

SOLID: Infusing Symbolic Logic into LLMs via Dual-Verified Backward Reasoning

Wei jie Yu, Yuhang Dong, Feifei Ma*

Laboratory of Parallel Software and Computational Science, ISCAS, Beijing, China

University of Chinese Academy of Sciences, Beijing, China

yuweijie25@mails.ucas.ac.cn, dongyuhang22@mails.ucas.ac.cn, maff@ios.ac.cn

Abstract

Large Language Models (LLMs) often struggle with rigorous logical reasoning, suffering from hallucinations and inconsistent deduction. Chain-of-Thought (CoT) *prompting* relies heavily on the frozen model’s inherent capabilities, while neuro-symbolic approaches face a “translation reliability” bottleneck in which semantic parsing errors mislead external solvers. Recent work attempts to *infuse* symbolic capabilities via fine-tuning, but typically mimics *forward reasoning*, suffering from search-space explosion and irrelevant intermediate facts that dilute attention. We propose **SOLID** (Symbolic Oriented Logic Infusion via Dual-verification), a framework that synthesizes high-quality supervision through a Dual-Verification Mechanism: bidirectional consistency checks filter out hallucinated translations. Crucially, instead of forward chaining, we execute validated translations to generate *backward reasoning traces*—verifying necessary premises from the goal. By fine-tuning on these focused, goal-oriented paths, SOLID infuses the model with the efficient deduction process of a symbolic engine. Experiments on PRONTOQA, LOGIC-NLI, and PROOFWRITER demonstrate state-of-the-art accuracy, significantly outperforming CoT baselines, solver-based methods, and forward-chaining variants. Code and data are available at: <https://github.com/ejie22/SOLID>.

1 Introduction

Large Language Models (LLMs) have achieved remarkable success across various natural language processing tasks, exhibiting emergent abilities in few-shot learning and instruction following (Brown et al., 2020; Ouyang et al., 2022). However, despite their fluency, LLMs continue to struggle with rigorous multi-step logical reasoning (Cheng et al., 2025; Saparov and He, 2023). They are prone to hallucinations and unfaithful reasoning processes,

particularly in scenarios requiring strict adherence to logical rules, such as mathematical proving or legal reasoning (Ji et al., 2023; Turpin et al., 2023).

To bridge this gap, Chain-of-Thought (CoT) prompting (Wei et al., 2022; Kojima et al., 2022) has emerged as a popular technique to elicit reasoning, and now anchors a broad family of prompt-engineering strategies (Liu et al., 2026). By encouraging the model to generate intermediate steps, CoT significantly improves performance. However, prompting strategies rely fundamentally on the *frozen* model’s parametric knowledge. They lack explicit alignment with symbolic logic, often leading to plausible-sounding but logically invalid deduction steps (Saparov and He, 2023; Turpin et al., 2023). Furthermore, standard CoT typically suffers from a lack of goal orientation, causing models to lose track of the objective in complex search spaces (Kazemi et al., 2023; Yao et al., 2023).

Alternatively, Neuro-Symbolic approaches seek to combine LLMs with external deductive solvers to guarantee correctness. In frameworks such as Logic-LM (Pan et al., 2023) and Program of Thoughts (Chen et al., 2023), the LLM acts as a semantic parser, translating natural language into formal logic (e.g., Prolog) for execution. While theoretically rigorous, these systems face a critical “translation reliability” bottleneck. As noted by prior studies (Feng et al., 2024; Lyu et al., 2023), if the LLM generates an incorrect formal representation—such as misinterpreting a quantifier—the external solver will inevitably produce a valid but factually wrong conclusion, or simply fail to execute. Moreover, relying on external tools restricts the model’s flexibility.

To address these dependencies, recent works infuse symbolic capabilities into LLMs through instruction tuning (Feng et al., 2024), part of a wider effort to train compact models to reason by fine-tuning them on traces produced by a stronger source (Wang et al., 2026). However, these

* Corresponding author.

methods predominantly rely on Forward Chaining (bottom-up reasoning), starting from known facts and iteratively deriving new conclusions. Forward reasoning suffers from search-space explosion in complex multi-hop scenarios: by exhaustively generating all possible implications regardless of their relevance to the goal, it produces *irrelevant intermediate facts* that dilute the model’s attention as the reasoning chain lengthens. Prior backward-chaining strategies (Kazemi et al., 2023; Lee and Hwang, 2025) alleviate such inefficiencies but remain confined to prompting or solver-augmented paradigms, failing to achieve parameter-level infusion.

In this paper, we propose **SOLID** (Symbolic Oriented Logic Infusion via Dual-verification), which aligns LLMs with a Horn-like fragment of First-Order Logic (FOL) through verifiable backward reasoning. By learning to reason recursively from the conclusion back to necessary premises, SOLID naturally prunes irrelevant branches and maintains a *focused, minimal* reasoning path even in deep deduction. Specifically, a teacher LLM translates logical problems into formal specifications, and a Dual-Verification Mechanism uses an automated solver to filter out translation errors via bidirectional consistency. The solver then produces backward reasoning traces from the validated specifications; fine-tuning on these goal-directed paths infuses the student LLM with the solver’s deductive process while removing the need for external tools at inference.

Our contributions are summarized as follows:

- We propose **SOLID**, the first framework to fine-tune LLMs on full solver-generated backward reasoning traces. By training on the entire search history—including failed attempts and dead ends—SOLID trains models to generate explicit failure-detection and backtracking steps, and addresses search-space explosion.
- We develop a specialized backward solver for a Horn-like FOL fragment, paired with a Dual-Verification Mechanism that validates neuro-symbolic translations via bidirectional consistency. The resulting supervision consists of label-consistent translations paired with solver-verified, logically sound traces.
- We conduct extensive experiments on three benchmarks: PRONTOQA (Saparov and He,

2023), LOGICNLI (Tian et al., 2021), and PROOFWRITER (Tafjord et al., 2021), where SOLID achieves state-of-the-art performance, significantly outperforming existing baselines.

2 Preliminaries

2.1 Problem Formulation

We define the logical reasoning task as determining the semantic relationship between a context $\mathcal{C}$ (comprising facts $\mathcal{F}$ and rules $\mathcal{R}$) and a query S . Adopting a unified provability schema, we classify this relationship into four states—*Entailment*, *Contradiction*, *Neutral*, and *Self-Contradiction*—based on the joint provability of S and $\neg S$ (see Figure 1).

To support rigorous deduction, natural language inputs are mapped into First-Order Logic (FOL)—specifically, the Horn-like fragment characterized in Appendix A—where facts are represented as ground atoms (e.g., $Red(Apple)$) and rules as logical implications. To streamline resolution, we normalize rules into standard forms depending on whether the quantified variable appears free in the consequent.

For rules where the consequent does *not* depend on the quantified variable, we apply the following standard FOL equivalence to convert existential antecedents into universally quantified implications:

$$(\exists x. P(x)) \rightarrow Q \equiv \forall x. (P(x) \rightarrow Q) \quad (1)$$

This side-condition is essential: the equivalence holds *only* when x does not occur free in Q . For rules where the consequent *does* contain x (e.g., “If someone is concerned and serious, then *he* is entire,” where $Q = Entire(x)$), we directly use the universally quantified form $\forall x. (P(x) \rightarrow C(x))$ without invoking the above rewriting. This case-by-case treatment ensures semantic fidelity across all rule types.

The purpose of the normalization is to give every rule a single search interface. After rewriting, an existential-antecedent rule carries one outermost $\forall$ over an implication, so it enters backward search through the same goal-to-head unification path as an ordinary universal rule and needs no dedicated proof strategy. Because the rewriting is an exact equivalence whenever its side condition holds, the normalized theory is logically identical to the original, and the solver’s soundness and completeness over the supported fragment (Appendix A) carry over unchanged.

	Proved $\neg S$? No	Proved $\neg S$? Yes
Proved S ? Yes	Entailment (True) $\mathcal{C} \vdash S \wedge \mathcal{C} \not\vdash \neg S$	Self-Contradiction (Inconsistent) $\mathcal{C} \vdash S \wedge \mathcal{C} \vdash \neg S$
Proved S ? No	Neutral (Unknown) $\mathcal{C} \not\vdash S \wedge \mathcal{C} \not\vdash \neg S$	Contradiction (False) $\mathcal{C} \not\vdash S \wedge \mathcal{C} \vdash \neg S$

Figure 1: Classification by the joint provability of S and $\neg S$ under context $\mathcal{C}$. $\vdash$: provable from $\mathcal{C}$; $\not\vdash$: not provable.

2.2 The Neuro-Symbolic Paradigm

Standard neuro-symbolic frameworks decouple reasoning into *semantic parsing* and *symbolic execution*. A neural parser P_θ translates input X into a formal specification Φ , which a deterministic solver E executes to derive the answer y . However, this pipeline suffers from a *Translation Bottleneck*: unlike robust neural models, symbolic solvers are brittle; even minor parsing errors cause execution failure, leading to severe error cascading. This bottleneck concerns *what* the solver executes; in our pipeline it is addressed at data-synthesis time by Dual-Verification (Section 3.2), and is moot at inference, where the student runs no parse-then-solve pipeline.

A separate question is *how* the solver searches. Forward Chaining (data-driven) often suffers from *search space explosion* by deriving irrelevant intermediate facts. In contrast, we adopt Backward Chaining (goal-driven), which initiates search from the query and recursively verifies only necessary premises. This approach naturally prunes irrelevant branches, yielding a focused and minimal reasoning trace essential for efficient question answering.

3 Methodology

The overall architecture of SOLID is illustrated in Figure 2. Our three-stage pipeline synthesizes high-quality supervision data: (1) **Neural Semantic Translation** (Section 3.1), (2) **Dual-Verification Mechanism** (Section 3.2), and (3) **Backward Trace Synthesis** (Section 3.3), whose output is used to fine-tune the Student LLM.

3.1 Neural Semantic Translation

The first step is to bridge the gap between informal natural language and formal symbolic representation. We employ a powerful off-the-shelf LLM (e.g., DeepSeek-V3) as the *Semantic Parser*, denoted as $\mathcal{M}_{teacher}$.

Given a dataset sample (X, y) , where the input X comprises a context $\mathcal{C}$ and a query statement S , and y is the gold provability label drawn from *Entailment*, *Contradiction*, *Neutral*, or *Self-Contradiction*, we construct a few-shot prompt $\mathcal{P}$ that instructs the model to: (1) decompose $\mathcal{C}$ into Facts $\mathcal{F}$ and Rules $\mathcal{R}$; and (2) parse $\mathcal{F}, \mathcal{R}$ and S into a symbolic format Φ . The entire natural language input is converted into a structured symbolic specification Φ , comprising the sets of normalized facts $\mathcal{F}_n$, rules $\mathcal{R}_n$, and the formal target statement S_n :

$$\Phi = (\mathcal{F}_n, \mathcal{R}_n, S_n) \leftarrow \mathcal{M}_{teacher}(X; \mathcal{P}) \quad (2)$$

Driven by few-shot instructions, the output follows standardized First-Order Logic formats, such as *Universal Rules* and *Ground Implications* (details in Appendix A). Note that while this format guarantees *syntactic* executability for the solver, *semantic* errors (e.g., hallucinations) may persist, necessitating the subsequent verification stage.

3.2 The Dual-Verification Mechanism

To safeguard against translation errors, we employ a Dual-Verification Mechanism that validates the generated symbolic context against the ground truth label y . Our Backward Solver performs **Dual-Branch Execution**, attempting to construct proof trees for both the statement S_n and its negation $\neg S_n$.

A translated sample is retained if and only if the solver’s execution outcomes strictly align with the expected logical profile defined by y , as illustrated in Figure 1. For instance, a label of *Entailment* requires the solver to prove S_n while failing to prove $\neg S_n$. This rigorous alignment effectively filters out hallucinated translations that are syntactically valid but semantically inconsistent with the original text. Stages 2 and 3 share a single solver engine (Algorithm 1, Appendix B): the same implementation produces these verification verdicts and the training traces of Section 3.3.

3.3 Backward Trace Synthesis

Once a symbolic specification Φ passes the Dual-Verification mechanism, it is deemed a high-quality

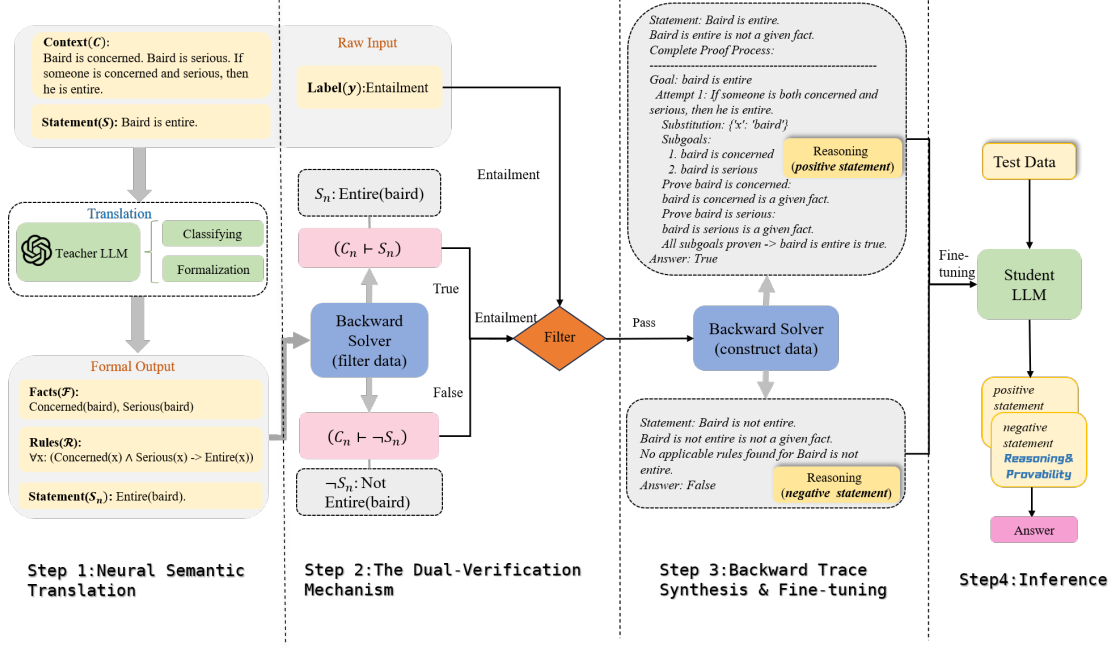


Figure 2: **Overview of the SOLID Framework.** The pipeline is organized into two distinct phases: **Training Data Synthesis (Steps 1–3)** and **Inference (Step 4)**. During the training phase: (1) **Neural Semantic Translation**: A Teacher LLM translates natural language into symbolic facts and rules; (2) **The Dual-Verification Mechanism**: To filter translation errors, the Backward Solver verifies logical consistency for both the positive statement (S_n) and its negation ($\neg S_n$); (3) **Backward Trace Synthesis & Fine-tuning**: The solver generates goal-oriented reasoning traces to supervise the Student LLM. Finally, in the **Inference Phase**: (4) **Inference**: The fine-tuned Student LLM applies its infused symbolic capabilities to perform solver-free reasoning on unseen test data.

representation of the ground truth. The final objective is to convert the solver’s execution path into a natural language Backward Chain-of-Thought (CoT), denoted as C_{bw} , which serves as the supervision signal for the student model. Figure 3 illustrates the dual-branch traces produced by the solver for a representative sample, with the five trace components colour-coded.

3.3.1 Backward Solver Execution Process

Our solver executes a rigorous goal-directed search to construct validity proofs. The complete execution procedure is formalized in Algorithm 1 (Appendix B). To address the efficiency bottlenecks common in naive backward chaining (e.g., infinite recursion or redundant search), we introduce a **Simplicity-First Strategy** that dynamically prioritizes reasoning paths.

Simplicity-First Rule Selection. When multiple rules can derive a goal G , random selection often leads to suboptimal paths. Instead, we retrieve all candidate rules Ω_G and rank them using a heuristic scoring function $\text{Score}(r) \in \{1, 2, 3\}$. Rules are categorized into three levels based on the state of their antecedents in the current knowledge base:

- **Priority 3: Directly Grounded (Highest).** The rule’s antecedents are already explicitly satisfied by known facts (e.g., all subconditions are in $\mathcal{F}$). Applying this rule requires no further search and is verified immediately.
- **Priority 2: Inferentially Plausible (Medium).** The antecedents are not facts, but they appear as consequents in other available rules. This indicates that a valid deductive chain likely exists, warranting further recursion.
- **Priority 1: Speculative (Lowest).** The antecedents are neither facts nor derivable from other rules. While not definitively false, this path represents the least promising search direction.

By sorting Ω_G in descending order of priority, the solver mimics human intuition: validating direct facts first, then deducing likely consequences, and minimizing speculative search. Ties are broken deterministically: rules sharing a score retain their order of appearance in the parsed knowledge base

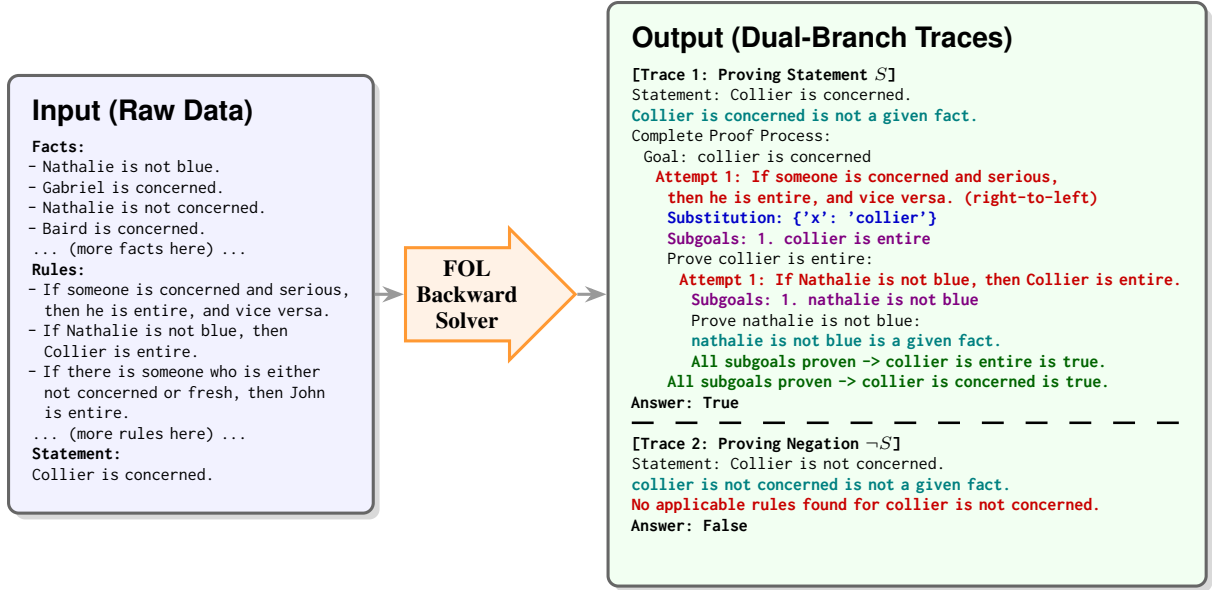


Figure 3: Visualization of the Solver-Generated Data. **Left:** The raw data. **Right:** The solver generates traces for both the statement (S) and its negation ($\neg S$). Top: A successful proof trace for S , annotated with: **Rule Selection (Red)**, **Substitution (Blue)**, **Subgoals (Purple)**, **Fact Check (Teal)**, **Verification (Green)**. Bottom: The solver correctly returns False for $\neg S$.

(we apply a stable sort), so trace generation is fully reproducible.

Execution Constraints and Optimization. Beyond prioritization, the solver enforces strict execution constraints to ensure robustness (see Algorithm 1 in Appendix B). We implement **Memoization** (using a global cache $\mathcal{M}$) to avoid redundant computations for visited states. Crucially, **Cycle Detection** is performed by maintaining a recursion stack $\mathcal{S}$; if the current goal G appears in $\mathcal{S}$, the branch is terminated to prevent infinite loops. Finally, logical **Short-Circuiting** is applied, where an AND-condition fails immediately upon the first unproven sub-goal, and an OR-condition succeeds as soon as a valid rule path is found.

3.3.2 Natural Language Trace Linearization

To train the student model, we serialize the hierarchical execution log $\mathcal{T}$ into a structured natural language format C_{bw} using a **Full Trace** strategy. Distinct from optimal-path-only methods, **our linearization preserves the entire search history, including all failed attempts and dead ends explicitly** (see Appendix E for a comparison). This design is pivotal: as verified in our Ablation Study (Section 4.4), exposing the model to such “trial-and-error” processes is critical for robust reasoning. To facilitate the infusion of backward chaining mechanics, we highlight five structural components in

the generated trace:

1. **Rule Selection (Red):** The trace logs the specific implication rule retrieved from the knowledge base (e.g., “Attempt 1: If someone is concerned...”), teaching the model to identify relevant premises for the current goal.
2. **Variable Substitution (Blue):** We explicitly state the unification binding σ (e.g., “Substitution: {'x': 'collier'}”), forcing the model to understand how abstract quantifiers apply to specific entities.
3. **Subgoals Decomposition (Purple):** The trace breaks down the antecedent of the selected rule into a list of required sub-proofs (e.g., “Subgoals: 1. collier is entire”), demonstrating the recursive nature of the problem.
4. **Fact Check (Teal):** When a goal matches the known facts, the trace explicitly confirms it (e.g., “nathalie is not blue is a given fact”), grounding the reasoning in the provided context.
5. **Logical Verification (Green):** The trace synthesizes the results of subgoals to validate the parent goal (e.g., “All subgoals proven $\rightarrow$ collier is concerned is true”), ensuring logical consistency.

For every raw input entry, our framework executes the solver on both the positive statement (S_n) and its logical negation ($\neg S_n$), thereby generating **two distinct instruction-tuning samples** from a single context. This mechanism effectively doubles the dataset scale.

3.4 Fine-tuning Open-source LLMs

The final phase is to infuse this symbolic reasoning capability into a student LLM, $\mathcal{M}_{student}$. We construct an instruction-tuning dataset $\mathcal{D}_{train} = \{(X, C_{bw})\}_{i=1}^N$ consisting of the natural language input, the solver-generated backward trace, and the final answer. By training on these high-quality, dual-verified traces, $\mathcal{M}_{student}$ learns to simulate the backward solver’s operation, achieving high-accuracy logical reasoning without accessing external tools during inference.

4 Experiments

4.1 Experimental Setup

Datasets. We select three benchmarks representing a gradient of logical complexity. We begin with **ProntoQA** (Saparov and He, 2023) (5-hop subset) to evaluate standard deduction under the Closed-World Assumption (CWA). Next, we employ **ProofWriter** (Tafjord et al., 2021) (Depth-5) to assess reasoning under the Open-World Assumption (OWA), which involves handling negation and unknown facts. Finally, we include **LogicNLI** (Tian et al., 2021) to test the generalization to complex logical connectors, such as bi-conditionals. Detailed statistics and training settings are provided in Appendix F.

Baselines. We evaluate SOLID against comprehensive state-of-the-art methods categorized into three paradigms. (1) **Prompt-based Methods:** Using GPT-4o as the backbone, we compare with *Standard CoT* (Wei et al., 2022), *SymbCoT* (Xu et al., 2024), and *LAMBADA* (Kazemi et al., 2023), which decomposes reasoning into modular backward-chaining sub-tasks. We also include *Aristotle* (Xu et al., 2025), a recent SOTA method utilizing a decompose-search-resolve architecture. (2) **Solver-based Methods:** We compare against *Logic-LM* (Pan et al., 2023) and *VERUS-LM* (Callewaert et al., 2026). Additionally, we evaluate *SymBa* (Lee and Hwang, 2025), which integrates an SLD Resolution solver to control the deduction process, invoking LLMs only for necessary information retrieval. (3) **Fine-tuning Methods:** We

select *LoGiPT* (Feng et al., 2024), which fine-tunes Llama models on solver-derived instruction data. Additionally, we report *SOLID (Forward)*, a variant of our model trained on forward-chaining traces, to explicitly verify the superiority of our backward-chaining strategy. Detailed configurations for all baselines are provided in Appendix G.

Implementation Details. We use **Llama-2-7B** and **Qwen-2.5-7B-Instruct** as our base models. Fine-tuning is performed using LoRA (Low-Rank Adaptation) with $r = 64$ and $\alpha = 128$. We train for 3 epochs with a learning rate of 5×10^{-5} . All experiments are conducted on a single Nvidia A100-40G GPU. The teacher model $\mathcal{M}_{teacher}$ used for semantic translation is DeepSeek-V3 throughout. To ensure reproducibility, all fine-tuning runs use a fixed random seed (seed= 42), and inference uses greedy decoding. Table 1 reports the seed-42 checkpoint; across three seeds the standard deviation is at most 2.4 points, so the margins reported below substantially exceed seed-induced variation (Appendix I).

4.2 Main Results

Table 1 summarizes the performance of SOLID compared to state-of-the-art baselines across three logical reasoning benchmarks. The results demonstrate that our framework achieves superior accuracy, consistently outperforming both GPT-4o-based methods and existing fine-tuning approaches.

Overall Performance. Our best model, **SOLID (Qwen-2.5-7B)**, establishes a new state-of-the-art with an impressive average accuracy of **97.9%**. It achieves saturation on ProntoQA (100.0%) and demonstrates remarkable robustness on the complex LogicNLI dataset (96.0%). Even when using the **Llama-2-7B** backbone, SOLID achieves an average accuracy of **94.8%**, surpassing the strongest baseline (Aristotle with GPT-4o) by a substantial margin of +9.3%.

Comparison with Prompt-based and Solver-based Methods. A key finding is that SOLID, despite utilizing significantly smaller open-source models (7B parameters), consistently outperforms methods relying on the powerful proprietary **GPT-4o**.

- On the challenging **LogicNLI** benchmark, which requires handling complex logical connectors (e.g., “if and only if”), prompt-based methods struggle (CoT: 61.0%, Aristotle:

Method	Backbone	ProntoQA	ProofWriter	LogicNLI	Avg
<i>Prompt-based Methods</i>					
Chain-of-Thought (CoT)	GPT-4o	98.0	77.2	61.0	78.7
SymbCoT (Xu et al., 2024)	GPT-4o	<u>99.4</u>	82.5	58.7	80.2
LAMBADA (Kazemi et al., 2023)	GPT-4 Turbo	96.0	69.7	–	–
Aristotle (Xu et al., 2025)	GPT-4o	99.0	88.7	<u>68.7</u>	<u>85.5</u>
<i>Solver-based Methods</i>					
Logic-LM (Pan et al., 2023)	GPT-4o	83.2	73.0	53.2	69.8
VERUS-LM (Callewaert et al., 2026)	GPT-4o	95.8	<u>93.8</u>	–	–
SymBa (Lee and Hwang, 2025)	GPT-4 Turbo	96.3	79.8	–	–
<i>Fine-tuning Methods</i>					
LoGiPT (Feng et al., 2024)	CodeLlama-13B	95.6	89.5	–	–
SOLID (Forward)	Llama-2-7B	99.8	70.7	31.7	67.4
SOLID (Ours)	Llama-2-7B	99.8	92.7	92.0	94.8
SOLID (Ours)	Qwen-2.5-7B	100.0	97.8	96.0	97.9
<i>Improvement vs. Best Baseline</i>	–	+0.6	+4.0	+27.3	+12.4

Table 1: Accuracy (%) comparison on ProntoQA, ProofWriter, and LogicNLI. **Bold** indicates the best overall performance; underline indicates the best baseline performance (excluding SOLID). The last row shows the absolute improvement of our best model over the strongest baseline. Results for LAMBADA and SymBa are sourced from Lee and Hwang (2025) using GPT-4 Turbo. Entries marked “–” indicate that the original work did not report results on the corresponding benchmark.

68.7%). In contrast, SOLID (Llama-2-7B) achieves **92.0%** accuracy. This indicates that fine-tuning on high-quality, solver-generated traces infuses the model with complex logical rules more effectively than in-context learning.

- Compared to **Solver-based methods** like Logic-LM (Avg 69.8%) and VERUS-LM, SOLID avoids the brittleness of neuro-symbolic translation errors. While VERUS-LM performs well on ProofWriter (93.8%), it is unable to handle LogicNLI. This is because its strict constraint inconsistency detection mechanism lacks the flexibility to resolve the self-contradictory scenarios inherent in this diagnostic benchmark.

Comparison with Fine-tuning Baselines. Compared to LoGiPT (Feng et al., 2024), which utilizes CodeLlama-13B, our SOLID (Llama-2-7B) achieves higher accuracy on both ProntoQA (+4.2%) and ProofWriter (+3.2%) despite using a smaller model backbone. To remove the backbone as a confound entirely, we additionally train SOLID on LoGiPT’s exact backbone, CodeLlama-13B; with the backbone held identical, supervision data becomes the only variable and the margins widen further (Appendix H). Both readings point the same way: our *Backward Solver* generates higher-quality supervision data than the traces used in LoGiPT.

Impact of Reasoning Direction. To explicitly verify the effectiveness of our backward-chaining strategy, we compare SOLID (Backward) against a variant trained with Forward Solver traces. *Both variants use the same Full-Trace linearization* (retaining failed attempts and dead ends) and the same teacher model, isolating reasoning *direction* as the only factor of variation. As shown in Table 1, the backward-chaining model consistently outperforms the forward-chaining variant (Avg **94.8%** vs. **67.4%**).

Crucially, the performance gap correlates strongly with task complexity: while identical on ProntoQA, it widens drastically on the complex LogicNLI (**92.0% vs. 31.7%**). Forward chaining operates in a data-driven, bottom-up manner, exhaustively deriving all possible facts regardless of their relevance to the query. In complex tasks requiring deep reasoning, this process generates a plethora of irrelevant intermediate facts. These redundant thoughts clutter the context and dilute the model’s attention, causing confusion and logical incoherence as the sequence length increases. To quantify this, we measured the average number of intermediate conclusions per problem under each strategy; the results are summarized in Table 2.

Forward chaining produces **7.3×** more intermediate conclusions than backward on ProntoQA, and **2.0×** more on the harder LogicNLI. The ratio confirms that forward chaining produces substantially more unfocused content, with the absolute volume

Benchmark	Forward	Backward	Ratio
ProntoQA	9.21	1.27	7.3×
LogicNLI	16.69	8.44	2.0×

Table 2: Average number of intermediate conclusions per problem under forward and backward chaining. The ratio (Forward/Backward) quantifies how much unfocused content forward chaining produces; the gap widens on the harder benchmark.

of spurious derivations growing on tasks where many irrelevant facts are derivable. In contrast, our goal-oriented backward reasoning naturally prunes irrelevant branches, maintaining a minimal and focused reasoning trace. This ensures that the model remains aligned with the query even during deep multi-hop inference, effectively mitigating the chaos caused by search space explosion.

4.3 Validity of the Generated Traces

Answer accuracy alone cannot distinguish a model that has internalized the solver’s procedure from one that guesses an answer and then produces a plausible-looking justification. Because our traces follow a rigid, machine-generated grammar, student outputs are mechanically parseable, and we audit them under three increasingly strict criteria, all computed over correctly-answered items. **Step-level validity** scores individual assertions against the natural-language context, with partial credit. **Branch-level validity** is all-or-nothing per reasoning branch: a single flawed assertion invalidates the whole branch. **Fully sound proofs** additionally replays each branch through our solver, and accepts it only when the reasoning is logically sound as well as well-formed—every inference must be re-derivable, so a branch that leans on a conclusion it never established fails even if its narrative reads correctly. The three criteria are nested; Appendix J gives the definitions in full.

Benchmark	Step-level	Branch-level	Fully sound
ProntoQA	100.00	100.00	100.00
ProofWriter	99.42	98.10	97.64
LogicNLI	99.64	98.90	96.50

Table 3: Trace validity of SOLID (Llama-2-7B) over correctly-answered items (%). **Step-level**: legal assertions over all assertions. **Branch-level**: issue-free branches over all branches. **Fully sound**: branches that are issue-free *and* survive solver replay, over all branches.

As Table 3 shows, step-level validity is 99.4–

Configuration	ProntoQA	ProofWriter	LogicNLI
Full SOLID	99.8	92.7	92.0
w/o Dual-Ver.	100.0 (+0.2)	92.0 (-0.7)	87.0 (-5.0)
w/o Full Traces	99.8 (0.0)	79.9 (-12.8)	75.0 (-17.0)
w/o S.F.	99.8 (0.0)	92.9 (+0.2)	90.3 (-1.7)

Table 4: Ablation results (accuracy %). Numbers in parentheses show the absolute change Δ vs. full SOLID. “w/o S.F.” = remove Simplicity-First strategy.

100% and branch-level validity 98.1–100%, so correct answers are overwhelmingly accompanied by derivations that survive an assertion-by-assertion check; the student is not producing post-hoc rationalizations for guessed answers. The residual gap to fully sound proofs, widest on LogicNLI (98.90 $\rightarrow$ 96.50), consists of branches that pass every assertion-level check yet contain an inference the solver cannot re-derive.

4.4 Ablation Study

To verify the contribution of our key components, we conduct ablation studies using the **Llama-2-7B** backbone, investigating three critical aspects: *Dual-Verification Mechanism*, *Full Search Traces*, and *Simplicity-First Strategy*. Table 4 summarizes the results.

Impact of Dual-Verification Mechanism. The Dual-Verification module acts as a strict filter, creating a critical trade-off between **data quantity** and **data quality**. Removing this module (“w/o Dual-Ver.”) retains more training samples but inevitably introduces noisy, invalid translations. The impact diverges significantly based on task complexity:

- On ProntoQA, performance remains saturated (100.0%). For simpler deductive tasks, the model benefits primarily from the increased **volume of data**, making it robust to minor noise; the gain from observing diverse reasoning templates outweighs the harm caused by occasional error entries.
- Conversely, on LogicNLI, performance drops significantly (−5.0%). This dataset involves intricate semantics where the connection between the natural language context and the reasoning proof is highly specific. The noise introduced by unfiltered data disrupts this delicate alignment, leading the model to hallucinate valid proofs for mismatched premises.

Necessity of Full Search Traces. Removing failed attempts and dead ends (“w/o Full Traces”) leads to a catastrophic performance collapse on LogicNLI (−17.0%) and ProofWriter (−12.8%). This highlights that for complex problem spaces, learning “*what not to do*”—specifically backtracking and error recognition—is just as critical as learning the correct path. By exposing the model to the solver’s trial-and-error process, SOLID effectively teaches the model to navigate and prune the search space, a capability that is lost when training solely on sanitized optimal solutions.

Analysis of Simplicity-First Strategy. The Simplicity-First strategy regulates the trade-off between **search efficiency** and the **richness of the reasoning process**. Removing this constraint (“w/o S.F”) forces the solver to follow the default retrieval order, leading to divergent outcomes:

- On ProntoQA, performance remains unchanged ($\Delta = 0.0$). Since the candidate rule set for each step is typically singular in this simple dataset, the heuristic ranking is indistinguishable from the default order.
- On ProofWriter, we observe a counter-intuitive improvement ($\Delta = +0.2\%$) without S.F. The default order produces less optimized, longer inference paths; the extended traces provide richer supervision signals, acting as a form of “process augmentation” that enhances model robustness.
- Conversely, on LogicNLI, performance declines ($\Delta = -1.7\%$). In this highly complex domain, the unoptimized default order leads to excessive path lengths and a proliferation of useless branches, manifesting as distracting noise rather than informative context.

5 Related Work

Chain-of-Thought and Backward Reasoning. While Chain-of-Thought (CoT) (Wei et al., 2022) enhances reasoning, it remains prone to hallucination and lacks goal-orientation. Advanced methods like Tree-of-Thoughts (Yao et al., 2023) and LAMBADA (Kazemi et al., 2023) introduce structured backward search to mitigate this, but they rely heavily on complex prompting strategies that incur high inference costs. In contrast, SOLID addresses these limitations by fine-tuning to infuse the model with rigorous backward chaining, effectively replacing

expensive runtime search with learned algorithmic stability.

Neuro-Symbolic Reasoning. Neuro-Symbolic AI typically combines neural parsers with external solvers to ensure robustness (Pan et al., 2023; Ye et al., 2023). Recent systems like SymBa (Lee and Hwang, 2025) deepen this by using solvers to explicitly control the reasoning process. However, these *solver-dependent* approaches introduce significant runtime latency and deployment complexity. SOLID proposes a philosophy of *Infusion*: we utilize the symbolic solver solely during training to synthesize supervision, allowing the LLM to absorb deductive logic directly into its parameters for efficient, solver-free inference.

6 Conclusion

We presented **SOLID**, a novel framework that infuses a Horn-like fragment of First-Order Logic into LLMs by fine-tuning on solver-generated backward reasoning traces. By integrating a goal-oriented Backward Solver with a Dual-Verification Mechanism, SOLID synthesizes label-consistent training data whose reasoning traces are solver-verified and logically sound. Experiments on ProntoQA, LogicNLI, and ProofWriter demonstrate that our approach achieves state-of-the-art performance, effectively handling complex reasoning without relying on external solvers during inference. Future work will extend this paradigm to broader mathematical reasoning domains.

Limitations

Formal expressiveness. Our Backward Solver is sound and complete only over the formula fragment described in Appendix A—Horn-like clauses under finite-domain Herbrand grounding with a Domain Closure Assumption on the parsed constants—so inputs requiring function symbols, infinite domains, equality reasoning, or higher-order quantification fall outside this fragment and cannot be used as training supervision under our pipeline. The restriction is deliberate rather than incidental: full first-order logic is undecidable, whereas reliable trace synthesis requires a procedure that is sound, complete and terminating. Extending SOLID to a richer fragment is therefore a matter of extending the solver, not of redesigning the framework.

Dependence on labelled data. SOLID operates in the standard supervised fine-tuning setting. Dual-

Verification consults each sample’s gold provability label to decide whether a translation is retained, so the pipeline converts existing labels into process-level supervision rather than removing the need for them. This requirement is shared with the fine-tuning baselines we compare against, but it does bound the settings in which SOLID applies. Extending the filter to unlabelled corpora—for instance by replacing the gold label with a consensus over independently generated translations—remains future work.

Evaluation scope. Our empirical evaluation is confined to three English-language, synthetically constructed logical reasoning benchmarks (Pron-toQA, ProofWriter, LogicNLI), whose controlled vocabularies and templated phrasings isolate deductive skill but do not reflect the full complexity of naturally occurring reasoning text; generalization to multilingual settings, real-world legal or mathematical proofs, and noisier domains is therefore not established by our current experiments.

Deployment without a solver. Removing the external solver at inference is what makes the student efficient, but it also means a trace is no longer correct by construction: the model emits fluent, well-formed derivations whether or not every step is valid. In high-stakes settings such as legal analysis, education, or mathematical proof, a confidently presented yet subtly invalid derivation can mislead more than an outright refusal, and fluency may be read as a signal of reliability that our training procedure does not license. We therefore do not advocate deploying solver-free reasoning models in such domains without a verification layer. Because our traces follow a rigid, machine-parseable grammar, the same backward solver used to synthesize supervision can be run as an independent proof checker over the student’s output, restoring an explicit correctness guarantee where one is required. We regard this solver-in-the-loop configuration, rather than unguarded solver-free inference, as the appropriate mode for sensitive applications.

Acknowledgments

We are grateful to the anonymous reviewers for their comments and suggestions. This work has been supported by the National Natural Science Foundation of China (NSFC) under grant No. 62572461.

References

- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. [Language models are few-shot learners](#). In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.
- Benjamin Callewaert, Simon Vandeveld, and Joost Vennekens. 2026. [VERUS-LM: a versatile framework for combining LLMs with symbolic reasoning](#). In *Proceedings 41st International Conference on Logic Programming (ICLP 2025)*, volume 439 of *Electronic Proceedings in Theoretical Computer Science*, pages 47–62. Open Publishing Association.
- Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. [Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks](#). *Transactions on Machine Learning Research*.
- Fengxiang Cheng, Haoxuan Li, Fenrong Liu, Robert van Rooij, Kun Zhang, and Zhouchen Lin. 2025. [Empowering LLMs with logical reasoning: A comprehensive survey](#). In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pages 10400–10408. International Joint Conferences on Artificial Intelligence Organization. Survey Track.
- 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*, pages 4026–4042, Mexico City, Mexico. Association for Computational Linguistics.
- Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. [Survey of hallucination in natural language generation](#). *ACM Computing Surveys*, 55(12):1–38.
- Mehran Kazemi, Najoung Kim, Deepti Bhatia, Xin Xu, and Deepak Ramachandran. 2023. [LAMBADA: Backward chaining for automated reasoning in natural language](#). In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6547–6568, Toronto, Canada. Association for Computational Linguistics.
- Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. [Large Language Models are Zero-Shot Reasoners](#). In *Advances in Neural Information Processing Systems*, volume 35, pages 22199–22213. Curran Associates, Inc.

- Jinu Lee and Wonseok Hwang. 2025. [SymBa: Symbolic backward chaining for structured natural language reasoning](#). In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 2468–2484, Albuquerque, New Mexico. Association for Computational Linguistics.
- Yao-Yang Liu, Zhen Zheng, Feng Zhang, Jin-Cheng Feng, Yi-Yang Fu, Ji-Dong Zhai, Bing-Sheng He, Xiao Zhang, and Xiao-Yong Du. 2026. [A comprehensive taxonomy of prompt engineering techniques for large language models](#). *Frontiers of Computer Science*, 20(3):2003601.
- 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.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. [Training language models to follow instructions with human feedback](#). In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc.
- Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. 2023. [Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning](#). In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3806–3824, Singapore. Association for Computational Linguistics.
- Abulhair Saparov and He He. 2023. [Language models are greedy reasoners: A systematic formal analysis of chain-of-thought](#). In *The Eleventh International Conference on Learning Representations*.
- Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. 2021. [ProofWriter: Generating implications, proofs, and abductive statements over natural language](#). In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3621–3634, Online. Association for Computational Linguistics.
- Jidong Tian, Yitian Li, Wenqing Chen, Liqiang Xiao, Hao He, and Yaohui Jin. 2021. [Diagnosing the first-order logical reasoning ability through LogicNLI](#). In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3738–3747, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.
- 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 *Advances in Neural Information Processing Systems*, volume 36, pages 74952–74965. Curran Associates, Inc.
- Chengyu Wang, Taolin Zhang, Richang Hong, and Jun Huang. 2026. [A short survey on small reasoning models: training, inference, applications, and research directions](#). *Frontiers of Computer Science*, 20(11):2011366.
- 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*, volume 35, pages 24824–24837. Curran Associates, Inc.
- 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 (Volume 1: Long Papers)*, pages 3052–3075, Vienna, Austria. Association for Computational Linguistics.
- 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 (Volume 1: Long Papers)*, pages 13326–13365, Bangkok, Thailand. Association for Computational Linguistics.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. [Tree of thoughts: Deliberate problem solving with large language models](#). In *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822. Curran Associates, Inc.
- 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*, volume 36, pages 45548–45580. Curran Associates, Inc.

A Symbolic Logic Formats

As mentioned in the main text (Section 3.1), our Semantic Parser translates natural language into three standard First-Order Logic formats. And quantifiers bind tightly: $\forall x. \varphi$ is parsed as $\forall x. (\varphi)$ where φ extends as far right as syntax permits. All inline rules are explicitly bracketed to avoid scope ambiguity.

- **Universal Rules:** The quantifier $\forall x$ scopes over the *entire* implication, resolving into a universal form in two distinct ways depending on the consequent.

Direct Formulation: When both the antecedent and the consequent contain x , the rule is expressed as:

$$\forall x. (A(x) \wedge B(x) \rightarrow C(x))$$

This general form is handled *directly* by the solver.

Existential Conversion. When the natural-language premise has the original form $(\exists x. A(x)) \rightarrow C(c)$ —i.e., an existential antecedent paired with a ground (constant-only) consequent—the side condition is satisfied. The normalization stage applies Eq. (1) once to rewrite the rule into the equivalent universal form:

$$(\exists x. A(x)) \rightarrow C(c) \equiv \forall x. (A(x) \rightarrow C(c))$$

Although both cases ultimately evaluate as universally quantified rules, their semantic origins and application of Eq. (1) make them operationally distinct.

- **Quantified Antecedents:** Rules where a universally quantified condition over the domain constitutes the antecedent of an implication whose consequent is a ground atom. Because the antecedent is itself a universally quantified statement, Eq. (1) does not apply. The solver verifies this antecedent by checking whether the condition holds for every instantiated constant in the known domain.

$$(\forall x. (A(x) \vee \neg B(x))) \rightarrow C(c)$$

- **Ground Implications:** Direct implications between ground atoms with no quantifiers.

$$A(c_1) \rightarrow B(c_2)$$

Handling Quantified Antecedents via Herbrand Grounding. A pure SLD-resolution procedure cannot directly verify a premise of the form $(\forall x. \varphi(x)) \rightarrow \psi$, because deciding whether $\forall x. \varphi(x)$ holds requires checking φ for *every* domain element, not unifying against the head of a clause. Algorithm 1 (Appendix B) omits this

preprocessing step for brevity; in our actual implementation, the `load_problem` routine performs **finite-domain Herbrand grounding** before the backward search begins. Concretely, let $D = \{c_1, \dots, c_n\}$ be the finite set of constants enumerated in the parsed problem (constants field). For each rule with a universally quantified antecedent, we rewrite

$$(\forall x. \varphi(x)) \rightarrow \psi \implies \left(\bigwedge_{c \in D} \varphi(c) \right) \rightarrow \psi.$$

Under the Domain Closure Assumption (the parsed constants list is exhaustive for the problem), this rewriting is sound and complete. After grounding, every clause becomes Horn-like over ground atoms, falling within the completeness range of SLD-resolution.

B Backward Chaining Algorithm

Algorithm 1 details the Backward Chaining solver from Section 3.3. It uses a Simplicity-First rule ordering, memoization, cycle detection, and short-circuit evaluation to prevent infinite recursion and redundant search.

Notations. Let the knowledge base be $\mathcal{K} = (\mathcal{F}, \mathcal{R})$, comprising facts $\mathcal{F}$ and rules $\mathcal{R}$. For a rule $r \in \mathcal{R}$, $\text{Cons}(r)$ and $\text{Ant}(r)$ denote its consequent (head) and antecedent (subgoals), respectively. The function $\text{Unify}(A, B)$ returns a logical substitution σ mapping A to B (or Fail if impossible), which $\text{Apply}(\sigma, S)$ applies to subgoals S . Finally, $\text{Score}(r)$ computes a rule’s complexity to prioritize structurally simpler rules.

C Filtered-Sample Case Studies

To make the role of Dual-Verification concrete, we examine the samples that fail to pass the filter. They split cleanly into two categories. The first reveals a genuine teacher-LLM translation error; the second reveals *label noise in the source dataset*, where the solver’s rejection acts as an implicit denoising mechanism.

Category (i): Teacher-LLM translation errors. A representative failure on LogicNLI starts from the natural-language premise:

“If there is nobody who is not not nice, then Matthew is not long.”

The teacher LLM produced

$$\forall x. (\text{Nice}(x) \rightarrow \neg \text{Long}(\text{Matthew})),$$

Algorithm 1 Backward Chaining (Simplicity-First)

Require: Goal G , Knowledge Base $\mathcal{K} = (\mathcal{F}, \mathcal{R})$,
Recursion stack $\mathcal{S}$, Global memo $\mathcal{M}$

Ensure: Boolean Validity (*True/False*)

```
1: procedure SOLVE( $G, \mathcal{S}$ )
2:   if  $G \in \mathcal{F}$  or  $\mathcal{M}[G] = \text{True}$  then
3:     return True  $\triangleright$  Base fact or memo hit
4:   if  $G \in \mathcal{S}$  or  $\mathcal{M}[G] = \text{False}$  then
5:     return False  $\triangleright$  Cycle detected or
      memo fail
6:    $\Omega_G \leftarrow \{r \in \mathcal{R} \mid \text{Unify}(\text{Cons}(r), G) \neq$ 
      Fail  $\}$ 
7:   if  $\Omega_G = \emptyset$  then
8:     return False  $\triangleright$  No applicable rules
9:    $\Omega_{\text{sorted}} \leftarrow \text{Sort}(\Omega_G, \text{Score}, \text{descending})$ 
10:  for each rule  $r \in \Omega_{\text{sorted}}$  do
11:     $\sigma \leftarrow \text{Unify}(\text{Cons}(r), G)$ 
12:     $\text{Subgoals} \leftarrow \text{Apply}(\sigma, \text{Ant}(r))$ 
13:     $\text{Success} \leftarrow \text{True}$ 
14:    for each  $g' \in \text{Subgoals}$  do
15:       $ok \leftarrow \text{SOLVE}(g', \mathcal{S} \cup \{G\})$ 
16:      if  $ok = \text{False}$  then
17:         $\text{Success} \leftarrow \text{False}$ 
18:      break  $\triangleright$  Short-circuit AND
19:    if  $\text{Success}$  then
20:       $\mathcal{M}[G] \leftarrow \text{True}$ 
21:      return True  $\triangleright$  Short-circuit OR
22:     $\mathcal{M}[G] \leftarrow \text{False}$ 
23:  return False
```

which both drops an outer negation (“nobody who is not not nice” is equivalent to “nobody who is nice”) and mis-scopes the quantifier (the antecedent should be a top-level $\forall$ -statement, not an internally quantified rule body). The semantically faithful translation is

$$(\forall x. \neg \text{Nice}(x)) \rightarrow \neg \text{Long}(\text{Matthew}).$$

Running the solver under both branches reveals the discrepancy between the gold provability profile and the buggy translation, so the sample is correctly discarded.

Category (ii): Dataset label noise. Consider a ProntoQA sample with facts $\{\text{Zumpus}(\text{Sally})\}$, rule $\forall x. (\text{Zumpus}(x) \rightarrow \neg \text{Feisty}(x))$, and query statement $\text{Feisty}(\text{Sally})$ *labelled* as Entailment. Any sound solver derives $\neg \text{Feisty}(\text{Sally})$ from these premises, and *cannot* derive $\text{Feisty}(\text{Sally})$; the correct label is therefore Contradiction, not Entailment. Dual-Verification rejects this sample be-

cause the solver-derived four-state profile does not match the (incorrect) gold label. As a side benefit, this filtering quietly removes mislabelled examples from the supervision corpus.

Soundness of the filter. Within the formula fragment supported by our solver (Appendix A), the backward chaining procedure is sound and complete. It therefore never discards a sample whose translation is semantically faithful *and* whose gold label is correct: such samples always satisfy the joint provability constraint by construction. All discarded samples fall into one of the two categories above.

D Pipeline Failure Handling and Computational Cost

What happens when a sample fails Dual-Verification? A translated sample $\Phi = (\mathcal{F}_n, \mathcal{R}_n, S_n)$ fails Dual-Verification when the joint solver outcome on S_n and $\neg S_n$ does *not* match the four-state profile dictated by the gold label y (Figure 1). **Failed samples are simply discarded**—they are never repaired, never retried, and never propagate to the fine-tuning corpus. We deliberately favour discarding over repair to avoid introducing translation noise into the supervision signal: a sample whose symbolic form contradicts its label is, by construction, untrustworthy. Discard rates remain modest in practice—ProntoQA **1.20%**, ProofWriter **3.96%**, LogicNLI **14.75%**—leaving an ample volume of high-quality training data after filtering (cf. Table 5 for the per-dataset retention figures).

Computational cost of dual-branch verification.

A common concern about dual-branch execution is that running the solver twice (once for S_n , once for $\neg S_n$) doubles the cost. We emphasize two facts that make this overhead negligible in practice:

- **One-time, offline.** Dual-Verification is part of *training-data synthesis*, executed exactly once per dataset before any fine-tuning. It does *not* run at inference time.
- **Wall-clock cost is small.** With a 40-process parallel solver pool on commodity CPU hardware, the total wall-clock time for the dual-branch pass is **2.91 s** on ProntoQA, **7.43 s** on ProofWriter, and **21.84 s** on LogicNLI. The cost is dwarfed by both the teacher-LLM translation cost and the subsequent LoRA fine-tuning cost.

Inference cost is unaffected: at test time the student LLM produces backward traces directly without invoking any solver, so deployment latency for SOLID is identical to that of any plain fine-tuned 7B model.

E Detailed Comparison of Linearization Strategies

In this section, we articulate the algorithmic differences between our **Full Trace (SOLID)** strategy and the traditional **Optimal-Path-Only** baseline. The core distinction lies in the termination condition and the selection criteria of the reasoning path.

E.1 Strategy Definitions

SOLID: Early Stopping with Full History. Our approach aims to simulate the genuine cognitive process of a solver facing an unseen problem. We employ an **Early Stopping** mechanism: the solver executes the search and terminates immediately upon finding the *first* valid derivation path. Crucially, the linearization preserves the chronological sequence of all operations up to that point, including failed attempts (dead ends) and circular branches encountered before the successful path. This exposes the student model to the necessary “trial-and-error” logic required for inference-time search.

Baseline: Optimal-Path Selection. The standard baseline focuses on data quality over process fidelity. It performs an exhaustive search to identify *all* possible valid proof trees for a given query. From this set, it selects the single “best” path based on a heuristic scoring mechanism and discards all failed attempts and suboptimal paths.

E.2 Heuristic Scoring Mechanism

To define the “Optimal Path” while strictly avoiding redundant reasoning and unnecessary derivation steps, the baseline employs a simplified scoring function $S(P)$. Instead of using multi-dimensional structural heuristics that often introduce mutual interference, this mechanism directly penalizes the total number of inference steps (i.e., non-leaf internal nodes) within the proof tree.

$$S(P) = \text{Base} - \lambda \cdot N_{\text{inf}}(P) \quad (3)$$

Where the components and metrics are formally defined as follows:

- **Base Score (Base):** A constant set to 100 points, serving as a baseline offset to ensure that the final score remains positive for downstream ranking and selection.
- **Inference Step Penalty ($N_{\text{inf}}(P)$):** Defined as the total count of internal reasoning nodes (non-leaf nodes) in the proof tree P , which strictly represents the number of rule applications executed during the derivation:

$$N_{\text{inf}}(P) = |\{n \in \text{nodes}(P) \mid \neg \text{is_leaf}(n)\}| \quad (4)$$

The parameter λ (e.g., $\lambda = 5$) represents the penalty weight per inference step. This penalty universally suppresses both overly deep recursive chains and overly wide branch expansions by directly targeting the overall structural complexity of the proof.

The baseline selects the optimized path $P^* = \arg \max_P S(P)$ as the definitive training target for the student model.

E.3 Visual Comparison

Figure 4 illustrates the difference on a sample query. The SOLID strategy (Left) reveals the “struggle” (Attempts 1 & 2 failed) before the “success” (Attempt 3), mimicking inference. The Baseline (Right) presents only the sanitized, high-score path (Attempt 3), hiding the search complexity.

F Dataset Details

We provide detailed statistics, construction protocols, and dual-verification pass rates for the three benchmarks used in our experiments.

ProntoQA. This synthetic dataset uses fictional ontology (e.g., “wumpus”) to isolate reasoning skills from commonsense priors under the Closed-World Assumption (CWA).

- **Training Set:** We merged all available training subsets and filtered out syntax-invalid entries, yielding **31,498** samples.
- **Test Set:** We utilized the standard **5-hop** subset (**500** samples) to evaluate deep deductive capabilities.

ProofWriter. Designed for multi-hop reasoning under the Open-World Assumption (OWA), this dataset requires the model to distinguish between *True*, *False*, and *Unknown*.

(a) SOLID: Early Stopping (Ours)

Stops at 1st success; retains preceding failures.

```

Goal: Garth is wild

Attempt 1: Someone being both wild and alive
is equivalent to being obedient. (right-to-left)
Substitution: {'x': 'Garth'}
Subgoals: 1. Garth is obedient
...
-> Failed: Cycle detected / cannot prove.

Attempt 2: If someone is not obedient and
angry, then he is both wild and not alive...
Substitution: {'x': 'Garth'}
...
-> Failed: cannot prove Garth is not obedient

Attempt 3: Someone who is either not angry
or electronic is always wild.
Substitution: {'x': 'Garth'}
Subgoals: 1. Garth is not angry,
2. Garth is electronic
Prove Garth is not angry:
... (Reasoning steps) ...
Premise satisfied (OR) ->
Garth is not angry is true.
Premise satisfied (OR) ->
Garth is wild is true.

Answer: True

```

(b) Baseline: Optimal Selection

Exhaustive search + Heuristic Filtering.

```

Goal: Garth is wild

Attempt 1: Someone who is either not angry
or electronic is always wild.
Substitution: {'x': 'Garth'}
Subgoals: 1. Garth is not angry,
2. Garth is electronic

Prove Garth is not angry:
Attempt 1: If someone is not wild or not
electronic, then he is not angry.
Substitution: {'x': 'Garth'}
Subgoals: 1. Garth is not wild...

Prove Garth is not wild:
Garth is not wild is a given fact.

Premise satisfied (OR) ->
Garth is not angry is true.

Premise satisfied (OR) ->
Garth is wild is true.

Answer: True

```

Selection Metric: The path above was selected because it maximizes the heuristic score (High Fact Ratio, Low Depth). Dead ends (Attempts 1 & 2) are removed, depriving the model of negative supervision.

Figure 4: **Impact of Linearization Strategies.** **Left:** SOLID employs *Early Stopping*, capturing the trial-and-error process (red text) leading up to the first valid proof. **Right:** The Optimal-Path method selects the highest-scoring proof, presenting a sanitized view that lacks search logic training.

- **Training Set:** We combined examples across all depths (0 to 5), resulting in a comprehensive set of **79,582** examples.
- **Test Set:** Evaluation is performed on the official **OWA Depth-5** test set (**600** samples) to assess robustness in handling unknown facts and negation.

LogicNLI. This diagnostic dataset focuses on intricate logical connectors, including bi-conditionals (“if and only if”) and exclusive disjunctions.

- **Training Set:** We processed the original training data to construct a high-quality corpus of **27,280** samples.
- **Test Set:** Following the standard evaluation protocol, we used a strict split of **300** test cases featuring complex logical structures.

Dual-Verification Pass Rates. Table 5 reports the fraction of translated samples that pass the Dual-Verification filter for each dataset. A higher pass rate indicates more reliable semantic parsing by the teacher LLM; a lower rate reflects harder translation challenges but also a stricter quality guarantee for retained samples.

	ProntoQA	ProofWriter	LogicNLI
Raw samples	31,880	82,864	32,000
Verified samples	31,498	79,582	27,280
Pass rate (%)	98.80	96.04	85.25

Table 5: Dual-Verification pass rates per dataset. Rejected samples contain translations that are syntactically valid but semantically inconsistent with the ground-truth label.

G Baseline Details

We evaluate SOLID against a comprehensive set of baselines categorized into three paradigms: prompt-based reasoning, solver-based neuro-symbolic methods, and fine-tuning approaches.

Prompt-based Methods. These methods utilize **GPT-4o/GPT-4 Turbo** as the backbone and rely on in-context learning to elicit reasoning without updating model parameters.

- **Chain-of-Thought (CoT):** The standard prompting strategy that encourages the model to generate intermediate natural language reasoning steps before deriving the final answer, using the “Let’s think step by step” prompt

structure.

- **SymbCoT:** A symbolic chain-of-thought method that prompts the LLM to translate natural language contexts into symbolic expressions first, followed by step-by-step symbolic deduction.
- **LAMBADA:** A modular backward-chaining framework that decomposes the reasoning process into four sub-modules: *Fact Check*, *Rule Selection*, *Goal Decomposition*, and *Sign Agreement*, each implemented via few-shot prompting.
- **Aristotle:** A state-of-the-art framework employing a “decompose-search-resolve” architecture that integrates symbolic logic rules to maintain logical consistency and reduce search errors during complex deduction.

Solver-based Methods. These approaches use LLMs primarily as semantic parsers, with reasoning offloaded to external deterministic solvers.

- **Logic-LM:** Translates natural language problems into symbolic programs (e.g., Python or Prolog) and offloads reasoning to a deterministic solver.
- **VERUS-LM:** Introduces a self-refinement loop that uses feedback from the reasoning engine (e.g., syntax errors or execution failures) to iteratively correct the generated logic.
- **SymBa:** Integrates a symbolic *SLD Resolution* solver with an LLM, letting the solver control the backward-chaining proof process and invoking the LLM only for specific information retrieval.

Fine-tuning Methods. These methods fine-tune open-source models on specialized logical reasoning datasets.

- **LoGiPT:** Fine-tuned on instruction data derived from the invisible reasoning processes of deductive solvers, aiming to teach the LLM to mimic strict solver execution logic.
- **SOLID (Forward):** An ablation variant of our framework trained on *forward-chaining* reasoning traces, allowing explicit comparison to verify the superiority of our goal-oriented *backward-chaining* strategy.

H Additional Student Backbones

We report two further student backbones, each answering a different question, with all other settings held fixed (LoRA $r=64$, $\alpha=128$, 3 epochs, $lr=5 \times 10^{-5}$, seed 42, single A100-40G). **Llama-3.1-8B-Instruct** asks whether SOLID’s gains transfer to more recent open backbones rather than being tied to Llama-2-7B. **CodeLlama-13B** is LoGiPT’s own backbone, included so that the fine-tuning comparison can be made with the backbone held identical. Table 6 reports both.

Student	ProntoQA	ProofW.	LogicNLI	Avg
<i>LoGiPT</i>	95.6	89.5	–	–
Llama-2-7B	99.8	92.7	92.0	94.8
Llama-3.1-8B	100.0	98.2	97.0	98.4
CodeLlama-13B	99.6	98.5	98.3	98.8

Table 6: SOLID under additional student backbones (accuracy %). *LoGiPT* (Feng et al., 2024), which itself uses CodeLlama-13B, is shown for reference. Llama-3.1-8B-Instruct improves average accuracy by **+3.6** points over Llama-2-7B. CodeLlama-13B is LoGiPT’s own backbone: holding it fixed leaves supervision data as the only variable.

A stronger student. The improvements of Llama-3.1-8B-Instruct over the Llama-2-7B configuration are consistent across all three datasets and are most pronounced on LogicNLI (+5.0) and ProofWriter (+5.5), the two harder benchmarks. This indicates that SOLID’s supervision quality is not bottlenecked by the student model: a stronger student translates the same dual-verified backward traces into stronger end-task accuracy. We therefore retain Llama-2-7B as our primary backbone in the main paper for direct comparability with LoGiPT, while reporting Llama-3.1-8B-Instruct here as evidence that the pipeline transfers cleanly to current-generation open backbones.

A controlled fine-tuning comparison. In Table 1, SOLID and LoGiPT differ in both backbone and supervision data, so the comparison—although conservative, since SOLID uses the *smaller* model—does not isolate either factor. Training SOLID on CodeLlama-13B removes the confound: with the backbone identical to LoGiPT’s, supervision data is the only remaining variable, and SOLID improves on LoGiPT by **+4.0** on ProntoQA and **+9.0** on ProofWriter (LoGiPT reports no LogicNLI result). The two readings agree.

With a smaller backbone SOLID already outperforms LoGiPT (+4.2 / +3.2), and with the backbone held fixed the margin widens rather than closes, which is what one expects if the gain originates in the supervision rather than in model capacity.

I Seed Variance

All results in the main paper use a single fixed seed (seed = 42; Section 4.1). To confirm that the reported margins are not an artifact of that choice, we re-trained SOLID (Llama-2-7B) under two further seeds, holding every other setting fixed, and re-scored all three runs under a single unified decoding stack for comparability. Table 7 reports the outcome.

Benchmark	Mean $\pm$ std (3 seeds)
ProntoQA	99.73 $\pm$ 0.12
ProofWriter	91.62 $\pm$ 2.12
LogicNLI	90.55 $\pm$ 2.41

Table 7: Accuracy (%) of SOLID (Llama-2-7B) across three random seeds. The seed-42 run is the checkpoint reported in Table 1.

Variation is negligible on the saturated ProntoQA benchmark and reaches at most 2.41 points on LogicNLI, the hardest of the three. The seed-42 checkpoint reported in Table 1 sits within one standard deviation of the mean on every benchmark. For context, the improvements SOLID reports over the strongest baseline are +27.3 on LogicNLI and +4.0 on ProofWriter, an order of magnitude larger than the seed-induced spread, so the central conclusions do not depend on the particular seed.

J Trace Validity Checker

This appendix defines the three criteria reported in Section 4.3. All are computed over correctly-answered items; a trace that cannot be parsed counts as invalid rather than being skipped.

Step-level validity (partial credit). The fraction of legal assertions over all assertions in the trace. Five kinds of assertion are counted, each checked individually against the natural-language context: rule citations, fact checks, subject-level substitution checks, the answer self-consistency check, and the non-degeneracy check.

Branch-level validity (all-or-nothing). The fraction of branches in which the step-level checks report no issue at all, over all branches. A single

flawed assertion invalidates the entire branch, so branch-level validity never exceeds step-level validity.

Fully sound proofs (with solver replay). The strictest criterion, and the only one that re-derives the reasoning rather than inspecting it. A branch is counted only if it is issue-free under the step-level checks *and* survives replay through our solver. Replay tests logical validity rather than well-formedness: every attempt must genuinely follow from the rule it cites under the binding it reports, and no step may rest on a conclusion that was never established earlier in the branch. A step that is perfectly grammatical and cites a real rule still fails if the inference does not hold. A branch therefore counts as fully sound only when both its form and its logic are without fault, so this criterion never exceeds branch-level validity.

K Out-of-Distribution Evaluation

A natural concern is whether SOLID’s fine-tuned student merely memorizes patterns specific to each training distribution, or whether it acquires a transferable backward-reasoning procedure. To probe this, we train SOLID (Qwen-2.5-7B-Instruct) exclusively on LOGICNLI traces and evaluate *zero-shot* on the PROOFWRITER Depth-5 OWA test set, without exposing the model to any ProofWriter sample during training. The two benchmarks differ along all major axes: world assumption (LogicNLI uses an explicit closed-world style with biconditionals; ProofWriter is open-world with explicit “unknown”), surface vocabulary, rule connector distribution, and chain depth.

Setting	Train	Test (ProofW.)
SOLID (in-distribution)	ProofWriter	97.8
LoGiPT (in-distribution)	ProofWriter	89.5
SOLID (OOD, ours)	LogicNLI	87.4

Table 8: Out-of-distribution transfer on ProofWriter Depth-5 OWA (accuracy %). SOLID trained only on LogicNLI nearly matches LoGiPT’s *in-distribution* performance on ProofWriter, indicating the model acquires a transferable backward reasoning procedure rather than dataset-specific surface patterns.

As Table 8 shows, the LogicNLI-trained SOLID achieves **87.4%** on ProofWriter Depth-5, falling only -10.4 points below its in-distribution counterpart and—more strikingly—falling only -2.1 points below LoGiPT, which was fine-tuned *on*

ProofWriter itself. The model has never seen a ProofWriter sample yet still nearly matches a baseline that was explicitly trained on the target distribution. We interpret this as evidence that the dual-verified backward traces transmit a reasoning skill that generalizes across logical-reasoning benchmarks, rather than a dataset-specific shortcut. The remaining gap is plausibly attributable to differences in surface phrasing (which the student adapts to via in-distribution training in the standard setting).