Prompt-Driven Verification of LLM-Generated Mathematical Solutions

Hengzhi Zhang
hz2663@columbia.edu

Abstract

We propose a self-verification pipeline that transforms an LLM's chain-of-thought into an explicit proof object, enabling the model to audit its own reasoning with only lightweight calls to itself (or an even smaller variant)—no external solver or larger teacher is required. Our method decomposes multi-step mathematical solutions into five modular stages: (1) solution generation via Vertex AI's Gemini backends, (2) sentence-level segmentation to isolate individual expressions, (3) rule attribution, where each derivation step is labeled with the specific theorem or identity applied, (4) application checking, in which the model verifies each step's logical validity against its stated premises, and (5) error localization via majority voting over multiple trials to pinpoint the earliest incorrect sentence. On an initial sample of ten hard, level-5 MATH problems, our pipeline localizes the first error with high accuracy while providing compact, JSON-structured justifications for each ruling. This traceable, data-rich output not only aids human reviewers but also yields labeled triples $\langle$premises, rule, conclusion$\rangle$ that can supervise future theorem-rationale learning. Our results demonstrate that modeling mathematical reasoning as explicit rule applications empowers LLMs to self-inspect and correct their own proofs, laying groundwork for more reliable, reflective reasoning agents.

1 Introduction

Large language models (LLMs) now routinely derive multi-step solutions to competition-level mathematics problems, yet they still hallucinate steps, mis-apply algebraic identities, or lose track of definitions. We present a *self-verification pipeline* that turns the model's own chain-of-thought into an *object of reasoning*: every generated sentence is treated as a candidate statement that must be justified by explicit applications of previously accepted sentences and general mathematical rules. The pipeline shows that an LLM can *audit itself* with only small, inexpensive calls to the same (or an even smaller) model—no external symbolic solver or larger teacher model is required.

2 Related Work

Chain-of-Thought Prompting Chain-of-Thought (CoT) prompting was first introduced by Wei et al. [Wei et al., 2022], who showed that guiding large language models to generate intermediate reasoning steps markedly improves performance on multi-step arithmetic and commonsense problems. Wang et al. [Wang et al., 2023] later proposed the *self-consistency* strategy, in which multiple CoT samples are generated and aggregated by majority-vote to further boost answer accuracy.

Self-Verification and Error Detection A line of work has explored using LLMs to verify their own reasoning. Weng et al. [Weng et al., 2023] introduced backward verification of CoT outputs, assigning interpretable validation scores to each reasoning chain and demonstrating gains on arithmetic, logical, and commonsense benchmarks. Building on this, Miao et al. [Miao et al., 2024] proposed *SelfCheck*, a zero-shot schema in which an LLM flags errors in its step-by-step reasoning and combines multiple checks via weighted voting to boost final accuracy on GSM8K, MathQA, and the MATH dataset [Hendrycks et al., 2021]. More recently, Chowdhury and Caragea [Chowdhury and Caragea, 2025] designed COT-STEP prompts to decompose reasoning, plus a zero-shot verifier that classifies the correctness of each step without any fine-tuning.

Deductive and Dataset-Driven Verification Ling et al. [Ling et al., 2023] introduced the *Natural Program* formalism for rigorous, stepwise deductive checking of CoT outputs, providing a template for exact logical verification. Hong et al. [Hong et al., 2024] offered an in-depth empirical study of LLMs' self-verification skills, constructing the *Fallacies* dataset to benchmark models' abilities to detect logical missteps in their own reasoning chains. Beyond algorithmic advances, Chen et al. [Chen et al., 2025] present a broad survey of CoT methods and long-form reasoning strategies, situating verification-guided pipelines within the evolving *Reasoning Era* of large language models.

Theorem Rationale Learning Sheng et al. [Sheng et al., 2025] introduce *Theorem Rationale* (TR), a framework that explicitly guides LLMs to select and apply the most pertinent mathematical theorems when solving complex problems. They construct a dedicated dataset of problem–theorem–solution triples and train models to generate a concise theorem rationale before each derivation step, yielding significant gains on high-school and collegiate mathematics benchmarks in AAAI-25.

3 Pipeline Overview

The code in supplementary material decomposes the task *"point to the first incorrect step in a large-language–generated solution to a MATH problem"* into five well-isolated stages, each implemented by an explicit module (Fig. 1). These stages are executed sequentially by the driver class `VerifyCotTheorems`, though any individual stage can be run in isolation for ablation studies or debugging. More detail could be found in 'https://github.com/henryzhang11/verify-cota'.

Solution generation. The `model.py` wrapper supports two Vertex AI backends—`gemini-2.0-flash-001` by default, and an optional higher quality leader model `gemini-2.5-flash-preview-04-17`. Calls to `VertexAI.generate` use a near-deterministic sampling policy (temperature 0.05, $p = 0.95$, $k = 20$) with an exponential back-off retry mechanism of up to six attempts before failing gracefully. Problems are drawn from the Hendrycks MATH dataset via helpers in `dataloader.py`, which expose methods to load the full training split, a hard (non-geometry level-5) subset, and toy instances for unit tests. The script `find_incorrect_solution.py` streams each problem to the model, extracts the `\boxed{...}` final answer, and records any mismatches against the official key in a JSONL file for downstream verification.

Sentence segmentation. Two prompts—`VerifyCotTheorems.parse_text` for the problem statement and `cleanup_answer` for the generated solution—ask the Gemini model to rewrite text so that every mathematical expression appears within a complete English sentence and each sentence sits on its own line. The model's response is delimited by back-ticks, split on newlines, and filtered for empties, yielding ordered lists `problem_sentences` and `solution_sentences`. These are concatenated into `all_sentences`, which serves as the indexed proof state for later stages.

Rule attribution. A lightweight classifier in `categorize` tags each solution sentence as either a *derivation* or *other*, using a majority vote over up to seven LLM passes (typically converging in three). For each derivation, the `name_theorem` module re-prompts the model to emit a single rigid JSON quadruplet containing: a concise rule description (e.g. "quadratic formula," "AM–GM inequality"), the indices of premises and any earlier conclusions reused, and the natural-language form of the claimed conclusion. These quadruplets are cached in `theorems_applied[k]` for sentence k.

Application checking. The `check_application` module prompts the Gemini model with each rule application's premises, rule name, and claimed conclusion. It returns for each application a Boolean `verdict_i` asserting whether the step is valid under first-order logic and a global `"relation"` label (RESTATE, CONTRADICT, or NEITHER) relative to the original sentence. A strict regular expression extracts the fenced JSON (`‘json ...‘`), with malformed or incomplete replies triggering up to five retries before aborting. Parsed results populate `application_correctness[k]` and `application_relevance[k]`.

Error localisation and majority voting. A sentence is deemed sound only if all its rule applications are correct and its final conclusion restates the original sentence. The earliest sentence failing this criterion is reported as the first mistake; if none fail, the proof is certified correct (index -1). To mitigate stochasticity in LLM grading, the entire pipeline (`find_first_mistake`) runs three times, pooling the reported indices and taking a majority vote—ties are broken by choosing the smallest index. The harness in `find_first_mistake.py` processes batches from `clean_sentences_2.txt` and logs aggregate statistics.

Implementation notes Robust logging is provided by `logger_setup.py`, which captures every prompt, response, and intermediate data structure at the DEBUG level while streaming progress at INFO, with automatic rotation above 100000 characters. Numerical answers undergo symbolic normalization in `math_equivalence.py`, ensuring canonical representation of fractions, radicals, and units before string comparison. For oracle baselines, `prepare_baseline_prompt_2.py` generates prompts for feeding into fresh LLMs to locate the first mistake with or without the official answer, quantifying the benefit of our rule-checking decomposition.

4 Reasoning as Rule Applications

A “rule” is any conditional map

$$\text{premises} \longrightarrow \text{conclusion},$$

such as “if $p \to q$ and p then q” or “if $a + b = c$ and $a, b \geq 0$ then $c \geq a$”.

As a side note, the model frequently do not follow the instruction ’let’s reason step by step’ if a step is defined by the application of a single theorem. Instead, large portions of the sentences generated by Gemini jumps several steps from the previous sentence, so it’s necessary to ask the checking model to generate a proof instead of pick a single theorem to show the next sentence based on previous sentences.

Forcing the model to name its rules yields:

- **Traceability** – Human reviewers can skim compact JSON objects instead of deciphering free-text proofs.
- **Data synthesis** – Each verified (or rejected) application becomes a labelled triple $\langle \text{premises}, \text{rule}, \text{conclusion}, \text{verdict} \rangle$ that favours future models which *inspect*, not skip, intermediate steps.
- **Modularity** – Additional checkers (e.g. for theorem correctness) can be layered without altering generation.

5 Experimental Setup

We load 850 *level-5* non-geometry problems from the public MATH train split via `load_MATH_hard` (geometry problems are not tested due to lack of proper diagrams as testing data and price of model processing images). After a test generation sweep, the naive expression equivalence checking function provided by the MATH paper flagged 60 problems to be incorrect. The first ten of these problems are used to conduct an initial test of our above pipeline. As a baseline we use **ChatGPT o4-mini** to decide the first faulty sentence followed by *direct human inspection* to ascertain the existence of the mistake.

6 Results

Across the ten-problem sample, three independent runs of our pipeline found the close to exact first mistake in **seven problems** compared to GPT-4o-mini. Asking for the exact same answer could be overly strict, as the below example demonstrats: On **Problem 1** (five dice probability) the pipeline flags sentence 23, whereas annotators flagged sentence 26; because sentence 23 repeats an already-defined probability, it is arguably the true first slip. Such abiguities in natural language could mean a sentence could be correct under one interpretation while being incorrect under another.

(1) Each of five, standard, six-sided dice is rolled once. (2) Two of the dice come up the same, but the other three are all different from those two and different from each other. (3) The pair is set aside, and the other three dice are re-rolled. (4) The dice are said to show a "full house" if three of the dice show the same value and the other two show the same value (and potentially, but not necessarily, all five dice show the same value). (5) What is the probability that after the second set of rolls, the dice show a full house? (6) Let the five dice be D_1, D_2, D_3, D_4, D_5. (7) We are given that two of the dice come up the same, and the other three are all different from those two and different from each other. (8) Let the value of the pair be x. (9) Then the other three dice have values a, b, c such that x, a, b, c are distinct. (10) We set aside the pair, and re-roll the other three dice. (11) We want to find the probability that after the second set of rolls, the dice show a full house. (12) Let the value of the pair be x. (13) The three dice that are re-rolled have values a, b, c such that x, a, b, c are distinct. (14) We re-roll the three dice. (15) Let the values of the three dice be d_1, d_2, d_3. (16) We want to have a full house, which means we have three of one value and two of another value. (17) We already have two dice with value x. (18) We want to have either three dice with value x and two dice with another value, or two dice with value x and three dice with another value. (19) Case 1: We have three dice with value x. (20) Then we need one more x. (21) The probability of rolling an x is $\frac{1}{6}$. (22) We need exactly one of d_1, d_2, d_3 to be x, and the other two to be the same value, different from x. (23) The probability of rolling exactly one x is $\binom{3}{1}(\frac{1}{6})^1(\frac{5}{6})^2 = 3 \cdot \frac{1}{6} \cdot \frac{25}{36} = \frac{75}{216}$. (24) The other two dice must be the same value, different from x. (25) There are 5 possible values for the other two dice. (26) The probability of rolling two of the same value, different from x, is $\binom{2}{2}(\frac{1}{6})^2(\frac{5}{6})^0 = \frac{1}{36}$. (27) So the probability of rolling exactly one x and two of the same value, different from x, is $\frac{75}{216} \cdot \frac{1}{36} = \frac{75}{7776}$. (28) The probability of rolling exactly one x is $\binom{3}{1}(\frac{1}{6})(\frac{5}{6})^2 = \frac{75}{216}$. (29) The probability of rolling two of the same value, different from x, is $5 \cdot (\frac{1}{6})^2 = \frac{5}{36}$. (30) The probability of rolling exactly one x and the other two are the same value, different from x, is $\frac{75}{216} \cdot \frac{5}{36} = \frac{375}{7776}$. (31) Case 2: We have two dice with value x. (32) We need to have three of the same value, different from x. (33) We need all three dice to be the same value, different from x. (34) There are 5 possible values for the three dice. (35) The probability of rolling three of the same value, different from x, is $5 \cdot (\frac{1}{6})^3 = \frac{5}{216}$. (36) Case 3: We have two dice with value x. (37) We need to have two of one value and one of another value, both different from x. (38) We need to have two of one value and one of another value, both different from x. (39) The probability of rolling two of one value and one of another value is $\binom{3}{2}(\frac{1}{6})^2(\frac{5}{6}) \cdot 5 = 3 \cdot \frac{1}{36} \cdot \frac{5}{6} \cdot 5 = \frac{75}{216}$. (40) The probability of rolling a full house is $\frac{5}{216} + \frac{75}{216} = \frac{80}{216} = \frac{10}{27}$. (41) Final Answer: The final answer is $\boxed{10/27}$

Trial 1's potentially incorrect sentences lists:

$$[[23, 26, 27, 39, 40], [18], [29, 30, 43, 55, 82], [8, 17, 21], [], [], [], [15, 17, 19, 22, 23, 54], [], [11]]$$

Trial 2's potentially incorrect sentences lists

$$[[23, 26, 27, 39, 40], [], [42, 43, 69, 77, 82], [17, 21], [20], [], [], [19, 22, 46], [], []]$$

Trial 3's potentially incorrect sentences lists

$$[[23, 27, 39, 40], [16], [30, 43, 47, 54, 82], [21], [], [83], [], [19, 22, 23, 46], [], [9, 11, 13]]$$

Ground truth first incorrect sentences

$$[26, 18, 44, 21, -1, -1, -1, 12, -1, -1]$$

Voting then take minimum

$$[23, 16, 43, 21, -1, -1, -1, 19, -1, 11]$$

To deal with such uncertainties and the possibility that language models make mistakes checking proofs, conduct multiple trials and use a voting mechanism to make the final decision on the first incorrect sentence. The algorithm collects potential mistake sentences across multiple runs and then counts the number of times each sentence is flagged potentially wrong. Sentences with the largest count are very likely to be false and therefore deemed to be the prime suspects. The smallest number among these sentences are then deemed to be the first incorrect sentence.

To justify this, observe that a language model could make mistake judging a sentence to be incorrect when it is in fact correct, but that probability is much smaller than the probability it judges an actual incorrect sentence to be incorrect. By the law of large numbers, using multiple trials, the total number of times an incorrect sentence is flagged would ultimately be much bigger than the total number of times a correct sentence is flagged. So, it makes sense to pessimistically pick out the first sentence that's flagged the largest number of times in all trials.

7 Discussion

The pipeline could potentially serve as real-time guardrail on problems drawn from domains the model hasn't seen—e.g. novel theorem families, atypical presentation styles, or entirely new subject areas—to test whether self-verification prevents spurious leaps outside the training distribution.

The preliminary sample is encouraging but too small for definitive claims.

8 Next Steps

1. Compress prompts to lower failure rates due to formatting overload.
2. Implement a tolerant JSON extractor to handle stray tokens.
3. Introduce `check_theorem` to validate the natural-language statement of each cited rule.
4. Evaluate the enlarged set of 100 + problems and analyze error patterns.

9 Conclusion

Modeling mathematical reasoning as an explicit sequence of rule applications lets an LLM audit itself, surface the earliest slip, and provide human-legible justifications. Even on a small public sample, the pipeline reaches an effective 70 % localisation accuracy—without consulting a larger teacher model and while remaining text-only.

References

Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wanxiang Che. Towards reasoning era: A survey of long chain-of-thought for reasoning large language models. *CoRR*, abs/2503.09567, 2025. doi: 10.48550/ARXIV.2503.09567. URL `https://doi.org/10.48550/arXiv.2503.09567`.

Jishnu Ray Chowdhury and Cornelia Caragea. Zero-shot verification-guided chain of thoughts. *CoRR*, abs/2501.13122, 2025. doi: 10.48550/ARXIV.2501.13122. URL `https://doi.org/10.48550/arXiv.2501.13122`.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In Joaquin Vanschoren and Sai-Kit Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL `https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html`.

Ruixin Hong, Hongming Zhang, Xinyu Pang, Dong Yu, and Changshui Zhang. A closer look at the self-verification abilities of large language models in logical reasoning. In Kevin Duh, Helena Gómez-Adorno, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 900–925. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.NAACL-LONG.52. URL `https://doi.org/10.18653/v1/2024.naacl-long.52`.

Zhan Ling, Yunhao Fang, Xuanlin Li, Zhiao Huang, Mingu Lee, Roland Memisevic, and Hao Su. Deductive verification of chain-of-thought reasoning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL `http://papers.nips.cc/paper_files/paper/2023/hash/72393bd47a35f5b3bee4c609e7bba733-Abstract-Conference.html`.

Ning Miao, Yee Whye Teh, and Tom Rainforth. Selfcheck: Using llms to zero-shot check their own step-by-step reasoning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL `https://openreview.net/forum?id=pTHfApDakA`.

Yu Sheng, Linjing Li, and Daniel Dajun Zeng. Learning theorem rationale for improving the mathematical reasoning capability of large language models. In Toby Walsh, Julie Shah, and Zico Kolter, editors, *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 15151–15159. AAAI Press, 2025. doi: 10.1609/AAAI.V39I14.33662. URL `https://doi.org/10.1609/aaai.v39i14.33662`.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL `https://openreview.net/forum?id=1PL1NIMMrw`.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL `http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html`.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. Large language models are better reasoners with self-verification. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 2550–2575. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-EMNLP.167. URL `https://doi.org/10.18653/v1/2023.findings-emnlp.167`.