Boosting LLM Math Performance with Structured Code Prompts and Symbolic Verification

Anonymous ACL submission

Abstract

While large language models (LLMs) have gained significant attention for mathematical problem-solving, many existing benchmarks require only shallow reasoning and have limited scope, hindering rigorous evaluation of their understanding of mathematical logic. To address this gap, we evaluate several LLMs on the more challenging **CONIC10K** dataset, which focuses on conic section problems. Using code prompts, fine-tuning, and decoding strategies, we improve performance, boosting Qwen-72B from 20.2% to 34.3%. Notably, DeepSeek-R1 achieves a new state-of-the-art accuracy of 97.3% with code prompting, up from 92.7%, demonstrating that *even high-performing models benefit from symbolic input when properly aligned*. We also develop an automated verification system that independently processes 41.9% of results, reducing human evaluation cost. Our results underscore the importance of structured symbolic prompting in enhancing mathematical reasoning and highlight the potential of code-based methods as a general framework for improving LLM performance on complex math tasks.

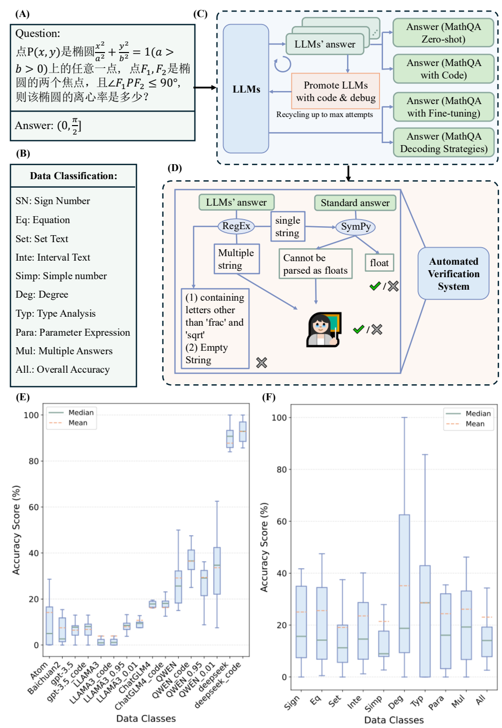

Figure 1: **Our verification process and test results.** In **CONIC10K**, LLMs generate answers to questions (A) via code prompts and fine-tuning (C). An automated verification system checks the answers, while unresolved cases are reviewed by humans, and the combined evaluation yields the final accuracy (D). Box plots show accuracy by model (E) and question type (F). Figure (B) shows our question categories.

1 Introduction

Recent advances in large language models (LLMs) have improved their mathematical reasoning capabilities. For instance, Claude 3.5 Sonnet (Budagam et al., 2024) reached 97.72% on GSM8k (Cobbe et al., 2021), while GPT-4 (Tan et al., 2025) reached 93.9% on SVAMP (Zhao et al., 2023) , and scored 94.3% on MATH23k (Zhou et al., 2023). However, these datasets have notable limitations: 1) many problems require only a few reasoning steps, allowing models to succeed using shallow heuristics (Hendrycks et al., 2021); and 2) their limited knowledge scope hinders evaluation on logic-intensive tasks and comprehensive assessment of mathematical understanding.

Despite advances in enhancing LLMs' mathematical reasoning through diverse methods (Gao et al., 2023; Gou et al., 2024; Jiang et al., 2023), current approaches still face limitations in fully evaluating and improving their capabilities. While natural language aids abstract reasoning, it struggles with precise computation and symbolic manipulation. In contrast, code excels at exact operations and can leverage external tools, but it remains unclear whether these methods truly help LLMs grasp complex concepts and reasoning tasks.

To address these challenges, we selected the more complex **CONIC10K** dataset (Wu et al., 2023), which focuses on Chinese conic section problems. We first evaluated several LLMs on this dataset, then enhanced their capabilities through code-based prompts and fine-tuning. Simultaneously, the answer format of LLMs tends to be very unstable. Comparing accuracy is difficult due to the conic curve dataset's complex answer forms (equations, sets, text). Therefore, we developed an **Automated Verification System** using RegEx and SymPy to measure the model's performance more accurately and efficiently.

Our contributions include: (1) Code-based prompting significantly enhances underperforming models (e.g., Qwen-7B improves from 20.2% to 34.3%); (2) DeepSeek-R1 improves from 92.7% to 97.3% with code prompting, achieving a new state-of-the-art, which demonstrates that strong models can further benefit from symbolic input when aligned with structured reasoning tasks; (3) The Automated Verification System we designed can independently process up to 866 records (out of 2069), greatly saving labor costs.

2 Related Work

For elementary-level problems, datasets such as GSM8K (Cobbe et al., 2021), SVAMP (Zhao et al., 2023), FineMath (Liu et al., 2024b), and CMATH (Wei et al., 2023) include both English and Chinese primary school problems. For more advanced tasks, datasets like MATH (Hendrycks et al., 2021), GAOKAO-Bench (Zhang et al., 2023), and MathBench (Liu et al., 2024a) cover algebra, geometry, etc. Beyond traditional datasets, emerging resources incorporate follow-up questioning (Mishra et al., 2024), fuzzy reasoning (Li et al., 2024b), multimodal understanding (Shi et al., 2024; Cherian et al., 2024), error detection (Singh et al., 2024; Sonkar et al., 2024; Zhang et al., 2024), and detailed evaluation (Qiao et al., 2024) to provide a more comprehensive evaluation of LLMs.

With the support of diverse mathematical datasets, researchers have explored various approaches to enhance the mathematical reasoning capabilities of LLMs. Yu et al. (2024) introduced model ensemble methods, Wang et al. (2024b); Mishra et al. (2024); Wang et al. (2024d) utilized reinforcement learning and Direct Preference Optimization. Wang et al. (2022, 2023) applied chain-of-thought techniques, Yao et al. (2023); Wang et al. (2024a); Feng et al. (2023) developed tree-based methods to reduce computational costs and Wang et al. (2024c) integrated LLMs with Lean4 for theorem proving and (Li et al., 2024a) achieved strong results via code-assisted training.

3 Methods

3.1 Dataset

An example of the **CONIC10K** dataset is presented in Appendix Figure 1(A), and details are shown in Appendix A. We only use question texts and answers. To facilitate a more detailed analysis from the perspectives of numerical computation, expression parsing, and structural modeling, we divide the answers into 10 categories as shown in Appendix Table 2 of Appendix A.

3.2 Tasks

We used the following tasks to evaluate several popular pretrained models' ability on **CONIC10K**. The whole workflow is shown in Appendix Figure 1(B).

MathQA zero-shot. We applied zero-shot inference for all models. The prompts we used in this task are designed to instruct the models to answer together with a rationale, confirm the answer, and output it separately to facilitate verification. Appendix Table 4 shows the prompts we used.

MathQA with code. Previous studies (Gao et al., 2020; Austin et al., 2021) have demonstrated that utilizing code in problem-solving can significantly improve the mathematical reasoning capabilities of models. Motivated by these findings, we prompt the model to generate code to solve the problem. If models fail to generate executable code, additional prompts are used iteratively, up to a predefined maximum number of attempts, to help them refine their output. If all attempts fail, models are then prompted to directly provide an answer, following the MathQA zero-shot strategy. Appendix Table 4 shows the prompts we used.

MathQA with fine-tuning and decoding strategies. Due to limited computational resources, we applied fine-tuning using LoRA (Hu et al., 2022) and evaluated model performance via the MathQA zero-shot task. Additionally, to examine the impact of decoding randomness on mathematical reasoning, we tested two temperature settings (0.95 and 0.01), as prior studies (Brown et al., 2020; Holtzman et al., 2020) have shown that higher temperatures may impair precision by increasing variability in next-token prediction.

Model	Sign	Eq	Set	Inte	Simp	Deg	Typ	Para	Mul	All
Atom (Llama-Chinese)										
Atom-7B	0.0	0.5	0.0	3.9	8.0	75	28.6	19.4	0.0	6.1
Baichuan2-chat (Yang et al., 2023)										
Baichuan2-chat-7B	0.0	2.1	0.0	1.2	2.7	12.5	28.6	9.7	15.4	2.6
GPT-3.5 (OpenAI)										
GPT-3.5-turbo	13.0	8.4	7.5	7.8	8.4	0.0	0.0	3.2	7.7	8.4
GPT-3.5-turbo with code	13.0	8.4	7.5	10.1	9.1	0.0	0.0	3.2	7.7	9.1
Llama 3-Chinese (Grattafiori et al., 2024)										
Llama 3-Chinese-7B	0.9	1.0	0.0	1.6	3.9	0.0	28.6	0.0	0.0	2.7
Llama 3-Chinese-7B with code	0.9	1.3	0.0	1.6	3.9	0.0	28.6	0.0	0.0	2.8
Llama 3-Chinese-7B ft with 0.95	9.6	9.2	7.5	13.2	8.1	12.5	0.0	6.5	3.8	8.9
Llama 3-Chinese-7B ft with 0.01	9.6	9.2	7.5	12.8	9.8	37.5	0.0	3.2	7.7	10.0
ChatGLM-4 (GLM et al., 2024)										
ChatGLM-4-9B	18.3	19.2	17.5	19.5	17.6	12.5	0.0	16.1	23.1	18.1
ChatGLM-4-9B with code	18.3	19.4	17.5	19.5	17.9	12.5	0.0	16.1	23.1	18.3
Qwen (Bai et al., 2023)										
Qwen-72B	23.5	27.8	15.0	16.0	17.6	50.0	57.1	32.0	32.3	20.2
Qwen-72B with code	41.7	47.5	37.5	40.1	27.9	25.0	42.9	32.3	35.6	34.3
Qwen-7B ft with 0.95	33.0	36.2	20.0	29.6	8.8	62.5	28.6	25.8	30.8	18.9
Qwen-7B ft with 0.01	40.9	33.9	20.0	28.4	7.5	62.5	42.9	35.5	46.2	18.4
DeepSeek-R1 (Guo et al., 2025)										
DeepSeek-R1	86.3	89.2	60.0	84.0	93.9	100.0	85.7	93.5	92.3	92.7
DeepSeek-R1 with code	**91.5**	**96.1**	**87.5**	**86.8**	**97.7**	**100.0**	**85.7**	**93.5**	**96.1**	**97.3**

Table 1: Comparison of different models. Number in the table is the accurate rate in this Category 'Sign': Sign Number, 'Eq': Equation, 'Set', Set Text, 'Inte': Interval Text, 'Simp' Simple number, 'Deg': Degree, 'Typ': Type Analysis, 'Para': Parameter Expression, 'Mul': Multiple Answers, 'All': Overall Accuracy. For specific examples of each category, please see Appendix 2

Automated verification system. As noted in (Wu et al., 2023), evaluating differently formatted answers is challenging and often requires human judgment. To address this, we design an automated verification system using regular expressions and SymPy to determine equivalence when the standard answer is a single number, including simple and complex forms (e.g., $\sqrt{2}$).

Specifically, we first clean the model's output using RegEx to remove Chinese characters and standardize number formats. If the output contains letters excluding 'sqrt' and 'frac', it is marked incorrect. Then, the prediction and the reference answer are converted to floats using SymPy for comparison. If conversion fails, human judgment is required. The evaluation workflow is shown in Figure 1(D). The Automated Verification System we designed can independently process up to 866 records and has an average of 756.43 out of 2069, which greatly saves labor costs.

4 Experiments

4.1 Models

We evaluated the performance of several LLMs on MathQA tasks introduced in Methods, including MathQA zero-shot, MathQA with code, MathQA with Fine-tuning, and Decoding Strategies. The models used for evaluation are as listed in Appendix Table 3.

When fine-tuning Qwen-7B and Llama 3, we use AdamW with a 5e-5 learning rate and cosine scheduling. Training is done with batch size 32 per GPU, gradient accumulation over 8 steps, and gradient clipping at a max norm of 1.0. LoRA rank is set to 16 with a scaling factor of 8. We use the original training set and test set, with a max input length of 1024, and limit samples to 10,000.

4.2 Answer Evaluation

For problems with single-number answers, we apply the automated verification system described earlier. For other problems with diverse answer formats (e.g. sqrt(2)x+5y-1=0, y=\frac{\sqrt{2}}{5}...), we rely on human evaluation. Since over half the dataset involves single-number answers, manual workload is greatly reduced.

4.3 Results

Results are shown in the Table 1 and Figure 1 (E-F).

Most baseline LLMs perform poorly. Among all base models, most perform poorly in the MathQA Zero-shot task. Atom-7B and Baichuan2-chat-7B, for example, achieve overall accuracies of only 6.1% and 2.6%, respectively. Even widely used models such as GPT-3.5-turbo reach just 8.4% accuracy, indicating severe limitations in solving structured math problems out-of-the-box.

Code prompting brings gains. Code-based prompting consistently improves performance across multiple models. For instance, GPT-3.5-turbo increases from 8.4% to 9.1%, and ChatGLM-4-9B achieves a boost from 18.1% to 18.3%. While the improvements may appear modest in lower-performing models, the trend is consistent. In higher-capacity models such as Qwen-72B, code prompting increases accuracy from 20.2% to 34.3%, a substantial gain of 14.1 percentage points. Code prompts guide models to translate natural language problems into explicit mathematical programs, mitigating instability in numerical and symbolic processing (See an example in Appendix E).

DeepSeek-R1 achieves state-of-the-art accuracy. DeepSeek-R1 demonstrates exceptional performance, achieving 92.7% in the base setting and further improving to 97.3% with code prompts. It leads across almost all categories. Notably, although DeepSeek-R1 has already been extensively trained on reasoning-intensive tasks, our code-based prompting strategy still yields measurable gains. This suggests that even for highly capable models, introducing structured symbolic input can further enhance precision, serving as a complementary enhancement.

Task-wise difficulty distribution. Categories like Degree, Equation, and Set Text are challenging for most models, especially those lacking explicit symbolic reasoning. Conversely, Simple Number and Type Analysis sees relatively higher accuracy even among weaker models, suggesting that numerical lookup or pattern recognition may suffice in these categories.

5 Discussions

We argue that LLMs face challenges in math due to the gap between probabilistic generation and deterministic reasoning.

First, LLMs generate text by predicting the most probable next word based on learned probability distributions. This mechanism contrasts with the strict, deterministic logic required in mathematical reasoning. Math problems typically require a single correct answer and unambiguous deductive steps. In contrast, LLMs allow multiple plausible outputs, introducing uncertainty that compromises the consistency of multi-step reasoning.

Meanwhile, LLMs primarily rely on statistical correlations and pattern matching, making it difficult to verify whether they truly understand mathematical concepts. The poor performance of some smaller models on complex tasks may indicate a lack of genuine understanding.

These limitations highlight a fundamental gap between probabilistic language modeling and the deterministic nature of mathematical reasoning, suggesting that additional structures or specialized training may be necessary for reliable performance in mathematical tasks.

6 Conclusion

We benchmark popular LLMs on the CONIC10K dataset. Our experiments show that most baseline models—except for DeepSeek-R1 (92.7%)—perform significantly below expectations, with accuracies below 20%, highlighting the challenge that structured math tasks pose for general-purpose LLMs. To address this, we explore three enhancement strategies—code prompting, fine-tuning, and decoding control—which substantially improve weaker models (e.g., Qwen-7B from 20.2% to 34.3%). Notably, even DeepSeek-R1, which already achieves state-of-the-art performance among all models, benefits further from our methods, reaching 97.3% with code prompts. This demonstrates that, when properly aligned, symbolic prompting can reinforce the reasoning capabilities of even the strongest models. Our work provides timely and practical insights into the current strengths and limitations of LLMs in mathematical domains, serving as a valuable reference for future benchmark design and model adaptation. Additionally, we develop an automated verification system that independently handles 41.9% of examples, significantly reducing human evaluation workload.

Limitations

Our current approach executes code outputs directly from LLMs, but the actual execution results may deviate from the model's intended logic due to syntax errors, undefined variables, or semantic inconsistencies. To ensure stability and correctness, future work should incorporate an external code interpreter or compiler to validate and refine execution outputs through a feedback loop driven by runtime errors or mismatch signals. Additionally, decoding temperature shows inconsistent effects—lower temperatures benefit some models (e.g., Llama 3-Chinese-7B), while others (e.g., Qwen-7B) perform better with higher values. Given that our current experiments cover only a limited range of models and tasks, conclusions regarding decoding strategies remain preliminary, calling for broader future exploration. Moreover, existing models are not trained on formal mathematical corpora (e.g., Lean), limiting their exposure to rigorous proofs and confining their reasoning to informal natural language. Finally, our pipeline lacks intermediate verification mechanisms; if an early reasoning step is flawed, subsequent steps propagate the error, reflecting a common challenge in multi-step LLM-based reasoning where outputs are not self-validated during generation.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, and 1 others. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Devichand Budagam, Ashutosh Kumar, Mahsa Khoshnoodi, Sankalp KJ, Vinija Jain, and Aman Chadha. 2024. Hierarchical prompting taxonomy: A universal evaluation framework for large language models aligned with human cognitive principles. *arXiv preprint arXiv:2406.12644*.

Anoop Cherian, Kuan-Chuan Peng, Suhas Lohit, Joanna Matthiesen, Kevin Smith, and Josh Tenenbaum. 2024. Evaluating large vision-and-language models on children's mathematical olympiads. *Advances in Neural Information Processing Systems*, 37:15779–15800.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. 2023. Alphazero-like tree-search can guide large language model decoding and training. *arXiv preprint arXiv:2309.17179*.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2020. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, and 1 others. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2024. Tora: A tool-integrated reasoning agent for mathematical problem solving. *arXiv preprint arXiv:2309.17452*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Weisen Jiang, Han Shi, Longhui Yu, Zhengying Liu, Yu Zhang, Zhenguo Li, and James T. Kwok. 2023. Forward-backward reasoning in large language models for mathematical verification. *arXiv preprint arXiv:2308.07758*.

Chengpeng Li, Guanting Dong, Mingfeng Xue, Ru Peng, Xiang Wang, and Dayiheng Liu. 2024a. Dotamath: Decomposition of thought with code assistance and self-correction for mathematical reasoning. *arXiv preprint arXiv:2407.04078*.

Yiyuan Li, Shichao Sun, and Pengfei Liu. 2024b. Frog: Evaluating fuzzy reasoning of generalized quantifiers in large language models. *arXiv preprint arXiv:2407.01046*.

Hongwei Liu, Zilong Zheng, Yuxuan Qiao, Haodong Duan, Zhiwei Fei, Fengzhe Zhou, Wenwei Zhang, Songyang Zhang, Dahua Lin, and Kai Chen. 2024a. Mathbench: Evaluating the theory and application proficiency of llms with a hierarchical mathematics benchmark. *arXiv preprint arXiv:2405.12209*.

Yan Liu, Renren Jin, Ling Shi, Zheng Yao, and Deyi Xiong. 2024b. Finemath: A fine-grained mathematical evaluation benchmark for chinese large language models. *arXiv preprint arXiv:2403.07747*.

AtomEcho Llama-Chinese. Atom-7b language model.

Shubhra Mishra, Gabriel Poesia, Belinda Mo, and Noah D Goodman. 2024. Mathcamps: Fine-grained synthesis of mathematical problems from human curricula. *arXiv preprint arXiv:2407.00900*.

OpenAI. Gpt-3.5-turbo.

Runqi Qiao, Qiuna Tan, Guanting Dong, Minhui Wu, Chong Sun, Xiaoshuai Song, Zhuoma GongQue, Shanglin Lei, Zhe Wei, Miaoxuan Zhang, and 1 others. 2024. We-math: Does your large multimodal model achieve human-like mathematical reasoning? *arXiv preprint arXiv:2407.01284*.

Wenhao Shi, Zhiqiang Hu, Yi Bin, Junhua Liu, Yang Yang, See-Kiong Ng, Lidong Bing, and Roy Ka-Wei Lee. 2024. Math-llava: Bootstrapping mathematical reasoning for multimodal large language models. *arXiv preprint arXiv:2406.17294*.

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*.

Shashank Sonkar, Naiming Liu, MyCo Le, and Richard Baraniuk. 2024. Malalgoqa: Pedagogical evaluation of counterfactual reasoning in large language models and implications for ai in education. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 15554–15567.

Wenting Tan, Dongxiao Chen, Jieting Xue, Zihao Wang, and Taijie Chen. 2025. Teaching-inspired integrated prompting framework: A novel approach for enhancing reasoning in large language models. In *Proceedings of the 31st International Conference on Computational Linguistics: Industry Track*, pages 827–839, Abu Dhabi, UAE. Association for Computational Linguistics.

Ante Wang, Linfeng Song, Ye Tian, Baolin Peng, Dian Yu, Haitao Mi, Jinsong Su, and Dong Yu. 2024a. Litesearch: Efficacious tree search for llm. *arXiv preprint arXiv:2407.00320*.

Chaojie Wang, Yanchen Deng, Zhiyi Lyu, Liang Zeng, Jujie He, Shuicheng Yan, and Bo An. 2024b. Q*: Improving multi-step reasoning for llms with deliberative planning. *arXiv preprint arXiv:2406.14283*.

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. 2023. Mathcoder: Seamless code integration in llms for enhanced mathematical reasoning. *arXiv preprint arXiv:2310.03731*.

Ruida Wang, Jipeng Zhang, Yizhen Jia, Rui Pan, Shizhe Diao, Renjie Pi, and Tong Zhang. 2024c. Theoremllama: Transforming general-purpose llms into lean4 experts. *arXiv preprint arXiv:2407.03203*.

Tianduo Wang, Shichen Li, and Wei Lu. 2024d. Self-training with direct preference optimization improves chain-of-thought reasoning. *arXiv preprint arXiv:2407.18248*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Tianwen Wei, Jian Luan, Wei Liu, Shuang Dong, and Bin Wang. 2023. Cmath: Can your language model pass chinese elementary school math test? *arXiv preprint arXiv:2306.16636*.

Haoyi Wu, Wenyang Hui, Yezeng Chen, Weiqi Wu, Kewei Tu, and Yi Zhou. 2023. Conic10k: A challenging math problem understanding and reasoning dataset. *arXiv preprint arXiv:2311.05113*.

Aiyuan Yang, Bin Xiao, Bingning Wang, Borong Zhang, Ce Bian, Chao Yin, Chenxu Lv, Da Pan, Dian Wang, Dong Yan, and 1 others. 2023. Baichuan 2: Open large-scale language models. *arXiv preprint arXiv:2309.10305*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Yao-Ching Yu, Chun-Chih Kuo, Ziqi Ye, Yu-Cheng Chang, and Yueh-Se Li. 2024. Breaking the ceiling of the llm community by treating token generation as a classification for ensembling. *arXiv preprint arXiv:2406.12585*.

Xiaotian Zhang, Chunyang Li, Yi Zong, Zhengyu Ying, Liang He, and Xipeng Qiu. 2023. Evaluating the performance of large language models on gaokao benchmark. *arXiv preprint arXiv:2305.12474*.

Zhihan Zhang, Tao Ge, Zhenwen Liang, Wenhao Yu, Dian Yu, Mengzhao Jia, Dong Yu, and Meng Jiang. 2024. Learn beyond the answer: Training language models with reflection for mathematical reasoning. *arXiv preprint arXiv:2406.12050*.

James Xu Zhao, Yuxi Xie, Kenji Kawaguchi, Junxian He, and Michael Qizhe Xie. 2023. Automatic model selection with large language models for reasoning. *arXiv preprint arXiv:2305.14333*.

Zihao Zhou, Maizhen Ning, Qiufeng Wang, Jie Yao, Wei Wang, Xiaowei Huang, and Kaizhu Huang. 2023. Learning by analogy: Diverse questions generation in math word problem. *arXiv preprint arXiv:2306.09064*.

A Data Description

We employ the **Conic10K** benchmark dataset, a publicly available resource distributed under the permissive MIT License (Copyright © 2023 Alessandro Ristori), permitting unrestricted academic use. The **Conic10K** dataset is constructed from open-sourced conic section questions collected from two Chinese high school education websites, originally presented in image format. Each question image contains the question text, rationale, and answer. The text in each image has been extracted by the dataset authors to construct a structured, text-based dataset. The dataset comprises 9,826 questions in total, with 7,757 allocated to the training set and 2,069 to the test set. To enable more fine-grained analysis in terms of numerical computation, expression parsing, and structural modeling, we divide the answers into 10 categories as shown in Appendix Table 2. The distribution of different categories is shown in Appendix Figure 2.

Category	Examples	Description
Sign Number	$y = pm * x, pm * 2$	Contains a positive and negative sign
Equation	$x^2 + y^2 = 1$	Equations
Set Text	$\{0, 1\}$	Contains multiple answers
Interval Text	$[-1, 1], (1, 0)$	Intervals and coordinate
Simple Number	$1/3, \sqrt{5} - 1$	Numerical values
Degree	'ApplyUnit(120, degree)'	An angle values
Type Analysis	'ellipse'	Curve classification
Void Answer	' '	Data without answers
Parameter Expression	a/3	Contains parameters
Multiple Answers	'-1; [-1,2]'	Text contains multiple questions

Table 2: Answer categories with examples and description.

B Model setting

Appendix Table 3 summarizes the models used in our experiments, detailing key attributes such as model size, multilingual capability, primary language orientation (e.g., Chinese-oriented or English-oriented), and the types of tasks (e.g., zero-shot, code-based, or fine-tuned). Appendix Table 4 presents our prompt designs for the MathQA tasks, covering both zero-shot and code-based settings, and outlines the instructions provided at each interaction step.

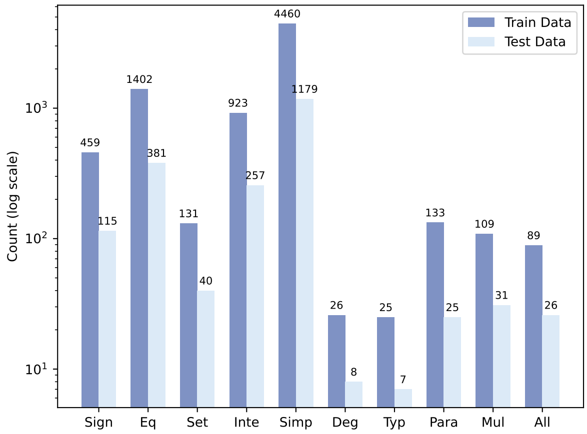

Figure 2: Log-scaled distribution of the train and text data on answer categories.

C Hardware and Software Environment

All experiments were conducted on a system equipped with NVIDIA RTX 4090 GPUs and 24 GB of RAM for fine-tuning. Model evaluation on the test set was performed using the CPU only. The computational workflow comprised two phases: (1) Fine-tuning was performed for Llama and Qwen-7B (32 total GPU-hours at 8 hours per experiment), and (2) Model evaluation was conducted on CPU nodes (approximately 60 compute-hours).

The implementation was based on PyTorch, with support from additional libraries such as Hugging Face Transformers and the OpenAI API for model evaluation.

D Error Analysis

During our evaluation on the **Conic10K** dataset, manual inspection reveals that LLMs make a wide range of errors, reflecting not only deficiencies in mathematical reasoning but also in fundamental understanding. These errors can be broadly categorized as follows:

(1) Code-to-math translation errors — failures in converting between programming syntax and algebraic expressions, such as misusing variable names,

Model	Size	Multi-language	Chinese-Oriented	Tasks
GPT-3.5-turbo	Unknown	✔	✗	z/c
Baichuan2-chat	7B	✔	✔	z
ChatGLM	9B	✗	✔	z/c
Llama 3-Chinese	7B	✔	✔	z/c/f
Qwen	7B	✔	✔	f/t
Qwen	72B	✔	✔	z/c
Atom	7B	✔	✔	z
DeepSeek-R1	660B	✔	✔	z/c

Table 3: **Models used in our experiments.** Chinese-oriented refers to whether methods such as increasing the proportion of Chinese data and designing a tokenizer specifically for Chinese are employed to enhance performance on Chinese tasks. Task describes the specific tasks for which the model is used, where **z** stands for MathQA zero-shot, **c** stands for MathQA with code, **f** stands for fine-tuning with **CONIC10K**, and **t** stands for MathQA with low temperature.

Table 4: Prompts used in different MathQA tasks

Task Type	Step	Prompt Content
	System	You are a talented mathematician
MathQA zero-shot	Question	"Math problem from **CONIC10K**"
	Organize Answersg	Please provide your final answer and ensure that this answer is in its simplest form, without any additional content; just output the answer itself.
MathQA with code	System	You are a programmer, you use python, sympy to solve problems, set variables, set an Interval to calculate the range of questions, expressions, and use solve() function to calculate. Only give a python code. (code cannot have external inputs and graphical outputs)
	Question	"Math problem from **CONIC10K**"
	No Code	Please try to solve the problem using code, "math problem from **CONIC10K**"
	Run Time Error	"Error information", Please check the code carefully.

misinterpreting math functions, or misunderstanding abstract formulations. (2) Common-sense mistakes — incorrect application of known mathematical rules or misuse of standard terminology. (3) Case analysis failures — Omission of necessary conditional reasoning, such as neglecting domain restrictions or failing to consider sign-based cases. (4) Boundary-related errors — misjudging open or closed intervals, or incorrectly handling endpoint inclusion.

In the following subsections, we present representative examples of several error types to illustrate common failure patterns in LLM-generated solutions.

D.1 Code-to-math translation errors

The problem of transformation between programming languages and algebraic languages exists, such as the following math problem:

> **It is known that** O is the origin of the coordinates, the focus of the parabola C:$y^2 = 2px(p > 0)$ is F, and the P is a point above C, PF is perpendicular to the x axis, Q is a point on the x axis, and $PQ \perp OP$. If $|FQ| = 6$, then the alignment equation for C is?

The code provided by LLM go to Appendix Code 1:

Listing 1: Wrong transform(By QWEN 72B)

```python
from sympy import symbols, Eq, solve
# Define symbols
p = symbols('p', positive=True)
# Given |FQ|
FQ = 6
# For the parabola y^2 = 2px, focus F is at (p/2, 0)
# Since PF is perpendicular to the x-axis, P has coordinates (p/2, sqrt(2p*(p/2))) = (p/2, p)
# PQ is perpendicular to OP, so Q is on the x-axis and has coordinates (p, 0)
# The distance from F to Q is given as |FQ| = 6
# Thus, |FQ| = |p - p/2| = |p/2| = 6, which gives us p = 12
# Calculate p
p_value = solve(Eq(p / 2, FQ), p)[0]
# The directrix or latus rectum of the parabola is at x = -p/2
directrix = -p_value / 2
# Equation of the directrix
print(directrix)
```

In this problem, the model is expected to derive the algebraic equations for the conic sections. However, it instead outputs a numerical value. Specifically, the question asks to generate the linear equations for the parabola, but the model only computes and outputs the value of the directrix ($x = -p/2$) without expressing it symbolically.

This failure likely stems from the model's confusion between numerical computation and symbolic expression generation. Although some intermediate reasoning steps are partially correct, the model prematurely terminates the process at a numeric result. This behavior reflects a limited understanding of task intent, particularly the distinction between symbolic synthesis and numerical solving, when symbolic reasoning and numerical evaluation are interwoven.

D.2 Knowledge-based errors

Model: Qwen-72B

math problem: Knowing that any point P on the parabola $y^2 = 4x$, the distance from the point P to the y axis is d, and for a given point $A(4, 5)$, then What is the minimum value of $|PA|+d$?

answer:$\sqrt{34} - 1$

LLM answer: To solve this problem, we can make use of the geometric properties of parabolas and the methods of analytic geometry. The given parabolic equation is $y^2 = 4x$, which is a standard parabola with an opening to the right with its focus at $(1, 0)$ and a collimation of $x = -1$.

For any point on the parabola $P(x, y)$, the distance d from the y axis is $|x|$. However, under the up-and-down symmetry of the parabola, x is always non-negative, so $d = x$.

Requires $|PA|$The minimum value of d, where $A(4, 5)$, we can observe that according to the definition of a parabola, the distance from the point P to the focal point is equal to the distance from the point P to the alignment. This means that if we are able to convert $|PA|d$ is converted to the sum of the distances from the point P to a certain point, we can use this property to simplify the problem.

Bearing in mind that the focal point of the parabola is $F(1, 0)$, then the distance from point P to the focal point F is equal to the distance from point P to the alignment $x = -1$. Therefore, $|PF| = x1$.

Now, we're looking for $|PA|d = |PA|$, the minimum value of x. Note, $|PA|x$ can actually be seen as the sum of the horizontal distances from the point A to the point P to the alignment. This can be further explained as the length of the path from A to the point P on the parabola and from P in a direction parallel to the alignment.

In order to make this path the shortest, according to the "law of reflection" in geometry (the light rays start from one point and are reflected to another point and have the smallest path length), we want this path to be like the light rays that start from A and are reflected by the alignment and reach the focal point F. This is because when the P point is in the direction of the extension line from A to F, $|PA|x$ has the smallest value.

So, we calculate the distance from A to F, as $|PA| The minimum value of x$. The distance from the point $A(4, 5)$ to the focus $F(1, 0)$ is

$$|AF| = \sqrt{(4-1)^2(5-0)^2} = \sqrt{925} = \sqrt{34}$$

So, $|PA|$ The minimum value of d is $\boxed{\sqrt{34}}$.

The analysis is correct, but LLM ignores a fact: the distance from the point on the parabola to the alignment is $|x| + 1$, not $|x|$!

This is a representative example of a *knowledge-based reasoning error*, where the model adopts a valid high-level strategy but fails to apply a fundamental mathematical fact—in this case, the correct expression for the distance from a point to a vertical line (i.e., $|x-(-1)| = x+1$). Unlike purely logical failures, this type of error arises from incomplete or imprecise recall of domain-specific definitions.

Such mistakes are common in LLMs, whose reasoning is built on surface-level statistical associations rather than grounded symbolic manipulation. While the model may imitate expert-like problem-solving patterns, it lacks persistent internal representations of formal mathematical concepts. Consequently, it often exhibits *partially correct but incomplete* reasoning: structurally coherent, yet factually flawed due to misapplied or omitted definitions.

This case illustrates a broader challenge in LLM-based mathematical reasoning: even when global strategies are appropriate, success hinges on local factual precision. Minor definitional inaccuracies—particularly in geometry or algebra—can lead to incorrect conclusions despite otherwise coherent reasoning paths.

D.3 Case analysis failures

Model: Qwen-72B

math problem: The set point $F_1(0,-3)$, $F_2(0,3)$, and the point P satisfies the condition $|PF_1|+|PF_2| = t+\frac{9}{t}$ (t is constant, and $t>0$), then what is the trajectory of the point P?

answer: Line segment F1F2 or ellipse

LLM answer: Due to $|PF_1|+|PF_2| = t+\frac{9}{t}$ is a constant ($t>0$ is given by the question), and the value is greater than $F_1F_2 = 6$ (because according to the AM-GM inequality, $t+\frac{9}{t} \geq 2\sqrt{t\cdot\frac{9}{t}} = 6$, and the equal sign holds if and only if $t=3$). According to the definition of an ellipse, if the sum of the distances from a point on a plane to two fixed points (focal points) is constant, and this constant is greater than the distance between the two focal points, then the trajectory of this point is an ellipse. Thus, the trajectory of the point P is an ellipse focused on F_1 and F_2.

We can see in the analysis that one of the answers *segment F1F2* is missing!

This error illustrates a common tendency of LLMs: generalizing from dominant patterns, such as associating a constant distance sum with ellipses, without explicitly reasoning through conditional branches. In problems where the outcome depends on parameter thresholds or equality constraints (e.g., piecewise or boundary-sensitive definitions), the model often defaults to the most prototypical case. This pattern-matching behavior results in incomplete reasoning, particularly when special cases demand distinct logical treatment beyond general rules.

E With Code, We Get Better Answer!

When we ask the large model to output the solution code to the prompt, we find that there is model analysis in Appendix Code 2, and the purpose of asking the large model to give the code is to make it more accurate in the operation.

Listing 2: AI combines code without losing analysis(By QWEN 72B)

```
# Hyperbola asymptotes
m = 4/3

# Since c is the same for both conics,
    we find a^2 and b^2 for the
    hyperbola
# For a hyperbola, c^2 = a^2 + b^2,
    and the slopes of the asymptotes
    are b/a
# We know c and m=b/a, so we can solve
    for a^2 and b^2

# Set up equations based on the
    relationship between a, b, and c
    for the hyperbola
eq1 = Eq(a**2 + b**2, c_ellipse**2)
eq2 = Eq(b/a, m)
```

Here is a complete example next.

math problem: For the problem D.2 that LLM makes a mistake, the code helps it!

The code generated by LLM for this problem is as follows 3:

Listing 3: code for Q1(By QWEN 72B)

```
from sympy import symbols, diff, solve
    , sqrt, Abs
# Define the variable
x = symbols('x', real=True, positive=
    True)
# Given parabola equation y^2 = 4x, we
    express y in terms of x
y = 2*sqrt(x)
# Distance from point P on the
    parabola to point A(4,5)
PA = sqrt((x - 4)**2 + (y - 5)**2)
# Distance d from point P to the y-
    axis
d = Abs(x)
# The expression for |PA| + d
expression = PA + d
```

```python
# Take the derivative of the expression with respect to x
derivative = diff(expression, x)
# Solve the derivative equal to zero to find critical points
critical_points = solve(derivative, x)
# Check if there is a valid solution that minimizes the expression
if critical_points:
    min_value = expression.subs(x, critical_points[0])
else:
    # If no critical point is found within the domain, check the boundary conditions
    # For this problem, the boundary condition would be that as x approaches infinity,
    # which is not applicable here since we are looking for a minimum.
    # Thus, we assume there's a typo or misunderstanding in the problem statement.
    min_value = None
# Since we expect a numerical result and our approach involves symbolic computation,
# We will attempt to find the minimum value numerically by evaluating the expression
# at a point close to the critical point obtained symbolically.
if min_value is not None:
    min_value = min_value.evalf()
else:
    min_value = "Undefined under given constraints"
min_value
```