Towards Reliable Proof Generation with LLMs: A Neuro-Symbolic Approach

Anonymous ACL submission

Abstract

Large language models (LLMs) struggle with formal domains that require rigorous logical deduction and symbolic reasoning, such as mathematical proof generation. We propose a neuro-symbolic approach that combines LLMs' generative strengths with structured components to overcome this challenge. As a proof-of-concept, we focus on geometry problems. Our approach is two-fold: (1) We retrieve *analogous problems* and use their proofs to guide the LLM, and (2) a *formal verifier* evaluates the generated proofs and provides feedback, helping the model fix incorrect proofs.

We demonstrate that our method significantly improves proof accuracy for OpenAI's o1 model (58%-70% improvement); both analogous problems and the verifier's feedback contribute to these gains. More broadly, shifting to LLMs that generate provably correct conclusions could dramatically improve their reliability, accuracy and consistency, unlocking complex tasks and critical real-world applications that require trustworthiness.

1 Introduction

Despite their remarkable performance across a wide range of tasks, LLMs still struggle in formal domains such as mathematical proofs. This stems primarily from their inherent architecture, which relies on probabilistic sequence generation based on patterns learned from vast textual datasets.

Mathematical proofs demand rigorous logical deduction, symbolic manipulation, and an understanding of abstract concepts that go beyond statistical correlations in language. The requirement for absolute truth and the absence of ambiguity in mathematical reasoning present a significant challenge for models trained to generate *plausible* text rather than formally valid inferences (Singh et al., 2024; Pan et al., 2025).

In addition, the often lengthy nature of proofs necessitates a level of sustained logical coherence and hierarchical reasoning that is hard for current LLMs. Recent work has shown that altering even superficial aspects of mathematical problems results in significant performance drops (Mirzadeh et al., 2024), suggesting that their success often hinges on pattern matching rather than genuine mathematical reasoning.

Enabling LLMs to generate rigorous and verifiable proofs can dramatically boost LLMs' reliability, accuracy and consistency, unlocking applications in mathematics, science and education (Welleck et al., 2022; Gupta et al., 2025; Kumar et al., 2023), as well as many safety-critical and security-critical tasks.

In this work, we introduce a neuro-symbolic approach that combines the generative strengths of LLMs with two complementary structured components: (1) **analogical guidance** and (2) **symbolic verification**. See Figure 1 for an illustration.

The first component retrieves analogous problems and their proofs to guide the model. This is inspired by both cognitive science, where analogy is recognized as a fundamental mechanism underlying human problem-solving and generalization (Gentner, 1983; Holyoak and Thagard, 1996), and by a recent work showing that when LLMs solve grade-school math word problems, asking them to think of analogous problems and their solutions can significantly improve performance (Yasunaga et al., 2023). In our setting, proofs for analogous problems both provide a better starting point for constructing a new proof, and also allow us to identify the most relevant theorems to show the model, substantially reducing costs.

To complement this, our second component employs a symbolic verifier that checks the generated proofs and provides structured feedback. This feedback drives an iterative loop, enabling the model to revise its output until a valid proof is obtained.

As a proof of concept, we focus on Euclidean geometry – a domain that is symbolic, verifiable,

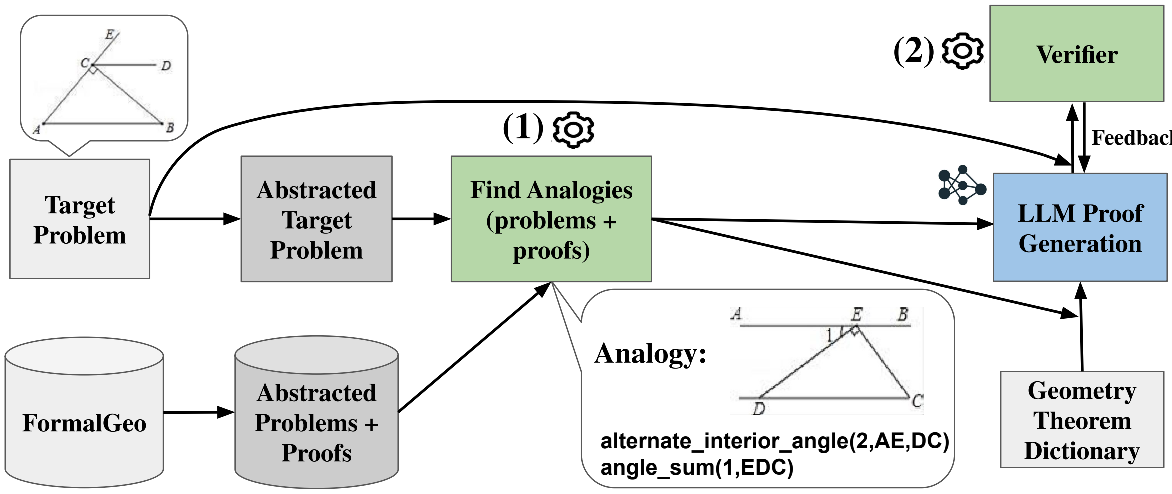

Figure 1: **Our neuro-symbolic approach.** Given a target problem from the FormalGeo-7k dataset, we first convert it into an *abstract form* by replacing entity names (e.g., lines, angles) and specific numeric values with placeholders (§3.1). We then *retrieve structurally similar problems* from the abstracted dataset by computing Jaccard similarity over key formal components: **construction** (entities and geometric relations), **conditions** (e.g., angle equalities, segment lengths), and **goal** (the conclusion to be proven). This is based on the observation that structurally similar problems often share proof patterns (§3.2). The retrieved problems, along with their corresponding formal proofs, are presented to an LLM as *in-context examples*, together with the available theorems from the Geometry Theorem Dictionary, to guide *proof generation* for the target problem (§3.3). Finally, a *symbolic verifier* iteratively checks the generated proof and provides feedback until a correct proof is produced or a retry limit is reached (§3.4).

and rich in structural analogies. Note that our primary objective in this paper is to evaluate whether our symbolic augmentations can improve the proof-generation capabilities of *general-purpose LLMs*. As such, while geometry is the domain we test our ideas in, our focus is *not* on competing with state-of-the-art, specialized geometry solvers. Rather, we wish to quantify the gains enabled by our method. **Our main contributions are:**

- We propose a neuro-symbolic system that aids LLMs in proof generation by providing analogical guidance and verification feedback.
- We design a symbolic verifier tailored to geometry proofs with expressive feedback. In contrast to other works, we evaluate the **entire proof**, not just the final numeric answer.
- Our method significantly improves proof accuracy, achieving gains of 58%–70% over OpenAI's o1 model. Both analogical guidance and the verifier contribute to performance.
- Our method reduces costs via focused context construction, reducing the theorem dictionary from 18K to just 2.5K tokens on average.
- We will release code and data, including the evaluation scripts and processed data used in our experiments[1].

[1]URL redacted, will be available upon publication.

2 Problem Formulation

We demonstrate our ideas in the domain of Euclidean geometry. Our input is a geometry problem, described in both natural language and via a formal representation. The description includes the geometric entities involved (e.g., lines, angles), their relationships (e.g., perpendicular, collinear), and measurements or algebraic expressions over them. We also receive a *goal*, some quantity to be determined (e.g., the length of a line). In addition, the model has access to a dictionary of theorems that may be used in the proof.

The output is a formal proof that derives the goal from the given conditions and theorems, along with the final, numeric answer. The proof consists of steps, each applying a specific theorem from the dictionary. See Figure 2 for an example.

3 Approach

Our goal is to develop a system that assists LLMs in proof generation. As a proof of concept, we focus on geometry problems from the FormalGeo-7k dataset (Zhang et al., 2023), containing 6,981 SAT-level Euclidean geometry problems[2].

[2]Available at `https://github.com/FormalGeo/FormalGeo` under the MIT License. Used here solely for non-commercial academic research.

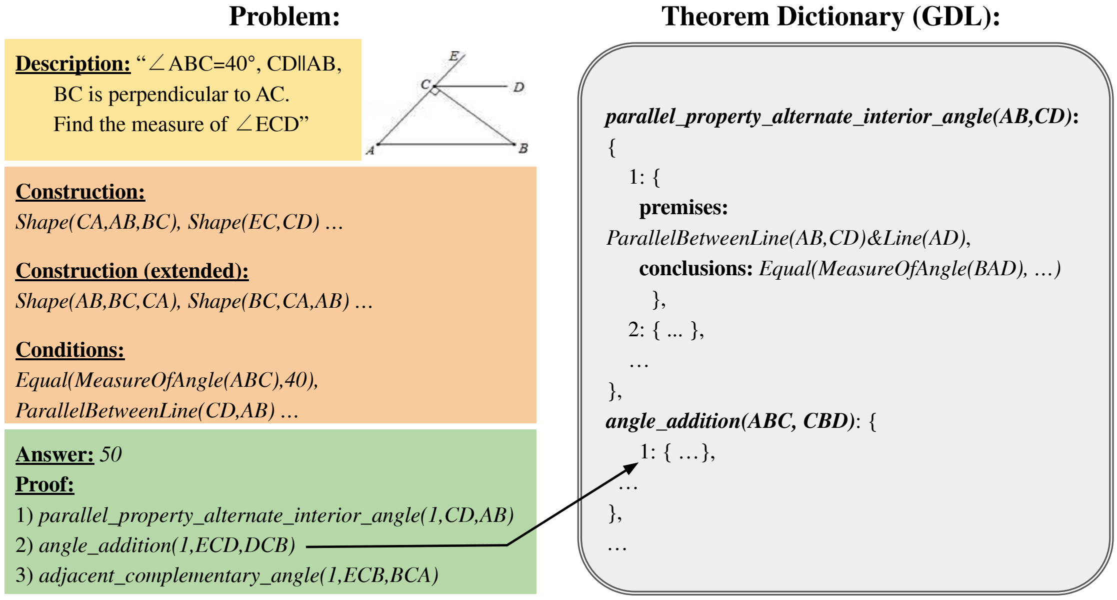

Figure 2: An example problem from FormalGeo-7k. **Left**: Problem inputs, including a natural language description and formal representations – construction (entities and relations), extended construction (inferred based on extension rules), conditions, and goal. The input also includes a numeric answer (or expression) and a formal proof, composed of steps, each invoking a theorem from the dictionary. **Right**: The Theorem Dictionary. Theorems include variation id, arguments, premises and conclusions. The full dataset also includes diagrams, but we do not use them, as a recent work (Zhang et al., 2024) showed that multimodal LLMs often struggle with the visual aspects of math problems.

Our approach is two-fold (see Figure 1). Given a target problem, we first build on the insight that structurally similar problems often admit similar proofs. We abstract all problems in the dataset (§3.1), retrieve **analogous problems** (similar to the target on an abstract level, §3.2), and use the top-ranked analogies and their corresponding proofs as few-shot examples (§3.3). Next, we employ a **symbolic verifier** that iteratively provides feedback on the validity of the generated proofs (§3.4).

3.1 Problem Abstraction

Analogous problems share a similar underlying structure, but could differ on surface-level details such as entity names or measurements. To identify such problems, the pipeline first abstracts the target problem and all problems in the dataset. We chose a very simple abstraction schema: entity names are replaced with "<word>" and numbers with "<num>". For example, "Equal(MeasureOfAngle(ABC),40)" becomes "Equal(MeasureOfAngle(<word>), <num>)". This process is applied to the formal representations of construction, conditions, and goal. Exploring more nuanced abstraction schemas (e.g., ones that keep information about shared symbols between words) is left for future work.

3.2 Analogous Problems Retrieval

Our goal in this section is to retrieve problems whose (known) proof is similar to the (unknown) target proof. We conjecture that providing the model with these proof examples in context will improve its ability to generate a correct proof.

Our underlying working hypothesis is that *similar problems often yield similar proofs*. More concretely, we posit we can identify problems with potentially useful proofs by finding analogous problems (i.e., problems which are structurally similar to the target problem). To do this, we train a regressor to predict proof similarity between two problems, based on their structural similarity.

Dataset and Training. For each pair of abstracted problems, we compute the Jaccard similarity over the multi-sets of *construction* and *condition* representations. We define *goal* similarity as 1 if the goals match exactly and 0 otherwise. These three features form the input to our model. Abstract proof similarity, computed again via Jaccard similarity, serves as the label.

The dataset contains $\sim$ 24.4M problem pairs (all combinations of 6,981 problems). Proof similarity scores are binned into five intervals of width 0.2. As the distribution is highly imbalanced, with 88%

k	Analogy		Random	
	Coverage ↑	Theorems ↓	Coverage ↑	Theorems ↓
20	88%	11.06	62%	32.92
50	93%	18.57	79%	49.87
100	96%	26.66	88%	66.98
150	98%	31.96	92%	82.82

Table 1: Average number of problems whose entire proof was covered by the union of theorems from their top-k analogies vs. random problems, over 100 random target problems (20 per level, 1–5). Analogies consistently achieve higher coverage despite fewer theorems.

of pairs in the first bin (0-0.2), we down-sample all bins to the size of the smallest (131K, 0.8-1), yielding a balanced dataset of 655K pairs. We split the data into 90% training and 10% evaluation.

We train a simple three-layer neural network using mean squared error (MSE) loss and the Adam optimizer, with a learning rate of 0.001, batch size of 32, and 10 epochs. See Appendix A for details.

Evaluation. Note that for our use case, we are only interested in whether we can predict very high proof similarity. We select all pairs with predicted similarity above 0.95 and measure the fraction whose ground-truth proof similarity is in the top two bins. While only 1.28% of pairs exceed this threshold, 71% of our predictions do.

We are encouraged by the results, which confirm that our regressor can identify proofs similar to the (unknown) target proof, based on the target problem's description alone.

3.3 LLM Proof Generation

We use a few-shot prompt (in-context learning) that begins with the target problem, including its textual description, construction, conditions and goal. This is followed by the top analogous problems selected using the regressor from Section 3.2, along with their full proofs and numeric answers. Additionally, the LLM is given a theorem dictionary, which defines geometry theorems in terms of formal premises and conclusions (see Figure 2). The LLM's task is to generate a correct proof, using only the theorems from the dictionary. See Appendix B for the prompt.

One challenge we encountered is that the full theorem dictionary contains 196 theorems (234 including variations), resulting in a large token count and, consequently, high costs. This also limits scalability, as adding more theorems would further increase the input size.

To address this problem, we propose a more efficient approach. We have just shown that analogous problems tend to have similar proofs; thus, we test whether we could similarly narrow down the dictionary to include only *theorems used in analogous proofs*. This reduction not only reduces costs, but also narrows the search space, helping the model focus on more relevant theorems.

To evaluate the effectiveness of this approach, we measure the extent to which the narrowed-down dictionary still captures all the theorems needed for the proof of the target problem. We sample 100 problems (20 from each level 1-5) and evaluate how many of their target proofs are completely covered using theorems from their top-k similar problems (retrieved with the regressor of Section 3.2), and compare to a random set of k problems.

Note that even if a proof of a problem includes a theorem not present in its analogous set, it might still be possible to construct a valid proof using only the covered theorems. Therefore, our coverage metric is a conservative estimate of effectiveness.

See Table 1 for results. Theorems from analogous problems consistently outperform the baseline of theorems from random k problems, providing higher coverage despite lower number of theorems. A larger k increases coverage but also expands the input; we find that $k = 100$ offers a good trade-off, achieving coverage of 96% with only 26.66 theorems on average, that is 13.6% of the full dictionary. Exploring other values of k is left for future work.

3.4 Verifier Iterative Feedback

We next endow the LLM with an external verifier that can guide it through a feedback loop. This is inspired by similar efforts showing that code-writing LLMs can improve with iterative corrections (Peng et al., 2025; Palavalli and Santolucito, 2024). After each attempt to produce a proof, the verifier provides natural-language feedback, specifying the first error found in the proof. The LLM incorporates it into its context for the next iteration. The loop continues until the proof is verified or the maximum number of retries is reached (see Section 4).

The verifier is a symbolic reasoning system capable of performing both formal logic checks and algebraic reasoning. Internally, it encodes proof steps and geometric constraints as logical formulas and algebraic expressions, and evaluates them using satisfiability modulo theories (SMT).

Importantly, the verifier does not know the solution. Instead, it assesses whether the numerical answer is entailed by the constraints imposed by the proof. If the proof is valid and the answer can indeed be inferred from it, the proof is accepted.

Error tiers. To analyze where the LLM struggles, we define three verifier-identified error tiers:

1. **Theorem call syntax violation**: Syntax errors. Common issues include undefined theorems or incorrect argument signatures.
2. **Premise violation**: Theorem calls that rely on premises that have not been derived from the problem description or preceding proof steps. Feedback identifies the missing premise and lists all premises derived so far.
3. **Goal not reached**: The proof contains no errors but does not reach the goal. Feedback indicates that either (1) the proof is underconstrained and multiple solutions exist, or (2) that the (unique) solution derived by the verifier differs from the LLM's answer.

See Appendix C for examples of error messages from the different tiers.

Implementation details. To identify tier-1 errors, we extended the implementation of Zhang et al. (2023). To identify tier-2 and tier-3 errors we use the Z3 Theorem Prover (De Moura and Bjørner, 2008), a state-of-the-art SMT solver, that encodes algebraic constraints derived from geometric properties and verifies their logical consistency.

We also augment Z3 with symbolic workarounds for trigonometric functions it does not natively support. See Appendix C for more details.

4 Experimental Setup

We evaluate the performance of our method on the FormalGeo-7K dataset. Our main research questions are as follows:

RQ1: Does our method lead to a higher rate of correct proofs generated by the LLM?

RQ2 (Ablation): What is the individual contribution of different components in our method?

> **A Note on Baselines for Comparison.** Our main goal is to assess whether our symbolic augmentations (analogy guidance and verification) can improve the ability of *existing LLMs* to generate formal proofs. Consequently, our focus is *not* on competing with specialized geometry solvers, but on quantifying the gains introduced by our method.

We now share details on our experimental setup.

Input problems. We randomly sample 10 problems for each difficulty level from 1 to 5 from the FormalGeo-7K dataset, resulting in a total of 50 problems. Levels correspond to the number of steps in the ground-truth proof.

Base model. We use OpenAI's o1 model with the default parameters as our base model. o1 is a state-of-the-art LLM known for its capabilities in mathematical reasoning, proof generation, and complex problem-solving (Jaech et al., 2024). We experimented with several models, including GPT-4, GPT-4o-mini, and GPT-4o (Achiam et al., 2023; Hurst et al., 2024), and observed that these models tend to produce proofs with a significant number of syntax errors. o1 consistently outperformed them, making it the strongest baseline for our task. We leave the evaluation of additional base models for future work.

Few-shot variants.

- **Analogy-based:** We provide k similar (analogous) problems and their proofs as few-shot examples, along with an abridged theorem dictionary which includes only theorems from the proofs of similar problems (Section 3.3).
- **Base model (non-analogy):** We provide k random problems and their proofs as few-shot examples, along with the complete dictionary.

A note about k. In our preliminary exploration, no model was able to correctly follow the proof format with zero-shot prompts (unsurprisingly). We have found that $k = 5$ works well in practice; testing the effect of k in more depth is left for future work.

Feedback iterations and multiple runs. As outlined in Section 3.4, we integrate the LLM into a *feedback loop* guided by a verifier, allowing the LLM up to five iterations (retries following feedback) per run. In addition, since LLMs sometimes get stuck (even with feedback), we allow starting from scratch up to three times per problem (i.e., three separate runs, with the default – and unchangeable – temperature parameter set to 1).

Metric. We report the fraction of problems for which the model produces a correct proof, for the following settings:

- **First run, no retries (%):** The model produces a correct proof on its first attempt.
- **First run, with retries (%):** Any of m retries (following feedback) in the first run succeeds.
- **Multiple runs, with retries (%):** Any attempt across n independent runs (each with up to m retries) succeeds.

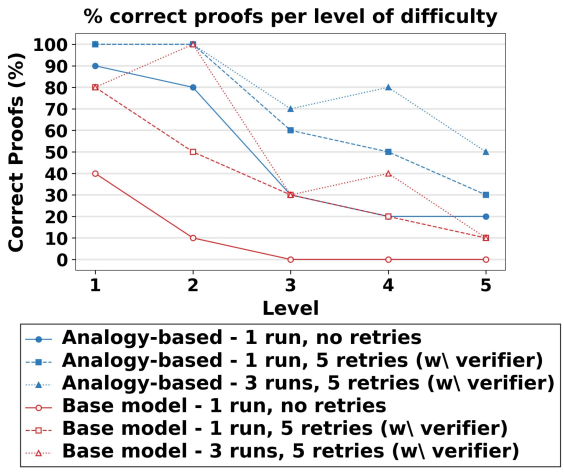

Figure 3: % correct proofs per level of difficulty (50 samples, 10 per level). Our analogy-based method outperforms the o1 base model (non-analogy) in all settings. Analogy retrieval, verifier feedback, and multiple runs each significantly contributes to performance. Our full pipeline (blue triangle) outperforms the baseline in every level, reaching an average aggregated accuracy of 80%. Even without multiple runs (blue square), performance remains strong at 68%, far exceeding the 10% of the base model baseline (red hollow circle).

Of course, more runs and retries increase costs; for our use case, we found that allowing a maximum of $m = 5$ and $n = 3$ offers a good tradeoff between performance and budget. We assess the effect of retries and runs in the next section.

5 Results

Figure 3 shows the results for the different settings.

5.1 RQ1: Performance gains of our approach.

We start by comparing our full pipeline (blue triangle: analogy + verifier, multiple runs with retries), to the most basic baseline (red hollow circle: non-analogy, no verifier, first run, no retries). Our full pipeline consistently outperforms the baseline at every level, achieving an average aggregated accuracy of 80%. In contrast, the base model achieves an average of only 10% (with 0% accuracy on levels 3 and above). This difference is statistically significant with p =5.82e-11 in the McNemar test at the 0.05 level (our method solved all problems the base model did, plus 35 others).

To isolate the contribution of multiple runs, we also evaluated our model after a single run (blue square). This setting also substantially outperformed the base-model baseline, achieving an overall accuracy of 68%, with p =3.73e-09 in the McNemar test at the 0.05 level. Thus, we conclude that our method significantly boosts the base model's success rate in generating correct proofs, even without multiple runs.

5.2 RQ2: Ablations.

We now assess the contribution of the different components of our pipeline: The analogy retrieval, the verifier, and the multiple runs.

Analogy retrieval. To measure the effect of analogy retrieval, we compare our method to the base model (non-analogy), under the same three settings: (1) first run, no retries, (2) first run, with retries (verifier), (3) multiple runs, with retries (verifier).

In setting (1) our method achieved an overall accuracy of 48%, substantially outperforming the baseline's 10% (blue circle vs. red hollow circle), p =3.81e-06 in the McNemar test at the 0.05 level. In setting (2), our method achieved 68% overall accuracy, while the base model reaches 38% (blue square vs. red hollow square), p =6.10e-05 in the McNemar test at the 0.05 level. In setting (3) we observe a 80% overall accuracy for our method vs. 52% for the base model (blue triangle vs. red hollow triangle). This improvement is also statistically significant, with p =1.22e-04 in the McNemar test at the 0.05 level. That is, analogy retrieval boosts model performance across all settings.

Verifier feedback. We now measure the impact of retries following feedback. As shown in Figure 3, allowing retries consistently improves results across all difficulty levels. For our analogy-based method, retries (blue square) yields an average gain of additional 20% over no retries (blue circle), with gains ranging from 10% to 30% at different levels. The improvement is even more pronounced for the base model (red hollow square vs. red hollow circle), where verifier feedback results in an average gain of 28%, ranging from 10% to 40% per level.

Multiple runs. In RQ1, we evaluated the effect of multiple runs when comparing our full method to the base model. We now analyze the impact of multiple runs for the same method (blue triangle vs. square, red hollow triangle vs. square). We find that additional runs improve performance for both our method and the base model. Notably, for our method multiple runs help more on harder problems (gains of 20-30% for levels 4-5), where the potential for improvement is greater.

To conclude, our method outperforms the baseline across all settings, with each component con-

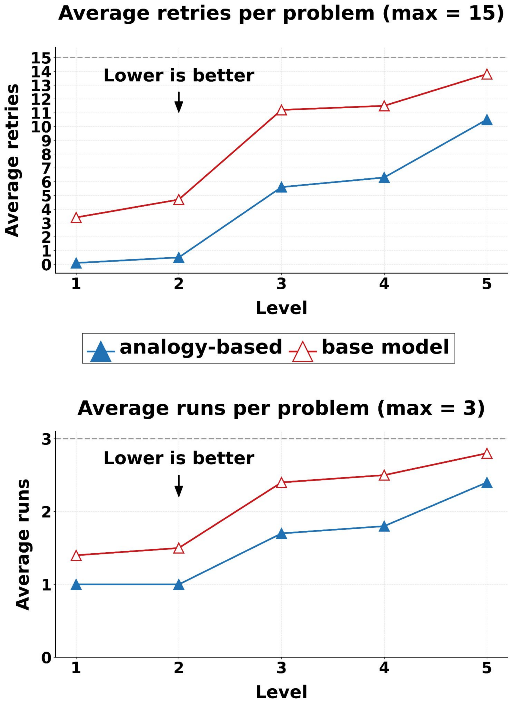

Figure 4: Average number of retries (top) and runs (bottom) per problem by difficulty level. The dashed line represents maximum allowed. Our analogy-based method consistently outperforms the base model, with fewer retries and runs across all levels.

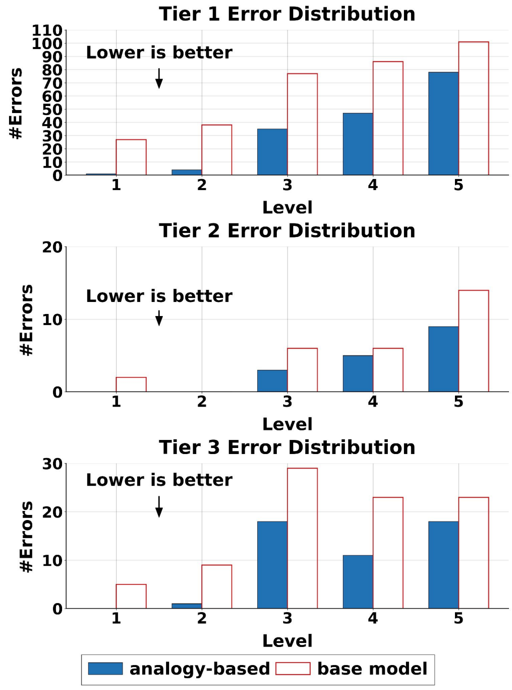

Figure 5: Error distribution by tier for our method vs. the base model. Our method reduces errors across all tiers, with tier-1 errors being most frequent in both.

tributing to performance. Although the verifier alone does enhance the base model's results, analogy retrieval supplies stronger initial proof candidates for the verifier to correct.

5.3 Further analysis.

Effect of m and n parameters. Figure 4 shows the average number of retries (top) and runs (bottom) per problem by difficulty level. Dashed line represents maximum allowed, but the method stops early if it reaches a valid proof. Our method consistently requires fewer retries and runs than the base model across all levels. On average, it uses 4.6 retries and 1.58 runs per problem, compared to 8.92 retries and 2.12 runs for the base model.

Proof accuracy given correct answers. We consider a numeric answer to be correct if it is correct in at least one retry across runs. By this definition, our full pipeline solved 100% of problems (50). Among these, it generated correct proofs for 40 problems (80% success rate). In contrast, the base model (multiple runs, retries) produced 45 correct answers (90%), with only 26 correct proofs (57.7% success rate, compared to 52% on all problems).

Interestingly, while the base model often finds the right answer, it fails to support it with a correct proof. We conjecture that the model often has some notion of the correct proof but struggles to precisely phrase it. By providing relevant proofs from analogous problems, our method helps the model close this gap, which is crucial for trustworthy systems. Notably, our method also improves the overall accuracy in finding the correct numerical answer (100% vs. 90%).

Stability of results. Our evaluation was conducted on a set of 50 problems (10 per difficulty level 1-5). To assess the stability of our results, we sample an additional 10 problems per level and rerun our method. Due to its high cost, we do not repeat this for the base-model baseline.[3] We observe that overall accuracy varies by only 3% per level on average, and conclude the results are stable. See Appendix D for per-level differences.

5.4 Error Analysis

As detailed in Section 3.4, we categorize verifier errors into three tiers. Figure 5 shows their distribution across difficulty levels.

As expected, our method yields fewer errors than the base model across all tiers and levels. Tier-1

[3] The baseline includes the entire theorem dictionary in the prompt, leading to high API call costs.

syntax errors are the most prevalent. These are followed by tier-3, where the proof fails to conclude the goal. Tier-2 – calling a theorem without satisfying one of the premises, are the least common. This trend holds for the base model as well. As noted in Section 3.4, tier-1 errors are frequent but easily resolved by the LLM after feedback.

6 Related Work

Neural Models for Proof Generation. Some prior work used LLMs to directly generate logical or mathematical proofs (Tafjord et al., 2020; Kazemi et al., 2022; Yue et al., 2023). In contrast, we do not train the LLM, but guide it by retrieving analogous problems and using their formal proofs to guide the proof generation.

Neuro-symbolic Methods. Neuro-symbolic approaches combine LLMs with symbolic tools. This strategy has been shown to be effective for some structured tasks (Szeider, 2024; Régin et al., 2024; Mittal et al., 2024; Wu and Mitra, 2024). Autoformalization methods (Song et al., 2024; Alhessi et al., 2025) such as LINC (Olausson et al., 2023) translate natural language into first-order logic using an LLM, followed by a symbolic theorem prover. We note that the focus of these works is on logical reasoning, and they are not readily equipped to handle the mathematical or algebraic elements within proofs.

One work in the domain of geometry is AlphaGeometry (Trinh et al., 2024), which combines symbolic deduction with a language model trained on 100M synthetic geometry examples to generate machine-verifiable proofs. In contrast, our method does not require training.

Analogical Reasoning. LLMs have been studied for analogical reasoning in diverse domains, including word analogies, narratives and processes, and visual scenarios. For a comprehensive dataset survey, see Sultan et al. (2024). However, relatively few works have explored analogical reasoning in mathematics. The most relevant to our work is Yasunaga et al. (2023), who propose *analogical prompting* as an alternative to chain-of-thought, relying on analogies generated by the model itself. In contrast, we retrieve structurally similar problems with verified proofs, without requiring the model to generate (potential incorrect) solutions itself.

Zhou et al. (2025) approach analogical reasoning as a self-supervised learning task, fine-tuning models to transfer solutions between structurally similar problems. The model is trained to solve a target problem conditioned on the solution to a related source problem, learning abstract transformation patterns from many pairs. In contrast, our method requires no additional training and operates entirely at inference time.

In another work, Lee et al. (2025) generate in-context examples for math word problems by creating numerical variants of the target problem. These examples are automatically created, unlabeled, and include only the problems, without solutions. In contrast, our focus is on generating formal mathematical proofs. Thus, in our setting altering numerical values offers virtually no benefits.

7 Conclusions and Future Work

We introduced a neuro-symbolic approach for proof generation, coupling LLMs with retrieval of analogous proofs and symbolic verification. The analogies help guide the LLM, and the verifier provides feedback on the generated proofs.

We test our ideas in the domain of Euclidean geometry. Our experiments on the FormalGeo-7k dataset show that our method significantly improves proof correctness over the base model, with each component contributing to performance gains. Our method also reduces costs by narrowing down the theorem dictionary using the analogous proofs. These results demonstrate the potential of neuro-symbolic methods for mathematical reasoning.

In the future, we plan to extend our approach beyond geometry to other areas, where automated proof verification could help validate complex derivations and ensure the consistency of models. We also see an interesting use case in education, where analogical retrieval can surface similar solved problems to guide students, and a symbolic verifier can offer targeted hints and feedback.

We also aim to explore more expressive forms of formal verification, such as model checking with temporal logics (LTL and CTL), enabling reasoning about dynamic systems that evolve over time, as well as more sophisticated mechanisms for analogy retrieval and structural similarity.

We hope this work inspires future research on neuro-symbolic systems that combine the flexibility of LLMs with the precision of formal reasoning. Our approach could provide a scalable blueprint for building reliable AI systems in STEM domains where correctness is crucial, allowing deployment in complex safety- and security-critical systems.

Ethical Considerations

Dataset. To protect privacy and ensure proper anonymization, we removed the names of individual annotators from the subset of problems used in our experiments and shared on GitHub.

Use of AI Assistants. We used GPT-4o and Claude 3.5 Sonnet for coding assistance, and GPT-4o for writing and rephrasing. We reviewed and edited all of the outputs to ensure they aligned with our design goals and preserved our original intent.

Limitations

General Limitations.

- **Generalization beyond geometry.** Our experiments focus on Euclidean geometry, and results may differ in other formal domains, which may involve different proof structures, theorem types, or symbolic representations. While our pipeline is designed to be adaptable to any domain that supports SMT-based verification, further evaluation is needed to confirm its effectiveness beyond geometry.

Analogy Retrieval Limitations.

- **Abstraction schema for analogy retrieval.** We use a simple abstraction method that replaces entity names and numbers with placeholders. While effective in many cases, this schema may miss deeper structural nuances or semantic relationships, leading to suboptimal or misleading analogies.

Verifier Limitations.

- **No support for inverse trigonometric.** A key limitation of our verification system is that the Z3 theorem prover does not support inverse trigonometric functions. While we implemented workarounds for direct functions such as sine and cosine, the system cannot verify cases requiring inverse operations (e.g., computing θ from $\cos(\theta) = 0.5$). This limits our ability to validate proofs where angle measures must be derived from trigonometric values. Future work could address this by integrating custom decision procedures for inverse trigonometric reasoning.

LLM Limitations.

- **No access to diagrams.** Our LLM uses only formal and textual descriptions, without visual inputs. Although our dataset includes rich symbolic annotations, covering geometric entities, relationships, constructions, and extended conditions, some errors arise due to the system's lack of access to visual diagrams. For instance, when reasoning about arcs, the model may incorrectly infer that $\mathrm{BOD} + \mathrm{BOA} = \mathrm{AOD}$ instead of the correct relation $\mathrm{BOD} = \mathrm{BOA} + \mathrm{AOD}$. Such mistakes are less frequent in reasoning about line, where properties like collinearity are explicitly annotated. In some cases, ambiguity in notation further complicates interpretation; for example, the angle name $\angle DOB$ could refer to the angle itself or its complement, depending on context.

  On the other hand, we note that a recent work (Zhang et al., 2024) has shown that multimodal LLMs often struggle with the visual aspects of math problems, often performing better without the images.

- **Closed LLM Dependency (OpenAI o1).** While this model delivers strong performance and is widely adopted, its architecture, training data, and training methodology are proprietary. It is also subject to deprecation or access restrictions, limiting transparency and long-term reliability.

- **LLMs are sensitive to prompt phrasing.** LLMs are sometimes sensitive to small changes to the prompts.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Yousef Alhessi, Sólrún Halla Einarsdóttir, George Granberry, Emily First, Moa Johansson, Sorin Lerner, and Nicholas Smallbone. 2025. Lemmanaid: Neuro-symbolic lemma conjecturing. *arXiv preprint arXiv:2504.04942*.

Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer.

Dedre Gentner. 1983. Structure-mapping: A theoretical framework for analogy. *Cognitive science*, 7(2):155–170.

Adit Gupta, Jennifer Reddig, Tommaso Calo, Daniel Weitekamp, and Christopher J MacLellan. 2025. Beyond final answers: Evaluating large language models for math tutoring. *arXiv preprint arXiv:2503.16460*.

Keith J Holyoak and Paul Thagard. 1996. *Mental leaps: Analogy in creative thought*. MIT press.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Mehran Kazemi, Najoung Kim, Deepti Bhatia, Xin Xu, and Deepak Ramachandran. 2022. Lambada: Backward chaining for automated reasoning in natural language. *arXiv preprint arXiv:2212.13894*.

Harsh Kumar, David M Rothschild, Daniel G Goldstein, and Jake M Hofman. 2023. Math education with large language models: Peril or promise? *Available at SSRN 4641653*.

Jaeseong Lee, Wei Yang, Gopal Gupta, and Shiyi Wei. 2025. Automatic mathematic in-context example generation for llm using multi-modal consistency. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 8908–8924.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2024. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*.

Chinmay Mittal, Krishna Kartik, Parag Singla, et al. 2024. Puzzlebench: Can llms solve challenging first-order combinatorial reasoning problems. *arXiv preprint arXiv:2402.02611*.

Theo X Olausson, Alex Gu, Benjamin Lipkin, Cedegao E Zhang, Armando Solar-Lezama, Joshua B Tenenbaum, and Roger Levy. 2023. Linc: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. *arXiv preprint arXiv:2310.15164*.

Mayur Amarnath Palavalli and Mark Santolucito. 2024. Using a feedback loop for llm-based infrastructure as code generation. *arXiv preprint arXiv:2411.19043*.

Zhuoshi Pan, Yu Li, Honglin Lin, Qizhi Pei, Zinan Tang, Wei Wu, Chenlin Ming, H Vicky Zhao, Conghui He, and Lijun Wu. 2025. Lemma: Learning from errors for mathematical advancement in llms. *arXiv preprint arXiv:2503.17439*.

Yun Peng, Akhilesh Deepak Gotmare, Michael R Lyu, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. 2025. Perfcodegen: Improving performance of llm generated code with execution feedback. In *2025 IEEE/ACM Second International Conference on AI Foundation Models and Software Engineering (Forge)*, pages 1–13. IEEE.

Florian Régin, Elisabetta De Maria, and Alexandre Bonlarron. 2024. Combining constraint programming reasoning with large language model predictions. *arXiv preprint arXiv:2407.13490*.

Joykirat Singh, Akshay Nambi, and Vibhav Vineet. 2024. Exposing the achilles' heel: Evaluating llms ability to handle mistakes in mathematical reasoning. *arXiv preprint arXiv:2406.10834*.

Peiyang Song, Kaiyu Yang, and Anima Anandkumar. 2024. Towards large language models as copilots for theorem proving in lean. *arXiv preprint arXiv:2404.12534*.

Oren Sultan, Yonatan Bitton, Ron Yosef, and Dafna Shahaf. 2024. Parallelparc: A scalable pipeline for generating natural-language analogies. *arXiv preprint arXiv:2403.01139*.

Stefan Szeider. 2024. Mcp-solver: Integrating language models with constraint programming systems. *arXiv preprint arXiv:2501.00539*.

Oyvind Tafjord, Bhavana Dalvi Mishra, and Peter Clark. 2020. Proofwriter: Generating implications, proofs, and abductive statements over natural language. *arXiv preprint arXiv:2012.13048*.

Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. 2024. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482.

Sean Welleck, Jiacheng Liu, Ximing Lu, Hannaneh Hajishirzi, and Yejin Choi. 2022. Naturalprover: Grounded mathematical proof generation with language models. *Advances in Neural Information Processing Systems*, 35:4913–4927.

Erik Wu and Sayan Mitra. 2024. Can llms plan paths with extra hints from solvers? *arXiv preprint arXiv:2410.05045*.

Michihiro Yasunaga, Xinyun Chen, Yujia Li, Panupong Pasupat, Jure Leskovec, Percy Liang, Ed H Chi, and Denny Zhou. 2023. Large language models as analogical reasoners. *arXiv preprint arXiv:2310.01714*.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2023. Mammoth: Building math generalist models through hybrid instruction tuning. *arXiv preprint arXiv:2309.05653*.

Renrui Zhang, Dongzhi Jiang, Yichi Zhang, Haokun Lin, Ziyu Guo, Pengshuo Qiu, Aojun Zhou, Pan Lu, Kai-Wei Chang, Yu Qiao, et al. 2024. Mathverse:

Does your multi-modal llm truly see the diagrams in visual math problems? In *European Conference on Computer Vision*, pages 169–186. Springer.

Xiaokai Zhang, Na Zhu, Yiming He, Jia Zou, Qike Huang, Xiaoxiao Jin, Yanjun Guo, Chenyang Mao, Yang Li, Zhe Zhu, et al. 2023. Formalgeo: An extensible formalized framework for olympiad geometric problem solving. *arXiv preprint arXiv:2310.18021*.

Ben Zhou, Sarthak Jain, Yi Zhang, Qiang Ning, Shuai Wang, Yassine Benajiba, and Dan Roth. 2025. Self-supervised analogical learning using language models. *arXiv preprint arXiv:2502.00996*.

A Analogous Problems Retrieval

We trained a neural network to predict proof similarity between two problems, using the Jaccard similarity of their proofs as the target. The model inputs a 3D feature vector capturing structural and semantic similarities: (1) Jaccard over abstract constructions, (2) Jaccard over abstract conditions, and (3) goal similarity – computed from abstracted problem representations. The model has three fully connected layers: 3→128 (ReLU), 128→64 (ReLU), and 64→1 (linear). It was trained for 10 epochs with batch size 32 using MSE loss and the Adam optimizer (learning rate 0.001).

B LLM Proof Generation

See Figure 6 for the system prompt provided to the LLM (OpenAI's o1 model), and Figure 7 for few-shot examples of analogous problems and their solutions (including proof, and answer).

C Verifier Implementation Details

Geometric reasoning module. We represent each problem state using a structured system of Python classes that encodes geometric facts, such as point orderings, segment lengths, and angle relationships. To enhance robustness and flexibility, we apply normalization procedures that map geometric objects to canonical, direction-agnostic forms. For example, treating ∠ ABC and ∠ CBA as equivalent.

Algebraic constraint solver. We use the Z3 Theorem Prover, a state-of-the-art SMT solver, to represent and enforce algebraic constraints implied by geometric statements. When a proof step establishes a fact like collinearity of points A, B, C, and D, the system automatically generates angle equality constraints (e.g., $\angle XAB = \angle XAC = \angle XAD$ for any reference point X).

Handling of Trigonometric Expressions. Z3 does not natively support trigonometric functions. To work around this limitation, we:

- Introduce symbolic variables for trigonometric expressions (e.g., `cos_ABC`).
- Add angles referenced by trigonometric terms to the system's internal constraint graph.
- Evaluate goals involving trigonometric expressions either by matching against known values or by reasoning symbolically if exact evaluation is not possible.

Verifier Examples Figure 8 presents a representative example of error outputted by the verifier, from each error tier.

D Stability of Results

Figure 9 shows the performance of our method on both the original 50 samples and the extended set of 100 samples, broken down by level. As shown, performance remains unchanged in levels 2, 3, and 5, while levels 1 show 5% difference, and level 4 4 show a 10% difference. Overall, the average difference across levels is 3%. We conclude our results are stable.

You are a mathematician expert in solving geometry problems.
Your task is to solve Problem B by constructing the correct sequence of theorems (THEOREM_SEQUENCE) to form its proof.

Inputs:
You are given:
Problem B, which you need to solve.
Five analogous problems (A1, A2, A3, A4, A5) that have similar proof structures. These are provided to help guide your approach to solving Problem B.
Geometry Theorem Dictionary (GDL): A dictionary containing various geometry theorems, each with its premise and conclusion. All theorems in the THEOREM_SEQUENCE must be selected from this dictionary.
GDL_DICTIONARY:
{GDL}
For each problem, you are provided the following data:
DESCRIPTION: A textual description of the problem.
CONSTRUCTION_CDL: The problem's construction in Condition Declaration Language (CDL).
TEXT_CDL: The text of the problem in CDL.
GOAL_CDL: The goal of the problem in CDL.
CONSTRUCTION_CDL_EXTENDED: An extended version of CONSTRUCTION_CDL.
SYMBOLS_AND_VALUES: Symbols with corresponding values, in the format (predicate;symbol;value).
EQUATIONS: Equations related to solving the problem, in the format (equation).
GOAL_SYMBOL: The symbol you are trying to solve for, in the format (symbol).
ANSWER: The calculated final answer, in the format (answer).
THEOREM_SEQUENCE: The sequence of theorems used in the proof, formatted as:
step_id <step_id>; <theorem>; <premise>; <conclusion>
step_id <step_id>; <theorem>; <premise>; <conclusion>

Your Task:
You need to solve Problem B by constructing the correct THEOREM_SEQUENCE, which should consist of theorems from the GDL. Ensure that each selected theorem logically follows the previous one and contributes to the goal of solving Problem B.

Output Format:
Your response must contain the following:
EQUATIONS: <equation> <equation> ...
GOAL_SYMBOL: <symbol>
ANSWER: <answer>
THEOREM_SEQUENCE:
<step_id>; <theorem>; <premise>; <conclusion>
<step_id>; <theorem>; <premise>; <conclusion>

Important Notes for the THEOREM_SEQUENCE:
Do not include the words "theorem", "premise", or "conclusion". Your sequence should only contain the step ID, theorem name, premise, and conclusion.
Use the exact theorem names and formats provided in the GDL.
Start with step_id = 1 and increment sequentially.
When referring to angles, use three letters (e.g., ABC for the angle at B). Be mindful of the order, as ABC is different from ACB. For polygons, list all distinct points in clockwise or counterclockwise order. For example, a polygon with points FGE can also be referred to as GEF or EFG.

Example of Correct THEOREM_SEQUENCE Format:
1; angle_addition(1,BFE,EFG);
Angle(BFE)&Angle(EFG)&Angle(BFG);
`["Equal(MeasureOfAngle(BFG),Add(MeasureOfAngle(BFE),MeasureOfAngle(EFG)))"]`
2; triangle_property_angle_sum(1,DFC); Polygon(DFC);
`["Equal(Add(MeasureOfAngle(DFC),MeasureOfAngle(FCD),MeasureOfAngle(CDF)),180)"]`

Reminder:
Ensure that the theorems you select come from the GDL, follow the correct format, and use the proper arguments (e.g., angle order and polygon points). Also, pay attention to the specific variation of the theorem (e.g., 1 for the first variation, 2 for the second, etc.).

Figure 6: System prompt given to the o1 LLM for generating a geometry proof for a target problem B. The model is provided with all details about the task, as well as a Geometry Theorem Dictionary (GDL) and five analogous problems (A1–A5), which are supplied as few-shot examples (see Figure 7).

Inputs for Problem A1:
DESCRIPTION:
As shown in the diagram, Div(LengthOfLine(AD)=LengthOfLine(DF)), Div(LengthOfLine(DF)=LengthOfLine(FB)), AG=15, DE is parallel to BC, DE is parallel to FG, FG || BC. Find the length of line CE.

CONSTRUCTION_CDL:
Shape(AD,DE,EA), ...
Collinear(ADFB), ...

TEXT_CDL:
Equal(Div(LengthOfLine(AD),LengthOfLine(DF)),3/2), ...
ParallelBetweenLine(DE,BC), ...

GOAL_CDL:
Value(LengthOfLine(CE))

CONSTRUCTION_CDL_EXTENDED:
Shape(DE,EA,AD), ...
Collinear(BFDA), ...
Point(A), ...
Line(AB), ...
Angle(ADF), ...
Polygon(ADE), ...
ParallelBetweenLine(CB,ED), ...

SYMBOLS_AND_VALUES:
LengthOfLine(AG);ll_ag;15, ...

Outputs for Problem A1:
EQUATIONS:
ll_ad/ll_df - 3/2, ...

GOAL_SYMBOL:
ll_ce

ANSWER:
9

THEOREM_SEQUENCE:
...

Inputs for Problem A2:
...

Inputs for Problem A3:
...

Inputs for Problem A4:
...

Inputs for Problem A5:
...

Outputs for Problem B:

Figure 7: Few-shot prompt shown to the o1 LLM, consisting of five analogous problems (A1–A5) along with their inputs and solutions – including the final answer and theorem sequence used in the proof. **Note:** "..." indicates omitted content for brevity.

```
Tier 1 (Syntax Violation)
--------------------------------------------------------------------------------
Theorem: parallel_property_ipsilateral_internal_angle(1,GA,HD)
You output the following premises: ParallelBetweenLine(GA,HD)&Line(AD)
But the correct premises: ParallelBetweenLine(GA,HD)&Line(GH)

Tier 2 (Premise Violation)
--------------------------------------------------------------------------------
- Error: You tried to use theorem: right_triangle_judgment_angle(1,BCD);
Polygon(BCD)&Equal(MeasureOfAngle(BCD),90);['RightTriangle(BCD)']

Missing premise: Angle measure 90° for triangle BCD is not established in the
premise.
Details: Premise provided: Polygon(BCD)&Equal(MeasureOfAngle(BCD),90)
- Available premises:
Perpendicular Lines: AD, BD
Collinear Points: ADC
Polygons: ABC, ABD, ACB, ADB, BCD, BDC
- Theorems related to the goal:
Step 1 - right_triangle_judgment_angle(1, BCD): RightTriangle(BCD)
- Solver constraints directly related to this goal:
|AB| = y
Please fix the proof.


Tier 3 (Goal Not Reached)
--------------------------------------------------------------------------------
- Goal: measure of angle ADB
- Model answer: 55.0
- Error: Your proof doesn't uniquely determine the value. You need to use additional
theorems to constrain the value.
- Available premises:
Collinear Points: AOB, BCD
Cocircular Points: A on circle O, AB on circle O, ABC on circle O, AC on circle O,
B on circle O, BC on circle O, C on circle O
Circles: O center: O
Circle Diameters: AB diameter of O
Tangent Lines: DA tangent to O
Polygons: ABD, ADB, ADCO, BCO, BOC, COAD, DCOA, OADC
- Theorems related to the goal: None found that constrain this goal
- Solver constraints directly related to this goal: ∠ADB ≤ 180, ∠ADB > 0,
∠ADC = ∠ADB
```

Figure 8: Examples of verifier errors across three tiers. **Tier 1**: syntax violations in theorem calls. **Tier 2**: missing or undefined premises. **Tier 3**: under-constrained reasoning that fails to derive the goal.

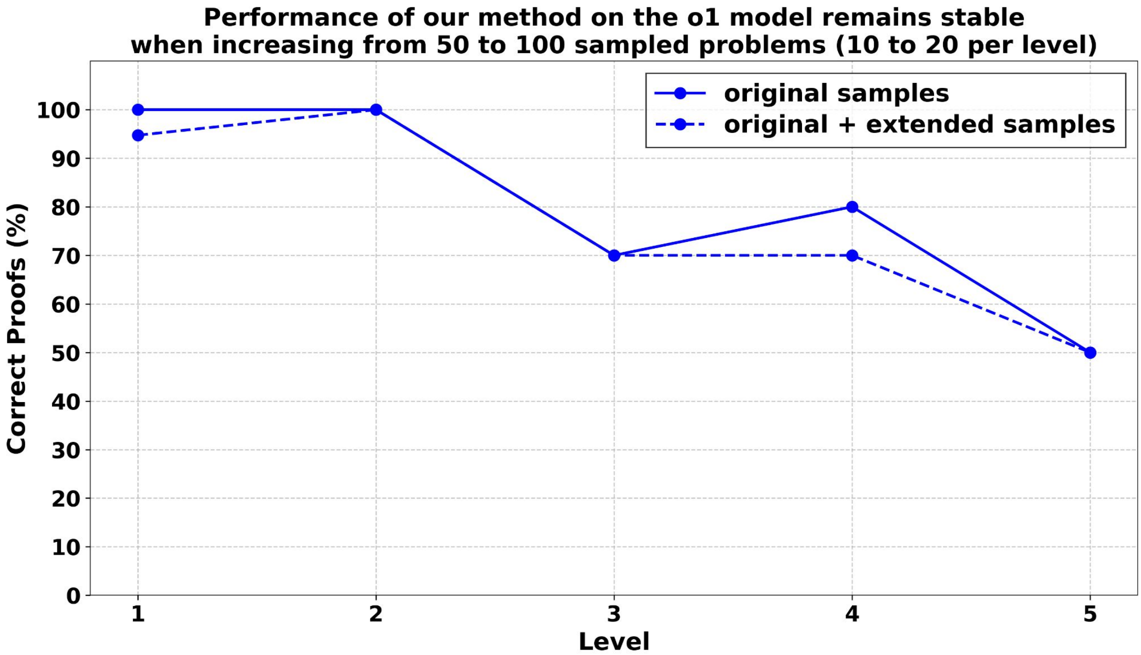

Figure 9: Accuracy remains stable when increasing the sample size from 50 to 100 (10 additional problems per level), with an average variation of only 3% per level. Per-level differences range from 0% (levels 2, 3, and 5) to 5% (level 1), and 10% (level 4). We conclude our results are stable.