 0
    end if
end function
```

Algorithm 4: Simplify a Subexpression with Depth < 2

```
function TRAVERSEANDSIMPLIFY(expr, d)
    complexity_count ← complexity_count + 1
    if simplified_once then return expr
    end if
    if GetDepth(expr) < d then
        for i ← 0 to d − 1 do
            if i < 2 then
                simplified_expr ← SimplifyLogic(expr, deep=False)
            else
                simplified_expr ← TraverseAndSimplify(expr, i)
            end if
            if simplified_expr ≠ expr then
                simplified_once ← True
                return simplified_expr
            end if
        end for
    end if
    if expr is Not then
        if expr.args[0] == True then return False
        end if
        if expr.args[0] == False then return True
        end if
        if expr.args[0] is Not then return expr.args[0].args[0]
        end if
        if expr.args[0] is Or then return And(Not(x) for x in args)
        end if
        if expr.args[0] is And then return Or(Not(x) for x in args)
        end if
        return Not(TraverseAndSimplify(expr.args[0], d))
    end if
    if expr is Implies then
        if expr.args[0] == expr.args[1] then return True
        end if
        if both args are atoms and simplifiable then return simplified Implies
        end if
        if GetDepth(expr) < 5 then return Or(Not(lhs), rhs)
        end if
        return Implies(TraverseAndSimplify(lhs, d), TraverseAndSimplify(rhs, d))
    end if
    if expr is And or Or then
        for arg in expr.args do
            if GetDepth(arg) < 2 then
                simplified ← SimplifyLogic(arg, deep=False)
                if simplified ≠ arg then
                    return expr.func(...with simplified arg...)
                end if
            end if
        end for
        return expr.func(TraverseAndSimplify(arg, d) for arg in expr.args)
    end if
    return expr
end function

function MAIN(formula)
    simplified_once ← False
    complexity_count ← 0
    result ← TraverseAndSimplify(formula, d)
    return result, complexity_count
end function
```

B Additional Dataset Distribution Information

Table 7: Summary of bucket statistics.

Bucket	#rules	Start	End	Δ
2	82,490	15.59	5.76	+9.83
4	217,298	16.06	7.23	+8.83
8	225,177	17.42	9.61	+7.82
12	87,316	19.91	11.75	+8.16
15	25,925	22.44	13.65	+8.79
20	1,451	25.42	14.68	+10.74
25	15	28.00	13.80	+14.20

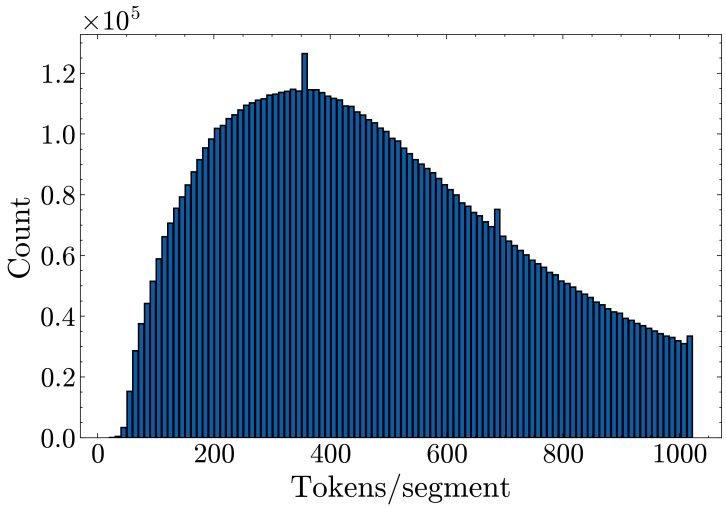
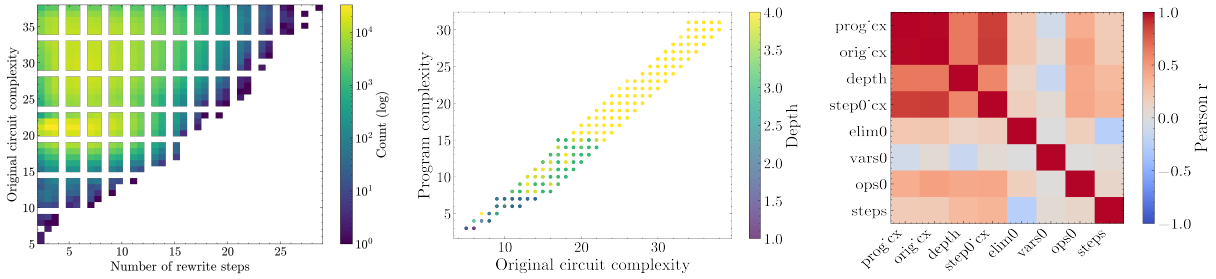


Figure 9: Training segment statistics. Distribution of token-counts per segment.



(a) Log-density heatmap of number of steps vs original complexity

(b) Scatter of original vs program complexity (colour=depth)

(c) Pearson correlation matrix of rule-level metrics

Figure 10: (a) Joint distribution of simplification steps and original circuit complexity; (b) Relationship between original and program complexity, coloured by derivation depth; (c) Correlations among key rule-level features (circuit/program complexity, depth, vars, ops, etc.).

C First Order Logic (FOL) Categories and Explanations

FOL Inference Rule	Symbolic Expression	Explanation
Bidirectional Dilemma (BD)	$((p \rightarrow q) \land (r \rightarrow s)),$ $(p \lor \neg s) \models (q \lor \neg r)$	If two conditional statements are true, given a true antecedent or a false consequent, the respective consequent is true or a respective antecedent is false.
Constructive Dilemma (CD)	$((p \rightarrow q) \land (r \rightarrow s)),$ $(p \lor r) \models (q \lor s)$	If two conditional statements are true and at least one of their antecedents is true, then at least one of their consequents is true.
Destructive Dilemma (DD)	$((p \rightarrow q) \land (r \rightarrow s)),$ $(\neg q \lor \neg s) \models (\neg p \lor \neg r)$	If two conditional statements are true, and one of their consequents must be false, then one of their antecedents must be false.
Disjunctive Syllogism (DS)	$((p \lor q) \land \neg p) \models q$	Disjunctive elimination. If we know one of two statements, p or q, to be true, and one of them is not true, the other must be true.
Hypothetical Syllogism (HS)	$((p \rightarrow q) \land (q \rightarrow r))$ $\models (p \rightarrow r)$	Chain argument rule or transitivity of implication.
Modus Ponens (MP)	$((p \rightarrow q) \land p) \models q$	Implication elimination rule. If p implies q and p is true, the statement can be replaced with q.
Modus Tollens (MT)	$((p \rightarrow q) \land \neg q) \models \neg p$	Implication elimination rule. If p implies q and q is false, the statement can be replaced with $\neg p$.
Universal Instantiation (UI)	$\forall x P(x) \implies \exists a P(a)$	If a statement P holds for a variable x, then there exists a particular value a for the statement to be true.
Existential Generalization (EG)	$\exists x P(x) \implies P(a)$	If a statement P holds true for some subset of variables x, then there is a particular value $x = a$ for which P holds true.
FOL proofs & general statements	–	–

Table 8: First Order Logic (FOL) inference rule categories and explanations.[2]

FOL Properties	Symbolic Expression
Distributive (Dist)	$(p \vee (q \wedge r)) \leftrightarrow ((p \vee q) \wedge (p \vee r))$ $(p \wedge (q \vee r)) \leftrightarrow ((p \wedge q) \vee (p \wedge r))$
Association (AS)	$(p \vee (q \vee r)) \leftrightarrow ((p \vee q) \vee r)$ $(p \wedge (q \wedge r)) \leftrightarrow ((p \wedge q) \wedge r)$
Tautology (TT)	$p \leftrightarrow (p \vee p)$ $p \leftrightarrow (p \wedge p)$
Transposition (TS)	$(p \rightarrow q) \leftrightarrow (\neg q \rightarrow \neg p)$
Importation (IM)	$(p \rightarrow (q \rightarrow r)) \leftrightarrow ((p \wedge q) \rightarrow r)$
Exportation (EX)	$((p \wedge q) \rightarrow r) \rightarrow (p \rightarrow (q \rightarrow r))$
Double Negation (DN)	$p \leftrightarrow \neg\neg p$
De Morgan's Law (DM)	$\neg(p \wedge q) \leftrightarrow (\neg p \vee \neg q)$ $\neg(p \vee q) \leftrightarrow (\neg p \wedge \neg q)$
Negation of XOR (NX)	$\neg(p \oplus q) \leftrightarrow (\neg p \oplus \neg q)$ $\neg(p \oplus q) \leftrightarrow (p \odot q)$
Negation of XNOR (NN)	$\neg(p \odot q) \leftrightarrow (\neg p \odot \neg q)$ $\neg(p \odot q) \leftrightarrow (p \oplus q)$

Table 9: First Order Logic (FOL) Basic Properties[3]

D Elimination Rules

E Complex FOL Expressions

F Training Data

G Sanity Check TRUE/FALSE Experiment Plot

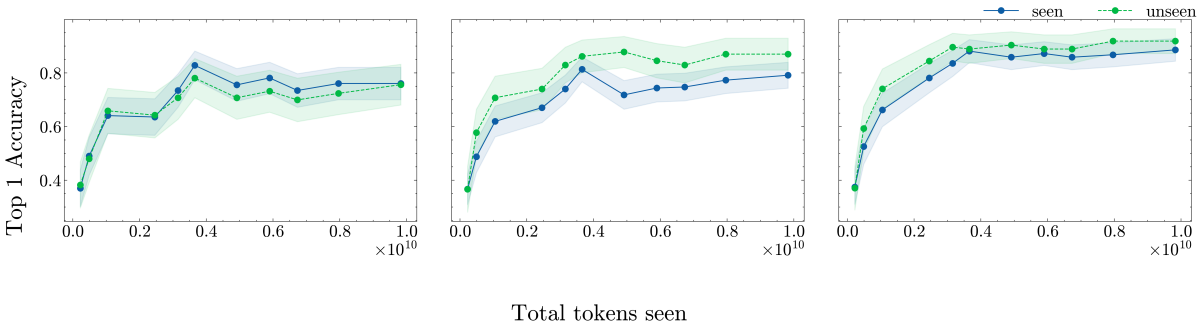


Figure 11: Top-1 accuracy vs. cumulative tokens for three complexity tertiles (0–21, 22–32, ≥33). Shaded regions denote 95% confidence interval.

H Task 2 Prompts

I Task 2 Example

	Symbolic Expression
E0	$p \vee \text{True} \leftrightarrow \text{True}$
E1	$p \vee \text{False} \leftrightarrow p$
E2	$p \wedge \text{True} \leftrightarrow p$
E3	$p \wedge \text{False} \leftrightarrow \text{False}$
E4	$\text{True} \vee p \leftrightarrow \text{True}$
E5	$\text{False} \vee p \leftrightarrow p$
E6	$\text{True} \wedge p \leftrightarrow p$
E7	$\text{False} \wedge p \leftrightarrow \text{False}$
E8	$p \vee p \leftrightarrow p$
E9	$p \wedge p \leftrightarrow p$
E10	$p \wedge \neg p \leftrightarrow \text{False}$
E11	$p \vee \neg p \leftrightarrow \text{True}$
E12	$\neg p \wedge p \leftrightarrow \text{False}$
E13	$\neg p \vee p \leftrightarrow \text{True}$
E14	$p \wedge (p \vee q) \leftrightarrow p$
E15	$p \wedge (\neg p \vee q) \leftrightarrow (p \wedge \neg p) \vee (p \wedge q)$ $\leftrightarrow \text{False} \vee (p \wedge q) \leftrightarrow p \wedge q$
E16	$p \wedge (\neg p \vee q) \leftrightarrow \text{False} \vee (p \wedge q) \leftrightarrow p \wedge q$
E17	$p \wedge (\neg p \vee q) \leftrightarrow (p \wedge \neg p) \vee (p \wedge q) \leftrightarrow p \wedge q$
E18	$p \vee (p \wedge q) \leftrightarrow p$
E19	$p \vee (p \wedge q \wedge r) \leftrightarrow p$
E20	$r \vee (p \wedge q \wedge r) \leftrightarrow r$
E21	$r \vee (p \wedge q \wedge r \wedge s) \leftrightarrow r$
E22	$p \vee (\neg p \wedge q) \leftrightarrow (p \vee \neg p) \wedge (p \vee q)$ $\leftrightarrow \text{True} \wedge (p \vee q) \leftrightarrow (p \vee q)$
E23	$p \vee (\neg p \wedge q) \leftrightarrow \text{True} \wedge (p \vee q) \leftrightarrow (p \vee q)$
E24	$p \vee (\neg p \wedge q) \leftrightarrow (p \vee \neg p) \wedge (p \vee q) \leftrightarrow (p \vee q)$
E25	$p \vee \neg(p \wedge q) \leftrightarrow p \vee (\neg p \vee \neg q)$ $\leftrightarrow (p \vee \neg p) \vee \neg q \leftrightarrow \text{True} \vee \neg q \leftrightarrow \text{True}$
E26	$p \vee \neg(p \wedge q) \leftrightarrow p \vee (\neg p \vee \neg q)$ $\leftrightarrow p \vee \neg p \vee \neg q \leftrightarrow \text{True} \vee \neg q \leftrightarrow \text{True}$
E27	$p \vee \neg(p \wedge q) \leftrightarrow (p \vee \neg p) \vee \neg q \leftrightarrow \text{True} \vee \neg q \leftrightarrow \text{True}$
E28	$p \vee \neg(p \wedge q) \leftrightarrow p \vee (\neg p \vee \neg q) \leftrightarrow \text{True} \vee \neg q \leftrightarrow \text{True}$
E29	$p \vee \neg(p \wedge q) \leftrightarrow p \vee (\neg p \vee \neg q) \leftrightarrow (p \vee \neg p) \vee \neg q \leftrightarrow \text{True}$
E30	$p \wedge \neg(p \vee q) \leftrightarrow p \wedge (\neg p \wedge \neg q) \leftrightarrow (p \wedge \neg p) \wedge \neg q$ $\leftrightarrow \text{False} \wedge \neg q \leftrightarrow \text{False}$
E31	$p \wedge \neg(p \vee q) \leftrightarrow (p \wedge \neg p) \wedge \neg q \leftrightarrow \text{False} \wedge \neg q \leftrightarrow \text{False}$
E32	$p \wedge \neg(p \vee q) \leftrightarrow p \wedge (\neg p \wedge \neg q) \leftrightarrow \text{False} \wedge \neg q \leftrightarrow \text{False}$
E33	$p \wedge \neg(p \vee q) \leftrightarrow p \wedge (\neg p \wedge \neg q) \leftrightarrow (p \wedge \neg p) \wedge \neg q \leftrightarrow \text{False}$

Table 10: First Order Logic (FOL) Elimination Rules[4]

	Symbolic Expression	Combination of FOL Rules	Included in Training
C1	$(((a \vee b) \rightarrow q) \wedge \neg q) \rightarrow (\neg a \wedge \neg b)$	MT + DM	✗
C2	$(((a \wedge \neg b) \rightarrow q) \wedge \neg q) \rightarrow (\neg a \vee b)$	MT + DM + DN	✓
C3	$(p \rightarrow q), (q \rightarrow r), (s \rightarrow t),$ $(\neg t \vee \neg r) \rightarrow (\neg p \vee \neg s)$	TS + DD	✓
C4	$(p \vee (q \wedge (a \vee b))) \leftrightarrow$ $((p \vee q) \wedge ((p \vee a) \vee b))$	DS + AS	✓
C5	$(p \wedge ((a \wedge b) \vee q \vee r)) \leftrightarrow$ $(((p \wedge a) \wedge b) \vee (p \wedge q) \vee (p \vee r)) \leftrightarrow$ $(((p \wedge a) \wedge b) \vee (p \wedge (q \vee r)))$	DS + AS	✓
C6	$((p \wedge q \wedge r) \vee (a \wedge p \wedge b) \vee (c \wedge d \wedge e)) \leftrightarrow$ $((p \wedge ((q \wedge r) \vee (a \wedge b))) \vee (c \wedge d \wedge e))$	DS + AS	✗
C7	$(p \vee (q \wedge r \wedge (a \vee b) \wedge s)) \leftrightarrow$ $((p \vee q) \wedge (p \vee r) \wedge (p \vee a \vee b) \wedge (p \vee s))$	DS + AS	✓
C8	$(p \vee (q \wedge (p \vee b) \wedge r)) \leftrightarrow$ $((p \vee q) \wedge (p \vee b) \wedge (p \vee r)) \leftrightarrow$ $(p \vee (q \wedge b \wedge r))$	DS + AS + TT	✓
C9	$\neg(p \vee (q \wedge (\neg a \vee b) \wedge \neg r)) \leftrightarrow$ $\neg((p \vee q) \wedge (p \vee \neg a \vee b) \wedge (p \vee \neg r)) \leftrightarrow$ $(\neg(p \vee q) \vee \neg(p \vee \neg a \vee b) \vee \neg(p \vee \neg r)) \leftrightarrow$ $((\neg p \wedge \neg q) \vee (\neg p \wedge a \wedge \neg b) \vee (\neg p \wedge r)) \leftrightarrow$ $(\neg p \wedge (\neg q \vee (a \wedge \neg b) \vee r))$	DS + DM + DN	✗
C10	$(\neg p \rightarrow q) \leftrightarrow (\neg q \rightarrow p)$	TS + DN	✓
C11	$(p \rightarrow \neg q) \leftrightarrow (q \rightarrow \neg p)$	TS + DN	✓
C12	$((a \wedge b) \rightarrow q) \leftrightarrow (\neg q \rightarrow (\neg a \vee \neg b))$	TS + DM	✗
C13	$(p \rightarrow (\neg a \vee \neg b)) \leftrightarrow ((a \wedge b) \rightarrow \neg p)$	TS + DM	✓
C14	$\neg((a \vee b) \oplus c \oplus d) \leftrightarrow (\neg(a \vee b) \oplus \neg c \oplus \neg d)$	DM + NX	✓
C15	$\neg(c \oplus (\neg a \vee b) \oplus d) \leftrightarrow (\neg c \oplus (a \wedge \neg b) \oplus \neg d)$	DM + NX	✗
C17	$\neg(p \odot q \odot (a \vee \neg b)) \leftrightarrow (\neg p \odot \neg q \odot (\neg a \wedge b))$	DM + NN	✓
C18	$((a \wedge b) \rightarrow q), ((a \wedge \neg c) \rightarrow s), (\neg q \vee \neg s) \rightarrow$ $((\neg a \vee \neg b) \vee (\neg a \vee c)) \rightarrow$ $(\neg a \vee \neg b \vee c) \rightarrow \neg(a \wedge b \wedge \neg c)$	DD + DN + DM + AS + TT	✗
C20	$((a \vee b) \rightarrow q), (r \rightarrow s), (a \vee b \vee r) \rightarrow (q \vee s)$	CD + AS	✓
C21	$(p \rightarrow (a \vee b)), (r \rightarrow s), (p \vee r) \rightarrow (a \vee (b \vee s))$	CD + AS	✓
C22	$(p \rightarrow q), ((a \vee \neg b) \rightarrow s), (p \vee \neg s) \rightarrow$ $(q \vee (\neg a \wedge b)) \rightarrow ((q \vee \neg a) \wedge (q \vee b))$	BD + DN + DM + DS	✗
C23	$(p \rightarrow q), ((\neg a \wedge \neg b) \rightarrow s), (p \vee \neg s) \rightarrow$ $q \vee \neg(\neg a \wedge \neg b) \rightarrow (q \vee a) \vee b$	BD + DM + AS	✓

Table 11: Complex FOL Expressions[5]. (BD = Bidirectional Dilemma, CD = Constructive Dilemma, DD = Destructive Dliemma, MT = Modus Tollens, DM = De Morgan's, DN = Double Negation, DS = Distribution, AS = Association, TS = Transposition, TT = Tautology, NN = Negation of XNOR, NX = Negation of XOR)

	# Examples	# Tokens		# Examples	# Tokens
BD	102.43K	778.57K	DM	740.92K	34.11M
CD	856.78K	71.70M	Dist	268.76K	17.97M
DD	806.15K	71.70M	XOR*	17.46K	793.75K
DS	321.31K	12.79M	XNOR*	15.42K	668.07K
HS	429.94K	23.20M	XOR-XNOR*	14.49K	577.84K
MP	237.80K	7.49M			
MT	285.19K	11.10M			
UI	123.00K	4.03M			
EG	9.10K	263.0K			
General/fol proof	2.27M	286.12M			
C2	94.36K	5.26M	E0	5.23K	83.92K
C3	108.56K	20.21M	E1	5.34K	141.32K
C4	102.73K	8.03M	E2	5.28K	147.74K
C5	107.64K	16.67M	E3	5.46K	88.60K
C7	109.03K	17.57M	E4	5.18K	82.91K
C8	97.90K	12.38M	E5	5.31K	146.08K
C10	102.43K	3.84M	E6	5.35K	148.27K
C11	86.92K	3.55M	E7	5.28K	84.61K
C13	93.22K	5.17M	E8	5.40K	188.11K
C14	109.20K	10.03M	E9	5.37K	194.38K
C17	108.53K	10.04M	E10	5.23K	140.78K
C20	109.10K	10.58M	E11	5.22K	139.83K
C21	109.37K	11.21M	E12	5.16K	134.20K
C23	109.45K	14.94M	E13	5.18K	135.07K
			E14	5.60K	135.07K
			E15	5.58K	720.92K
			E16	5.56K	441.38K
			E17	5.58K	569.09K
			E18	5.61K	256.89K
			E19	5.37K	291.75K
			E20	5.21K	286.32K
			E21	5.37K	352.57K
			E22	5.61K	734.95K
			E23	5.59K	470.14K
			E24	5.63K	586.41K
			E25	5.58K	703.58K
			E26	5.57K	714.06K
			E27	5.58K	501.50K
			E28	5.57K	502.50K
			E29	5.59K	625.27K
			E30	5.59K	720.77K
			E31	5.59K	499.31K
			E32	5.59K	498.49K
			E33	5.60K	648.88K
Complex Total	**1.45M**	**149.48M**	**Eliminations Total**	**185.09K**	**12.24M**
Random	15.26M	1.89B			
Total	**24.07M**	**2.67B**			

Table 12: Full Breakdown of the Training Dataset.[6]The labels are consistent with the FOL types described in Table 8, Table 9, Table 10, and Table 11. Note: not all basic properties (Table 9) of FOL were included explicitly in generation. This is because we qualitatively saw that the massive random generation sufficiently and implicitly (and sometimes explicitly) captured the basic properties.

*We explicitly included negations of XOR, negations of XNOR and the equivalences between XOR and XNOR.

1-Step Prompt	**2-Step Prompt**
You are given a first-order logic equivalence chain where the last step is missing (replaced by <BLANK>). Your task is to complete the chain by providing the final step that logically follows from the previous steps. The chain uses \Leftrightarrow to separate each step, and each step should be a valid logical expression. Respond with ONLY the final step that should replace <BLANK>, with no explanation, no quotes, no extra text. The final step should be a complete logical expression in parentheses.	You are given a first-order logic equivalence chain where the last 2 steps are missing (replaced by <BLANK>). Your task is to complete the chain by providing the final 2 steps that logically follow from the previous steps. The chain uses \Leftrightarrow to separate each step, and each step should be a valid logical expression. Respond with ONLY the final 2 steps separated by \Leftrightarrow that should replace the <BLANK>s, with no explanation, no quotes, no extra text. Each step should be a complete logical expression in parentheses.

Table 13: System prompts for the 1-step and 2-step diagnostic tasks.

Examples	Expressions and Evaluation Results
Step-2 correct, chain wrong	**Answer:** $P \Leftrightarrow \neg P \Leftrightarrow$ False **Prediction:** $P \Leftrightarrow Q \Leftrightarrow$ False $acc_{\text{2step\_step2}} = 1$ (both end with False) $acc_{\text{2step\_chain}} = 0$ $[(P \Leftrightarrow \neg P) \Leftrightarrow \text{False} \neq (P \Leftrightarrow Q) \Leftrightarrow \text{False}]$
Different Step-2, chain still correct	**Answer:** $P \Leftrightarrow \neg P \Leftrightarrow$ False **Prediction:** $P \Leftrightarrow (P \wedge \neg P) \Leftrightarrow$ False $acc_{\text{2step\_step2}} = 1$ (both end with False) $acc_{\text{2step\_chain}} = 1$ $[(P \Leftrightarrow \neg P) \Leftrightarrow \text{False} \equiv (P \Leftrightarrow (P \wedge \neg P)) \Leftrightarrow \text{False}]$

Table 14: Examples illustrating different evaluation outcomes for the *step completion* task.