Synthetic Proofs with Tool-Integrated Reasoning: Contrastive Alignment for LLM Mathematics with Lean

Mark Obozov
Research Center of the Artificial Intelligence Institute
Innopolis University
Innopolis, Russia
obozovmark9@gmail.com

Michael Diskin
HSE University
Moscow, Russia
michael.s.diskin@gmail.com

Aleksandr Beznosikov
Innopolis University
Innopolis, Russia

Alexander Gasnikov
Innopolis University
Innopolis, Russia

Serguei Barannikov
Skoltech, CNRS
Moscow, Russia

Abstract

Modern mathematical reasoning benchmarks primarily focus on answer finding rather than proof verification, creating a gap in evaluating the proving capabilities of large language models (LLMs). We present a methodology for generating diverse mathematical proof tasks using formal tools. Our approach combines Lean-based synthetic problem generation with a Tool-Integrated Reasoning (TiR) framework for partial (sampling-based) proof validation, and it uses contrastive preference optimization to align the model's proof outputs. Experiments on the Qwen-2.5 family of models demonstrate meaningful improvements in mathematical reasoning, particularly for smaller models. Our aligned models achieve up to a 57% higher success rate than baselines on the MiniF2F benchmark (across 0.5B, 1.5B, and 7B parameter models). These results highlight the potential of synthetic data and integrated validation for advancing LLM-based mathematical reasoning.

1 Introduction

Mathematical reasoning is a fundamental challenge for artificial intelligence. Despite significant advances in large language models (Li et al., 2024; Yang et al., 2024a; DeepSeek-AI, 2024), achieving robust proof-solving capabilities comparable to human mathematicians remains elusive. A core difficulty lies in the vast search space of proofs: any given statement can spawn an enormous graph of conjectures and implications. Exhaustive search in this space is infeasible — formal proof attempts like AlphaProof (Hubert et al., 2024) *can be* computationally expensive at scale for complex problems.

Recent approaches using LLMs show promise in their flexibility and reasoning ability. However, even models that excel at answer-focused tasks (e.g., the MATH benchmark (Hendrycks et al., 2021)) often struggle with formal proofs and higher-level mathematical reasoning (Tsoukalas et al., 2024; Glazer et al., 2024). This gap calls for methodologies that combine the adaptability of LLMs with the rigor of structured verification.

The contributions of this paper are as follows:

- **Tree-Based Conjecture Generation**: A tree-based algorithm for synthetic conjecture generation using Lean (de Moura et al., 2015) to produce diverse, valid mathematical problems.
- **Tool-Integrated Reasoning**: A Tool-Integrated Reasoning (TiR) framework for partial proof validation that does not require full formalization of the proofs.
- **Contrastive Alignment of Proofs**: A contrastive preference optimization approach (SimPO) to align the model's proof generation with preferred (correct) solutions.
- **New Proof Benchmark Dataset**: A synthetic dataset of 30,000 generated problems and proofs for benchmarking mathematical reasoning in LLMs.

In our pipeline, **Lean** primarily serves as a *generation and structuring environment*: it provides statements, neighborhood relations, and hints that scaffold synthesis. Model outputs are *informal proofs*. We rely on TiR and an LLM judge as *probabilistic* filters at scale; comprehensive formal checking in Lean is used only in a limited way here and is left to future work.

Our experiments with Qwen-2.5 models (Yang et al., 2024b) (0.5B–7B parameters) demonstrate significant improvements in mathematical reasoning, particularly for smaller models. These results highlight the potential of combining structured generation with flexible validation to advance mathematical reasoning in AI systems.

Proceedings of The 3rd Workshop on Mathematical Natural Language Processing (MathNLP 2025), pages 195–202

2 Background

Formal theorem proving has traditionally relied on search-based methods in proof assistants. Tools like Lean's Aesop tactic (Limperg and From, 2023) and the HyperTree proof search algorithm (Poesia et al., 2024a) can systematically explore proofs but struggle with the combinatorial explosion of possibilities in complex problems. More recently, large language models have been applied to theorem proving (Li et al., 2024; Yang et al., 2024a), but purely data-driven approaches still face challenges unless guided by formal structure. Synthetic data generation has shown promise: for instance, AlphaGeometry (Trinh et al., 2024) trained an LLM on two million procedurally generated geometry problems, yielding impressive problem-solving performance.

Aligning LLM outputs with desired solutions often requires learning from preferences. Reinforcement Learning from Human Feedback (RLHF) (Ouyang et al., 2022) showed that human feedback can effectively steer model behavior, and Direct Preference Optimization (DPO) (Rafailov et al., 2023) achieved this without a separate reward model. Recent contrastive approaches led to the Simple Preference Optimization (SimPO) (Meng et al., 2024) framework, which introduces a margin γ to ensure the model scores a preferred output higher than a dispreferred one by at least a fixed gap. This margin-based criterion is well-suited for mathematical proofs, where distinguishing correct reasoning from subtly flawed reasoning is essential; alternative preference objectives and recipes include KTO (Ethayarajh et al., 2024), LMSI (Huang et al., 2022), ORPO (Hong et al., 2024), and PPO (Schulman et al., 2017). Another complementary direction is integrating external tools into the reasoning loop. Tool-Integrated Reasoning (Gou et al., 2023) agents allow an LLM to call formal solvers or checkers during problem solving. We build on this idea by using formal environment checks to partially verify generated conjectures and proofs without requiring complete formalization.

Self-improving systems represent another promising direction, with recent work focusing on agents that enhance their mathematical capabilities by autonomously generating and filtering their own proofs (Lin et al., 2024; Huang et al., 2022; Poesia et al., 2024b), complemented by intrinsic motivation algorithms like Minimo (Poesia et al., 2024b) for exploring infinite action spaces without predefined goals.

Current evaluation methodologies present significant limitations; standard benchmarks (Liu et al., 2024; Hendrycks et al., 2021) typically rely on answer-matching approaches that fail to validate reasoning steps, while formal proof benchmarks such as PutnamBench and MiniF2F (Tsoukalas et al., 2024; Zheng et al., 2021) offer more rigorous assessment environments with varying difficulty distributions — PutnamBench featuring exceptionally challenging problems and MiniF2F providing a broader range suitable for evaluating models across different skill levels — though they still require bridging the gap between natural language reasoning and formal representations.

3 Synthetic Generation

3.1 Lean-Based Generation

We propose a tree-based algorithm to generate new conjectures from existing ones. The method treats mathematical statements as nodes in a proof graph and leverages known proofs to create harder related problems. Given an initial conjecture X_0 with a known proof Y, we perform a random walk of N steps starting at X_0, moving to a sequence of neighboring conjectures in the graph. This produces a trajectory of intermediate conjectures that lie progressively further from the root (axioms). We then reverse this trajectory and prepend the original proof Y, using X_0 as a lemma to derive a new conjecture. In essence, the algorithm finds a new target statement that is "beyond" X_0 in the proof tree and constructs a valid proof for it using Y. Algorithm 1 outlines the procedure.

The choice of neighbor selection in Algorithm 1 is crucial. We consider several strategies, including Lean hints and tooling (e.g., *Lean Copilot* (Song et al., 2024)), a formal environment (Peano (Poesia et al., 2024b)), and lightweight LLM-based heuristics such as symbol-overlap and predicted difficulty; the walk stops on a fixed budget or when predicted difficulty saturates.

3.2 Generating New Problems from Solutions

Another strategy for synthesis is to derive new problems from known solutions. Given a solved problem P_0 with solution S_0, we attempt to *invert* it: generate a new problem P_1 for which S_0 (or a minimally modified version) serves as a solution. This inverse-problem technique expands

Algorithm 1 Synthetic Conjecture Generation

Require: $T = (V, E)$, N, X_0, Y ▷ Proof graph, steps, initial conjecture, and its proof
Ensure: $X_0 \in V$ and $Y \subseteq V$ ▷ X_0 and all proof nodes in V

$X_{\text{current}} \leftarrow X_0$
$T_{\text{trace}} \leftarrow \{\}$ ▷ Initialize trace
for $i = 1$ to N **do**
  $X_{\text{next}} \leftarrow$ choose a neighbor of X_{current} ▷ $X_{\text{next}} \in N(X_{\text{current}})$
  $T_{\text{trace}} \leftarrow T_{\text{trace}} \cup \{X_{\text{next}}\}$
  $X_{\text{current}} \leftarrow X_{\text{next}}$
end for
reverse T_{trace}
$T_{\text{trace}} \leftarrow Y \cup T_{\text{trace}}$ ▷ Prepend original proof Y

our dataset with challenging new problems while retaining a valid solution path.

Illustration. If S_0 proves $u + v \geq 2\sqrt{uv}$ for $u, v \geq 0$, then substituting $u = a^2$, $v = b^2$ and adding a constraint (e.g., $a + b = 1$) yields a variant inequality where the same proof skeleton applies with adjusted premises.

3.3 Generating Rejected Values

For contrastive training, we need not only correct proofs but also plausible incorrect proofs as counterexamples. We employ three techniques to automatically generate such *rejected* outputs: (1) use a smaller or less capable LLM to answer the problem (weaker models often produce incorrect or suboptimal solutions); (2) if the main model produces a correct solution, prompt it to introduce a mistake into its reasoning (and if it fails to solve the problem, its failed attempt is used as R); (3) perturb the proof trajectory from our tree-based generator (Algorithm 1) to create a flawed solution path. These methods yield contrastive pairs where y_w is a correct proof and y_l is a similar but incorrect attempt.

3.4 Tool-Integrated Validation

We integrate a validation step using external tools to filter the generated data. Instead of asking the LLM to produce a fully formal proof, we prompt it to output a verification function f (e.g., a Python snippet) that returns 1 if a conjecture holds for given inputs and 0 otherwise. We then evaluate f on N randomly sampled inputs from a domain-specific environment to statistically test the conjecture; this is a *sampling-based* check and thus provides no absolute soundness guarantee. Algorithm 2 illustrates this process, which computes the fraction of inputs for which f returns true.

Algorithm 2 TiR-Based Validation

Require: $f : \{\dots\} \to \{1, 0\}$
Require: N ▷ Number of random trials
Require: $\mathcal{E}$ ▷ Input generator environment
Ensure: Empirical success rate of conjecture (fraction of inputs where f returns 1)

$success \leftarrow 0$
$failure \leftarrow 0$
for $i = 1$ to N **do**
  $x \leftarrow \mathcal{E}$.generate_input()
  $output \leftarrow f(x)$
  if $output = 1$ **then**
    $success \leftarrow success + 1$
  else
    $failure \leftarrow failure + 1$
  end if
end for
return $\frac{success}{N}$

If $f(x) = 1$ for all N sampled inputs (i.e., a success rate of 1.0), we consider the conjecture *validated for the sampled domain* $\mathcal{E}$; adversarial counterexamples may still exist (see Limitations). For example, given the inequality conjecture $a^2 + b^2 + c^2 \geq ab + ac + bc$, our TiR prompt produces a function $f(a, b, c)$ that computes $a^2 + b^2 + c^2$ and $ab + ac + bc$ and returns 1 if the inequality holds (0 otherwise). Running Algorithm 2 on this f with random numeric inputs quickly confirms the truth of the conjecture.

4 Training Framework

Using the synthetic data and validation techniques above, we construct training pairs for contrastive alignment. For each problem x, we obtain a correct proof y_w (verified by TiR) and a corresponding incorrect proof y_l (generated via the strategies in Section 3.3). We then fine-tune the model using a SimPO-based objective, which encourages the model to assign higher probability to y_w over y_l.

The training loss follows the SimPO formula-

tion:

$$\mathcal{L}_{\text{SimPO}}(\pi_\theta) = - \mathbb{E}_{(x,y_w,y_l)}\Big[\log \sigma\Big(\log \frac{\beta}{|y_w|} \log \pi_\theta(y_w|x) - \log \frac{\beta}{|y_l|} \log \pi_\theta(y_l|x) - \gamma\Big)\Big], \quad (1)$$

where $|y|$ denotes the length (number of tokens) of output y. We set $\beta = 2.0$ (length normalization against overly short outputs) and $\gamma = 0.5$ (margin between preferred and rejected scores) based on validation experiments. Here β provides a length-normalization factor for the log-probabilities (penalizing overly short answers), and γ enforces a minimum margin between the model's scores for the preferred and rejected outputs.

5 Experiments

5.1 Setup

We fine-tuned three Qwen-2.5 models (with 0.5B, 1.5B, and 7B parameters) on our synthetic proof dataset. The SimPO objective (using $\beta = 2.0,\ \gamma = 0.5$) was implemented with a modified `torchtune` library (torchtune maintainers and contributors, 2024). Each model was trained for 3 epochs on two A100 80GB GPUs, and we selected the best checkpoint based on validation performance. Gradient checkpointing and offloading were used to manage GPU memory during training.

For the synthetic data generation, we employed both Lean and pure-LLM environments. We also experimented with the Peano formal environment, but it produced low-quality conjectures with our algorithm and was excluded from the final dataset. Our TiR validation framework was implemented to handle a variety of mathematical structures (e.g., arithmetic, algebra, group theory, graph theory) and was used primarily to filter out invalid conjectures or proofs before they were added to the training data.

We evaluated our aligned models on the MiniF2F test benchmark, which consists of formal math problems disjoint from those used in training. Since our model outputs are informal proofs (not directly checkable by an automatic theorem prover), we employed a separate judge model to assess solution correctness. Specifically, we used a 32B distilled model (DeepSeek-R1-32B-Qwen-Distilled) as an automated verifier: given a problem and our model's solution, it decides whether the solution is correct (binary accept/reject). We report the *judge-accepted pass rate* in $[0, 1]$ averaged over MiniF2F and compare it to the base model's success rate (baseline).

Model	Size	Aligned	Baseline
Qwen-2.5	0.5B	**0.22**	0.14
Qwen-2.5	1.5B	**0.37**	0.29
Qwen-2.5	7B	**0.53**	0.47

Table 1: MiniF2F *judge-accepted pass rate* (0–1). "Aligned": SimPO-trained on our synthetic pairs; "Baseline": original pretrained model. Higher is better. Verifier: DeepSeek-R1-32B-Qwen-Distilled. *Scope:* single family (Qwen-2.5).

5.2 Results and Discussion

Our alignment approach markedly improved the models' problem-solving success rates. Table 1 summarizes each model's performance versus its unaligned baseline.

The results reveal several insights:

- **Consistent Gains**: Across all model sizes, our aligned models outperform their baselines. The smallest model (0.5B) enjoys the largest relative gain (about +57% relative improvement).
- **Scaling Effects**: The benefit of synthetic training persists as model size grows, though the relative improvement is more pronounced for smaller models. This suggests that smaller models gain proportionally more from our additional training data and alignment.
- **Validation Efficacy**: The TiR filtering appears effective in removing invalid proofs from the training corpus, which helps ensure the model learns from mostly correct and verifiable examples.

6 Scope and External Validity

Our empirical scope is intentionally narrow (Qwen-2.5, MiniF2F) to isolate pipeline effects. The components of our method (problem synthesis, TiR filtering, contrastive alignment) are model-agnostic; we expect transfer across families with two practical considerations: (i) evaluation — complementing the LLM judge with selective formal checking on representative subsets; (ii) domain coverage — extending $\mathcal{E}$ and validators beyond algebra/number

theory (e.g., geometry and graphs require domain-specific generators and metamorphic tests).

7 Conclusion

Our study demonstrates that combining synthetic proof generation with partial formal validation can substantially bolster an LLM's mathematical reasoning abilities. In particular, training on generated conjecture–proof pairs (with contrastive alignment) enabled even relatively small models to solve significantly more formal problems. This work represents a step toward AI systems that can not only generate and verify mathematical proofs but also gradually improve their own reasoning strategies.

To support reproducibility and accelerate progress in mathematical reasoning research, we will release our complete synthetic dataset of 30,000 problems and the Tool-Integrated Reasoning framework under open-source licenses. This includes the tree-based generation algorithms, validation mechanisms, and alignment methodology described in this paper.

In future work, we plan to enhance the synthetic generation process with chain-of-thought prompting to further improve conjecture quality, extend our framework to other domains such as geometry (which may require specialized validation techniques), strengthen the integration between LLM reasoning and formal verification (improving the TiR framework), and investigate more efficient training strategies to scale to larger models.

8 Limitations

1. **Sampling-based validation.** TiR offers high-precision but only *sampling-based* guarantees: a conjecture that passes N random trials in $\mathcal{E}$ may still be false (adversarial cases are possible).

2. **Domain coverage.** The synthetic set is skewed toward algebra/number theory; geometry, analysis, and combinatorics require domain-specific generators and validators that we do not cover here.

3. **LLM judge bias.** Evaluation relies on an automated LLM judge (binary accept/reject). Despite high spot-check agreement, residual bias may affect absolute scores.

4. **Empirical scope.** Experiments intentionally target one model family (Qwen-2.5) and one benchmark (MiniF2F) to isolate pipeline effects; cross-family/benchmark validation is left for follow-up work.

9 Acknowledgements

The study was supported by the Ministry of Economic Development of the Russian Federation (agreement No. 139-10-2025-034 dd. 19.06.2025, IGK 000000C313925P4D0002)

References

Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. 2015. The Lean theorem prover (system description). In *Automated Deduction – CADE-25*, volume 9195 of *Lecture Notes in Computer Science*, pages 378–388. Springer.

DeepSeek-AI. 2024. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. 2024. KTO: Model alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*.

Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, and et al. 2024. FrontierMath: A benchmark for evaluating advanced mathematical reasoning in AI. *arXiv preprint arXiv:2411.04872*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, and et al. 2023. ToRA: A tool-integrated reasoning agent for mathematical problem solving. *arXiv preprint arXiv:2309.17452*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, and et al. 2021. Measuring mathematical problem solving with the MATH dataset. In *Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*.

Jiwoo Hong, Noah Lee, and James Thorne. 2024. ORPO: Monolithic preference optimization without reference model. *arXiv preprint arXiv:2403.07691*.

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and et al. 2022. Large language models can self-improve. *arXiv preprint arXiv:2210.11610*.

Thomas Hubert, Rishi Mehta, Laurent Sartran, and Google DeepMind Team. 2024. AI achieves silver-medal standard solving International Mathematical Olympiad problems. *Nature*. News & Views.

Jia Li, Edward Shan, Xuefeng Wu, Yifei Ma, Rohan Murty, Himanshu Arora, and et al. 2024. Numina-Math: AI Mathematical Olympiad Progress Prize.

Jannis Limperg and Asta Halkjær From. 2023. Aesop: White-box best-first proof search for Lean. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '23)*.

Haohan Lin, Zhiqing Sun, Yiming Yang, and Sean Welleck. 2024. Lean-STaR: Learning to interleave thinking and proving. *arXiv preprint arXiv:2407.10040*.

Hongwei Liu, Zilong Zheng, Yuxuan Qiao, Haodong Duan, Zhiwei Fei, Fengzhe Zhou, and et al. 2024. MathBench: Evaluating the theory and application proficiency of LLMs with a hierarchical mathematics benchmark. *arXiv preprint arXiv:2405.12209*.

Yu Meng, Mengzhou Xia, and Danqi Chen. 2024. SimPO: Simple preference optimization with a reference-free reward. *arXiv preprint arXiv:2405.14734*.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, and et al. 2022. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*.

Gabriel Poesia, David Broman, Nick Haber, and Noah D. Goodman. 2024a. HyperTree proof search for neural theorem proving. *arXiv preprint arXiv:2205.11491*.

Gabriel Poesia, David Broman, Nick Haber, and Noah D. Goodman. 2024b. Learning formal mathematics from intrinsic motivation. *arXiv preprint arXiv:2407.00695*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

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

torchtune maintainers and contributors. 2024. torchtune: PyTorch's finetuning library.

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

George Tsoukalas, Jasper Lee, John Jennings, Jimmy Xin, Michelle Ding, Michael Jennings, and et al. 2024. PutnamBench: Evaluating neural theorem-provers on the Putnam Mathematical Competition. *arXiv preprint arXiv:2407.11214*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, and et al. 2024a. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, and et al. 2024b. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. 2021. miniF2F: A cross-system benchmark for formal Olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*.

A Examples of generated conjectures

Here are a few examples of generated problems with our algorithms.

InternLM step generator was strong in algebra problems generation with algorithm 1.

Basic problem

For $a, b, c \geq 0$ and $a+b+c=1$ prove that $1+12abc \geq 4(ab+bc+ac)$

We got a new problem and proof to it, which further on the G than starting one:

New problem

$1+12ab(1-a-b) \geq 4(ab+b(1-a-b)+(1-a-b)a)$

B GeoGen and AlphaGeometry synthetic

In our methods, we did not touch geometrical problems as they require slightly different approaches. Also, formalization of geometrical problems to Lean-like languages is a quite complicated task. Knowing this, we consider different approach to synthetical geometric task generation. We analyzed AlphaGeometry framework and examined its problems:

1. AlphaGeometry can't solve problems that require non-trivial additional constructions. For instance, median doubling.
2. Directed angles make it possible to solve really generalized problems.
3. Projective theorems (Pascal theorem, Pappus's theorem, etc.), usually can't be solved.
4. There is no numerical package that could help to calculate geometrical problems in coordinates.

Knowing these facts, we used a slightly modified version of GeoGen which included Humpy and Dumpy points and created several configurations to this algorithm. Then, we applied Qwen2.5 to translate problems to AlphaGeometry language. Finally, we searched for the solution with AlphaGeometry. With this algorithm pairs problem/solution might be conducted. Let's consider examples of generated problems:

While we were able to generate some qualitative geometrical problems with this algorithm, this framework is computationally heavy, so we do not highlight any alignment experiments related to the geometrical data.

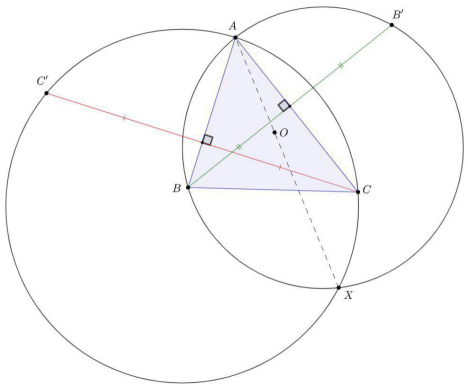

Figure 1: GeoGen example problem 1

C Real effects of small changes in evaluation

We examined the following question: Can we produce new evaluation sets by applying small changes to the questions similar to noise? For instance, given a simple task:

Original problem

On a circle, there are $2n$ points: n red and n blue. A red frog starts on one of the red points, and a blue frog starts on one of the blue points. Each minute, the red frog jumps clockwise to the next red point, and simultaneously, the blue frog jumps counterclockwise to the next blue point. Prove that for any initial positions of the frogs, it is always possible to draw a line such that the frogs are on opposite sides of the line at all times.

Problem (With changes)

On a circle, there are $2n$ points: n red and n blue. A red **turtle** starts on one of the red points, and a blue **turtle** starts on one of the blue points. Each **second**, the red **turtle hops** clockwise to the next red point, and simultaneously, the blue **turtle hops** counterclockwise to the next blue point. Prove that for any initial positions of the **turtles**, it is always possible to draw a line such that the **turtles** are on opposite sides of the line at all times.

With recent works there was an active attention to such mechanism as a potential way to produce new evaluation frameworks. In our work we show

that such changes are inefficient. Changes of the results from table 1 are bounded within $[-1, +1]$. Therefore, such a method of creating extra evaluation data is not really working and affecting LLM reasoning abilities.

D Examples of TiR validation function

In this section we provide a few examples of functions that were generated during the TiR conjecture validation procedure.

Conjecture 1 **(Inequality)** $a^2 + b^2 + c^2 >= ab + ac + bc$

```
def f(a, b, c):
    """
    Evaluates the conjecture a^2 + b^2 +
        c^2 >= ab + ac + bc.

    Args:
        a, b, c: Real numbers.

    Returns:
        1 if the conjecture holds, 0
            otherwise.
    """
    lhs = a**2 + b**2 + c**2
    rhs = a*b + a*c + b*c
    if lhs >= rhs:
        return 1
    else:
        return 0
```

Conjecture 2 **(Functional equation), IMO 2024** A function $f : \mathbb{Q} \to \mathbb{Q}$ is called aquaesulian if the following property holds: for every $x, y \in \mathbb{Q}$,

$$f(x + f(y)) = f(x) + y \quad \text{or} \quad f(f(x) + y) = x + f(y). \quad (2)$$

Show that there exists an integer c such that for any aquaesulian function f, there are at most c different rational numbers of the form $f(r)+f(-r)$ for some rational number r, and find the smallest possible value of c.

```
from fractions import Fraction

def aquaesulian_validator(x, y):
    """
    A validator for an aquaesulian
        function. This function will
        return 1 if the conjecture
    holds for the given x and y, and 0
        otherwise.

    Args:
        x: Rational number (Fraction)
        y: Rational number (Fraction)

    Returns:
        1 if conjecture holds, 0
            otherwise.
    """
    # Define the function f (this is a
        placeholder; define actual f
        based on the conjecture)
    # For example, assume f(x) = x (
        identity function for simplicity
         in testing)
    def f(z):
        return z  # Identity function

    # Check if f(x + f(y)) = f(x) + y or
         f(f(x) + y) = x + f(y)
    if f(x + f(y)) == f(x) + y:
        return 1  # First condition
            holds
    elif f(f(x) + y) == x + f(y):
        return 1  # Second condition
            holds
    else:
        return 0  # Neither condition
            holds
```

Conjecture 3 (**Graph theory**). There is a node in a planar graph with *degree* less than 5.

```
def f(graph):
    """
    Validator function for the
        conjecture: in a planar graph,
        there exists at least one node
    with degree less than 5.

    Parameters:
        graph (dict): A dictionary
            representing the graph as an
             adjacency list.
                      Each key is a node
                          , and the
                          value is a
                          list of its
                          neighbors.

    Returns:
        1 if there is a node with degree
             less than 5, 0 otherwise.
    """
    # Iterate over each node and check
        its degree
    for node, neighbors in graph.items()
        :
        if len(neighbors) < 5:
            return 1  # Conjecture holds
                : there exists a node
                with degree < 5

    return 0  # Conjecture does not hold
        : no node with degree < 5
```