Practical AI-Grading with Formal Verification: A Case Study of Russell-Bernays Axiom System

Anonymous ACL submission

Abstract

Employing Large Language Models (LLMs) for assessment and verification remains an unresolved problem. Our study shows that even within a simple logical system like the Russell–Bernays axiom system, state-of-the-art LLMs still frequently misjudge the validity of proofs. This reveals that probabilistic models are not good as reliable checkers, because they often accept arguments that sound right but are logically wrong, and sometimes reject correct steps for no good reason. To bridge this gap, we design a neuro-symbolic grading framework that decouples semantic interpretation from logical validation. Instead of relying on the model's internal judgment, our pipeline converts informal proofs into a Domain-Specific Language and leverages the Lean 4 kernel as a deterministic verifier, which provides formal feedback to help produce valid assessments. Ultimately, our work demonstrates that trustworthy verification cannot rely on probabilistic reasoning alone, but demands the feedback of symbolic kernels, which can be explored in more domains where trustworthiness must be guaranteed.

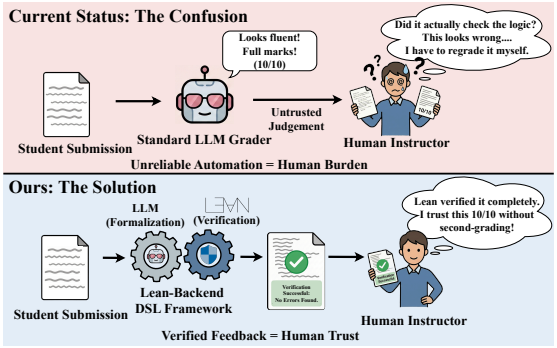

Figure 1: Framework overview. Top: Standard LLMs may produce untrustworthy scores. Bottom: Our Lean-backend DSL framework ensures logical correctness via formal verification, establishing human trust.

1 Introduction

The rapid advancement of LLMs has dramatically expanded their capability frontier, enabling impressive performance on complex reasoning and mathematical tasks. This progress naturally invites their application beyond mere problem-solving, toward roles as autonomous evaluators capable of judging solutions produced by humans or other AI systems (Vu et al., 2024; Raju et al., 2024; Gu et al., 2025). However, deploying LLMs in such high-stakes assessment capacities introduces significant challenges tied to their reliability and inherent propensity for hallucination (Dorner et al., 2025; Bavaresco et al., 2025; Panickssery et al., 2024; Gupta et al., 2025). While problem-solving can often leverage test-time feedback or sampling to improve output quality (Snell et al., 2025; Zuo et al., 2025), the act of evaluation itself demands consistent stability and trustworthy judgment (Bavaresco et al., 2025; Koo et al., 2024). Crucially, even if a model's assessment-accuracy is high, we lack a built-in mechanism to verify the correctness of any given assessment, which is a gap that leaves the risk of persuasive but erroneous feedback unresolved (Gupta et al., 2025).

This reliability gap is exposed in foundational domains like logical system. A prime example is the evaluation of proofs within compact systems such as Russell-Bernays axiom system—arguably one of the simplest and most well-defined logical frameworks. Yet, even here, LLMs exhibit critical and persistent failures. Driven by statistical likelihood rather than deductive rules, they frequently falter through dual modes: either failing to detect the use of external theorems, or misunderstanding the validity of inference steps. Conversely, while purely symbolic verifiers offer deterministic guarantees, they cannot interpret the informal expressions typical of human or LLM-generated proofs. Thus, a core challenge remains unresolved: achiev-

ing rigorous, automatic assessment for even the simplest informal logical systems. Solving this is not merely an academic exercise; it represents a fundamental step toward building trustworthy AI assessors capable of providing reliable feedback in educational and high-stakes scenarios.

To bridge the gap between the flexibility of LLMs and the rigor of formal verification, we propose a neuro-symbolic framework in Figure 1. Its core design decouples semantic interpretation, which is handled by an LLM, from logical verification, which is delegated to a formal proof assistant. A key innovation is the introduction of a domain-specific language (DSL) that acts as a structured intermediate representation. This DSL substantially simplifies the translation from natural-language proofs to a normalized, machine-readable form. Once expressed in the DSL, proofs can be automatically and deterministically converted into code for a general-purpose formal system (e.g., Lean 4). This enables the framework to leverage the underlying system's powerful capabilities—such as type-driven correctness checking and compiler-generated error diagnostics—while maintaining broad applicability. Finally, error messages from the formal backend are mapped back through the DSL layer into human-readable feedback, closing the loop from informal submission to verified, pedagogically useful assessment.

To validate the effectiveness of our framework, we focus on the domain of propositional logic derivation. The teaching and grading of formal logic serve as an ideal testbed for LLM-based assessment, as it demands strict syntactic adherence while handling varied and often subtly flawed student submissions. Within this domain, the Russell-Bernays axiom system provides a concrete and well-bounded instance. It operates under a minimal set of axioms and rules, offering a constrained environment where proofs admit many valid but error-prone paths. By implementing our neuro-symbolic grading pipeline for this system, we translate our design into practice. Experiments on real student submissions show that our method reliably outperforms pure LLM-based grading, significantly reducing erroneous evaluations.

Our contributions are as follows: (1) We propose a neuro-symbolic framework centered on a DSL that simplifies the translation of natural-language proofs into a machine-checkable form and conversely maps verification errors back into human-readable feedback. (2) We collect and curate a dataset of over 1,000 real-world student proofs from a discrete-mathematics course, and a carefully annotated subset with step-level scores and error-type labels, which can be used for evaluating automated reasoning and assessment systems. Our DSL-based method outperforms pure LLM baselines on this dataset in various experiments.

2 Related Work

Recent advancements in LLMs have led to significant breakthroughs in mathematical reasoning (Ho, 2025; Hubert et al., 2025). The paradigm is shifted from static prompting strategies to inference-time compute scaling and System 2 reasoning (Zhang et al., 2025). While earlier methods relied on Chain-of-Thought (Wei et al., 2022) and Self-Consistency (Wang et al., 2023), models like GPT-5.2 (OpenAI, 2025b), DeepSeek-R1 (Guo et al., 2025), and Gemini-3 (Google, 2025) now integrate reinforcement learning for reasoning directly into the generation process.

However, a critical "Generator-Validator Gap" persists: models capable of generating correct solutions often fail to reliably verify logical validity (West et al., 2024; Rodriguez et al., 2025). Unlike generation, which requires finding one valid path, verification demands a global understanding of logical entailment to detect subtle disconnects (Jiang et al., 2025; Song et al., 2025; Gu et al., 2025). Lacking this rigorous discrimination capability, LLMs fall back on probabilistic heuristics, often confusing surface-level fluency with logical soundness (Li et al., 2025). Such behavior renders pure LLMs unsuitable for high-stakes domains requiring deductive soundness (Chudziak and Kostka, 2025; Szymanski et al., 2025; Gupta et al., 2025).

To bridge the gap between informal reasoning and rigorous verification, researchers have turned to Autoformalization (Wu et al., 2022; Weng et al., 2025), which is to translate natural language mathematics into formal specifications of theorem provers like Lean (Moura and Ullrich, 2021), Rocq (Team, 2025b), or Isabelle (Nipkow et al., 2002). This paradigm underpins Neural Theorem Proving, where LLMs act as tactic generators to search for proofs in a formal environment (Chen et al., 2025; Ren et al., 2025; Lin et al., 2025).

Notable frameworks include Baldur (First et al., 2023), which employs whole-proof generation and refinement using compiler error messages as feedback signals. Draft, Sketch, and Prove (Jiang et al.,

2023; Cao et al., 2025), decouples high-level logical planning from low-level automation. LeanDojo (Yang et al., 2023) allows models to extract proof states and ASTs with the Lean kernel. Delta Prover (Zhou et al., 2025) also designs a DSL for interaction with Lean 4, similar to our approach. However, their DSL does not serve as an independent intermediate layer. Instead, it repackages a few commonly used but syntactically complex tactics in Lean 4, primarily to simplify specific expressions. Consequently, the model still needs to directly interact with Lean 4 through interfaces beyond this DSL when writing and revising proofs. While prior methods focus on automated theorem proving, we repurpose our pipeline for compiler-grounded diagnostics. By bridging informal logic to Lean via our DSL, we leverage the trusted Lean kernel to ensure that grading is grounded in strict compilation rather than probabilistic predictions.

Traditional Intelligent Tutoring Systems have long relied on brittle, handcrafted rules, limiting their scalability to complex domains. Conversely, first-generation GenAI tutors, while offering interactive flexibility, suffer from inherent unreliability in educational contexts (Van Vaerenbergh and Pérez-Suay, 2022; Chudziak and Kostka, 2025). Recent research converges on neuro-symbolic architectures that integrate neural semantic parsing with symbolic verification. For instance, some works (Hare and Tang, 2025; Jaldi et al., 2025) leverage educational ontologies and knowledge graphs as symbolic backbones, ensuring that instructional dialogue is factually grounded and pedagogically consistent. In the domain of formal logic, LeanTutor (Patel et al., 2025) exemplifies this synergy by translating proof states into natural language hints, grounding pedagogical suggestions in valid formal tactics.

Recent neuro-symbolic architectures, however, still face a fundamental trade-off between the generality of the symbolic system and the accuracy of LLM-based formalization. For instance, educational dialogue systems are often limited to simple, rule-based reasoning and cannot handle complex derivations like mathematical proofs. Conversely, tools such as LeanTutor remain constrained by the difficulty of reliably converting informal student work into a verifiable formal representation.

Our approach introduces an intermediate DSL that decouples these two challenges and enables both accurate formalization and explainable error reporting.

3 Preliminaries

3.1 Discrete Mathematics, Propositional Logic, and Russell-Bernays Axiom System

For many students, discrete mathematics is their first real introduction to formal logic. The challenge in propositional logic lies in shifting focus from the truth of a statement to the rigor of its derivation. In an axiomatic system, this means treating a proof as a chain of syntactic transformations, where semantic meaning takes a backseat to the mechanical application of rules.

Russell-Bernays axiom system and proof style. The Russell-Bernays axiom system we disscussed adopts a Hilbert-style presentation. Proofs consist of linear derivations constructed from two core components: (i) a small set of axiom schemata, and (ii) three fundamental transformation rules.

The permissible rules are:

- `Modus Ponens` (MP): From proofs of A and $A \to B$, derive the proof of B.
- `Substitution`: Replace a proof's meta-variables with well-formed formulas to get the proof of the new formula.
- `Replacement`: Derive a new formula by replacing any sub-formula with a logically equivalent expression, based on a definition (e.g., replacing $A \to B$ with $\neg A \vee B$).

Additionally, students are permitted to cite results proved in earlier subproblems, but only if those results have been explicitly introduced and designated as reusable lemmas.

3.2 Lean 4 as a Formal Foundation

We employ Lean 4 (Moura and Ullrich, 2021) as the formal foundation for our verification pipeline. Lean 4 is a proof assistant and a functional programming language based on dependent type theory, which unifies program verification and mathematical proof within a single, consistent framework. Compared to manually implementing the Russell system's rules in a general-purpose language like Python, Lean 4 offers two critical advantages for our task. First, its powerful type system can precisely express the structure of logical propositions and proofs, guaranteeing the syntactic compliance of derivations at the most fundamental level. Second, its rich metaprogramming and tactic framework enables the construction of efficient, scalable automation for proof synthesis and checking.

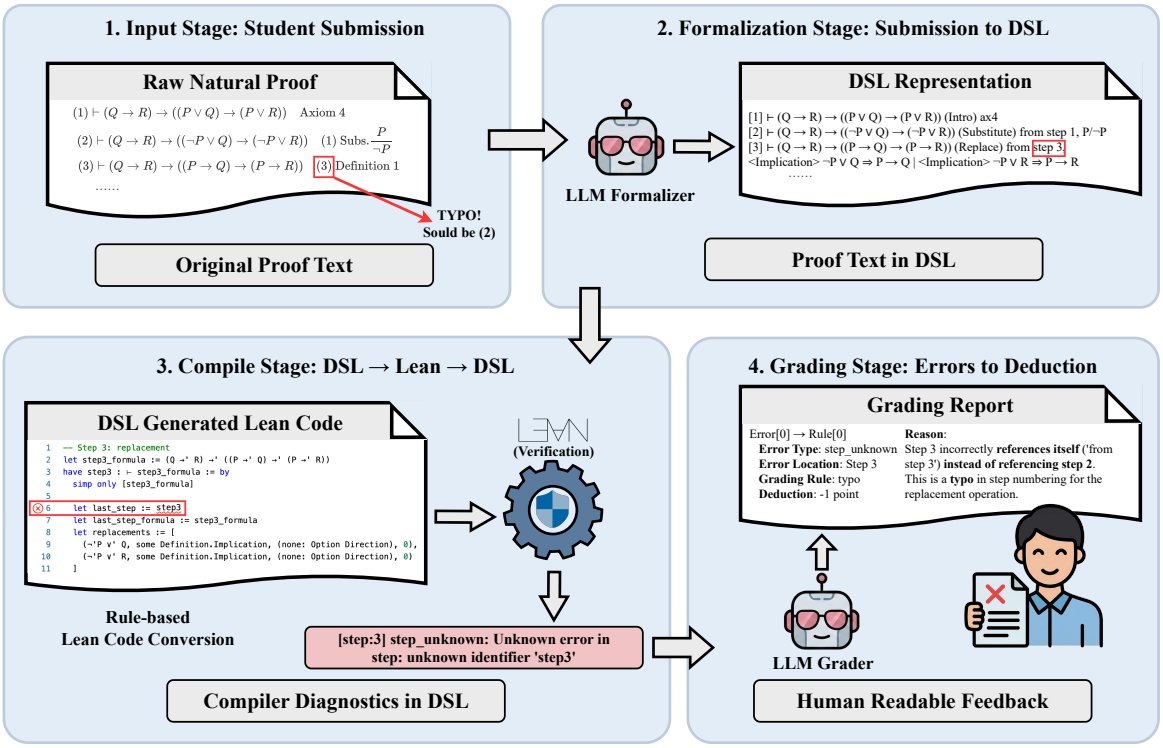

Figure 2: The four-stage pipeline of our neuro-symbolic grading framework. (1) Input: The system receives a raw natural proof. (2) Formalization: An LLM formalizes the proof into DSL representation. (3) Verification Stage: The DSL is converted into Lean code for verification; compiler diagnostics are then mapped back to the DSL. (4) Grading Stage: An LLM Grader maps DSL diagnostics to specific error categories and point deductions with reason.

4 Method

Our pipeline is built upon a neuro-symbolic framework that integrates the Lean 4 theorem prover with a custom DSL. The architecture, outlined in Figure 2, consists of three core components that operate from the bottom up: (1) a formally verified kernel of the Russell-Bernays axiom system implemented in Lean 4; (2) a DSL layer, along with a hybrid formalization process that bridges natural-language proofs to the kernel; and (3) an LLM-based assessment module that utilizes formal verification feedback for accurate grading.

4.1 Russell–Bernays Axiom System in Lean

We implemented the Russell–Bernays axiom system within Lean 4's type system. We first defined a custom `Formula` type and a corresponding `Provable` relation, thereby avoiding any interference with Lean's native `Prop` type. On this foundation, we implemented the primitive transformation functions and then declared the four Russell axioms, the three transformation rules, and an additional rule for introducing previously proved theorems as Lean `axioms`. This design exposes only the permitted Russell-style inferences to the rest of the system. As a result, proofs can directly employ Lean's efficient tactics, such as `simp`, `simp only`, and `exact`, and naturally follow Lean's native theorem-proving pattern, making the formalization both rigorous and practically convenient.

4.2 DSL and Hybrid Formalization

DSL Design. Built upon the Russell kernel, our DSL supports the three core transformation rules and the introduction of previously established theorems from the Russell system, as defined in Section 3.1. Each proof step is represented as a valid operation type coupled with its required parameters. Every such operation can be translated deterministically into Lean 4 source code via rule-based templates. These generated code fragments are then assembled into a complete, compilable Lean proof for subsequent verification.

Hybrid Formalization. This implementation creates an abstraction layer above Lean source code. The formalization from noisy, natural-language proof text into DSL is performed by a general-purpose LLM (e.g., DeepSeek-V3.2 (DeepSeek-AI,

2025)). The model's task is to segment the proof into individual steps, classify each step's operation type, and extract the corresponding parameters to produce a list of DSL steps. Once the DSL step sequence is obtained, its translation to Lean code, compilation, and the generation of feedback are entirely rule-based and error-free.

4.3 Formally Verified AI Grading

After compilation, low-level feedback from Lean is first mapped back through the DSL layer to produce human-readable error messages linked to specific proof steps. A separate LLM grader then receives the original proof text, the formalized DSL steps, these parsed error messages, and the detailed grading rubric. Based on this information, the grader is prompted to identify where points should be deducted at each step and for the overall proof.

5 Experiments

5.1 Experimental Setup

We describe four key aspects of our experimental design: the comparison methods, including ablations, the dataset for evaluation, the metrics, and the model selection.

5.1.1 Comparison Methods

To systematically evaluate the contribution of each component in our framework, we compare four grading configurations. The key difference among them lies in the additional information provided to the LLM grader, as summarized in Table 1.

Table 1: Information for each method. All methods receive the original proof text and the grading rubric.

Method	DSL Steps	DSL Errors	Std. Proof
Baseline (LLM-only)	×	×	×
Ours (Full DSL)	✓	✓	×
DSL w/o Errors	✓	×	×
Baseline w/ Std. Proof	×	×	✓

The **Baseline (LLM-only)** method relies solely on the original text without any structural or formal aids. **Ours (Full DSL)** uses both the structured proof steps and the formal verification feedback. The two ablations isolate specific components: **DSL w/o Errors** removes the compiler feedback, while **Baseline w/ Std. Proof** applies preprocessing to clean the proof's formatting and fill in permitted default parameters, isolating the effect of textual standardization.

Since most of the selected models are reasoning models, we impose no maximum token limit for the formalization and grading tasks. For models that support temperature, we set it to 0.2. In the proof standardization, a maximum token limit of 4096 and a temperature of 0 are applied. All results reported are from a single generation run.

5.1.2 Russell-Bernays Axiom Dataset

We collected over 1,000 raw student submissions across five problem sets. Following deduplication and manual verification of LaTeX formatting, we retained approximately half as valid proofs. To construct a representative evaluation subset, we scored all valid proofs using both our method and the baseline, then uniformly sampled 200 proofs based on their average score and scoring discrepancy for manual annotation. Each annotation provides step-level error types; see Appendix A for full details.

5.1.3 Metrics

We evaluate grading performance by comparing the set of predicted deductions against the ground truth annotations. For a given proof, let $\mathcal{P} = \{(s_i, t_i)\}$ denote the set of predicted deductions, where each pair indicates an error at step s_i of type t_i. Similarly, let $\mathcal{G} = \{(s_j, t_j)\}$ be the ground truth set. A prediction matches a ground truth item if both its step index and error type are identical ($s_i = s_j$ and $t_i = t_j$).

We report two suites of metrics: the first treats all error deductions equally, while the second weights each deduction by the point value of its corresponding grading rule. The following definitions apply to the unweighted case.

Micro- & Macro-F1. Micro-F1 aggregates TP, FP, and FN across all proofs before calculation, while Macro-F1 averages per-proof F1 scores with Laplace smoothing ($\epsilon = 0.1$):

$$\text{Micro-F1} = \frac{2 \cdot \sum \text{TP}}{2 \cdot \sum \text{TP} + \sum \text{FP} + \sum \text{FN}}.$$

$$\text{Macro-F1} = \frac{1}{N} \sum_{k=1}^{N} \text{F1}_k.$$

Jaccard Similarity. This measures the overlap between predicted and ground truth sets, with smoothing to handle edge cases:

$$\text{Jaccard}(\mathcal{P}, \mathcal{G}) = \frac{|\mathcal{P} \cap \mathcal{G}| + \epsilon}{|\mathcal{P} \cup \mathcal{G}| + \epsilon}, \quad \epsilon = 0.1.$$

Symmetric Difference Similarity. This metric penalizes discrepancies using an inverse scaling function:

$$\text{SymDiff}(\mathcal{P}, \mathcal{G}) = \frac{1}{1 + \lambda \cdot |\mathcal{P} \triangle \mathcal{G}|},$$

where $\lambda = 1.0$ controls the penalty rate.

For the weighted variants, we replace set cardinalities with sums of deduction point values: $|\mathcal{S}| \to \sum_{d \in \mathcal{S}} w(d)$, where $w(d)$ is the grading rule point value for deduction d.

5.1.4 Model Selection

We evaluate models ranging from closed-source (GPT-5 (OpenAI, 2025a), Qwen-Max (Team, 2025a), Kimi-K2 (Team et al., 2025b), Seed-1.6 (Seed et al., 2025)) to open-source (DeepSeek-V3.2 (DeepSeek-AI, 2025), GLM-4.7 (Team et al., 2025a)), including both reasoning and non-reasoning generation types like GPT-4o-mini (OpenAI, 2024), Seed-1.6-Flash.

5.2 Formalization Accuracy

To evaluate the LLM's accuracy in translating original proofs into DSL sequences, we performed additional manual annotation to verify the correctness of each formalized step. We pick the results from DeepSeek-V3.2[1] for evaluation. Errors were categorized not only as incorrect translations of operation types or parameters, but also included cases where the model implicitly corrected a student's typographical error, incorrectly filled in permissible default parameters, or failed to assign the required default value to an invalid parameter.

Table 2: Formalization accuracy of DeepSeek-V3.2 by action type. 'Nop' indicates invalid steps.

Action Type	Total Steps	Errors	Accuracy
Intro	528	9	98.30%
MP	332	5	98.49%
Nop	20	1	95.00%
Replace	320	7	97.81%
Substitute	538	6	98.88%
Overall	**1738**	**28**	**98.39%**

The results in Table 2 show that DeepSeek-V3.2 achieves consistently high success rates in formalization, with overall accuracy exceeding 98%. This demonstrates the effectiveness of the DSL. We note that the `Replace` operation exhibits a lower accuracy compared to others. This is expected, as `Replace` requires the model to infer and fill in parameters omitted in the informal proof, introducing a greater chance for misinterpretation.

Furthermore, by analyzing the steps where formalization errors occurred, we observe that despite being instructed otherwise, a prevalent error pattern is the model's implicit correction of student typos, thereby altering the intended meaning of the original proof. Of the 8 steps where the DSL assigned no deduction but the human grader did, 5 were marked by the grader as typos. Similarly, among the 10 steps where both the DSL and the human grader assigned deductions, 8 were also typographical errors. This indicates that a proportion of formalization errors arises not from a failure to understand the logical structure, but from the model's over-correction of typos in the student's original text.

5.3 Results

We present the evaluation of our framework on the annotated dataset of student proofs based on a series of models, including overall grading performance and a detailed analysis, as well as an ablation study that quantifies the contribution of individual components: proof structuring via the DSL and compiler-generated error messages.

5.3.1 Overall Grading Performance

Table 3 compares the grading performance of eight foundation models when using our DSL-based method versus the LLM-only baseline. Across most models, formal verification feedback yields a clear improvement.

The largest gains appear with DeepSeek-V3.2 (Also shown in Figure 4) and GLM-4.7 . Weaker models such as GPT-4o-mini and Seed-1.6-Flash also show substantial benefits, with gains exceeding +0.15 on several macro-level metrics. This demonstrates that the DSL feedback effectively compensates for weaker native reasoning.

Two exceptions are GPT-5 and Kimi-K2, where the DSL method offers little to no improvement or even slight degradation. Since they often produce extremely long reasoning processes and take much more time compared to the DSL method, we hypothesize that these models already employ extensive internal reasoning, which may not integrate effectively with the structured feedback, leaving limited room for further gain.

[1] DeepSeek-V3.2 performs best in grading results.

Table 3: Comparison of alignment with human judgment across different foundation models, using both unweighted and weighted metrics: Baseline vs. DSL-based method.

Model	Micro-F1 (↑)		Macro-F1 (↑)		Jaccard (↑)		Sym-Diff Sim (↑)	
	Base.	DSL	Base.	DSL	Base.	DSL	Base.	DSL
Unweighted								
DeepSeek-V3.2	0.587	**0.672** +0.085	0.667	**0.783** +0.116	0.621	**0.745** +0.125	0.699	**0.787** +0.087
GPT-5	0.655	0.631 -0.024	0.768	0.754 -0.014	0.726	0.714 -0.013	0.762	0.757 -0.004
GLM-4.7	0.473	0.611 +0.139	0.584	0.741 +0.157	0.533	0.704 +0.171	0.625	0.762 +0.137
Kimi-K2	0.634	0.603 -0.031	0.738	0.753 +0.015	0.696	0.717 +0.021	0.754	0.762 +0.008
Qwen-Max	0.593	0.598 +0.006	0.698	0.723 +0.025	0.655	0.682 +0.027	0.727	0.745 +0.018
Seed-1.6	0.430	0.576 +0.147	0.600	0.712 +0.112	0.541	0.667 +0.125	0.611	0.723 +0.113
Seed-1.6-Flash	0.246	0.305 +0.059	0.347	0.505 +0.158	0.282	0.455 +0.173	0.394	0.585 +0.192
GPT-4o-mini	0.153	0.151 -0.002	0.228	0.357 +0.128	0.170	0.308 +0.138	0.314	0.466 +0.152
Weighted								
DeepSeek-V3.2	0.597	**0.659** +0.062	0.656	**0.770** +0.114	0.620	**0.737** +0.118	0.630	**0.743** +0.113
GPT-5	0.639	0.605 -0.034	0.754	0.739 -0.015	0.717	0.704 -0.013	0.719	0.709 -0.009
GLM-4.7	0.478	0.624 +0.146	0.571	0.737 +0.165	0.530	0.704 +0.174	0.558	0.725 +0.167
Kimi-K2	0.642	0.625 -0.017	0.736	0.745 +0.009	0.699	0.713 +0.014	0.711	0.719 +0.008
Qwen-Max	0.591	0.583 -0.008	0.690	0.707 +0.017	0.654	0.673 +0.019	0.678	0.698 +0.021
Seed-1.6	0.450	0.598 +0.147	0.588	0.706 +0.118	0.539	0.667 +0.128	0.545	0.677 +0.133
Seed-1.6-Flash	0.262	0.347 +0.086	0.322	0.490 +0.168	0.270	0.451 +0.181	0.282	0.507 +0.226
GPT-4o-mini	0.165	0.186 +0.021	0.205	0.339 +0.134	0.159	0.302 +0.143	0.206	0.371 +0.165

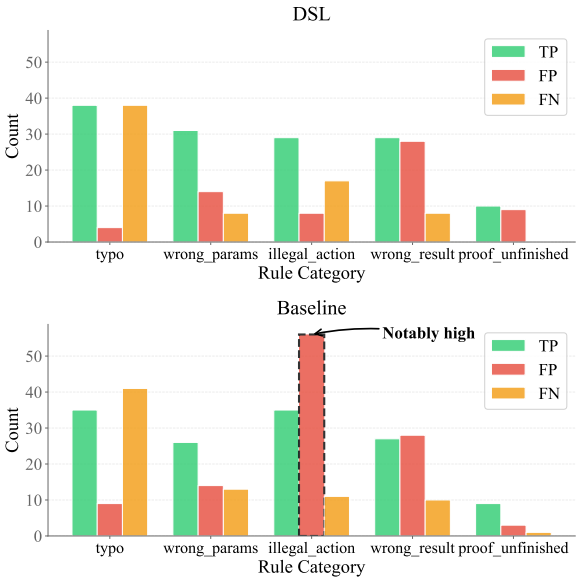

Figure 3: Distribution of TP, FP, and FN by error category for our method and Baseline.

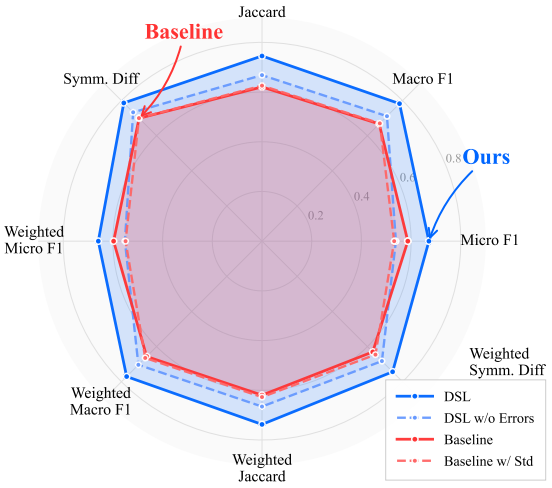

Figure 4: Radar chart comparing the overall performance of four methods.

5.3.2 Comparison with Baseline

To examine the differences between the two approaches in detail, we select the highest-performing model, DeepSeek-V3.2, for fine-grained comparison and subsequent ablation studies. Figure 3 details the breakdown of TP, FP, FN across five consolidated error categories. The comparison reveals distinct error-handling profiles for the two methods. The original grading rules and the categories can be found in Appendix A.2.

A key failure mode of the baseline is its excessive number of FPs in the `illegal_action` category, far exceeding that of our method. Without rule-grounded verification, the baseline often hallucinates violations of inference rules, penalizing many logically valid steps.

Both methods exhibit relatively high FN counts for typo errors. Detecting typos is inherently challenging, as it requires the grader to discern whether

a superficial inconsistency can be trivially corrected without altering the intended logical meaning, which is a task that demands global proof context and is difficult even for human annotators.

5.4 Ablation Study

To disentangle the contribution of proof structuring from that of formal verification feedback, we evaluate two ablated variants alongside the full pipeline and the pure baseline, as mentioned in 5.1.1.

The results, summarized in Figure 4, show that the complete DSL method (blue) consistently outperforms all other configurations across every metric. Crucially, the two ablated variants perform much closer to the pure baseline (red) than to the full DSL method. This indicates that the primary gain of our approach stems from the formal verification feedback, not merely from proof restructuring or surface-level text normalization. Together, these ablations validate that the core advantage of our neuro-symbolic framework lies in its ability to ground the LLM's assessment in deterministic, compiler-checked diagnostics.

6 Case Study

In this section, we present a comparative analysis of our method and the baseline through a concrete example; see Appendix C for more examples.

Goal: $\vdash P \vee \neg P$
Natural Proof:

$$
\begin{aligned}
&(1) \vdash P \to (P \vee Q) && \text{Ax. 2}\\
&(2) \vdash (P \vee P) \to P && \text{Ax. 1}\\
&(3) \vdash P \to (P \vee P) && \text{Subs. (1)}, \frac{Q}{P}\\
&(4) \vdash P \to P && \text{MP (3) (2)}\\
&\dots
\end{aligned}
$$

Proof 1. Error in (4), wrong using MP.

```
Success: False
Theorem Name: homework_3
Formula: P ∨ ¬P
Proof Steps: 4
[4] ⊢ P → P (MP) minor: step 3, major:
    step 2
[step:4] step_standard (MPPremiseMismatch):
    MP Error: Step 3 (minor premise) does
    not match the antecedent of step 2
    (major premise)

DSL Grading:
-3 points (illegal_mp)
```

(cont.)

```
Reason: Modus ponens incorrectly applied:
    minor premise P → (P ∨ P) does not
    match antecedent (P ∨ P) of major
    premise (P ∨ P) → P.

Baseline Grading:
No deductions (perfect proof).
```

Grade 1. Baseline did not find the error.

In the proof, step (4) incorrectly applies MP, essentially attempting to use syllogism, which is an invalid transformation rule. As shown in the grading, because the Lean code compiled from the DSL steps fails, the DSL-based grader detects that steps (2) and (3) do not constitute a legal MP application and flags the step as illegal. In contrast, the baseline grader fails to identify this error, as it confuses the rules of the Russell–Bernays axiom system with the inference rules of general propositional logic. We note that the scenario described above does not frequently occur with a strong LLM baseline, which can generally identify errors according to the prescribed axiomatic rules. However, since such failures remain probable, a rule-based verification layer significantly reduces the risk of their occurrence.

7 Conclusion

Our work focuses on employing LLMs for verification and grading tasks. We propose a neuro-symbolic pipeline that transforms natural language proofs into an intermediate Domain-Specific Language before compiling them for formal verification, and retrieves feedback from the formal system to aid LLM grading. Our framework has been validated on the Russell–Bernays axiom system within propositional logic, demonstrating more robust and reliable performance compared to purely LLM-based grading on a dataset from real-world student submissions. This approach can be extended to other domains where logical structure can be extracted via LLMs and then rigorously verified through rule-based formal systems, thereby enabling safer and more effective use of LLMs' broad knowledge in high-stakes scenarios such as education.

8 Limitations

Our work has three main limitations that point to areas for future work.

First, the accuracy of the entire system depends heavily on the first step: translating the student's proof into DSL. Although the DSL makes this easier, the LLM can still make mistakes, especially when the student's writing is messy or skips steps. Sometimes the LLM even "corrects" what it sees as typos, accidentally changing the meaning of the proof.

Second, for our experiments we used a simple rule where each type of error results in a fixed point deduction. This makes scoring clear and easy to measure, but real teaching assistants often adjust deductions based on how serious an error is within the whole proof. Judging error severity is a harder task that requires understanding the proof globally, which we leave for future work. Our current focus is on reliably finding errors in the first place.

Finally, all of our experiments were done within the Russell–Bernays axiom system because that was the source of our student data. It is a small, well-defined system that is a good starting point for our research. However, our framework is designed to be flexible. Similar axiom systems (e.g., Frege or Łukasiewicz) or general logical reasoning could be supported in the future by adapting the DSL and the Lean kernel, extending the approach to more practical domains.

9 Ethical Considerations

The development of AI-based automated graders, particularly those intended for educational assessment, carries broader ethical implications beyond data privacy. Our work prompts consideration of the following points.

While our framework reduces hallucination by grounding feedback in formal verification, its fairness across diverse student populations remains an open question. The initial proof translation is performed by a general-purpose LLM, which may encode societal biases that could subtly influence how it interprets linguistic style or informal reasoning patterns. A rigorous audit for demographic fairness is an essential next step before any real-world deployment.

Also, the societal impact of such tools must be carefully managed. On one hand, they can make detailed, immediate feedback scalable, potentially democratizing access to high-quality educational resources. On the other hand, over-reliance on automated scoring could inadvertently narrow the range of "acceptable" proof styles or prioritize syntactic correctness over creative problem-solving. It is crucial to position these systems as assistants that handle routine verification, thereby freeing instructor time for more nuanced, pedagogical interactions that foster deeper understanding.

References

Anna Bavaresco, Raffaella Bernardi, Leonardo Bertolazzi, Desmond Elliott, Raquel Fernández, Albert Gatt, Esam Ghaleb, Mario Giulianelli, Michael Hanna, Alexander Koller, Andre Martins, Philipp Mondorf, Vera Neplenbroek, Sandro Pezzelle, Barbara Plank, David Schlangen, Alessandro Suglia, Aditya K Surikuchi, Ece Takmaz, and Alberto Testoni. 2025. LLMs instead of human judges? a large scale empirical study across 20 NLP evaluation tasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 238–255, Vienna, Austria. Association for Computational Linguistics.

Chenrui Cao, Liangcheng Song, Zenan Li, Xinyi Le, Xian Zhang, Hui Xue, and Fan Yang. 2025. Reviving dsp for advanced theorem proving in the era of reasoning models. *Preprint*, arXiv:2506.11487.

Jiangjie Chen, Wenxiang Chen, Jiacheng Du, Jinyi Hu, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Wenlei Shi, Zhihong Wang, Mingxuan Wang, Chenrui Wei, Shufa Wei, Huajian Xin, Fan Yang, Weihao Gao, Zheng Yuan, Tianyang Zhan, and 3 others. 2025. Seed-prover 1.5: Mastering undergraduate-level theorem proving via learning from experience. *Preprint*, arXiv:2512.17260.

Jarosław A. Chudziak and Adam Kostka. 2025. Ai-powered math tutoring: Platform for personalized and adaptive education. *Preprint*, arXiv:2507.12484.

DeepSeek-AI. 2025. Deepseek-v3.2: Pushing the frontier of open large language models.

Florian E. Dorner, Vivian Yvonne Nastl, and Moritz Hardt. 2025. Limits to scalable evaluation at the frontier: LLM as judge won't beat twice the data. In *The Thirteenth International Conference on Learning Representations*.

Emily First, Markus N. Rabe, Talia Ringer, and Yuriy Brun. 2023. Baldur: Whole-proof generation and repair with large language models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2023, page 1229–1241, New York, NY, USA. Association for Computing Machinery.

Google. 2025. A new era of intelligence with gemini 3. Accessed: 2026-01-03.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. 2025. A survey on llm-as-a-judge. *Preprint*, arXiv:2411.15594.

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.

Adit Gupta, Jennifer Reddig, Tommaso Calò, Daniel Weitekamp, and Christopher J. MacLellan. 2025. Beyond final answers: Evaluating large language models for math tutoring. In *Artificial Intelligence in Education: 26th International Conference, AIED 2025, Palermo, Italy, July 22–26, 2025, Proceedings, Part I*, page 323–337, Berlin, Heidelberg. Springer-Verlag.

Ryan Hare and Ying Tang. 2025. Toward generalized autonomous agents: A neuro-symbolic ai framework for integrating social and technical support in education. *Preprint*, arXiv:2508.18406.

Anson Ho. 2025. Is ai already superhuman on frontiermath? Accessed: 2026-01-03.

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z. Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, Ottavia Bertolli, Tom Zahavy, Amol Mandhane, Jessica Yung, Iuliya Beloshapka, Borja Ibarz, Vivek Veeriah, Lei Yu, Oliver Nash, and 19 others. 2025. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*.

Chris Davis Jaldi, Eleni Ilkou, Noah Schroeder, and Cogan Shimizu. 2025. Education in the era of neurosymbolic AI. *Journal of Web Semantics*, 85:100857.

Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Timothee Lacroix, Jiacheng Liu, Wenda Li, Mateja Jamnik, Guillaume Lample, and Yuhuai Wu. 2023. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *The Eleventh International Conference on Learning Representations*.

Dongwei Jiang, Jingyu Zhang, Orion Weller, Nathaniel Weir, Benjamin Van Durme, and Daniel Khashabi. 2025. Self-[in]correct: Llms struggle with discriminating self-generated responses. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(23):24266–24275.

Ryan Koo, Minhwa Lee, Vipul Raheja, Jong Inn Park, Zae Myung Kim, and Dongyeop Kang. 2024. Benchmarking cognitive biases in large language models as evaluators. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 517–545, Bangkok, Thailand. Association for Computational Linguistics.

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, Kai Shu, Lu Cheng, and Huan Liu. 2025. From generation to judgment: Opportunities and challenges of LLM-as-a-judge. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 2757–2791, Suzhou, China. Association for Computational Linguistics.

Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, Jiayun Wu, Jiri Gesi, Ximing Lu, David Acuna, Kaiyu Yang, Hongzhou Lin, Yejin Choi, Danqi Chen, Sanjeev Arora, and Chi Jin. 2025. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. *Preprint*, arXiv:2508.03613.

Leonardo de Moura and Sebastian Ullrich. 2021. The Lean 4 Theorem Prover and Programming Language. In *Automated Deduction – CADE 28*, volume 12699 of *Lecture Notes in Computer Science*, pages 625–635, Cham. Springer International Publishing.

Tobias Nipkow, Markus Wenzel, and Lawrence C Paulson. 2002. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer.

OpenAI. 2024. Gpt-4o mini: advancing cost-efficient intelligence. Accessed: 2026-01-03.

OpenAI. 2025a. Gpt-5 system card. Accessed: 2026-01-03.

OpenAI. 2025b. Introducing gpt-5.2. Accessed: 2026-01-03.

Arjun Panickssery, Samuel R. Bowman, and Shi Feng. 2024. Llm evaluators recognize and favor their own generations. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

Manooshree Patel, Rayna Bhattacharyya, Thomas Lu, Arnav Mehta, Niels Voss, Narges Norouzi, and Gireeja Ranade. 2025. Leantutor: A lean-verified tutor for mathematical proofs. In *2nd AI for Math Workshop @ ICML 2025*.

Ravi Shanker Raju, Swayambhoo Jain, Bo Li, Jonathan Lingjie Li, and Urmish Thakker. 2024. Constructing domain-specific evaluation sets for LLM-as-a-judge. In *Proceedings of the 1st Workshop on Customizable NLP: Progress and Challenges in Customizing NLP for a Domain, Application, Group, or Individual (CustomNLP4U)*, pages 167–181, Miami, Florida, USA. Association for Computational Linguistics.

Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao,

Daya Guo, and Chong Ruan. 2025. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *Preprint*, arXiv:2504.21801.

Juan Diego Rodriguez, Wenxuan Ding, Katrin Erk, and Greg Durrett. 2025. Rankalign: A ranking view of the generator-validator gap in large language models. In *Second Conference on Language Modeling*.

ByteDance Seed, :, Jiaze Chen, Tiantian Fan, Xin Liu, Lingjun Liu, Zhiqi Lin, Mingxuan Wang, Chengyi Wang, Xiangpeng Wei, Wenyuan Xu, Yufeng Yuan, Yu Yue, Lin Yan, Qiying Yu, Xiaochen Zuo, Chi Zhang, Ruofei Zhu, Zhecheng An, and 255 others. 2025. Seed1.5-thinking: Advancing superb reasoning models with reinforcement learning. *Preprint*, arXiv:2504.13914.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*.

Yuda Song, Hanlin Zhang, Carson Eisenach, Sham M. Kakade, Dean Foster, and Udaya Ghai. 2025. Mind the gap: Examining the self-improvement capabilities of large language models. In *The Thirteenth International Conference on Learning Representations*.

Annalisa Szymanski, Noah Ziems, Heather A. Eicher-Miller, Toby Jia-Jun Li, Meng Jiang, and Ronald A. Metoyer. 2025. Limitations of the llm-as-a-judge approach for evaluating llm outputs in expert knowledge tasks. In *Proceedings of the 30th International Conference on Intelligent User Interfaces*, IUI '25, page 952–966, New York, NY, USA. Association for Computing Machinery.

GLM Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, and 152 others. 2025a. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *Preprint*, arXiv:2508.06471.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, and 150 others. 2025b. Kimi k2: Open agentic intelligence. *Preprint*, arXiv:2507.20534.

Qwen Team. 2025a. Qwen3-max: Just scale it. Accessed: 2026-01-03.

The Rocq Development Team. 2025b. The rocq prover.

Steven Van Vaerenbergh and Adrián Pérez-Suay. 2022. A classification of artificial intelligence systems for mathematics education. In Philippe R. Richard, M. Pilar Vélez, and Steven Van Vaerenbergh, editors, *Mathematics Education in the Age of Artificial Intelligence: How Artificial Intelligence Can Serve Mathematical Human Learning*, pages 89–106. Springer International Publishing, Cham.

Tu Vu, Kalpesh Krishna, Salaheddin Alzubi, Chris Tar, Manaal Faruqui, and Yun-Hsuan Sung. 2024. Foundational autoraters: Taming large language models for better automatic evaluation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 17086–17105, Miami, Florida, USA. Association for Computational Linguistics.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Ke Weng, Lun Du, Sirui Li, Wangyue Lu, Haozhe Sun, Hengyu Liu, and Tiancheng Zhang. 2025. Autoformalization in the era of large language models: A survey. *Preprint*, arXiv:2505.23486.

Peter West, Ximing Lu, Nouha Dziri, Faeze Brahman, Linjie Li, Jena D. Hwang, Liwei Jiang, Jillian Fisher, Abhilasha Ravichander, Khyathi Chandu, Benjamin Newman, Pang Wei Koh, Allyson Ettinger, and Yejin Choi. 2024. The generative AI paradox: "what it can create, it may not understand". In *The Twelfth International Conference on Learning Representations*.

Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Norman Rabe, Charles E Staats, Mateja Jamnik, and Christian Szegedy. 2022. Autoformalization with large language models. In *Advances in Neural Information Processing Systems*.

Kaiyu Yang, Aidan M 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 *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Duzhen Zhang, Zhong-Zhi Li, Ming-Liang Zhang, Jiaxin Zhang, Zengyan Liu, Yuxuan Yao, Haotian Xu, Junhao Zheng, Xiuyi Chen, Yingying Zhang, and 1 others. 2025. From system 1 to system 2: a survey of reasoning large language models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Yichi Zhou, Jianqiu Zhao, Yongxin Zhang, Bohan Wang, Siran Wang, Luoxin Chen, Jiahui Wang, Haowei

Chen, Allan Jie, Xinbo Zhang, Haocheng Wang, Luong Trung, Rong Ye, Phan Nhat Hoang, Huishuai Zhang, Peng Sun, and Hang Li. 2025. Solving formal math problems by decomposition and iterative reflection. *Preprint*, arXiv:2507.15225.

Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, Biqing Qi, Youbang Sun, Zhiyuan Ma, Lifan Yuan, Ning Ding, and Bowen Zhou. 2025. TTRL: Test-time reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

A Dataset Details

A.1 Data Collection

We collected submission data from a discrete-mathematics course at a major research university, where students completed proof assignments based on the Russell–Bernays axiom system via an Online Judge platform. A total of 385 students participated across five distinct problem sets (Set 1–5). Students submitted their proofs in LaTeX format, following the syntactic rules of the Russell system.

Before using the data, we obtained explicit consent from the students, explaining that their anonymized submissions would be used solely for research on AI-assisted grading and teaching. Students could opt out without penalty, and no compensation was offered. All submissions were fully anonymized—removing identifiers such as names, student IDs, and timestamps—retaining only a unique submission ID and the proof text itself. The study protocol was approved by the course instructor and complied with institutional guidelines for educational data use.

Due to the flexible nature of axiomatic proofs, many valid derivation paths exist for the same theorem. We collected approximately 500 raw submissions per problem set. After applying text-based deduplication and verifying that each submission could be properly rendered as LaTeX, about 300 unique, well-formatted proofs remained per set. Detailed per-set statistics are provided in Table 4. Each set is treated as an independent proof-grading task. To simulate the incremental nature of homework, the model and the Lean kernel were supplied with lemmas proved in earlier sets, allowing those results to be cited directly in subsequent proofs.

Table 4: Dataset statistics after processing.

	Set 1	**Set 2**	**Set 3**	**Set 4**	**Set 5**	**Total**
Raw	495	492	489	480	470	2,426
Deduplicated	337	245	362	328	372	1,644
Sampled	40	40	40	40	40	200

A.2 Data Annotation

After cleaning the dataset, we performed manual grading annotations. To ensure diversity in the annotated sample, we first graded all proofs using both the Baseline and DSL methods (implemented with DeepSeek-V3.2). We then sampled 40 proofs per problem set based on the average score and the

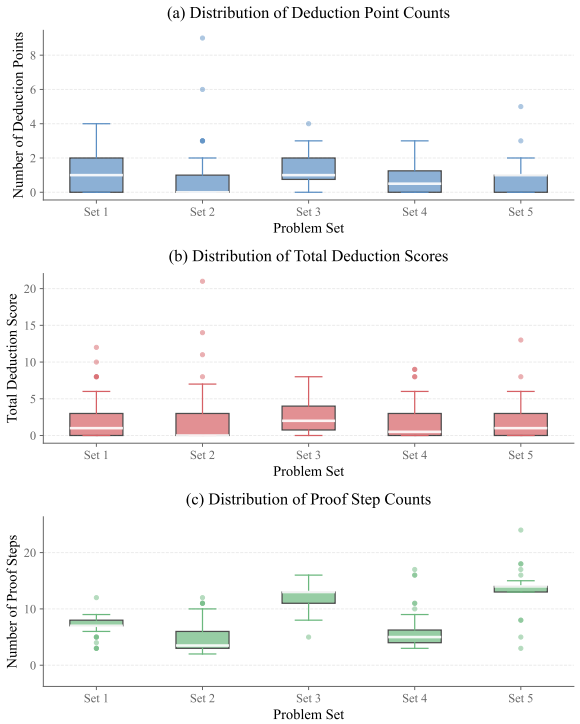

Figure 5: Distribution of the annotated proof set: number of error points, total deduction points, and proof length.

score discrepancy between the two methods. The detailed rubric used for annotation is illustrated in Figure 13. For clarity in later analysis, we group the original rule indices into five error categories: `typo` (rule 0), `wrong_params` (rules 1, 3, 4, 5), `illegal_action` (rules 6, 7, 9), `wrong_result` (rule 2, 8), and `proof_unfinished` (rule 10).

Three human annotators performed the grading. Figure 5 summarizes the characteristics of the annotated proofs, showing the distributions of the number of error points, total deduction points, and proof length.

B Extra Results

Figure 6 compares the deductions assigned by our method and the baseline against human annotations. The DSL-based method achieves a substantially higher correlation with human scores and a lower mean absolute error than the baseline. Visually, the baseline's predictions exhibit a systematic upward bias, indicating a tendency to over-deduct. This likely stems from the baseline's lack of rule-grounded verification: when asked to validate each inference step without formal support, the model often hallucinates errors in otherwise correct reasoning. Since most steps in a proof are valid, this results in an overall pattern of excessive penalization.

In contrast, the predictions of our DSL-based method are distributed more symmetrically around the diagonal, with no strong directional bias. The slight under-deduction observed in a fraction of points can be traced back to the formalization stage, where the LLM occasionally "corrects" student typos or fills in permissible defaults, as noted in Section 5.2. Overall, the integration of formal verification effectively mitigates the systematic over-penalization of the pure LLM grader and yields deductions that align much more closely with human judgment.

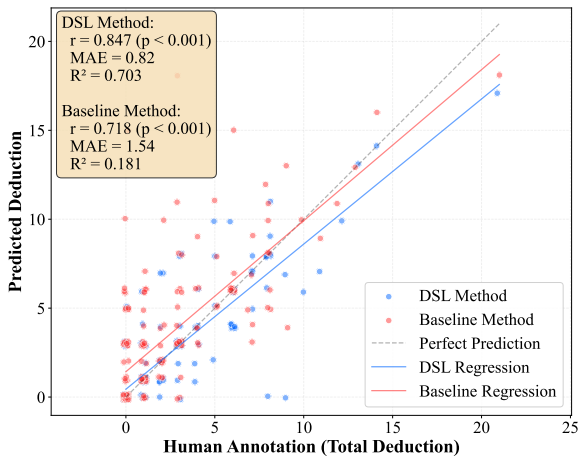

Figure 6: Predicted vs. human-annotated deductions for our method and Baseline.

C Case Studies

Goal: $\vdash P \to (Q \vee P)$
Natural Proof (excerpt):

(1) $\vdash (Q \to R) \to ((P \vee Q) \to (P \vee R))$
Ax. 4

(2) $\vdash (Q \to R) \to ((\neg P \vee Q) \to (\neg P \vee R))$
Subs. (1), $\frac{P}{\neg P}$

(3) $\vdash (Q \to R) \to ((P \to Q) \to (P \to R))$
Repl. (2), $(\neg P \vee Q) \Rightarrow (P \to Q)$

(4) $\vdash ((P \vee Q) \to (Q \vee P))$
$\to ((P \to (P \vee Q)) \to (P \to (Q \vee P)))$
Subs. (3), $\frac{Q}{P \vee Q}$, $\frac{R}{Q \vee P}$

(5) $\vdash (P \vee Q) \to (Q \vee P)$
Ax. 3

(6) $\vdash (P \to (P \vee Q)) \to (P \to (Q \vee P))$
MP (5) (4)

(7) $\vdash P \to (P \vee Q)$
(typo) cited as "Theorem 2"; should be Ax. 2

(8) $\vdash P \to (Q \vee P)$
MP (7) (6)

Proof 2. Typo in Step (7): cites "Theorem 2" instead of "Axiom 2"; DSL did not penalize.

```
Case 2 (DSL worse than baseline)
Theorem ID: 1 | Submission ID: 29540

DSL Grading:
Total Deduction: 0 points
No deductions (perfect proof)

Baseline Grading:
Total Deduction: 1 point
Deduction:
- Location: Step 7
- Rule Type: typo
- Deduction: -1 points
- Reason: Step 7 references 'Theorem 2'
    instead of 'Axiom 2'.
        This is a typo in theorem/axiom
    naming.
```

Grade 2. DSL missed the typo; baseline deducted 1 point.

As DSL-based grading relies on the formalizer, when the formalizer incorrectly "corrects" an error in the proof or resolves an ambiguous part by filling in the correct content, the grader can easily overlook such inadvertently fixed mistakes. For example, in Proof 2, step (7) writes "Theorem 2" instead of "Axiom 2", but the formalizer, recognizing that the correct reference should be Axiom 2, translates it as such, thereby failing to preserve a faithful translation of the original submission. Currently, due to cost considerations, all steps are formalized in a single pass; performing multiple formalization passes would likely yield better results.

D LLM Prompts

Due to space constraints, only excerpts of the prompts are presented here. For the complete, unedited prompts and their implementation details, please refer to the supplementary material.

D.1 Formalization Prompt

System Prompt For Formalizer

You are an expert in formal logic and the Russell axiom system. Your task is to formalize natural language proof steps into structured proof actions.
The Russell axiom system uses the following operators:

- $\neg$ (negation)
- $\wedge$ (conjunction/and)
- $\vee$ (disjunction/or)
- $\rightarrow$ (implication)
- $\leftrightarrow$ (equivalence/if and only if)

The system has 4 axioms:

- ax1: $(P \vee P) \rightarrow P$ (Tautology/Idempotence)
- ax2: $P \rightarrow (P \vee Q)$ (Disjunction Introduction)
- ax3: $(P \vee Q) \rightarrow (Q \vee P)$ (Disjunction Commutativity)
- ax4: $(Q \rightarrow R) \rightarrow ((P \vee Q) \rightarrow (P \vee R))$ (Implication Distribution)

The system supports 3 transformation rules plus theorem introduction.
You must respond with ONLY a JSON object containing the formalized proof.

Figure 7: The full system prompt used for the formalization task.

User Prompt For Formalizer

Task: Formalize a Russell Axiom System Proof

We need to formalize a proof in the Russell axiom system to verify its correctness. You will be given a theorem statement and its natural language proof steps, and you need to identify atoms and formalize each step with appropriate parameters.

Theorem to Prove

Theorem Name: {theorem_name}
Formula: {theorem_formula}

Background Information

1. Pre-defined Atom Formulas

We have three pre-defined atom formulas: **P, Q, R**
The theorem statement has introduced the following additional atoms: {theorem_atoms if theorem_atoms else "None"}

2. Available Axioms and Lemmas

The following axioms and previously proved lemmas are available for use in proofs:
{theorems_section}

3. Transformation Rules

CRITICAL REQUIREMENTS - Faithful Translation:

1. **Preserve Original Step Descriptions**: ...
2. **Use Original Step Formula**: ...

We have 5 types of proof actions:

Action 1: Introduce (intro) **Purpose**: Introduce an axiom or previously proved theorem into the proof.
...
IMPORTANT - Matching Theorem Names:

1. **Match step description with available list**: Take the theorem name mentioned in the step description (e.g., "axiom 4", "theorem 3.2.1") and try to match it with the list of available axioms and lemmas provided above. Use the exact Lean name from the list (e.g., `ax1`, `ax2`, `ax3`, `ax4`).

   ...

2. **If no match found in the list**: If the step description mentions a theorem/axiom that is NOT in the available list, create a reasonable Lean-style identifier based on the step's description text. Follow these naming conventions:

   ...

Action 2: Substitute (substitute) **Purpose**: Apply substitution to a previous step's formula by replacing atom variables.
...

Action 3: Replace (replace) **Purpose**: Apply definition-based replacement to convert between equivalent forms.
...
IMPORTANT REQUIREMENTS:

1. **Each replacement targets ONE specific subformula**: Do NOT describe the transformation of the entire formula. Each `replacement` object should target one specific subformula that matches the definition pattern.

2. **Multiple locations = Multiple replacements**: If the same definition needs to be applied at multiple locations in the formula, create a SEPARATE replacement entry for each location.

3. **Identify A, B, and direction from the definition pattern**:
   - For Implication: `A → B ⟺ ¬A ∨ B`
     - Left side: `A → B`
     - Right side: `¬A ∨ B`
   - For Conjunction: `A ∧ B ⟺ ¬(¬A ∨ ¬B)`
     - Left side: `A ∧ B`
     - Right side: `¬(¬A ∨ ¬B)`
   - For Equivalence: `A ↔ B ⟺ (A → B) ∧ (B → A)`
     - Left side: `A ↔ B`
     - Right side: `(A → B) ∧ (B → A)`
   - Identify which side appears in the from_step formula and specify it with the `from` parameter
   - Do NOT include any surrounding context or operators that are not part of A or B themselves

4. **Find differences by comparison**: Compare the `from_step` formula with the current step formula to identify ALL locations that changed. Each changed location should have its own replacement entry.

Parameter Details:

- `from_step`: The step ID to apply replacement to, starting from 1, just refer to the step mentioned inside the step description. Use the last step if not specified.
- `A`: The first component in the definition pattern (e.g., for Implication, this is the antecedent)
- `B`: The second component in the definition pattern (e.g., for Implication, this is the consequent)
- `definition`: Which definition to use (string: "Implication", "Conjunction", "Equivalence")
- `from`: Which side of the definition appears in from_step formula - "left" for left side, "right" for right side
- `position`: Which occurrence of the pattern with these A and B values to replace (0-based, usually 0 for the first occurrence)

  ...

CRITICAL: Parentheses in A and B:

- When A or B contains a complex formula (e.g., conjunction, disjunction, implication), you MUST include the parentheses
- For example, if B is a conjunction like `(Q ∧ P)`, you must write `"B": "(Q ∧ P)"` with the parentheses
- If A or B is just a single atom (e.g., P, Q, R), no parentheses are needed
- Example: For `¬P ∨ (Q ∧ P) → P → (Q ∧ P)`, A="P" (no parentheses needed) and B="(Q ∧ P)" (parentheses required)

...

Action 4: Modus Ponens (mp) **Purpose**: Apply modus ponens rule: from `A` and `A → B`, derive `B`.
...

- `minor_step`: The step containing the antecedent (A), starting from 1
- `major_step`: The step containing the implication (A → B), starting from 1

Note: Natural language may not specify which is major/minor. You must examine the formulas from both steps and identify which one is the implication (major) and which is its antecedent (minor).

Action 5: No Operation (nop) **Purpose**: Placeholder for invalid proof steps that don't represent actual transformations, but have their index maintained for consistency.
...
Note: Use this for steps that cannot be formalized as intro, substitute, replace, or mp actions, or when a line is missing from the proof. The step will be marked as a placeholder and won't affect other steps (like their ids)

General Requirements

These requirements apply to all actions (intro, substitute, replace, mp):

1. **Invalid Step References**: If a natural language description references a step that cannot be resolved to a valid step number in the range [1, current_id-1], set it to **0** to indicate an invalid reference. Or if it doesn't explicitly point out the referenced step id in the substitute or replace action, use the last step if possible.
    - **Common case 1**: Attempting to reference an axiom or theorem directly in substitute, replace, or mp actions without first introducing it. For example, "substitute from axiom 1" when axiom 1 has not been introduced with an `intro` action and thus has no valid step id. In this case, set `from_step` to 0.
    - **Common case 2**: The description mentions the operation but doesn't specify which step to apply it to in the MP action. For example, when missing the major step id or the minor step id, use **0** for the missing step id.
    - **Common case 3**: For substitute and replace, use the last step id as the default value when `from_step` is not specified. For example, "substitute P with Q" without mentioning which step to substitute from, set `from_step` to the last step id. However, if the proof text explicitly mentions a step id (even if it's incorrect), use that exact id faithfully - do not change it to the last step id.

Your Task

Analyze the proof text below and:

1. **Identify New Atoms**: Scan the proof and identify any NEW atom formulas used beyond P, Q, R and the atoms already introduced in the theorem statement ({theorem_atoms}). List them in the `atoms` field. Only pick atom formulas, ¬P is not an atom.
2. **Split into Steps**: Break the proof into individual numbered steps. Each step should have:
   - Id (starting from 1)
   - A formula (the result of that step)
   - A natural language description of the transformation
3. **Formalize Each Step**: For each step, identify:
   - The type of action (intro, substitute, replace, mp)
   - The required parameters based on the action type
   - The formula this step produces
4. **Reasoning Process**: Before generating the final JSON, provide a step-by-step analysis showing:
   - How you identified the atom formulas
   - How you split the proof into steps
   - For each step: what action type it is, what parameters it needs, and why

Output Format

First, provide your reasoning process explaining how you analyzed the proof.
Then, respond with ONLY a JSON object wrapped in `json` markers, with these fields:
Important Formula Format Rules:

- Use only the FIVE operators: ¬, ∧, ∨, →, ↔
- Use parentheses for grouping
- DO NOT include the ⊢ symbol
- Example: "(P → Q) → ((Q → R) → (P → R))" not "⊢ (P → Q) → ((Q → R) → (P → R))"

Proof Text to Formalize

{proof_text}
Now please analyze this proof step by step and provide the formalized JSON output.

Figure 8: The user prompt used for the formalization task, details can be found in the supplementary codes.

D.2 Grading Prompt

System Prompt For Grader (with DSL)

You are an expert teaching assistant specializing in grading proofs in the Russell axiom system. Your task is to evaluate student proofs by matching detected errors to predefined grading rubric deduction points.
The Russell axiom system uses these logical operators:

- $\neg$ (negation)
- $\wedge$ (conjunction)
- $\vee$ (disjunction)
- $\rightarrow$ (implication)
- $\leftrightarrow$ (equivalence)

The system has 4 axioms:

- ax1: $(P \vee P) \rightarrow P$ (Tautology)
- ax2: $P \rightarrow (P \vee Q)$ (Disjunction Introduction)
- ax3: $(P \vee Q) \rightarrow (Q \vee P)$ (Disjunction Commutativity)
- ax4: $(Q \rightarrow R) \rightarrow ((P \vee Q) \rightarrow (P \vee R))$ (Implication Distribution)

And 3 definitions:

- Implication: $(A \rightarrow B) \iff (\neg A \vee B)$
- Conjunction: $(A \wedge B) \iff \neg(\neg A \vee \neg B)$
- Equivalence: $(A \leftrightarrow B) \iff ((A \rightarrow B) \wedge (B \rightarrow A))$

The system has 4 proof rules for constructing valid proofs:

1. **Introduction Rule**: An axiom or previously proved theorem can be directly introduced as a step in the proof. The introduced formula is the conclusion of that axiom/theorem.
2. **Substitution Rule**: If a formula is provable, then replacing one or more atomic variables (must be atomic, not compound formulas) in it with arbitrary formulas yields a formula that is also provable. For example, from $P \rightarrow (P \vee Q)$, substituting P with $\neg R$ gives $\neg R \rightarrow (\neg R \vee Q)$.
3. **Replacement Rule**: If a formula is provable, then applying one of the three definitions to replace certain substructures with their equivalent forms yields a formula that is also provable. For example, using the implication definition, $(\neg P \vee Q)$ can be converted to $(P \rightarrow Q)$.
4. **Modus Ponens Rule** If A is provable and $(A \rightarrow B)$ is provable, then B is provable. This is the fundamental inference rule for deriving new conclusions.

When evaluating proofs, you should carefully analyze the errors and match each one to the most appropriate grading rule from the provided rubric.

Figure 9: The full system prompt used for the grading task (with DSL).

User Prompt For Grader (with DSL)

Task: Grade a Russell Axiom System Proof

You will be provided with:

- **Theorem statement** to be proven
- **Natural language proof** written by a student
- **DSL-formatted proof steps** (Domain Specific Language above Lean 4)
- **Error details** detected by the formal verification system (as reference)
- **Grading rules** for deduction

Your task is to analyze each step of the proof, using the DSL translation equivalence and provided reference errors, identify all step-level errors and final global conclusion errors, and match them to the most appropriate grading rules.

Background

Student proofs are formalized into the LEAN verification system through an intermediate DSL (Domain Specific Language). Proofs that pass verification deserve full scores. When errors occur, you must analyze them and assign appropriate deductions based on the grading rubric.
Below are the essential elements you need to understand:

1. Available Axioms and Lemmas

The following axioms and previously proved lemmas are available for use in proofs:
{theorems_section}

2. Available Proof Actions

We will present you with **DSL code** instead of raw LEAN code. It contains 5 types of proof actions:

Action 1: Introduce Axiom/Theorem ...

Action 2: Substitute ...

Action 3: Replace ...

Action 4: Modus Ponens ...

Action 5: No Operation (nop) ...
...

3. Formalization

When translating natural language proofs into DSL, the formalization process may not be completely equivalent to the original text in the following cases:

1. **Invalid or Missing Parameters in Original Proof**: ...
2. **Translation Errors**: ...

4. Error Types

The LEAN system generates errors that are parsed into human-readable format by the DSL. The main error categories include:

- **STEP_STANDARD**: Errors related to proof steps, including invalid parameters for each action type (intro, substitute, replace, mp), or cases where the derived formula differs from the expected goal.
- **STEP_PROOF_GOAL**: The step's result does not match its intended goal.
- **STEP_UNKNOWN**: Unclassified step errors.
- **STEP_NOP**: The step is a placeholder with no valid proof transformation.
- **GOAL_MATCH**: The final proof result does not match the theorem's goal.
- **GOAL_UNKNOWN**: Unclassified goal errors.
- **Atom/Statement/Header Errors**: Errors in atom variable declarations, theorem statements, or file headers. These may indicate typos (e.g., syntactically incorrect formulas in a step) or system anomalies, and should be distinguished accordingly.

...

5. Grading Principles

Our goal is to identify all errors in the proof and match each to an appropriate grading rule category. Not all errors may have a corresponding rule - only include errors that match a grading rule in your output. For each identified error, follow these principles:

1. **Distinguish Typos from Fake Proofs**: ...
2. **Avoid Error Propagation**: ...
3. **Cases for Unmatched Errors (DSL-based method specific)**: ...

Your Task

Analyze the provided **whole result**, which includes: the theorem statement, the student's natural language proof, the DSL translated steps, and the error details.
Analyze the proof step by step:

1. **Compare Original Proof with DSL Translation**: ...
2. **Analyze Errors for Matched Steps**: ...
3. **Check Final Conclusion**: ...
4. **Important Notes**: ...

Output Format

...

Grading Rule Types

Here are the actual categories of student errors used for deductions:
{rules_section}

Whole Result

{theorem_formatted_result}

Figure 10: The user prompt used for the grading task (with DSL), details can be found in the supplementary codes.

System Prompt For Grader (without DSL)

You are an expert teaching assistant specializing in grading proofs in the Russell axiom system. Your task is to evaluate student proofs by matching detected errors to predefined grading rubric deduction points.
The Russell axiom system uses these logical operators:

- $\neg$ (negation)
- $\wedge$ (conjunction)
- $\vee$ (disjunction)
- $\rightarrow$ (implication)
- $\leftrightarrow$ (equivalence)

The system has 4 axioms:

- ax1: $(P \vee P) \rightarrow P$ (Tautology)
- ax2: $P \rightarrow (P \vee Q)$ (Disjunction Introduction)
- ax3: $(P \vee Q) \rightarrow (Q \vee P)$ (Disjunction Commutativity)
- ax4: $(Q \rightarrow R) \rightarrow ((P \vee Q) \rightarrow (P \vee R))$ (Implication Distribution)

And 3 definitions:

- Implication: $(A \rightarrow B) \iff (\neg A \vee B)$
- Conjunction: $(A \wedge B) \iff \neg(\neg A \vee \neg B)$
- Equivalence: $(A \leftrightarrow B) \iff ((A \rightarrow B) \wedge (B \rightarrow A))$

The system has 4 proof rules for constructing valid proofs:

1. **Introduction Rule**: An axiom or previously proved theorem can be directly introduced as a step in the proof. The introduced formula is the conclusion of that axiom/theorem.
2. **Substitution Rule**: If a formula is provable, then replacing one or more atomic variables (must be atomic, not compound formulas) in it with arbitrary formulas yields a formula that is also provable. For example, from $P \rightarrow (P \vee Q)$, substituting P with $\neg R$ gives $\neg R \rightarrow (\neg R \vee Q)$.
3. **Replacement Rule**: If a formula is provable, then applying one of the three definitions to replace certain substructures with their equivalent forms yields a formula that is also provable. For example, using the implication definition, $(\neg P \vee Q)$ can be converted to $(P \rightarrow Q)$.
4. **Modus Ponens Rule** If A is provable and $(A \rightarrow B)$ is provable, then B is provable. This is the fundamental inference rule for deriving new conclusions.

When evaluating proofs, you should carefully analyze each step to identify errors and match them to the most appropriate grading rule from the provided rubric.

Figure 11: The full system prompt used for the grading task (without DSL).

User Prompt For Grader (without DSL)

Task: Grade a Russell Axiom System Proof

You will be provided with:

- **Theorem statement** to be proven
- **Natural language proof** written by a student
- **Grading rules** for deduction

Your task is to analyze the proof, identify all errors, and match each one to the most appropriate grading rule.

Background

Student proofs should correctly apply the Russell axiom system rules to derive the theorem statement. You will evaluate the proof by checking each step for errors and assigning appropriate deductions based on the grading rubric.
Below are the essential elements you need to understand:

1. Available Axioms and Lemmas

The following axioms and previously proved lemmas are available for use in proofs:
{theorems_section}

2. Proof Step Patterns

Students write proofs as numbered steps in natural language. Valid types include:

Introduce Axiom/Theorem ...

Substitution ...

Replacement ...

Modus Ponens ...

3. Error Types

When analyzing the proof, you should identify errors in two categories:

Step-Related Errors (with step_index) These errors occur at specific proof steps:

1. **Invalid Step References**: A substitute, replace, or mp operation doesn't reference a valid prior step (e.g., references a step that doesn't exist yet, or doesn't specify which step to use).
2. **Unknown Theorem**: An intro step references an axiom or theorem that is not in the available list.
3. **Invalid Parameters**: Other parameter errors such as attempting to substitute non-atomic formulas, incorrect substitution notation, etc.
4. **Other Rule Violations**: Any errors defined in the grading rules (e.g., directly using axioms without introduction, misapplication of definitions, illegal transformations).
5. **Formula Mismatch**: The result of the operation described in the step doesn't match the formula written in that step.

Global Errors (step_index is null) These errors are not tied to a specific step:

- **Proof Incomplete**: The final conclusion of the proof does not match the theorem's target formula.

4. Grading Principles

Our goal is to identify all errors in the proof and match each to an appropriate grading rule category. Not all errors may have a corresponding rule - only include errors that match a grading rule in your output. For each identified error, follow these principles:

1. **Distinguish Typos from Fake Proofs**: ...
2. **Avoid Error Propagation**: ...

Your Task

Analyze the provided proof step by step:

1. **Examine Each Step Sequentially**: ...
2. **Check Final Conclusion**: ...
3. **Important Notes**: ...

Output Format

First, provide your reasoning process showing step-by-step analysis of each proof step and any errors found.
Then, respond with ONLY a JSON object wrapped in `json` markers, with these fields:
...

Grading Rule Types

Here are the actual categories of student errors used for deductions:
{rules_section}

Whole Result

{whole_result}
...

Standardized Proof Text

This is the proof text cleaned up and standardized according to the standard step format rules. It can be used as a reference to help understand the student's intent when the original proof has formatting issues or omitted parameters.
Note: This standardized version is for reference only. You should still grade based on the original proof text shown in the "Whole Result" section above, as it represents what the student actually wrote.
...

Figure 12: The user prompt used for the grading task (without DSL), details can be found in the supplementary codes.

Error Deduction Rules JSON


```
[
    {
        "type": "typo",
        "description": "Typo/Symbol Error with correct underlying proof:
            Errors that affect the meaning of the proof, such as missing
            necessary parentheses, incorrect axiom/theorem numbering or naming
            , wrong step numbers in operations (substitution, modus ponens,
            replacement). For omitted predecessor step numbers: if it's a
            substitution or replacement operation and the omitted step is the
            immediately previous step, it's NOT considered a typo; however,
            for modus ponens operations, step numbers must be explicitly
            provided, and omission is considered a typo. Note: Text formatting
             errors that do NOT affect mathematical meaning (e.g., missing
            step numbering in front of a step, missing proof symbol \vdash)
            should NOT be penalized as typos.",
        "deduction": 1
    },
    {
        "type": "theorem_not_introduced",
        "description": "Available Axiom/Lemma Not Introduced: Directly
            referencing provided axioms or available lemmas in operations (
            substitution, replacement, modus ponens) without first introducing
             them as a separate step. For example, writing "apply modus ponens
             with axiom 4 and step (3)" or "substitute from axiom 2" without
            first writing a line to introduce that axiom. Note: This rule
            applies only to axioms/lemmas that ARE provided in the problem. If
             the proof references a theorem that is NOT provided (e.g.,
            external theorems, unproven lemmas), it should be classified as
            external_theorem_reference instead.",
        "deduction": 1
    },
    {
        "type": "introduction_mismatch",
        "description": "Introduction Formula Mismatch: The intro operation
            produces a formula that differs from the original axiom/lemma.
            Intro must output the exact original formula without any
            modifications (no substitution, replacement, or other
            transformations). If modifications are needed, first introduce the
             original formula, then apply transformations in separate steps.
            Example: Axiom 2 "P \rightarrow (P \lor Q)" cannot be directly
            introduced as "A \rightarrow (A \lor B)".",
        "deduction": 1
    },
    {
        "type": "external_theorem_reference",
        "description": "External Theorem Reference: Explicitly citing or
            introducing theorem names/numbers NOT provided in the problem (e.g
            ., "by Theorem 3.2.1", "according to associative law", "introduce
            De Morgan's law"). The theorem name must be explicitly mentioned
            in an intro operation or referenced in other operations (
            substitute, replacement, modus ponens). Key distinctions: (1) vs
            illegal_transformation - this rule requires explicit theorem
            citation; if using unprovable transformations without citing
            sources, use illegal_transformation instead; (2) vs
            theorem_not_introduced/introduction_mismatch - those apply to
            PROVIDED axioms/lemmas used incorrectly, while this applies to
            UNPROVIDED theorems.",
        "deduction": 3
    },
    {
        "type": "substitution_params_mismatch",
        "description": "Substitution Parameter Error: Errors in substitution
            parameters including: (1) Invalid substitutions - referencing
            atomic formulas that don't exist in the source formula; (2)
```

```
            Incorrect parameter order - writing A/P instead of P/A when
            intending to substitute P with A. Note: Distinguish from typo -
            missing one mapping is a parameter error, but if a single
            substitution has an obvious typo in the formula (e.g., writing Q
            instead of R), that's a typo.",
        "deduction": 1
    },
    {
        "type": "illegal_substitution",
        "description": "Illegal Substitution Rule: Attempting to substitute
            FOR a non-atomic formula. The substitution rule only allows
            substituting FOR atomic variables - you can only substitute an
            atomic variable (P, Q, R, etc.) with some other formula, but you
            cannot substitute a compound formula like (\neg P) or (A\
            rightarrowB) with something else. In substitution notation P/A (or
             \\frac{P}{A}), what is being substituted (left side P) must be
            atomic, while what it becomes (right side A) can be any formula (
            atomic or compound). For example, 'P/(\neg Q\lorR)' (substitute
            atomic P with compound formula) is legal, but '(\neg P)/Q' (
            substitute compound \neg P with atomic Q) or '(A\rightarrowB)/C' (
            substitute compound A\rightarrowB with C) are illegal because the
            left side is not atomic.",
        "deduction": 5
    },
    {
        "type": "illegal_mp",
        "description": "Modus Ponens Error: Incorrect application of MP rule
            where the major premise is not an implication, or the minor
            premise doesn't match the antecedent of the major premise, and it'
            s not a typo (no correct major or minor premise is available).",
        "deduction": 3
    },
    {
        "type": "illegal_replacement",
        "description": "Illegal Replacement: The specified definition number
            cannot be applied to transform the given formula. This means the
            replacement operation itself is invalid because the definition
            pattern does not match the formula structure. Key distinction from
             step_goal: if the replacement operation can be performed but
            yields a different result than the stated goal, it should be
            classified as step_goal instead.",
        "deduction": 3
    },
    {
        "type": "step_goal",
        "description": "Step Goal Mismatch: The described operation (
            introduction, substitution, replacement, or modus ponens) is
            correctly performed with valid parameters (no typos), but the
            resulting conclusion differs from this step's goal. The operation
            itself is legal but cannot achieve the intended result. For
            example, the specified definition cannot convert the original
            formula to the target formula.",
        "deduction": 3
    },
    {
        "type": "illegal_transformation",
        "description": "Illegal Transformation: Using unprovable logical
            transformations implicitly without citing theorem sources. This
            includes applying non-Russell system rules (e.g., De Morgan's laws
            , transitivity, double negation) or unproven transformations
            without explicit justification. Key distinction from
            external_theorem_reference: this rule applies when transformations
             are made without citing any source; if a step explicitly cites a
            theorem name/number (e.g., "by Theorem 3.2.1"), it should be
            classified as external_theorem_reference instead.",
        "deduction": 5
```

```
    },
    {
        "type": "proof_unfinished",
        "description": "Proof Unfinished: The proof is incomplete - the final
            step's conclusion does not match the theorem's required goal
            formula. This error applies when the last step, even if all other
            errors were corrected, still would not yield the target conclusion
            . However, if the last step has an error (e.g., typo or incorrect
            operation) that, once fixed, would produce the correct conclusion,
             then only the step error should be reported, not this global
            error. Important: This error is independent of errors in previous
            steps - if the final step's conclusion matches the target formula,
             do NOT report this error, regardless of whether there are errors
            in earlier steps.",
        "deduction": 5
    }
]
```

Figure 13: The error deduction rules.

E Use of AI Assistants

We acknowledge the use of AI assistants (e.g., Gemini, Claude Code) to facilitate code implementation, refine textual presentation, and assist in the preliminary analysis of partial experimental results. Crucially, all AI-generated content, including code snippets, statistical interpretations, and polished text—was subjected to rigorous human verification. The authors reviewed all outputs to ensure accuracy, scientific integrity, and alignment with the paper's logical flow.