Learning How to Use Tools, Not Just When: Pattern-Aware Tool-Integrated Reasoning

Ningning Xu[2] **Yuxuan Jiang**[1] **Shubhashis Roy Dipta**[1]
[1]University of Maryland, Baltimore County
[2]University of Georgia
yuxuanj1@umbc.edu

Abstract

Tool-integrated reasoning (TIR) has become a key approach for improving large reasoning models (LRMs) on complex problems. Prior work has mainly studied when to invoke tools, while overlooking how tools are applied. We identify two common patterns: a *calculator-pattern* that uses code for direct computation, and an *algorithmic-pattern* that encodes problems as programs. Misaligned choices often cause failures even when reasoning is sound. We propose a two-stage framework that first builds code competence from both patterns and then aligns pattern selection with teacher preferences. Across challenging math datasets, our pattern-aware method substantially improves both code usage and accuracy—for instance, raising Code@1 on MATH500 from 64.0% to 70.5% and on AIME24 from 26.7% to 50.0%. These gains highlight the effectiveness of a pattern-aware approach for tool-integrated reasoning.

1 Introduction

Tool-integrated reasoning (TIR) has become a powerful paradigm for enhancing large reasoning models (LRMs) [19, 4, 21]. By interacting with external verifiers such as code interpreters, TIR enables models to produce executable reasoning steps, making answers more accurate, faithful, and verifiable [23]. Recent progress has shown gains from better timing of tool invocation, multi-round tool calls, and tighter integration of external computation [5, 19].

While effective, a critical question remains: **can models reliably decide how to use tools once invoked?** We provide an initial exploration of this problem. Models often misalign their tool-use strategies with problem demands, applying code in a mechanical and context-insensitive way that prevents the tool from working effectively. As Figure 1 illustrates, on the factorial problem $1000! \div (800! \times 2!)$ the model used the *calculator-pattern* and failed due to overflow, while on the "first 10-digit prime in π" problem it used the *algorithmic-pattern* and stalled without output. *In both cases, switching to the complementary pattern succeeds*, indicating that the failures stem from *pattern selection* rather than a lack of reasoning ability.

To address this challenge, we propose a two-stage learning framework. In the first stage, *code competence acquisition*, the model learns to generate reliable code from both calculator- and algorithmic-pattern data, without teacher preferences, ensuring stable tool use. In the second stage, *pattern preference alignment*, teacher signals guide the model to choose the more suitable pattern for each problem. Together, these stages equip LRMs to not only invoke tools effectively but also adapt their strategy of use.

Our experiments on challenging math benchmarks show substantial and consistent gains. The model's ability to produce executable code (@1) increases from 64.0% to 70.5% on

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: MATH-AI.

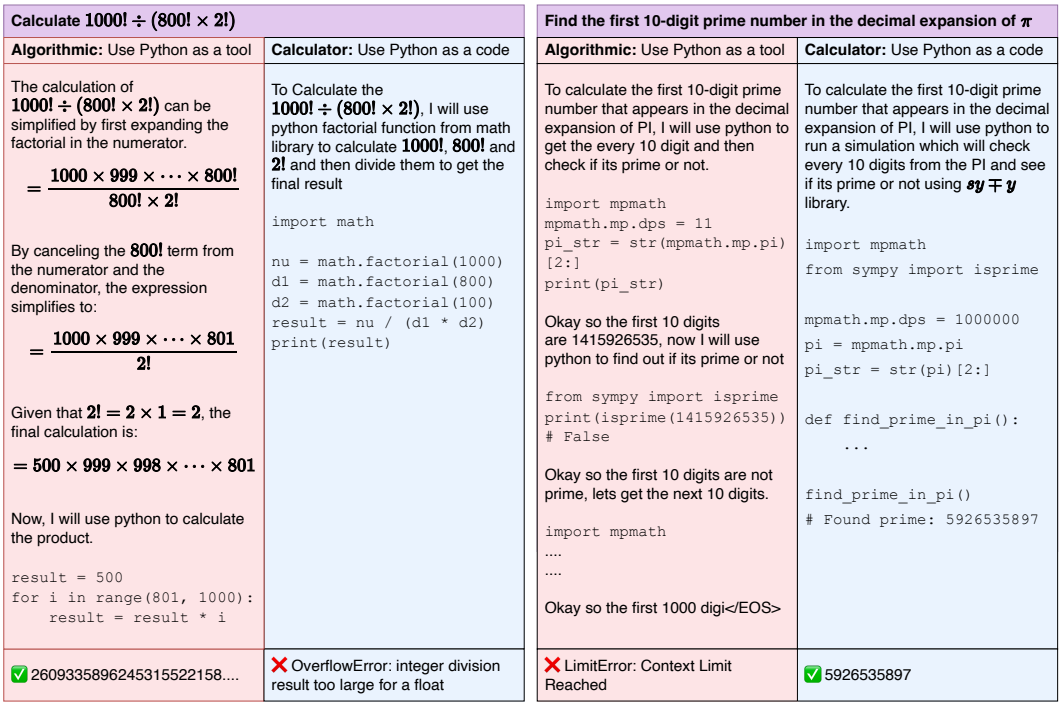

Figure 1: **Pattern mismatch in tool-integrated reasoning.** Left: in $1000! \div (800! \times 2!)$, an algebraic approach succeeds while direct computation overflows. Right: in finding the first 10-digit prime in π, a symbolic approach fails from context limits while a scanning approach succeeds. These cases show that success depends on the chosen tool-use pattern.

MATH500, indicating that *code competence acquisition* markedly stabilizes tool use. At the same time, overall problem-solving accuracy rises from 26.7% to 50.0% on AIME24, demonstrating that *pattern preference alignment* further improves strategy selection and reduces failures due to mismatched tool use. Together, these results validate that modeling *how* tools are used—beyond merely invoking them—yields large improvements in both code reliability and end-task performance.

This work makes three contributions. (1) We identify and formalize *pattern mismatch* in TIR, showing that misaligned tool-use strategies can derail otherwise sound reasoning. (2) We propose a two-stage framework that first builds code competence from both calculator- and algorithmic-pattern data (without teacher preferences) and then aligns pattern choice with teacher signals. (3) We provide empirical evidence on MATH500 and AIME24 that this approach substantially improves code@1 (64.0% → 70.5%) and accuracy (26.7% → 50.0%), and we illustrate how switching patterns flips failure cases into successes (Fig. 1).

2 Related Works

2.1 Reasoning with LRMs

LRMs have demonstrated better performance when explicitly generate intermediate reasoning steps[20]. Subsequent work has focused on improving the quality of these reasoning paths, either by introducing diversity through self-consistency [18], refining intermediate steps via process supervision [13, 11], or encouraging verification of intermediate results [22, 17].

2.2 Tool-Integrated Reasoning

Tool-integrated reasoning has emerged as a powerful paradigm for enhancing large reasoning models (LRMs) [19, 4], enabling executable reasoning steps through external tools such as Python interpreters [23]. Early work like PoT [3], PAL [6], and MathPrompter [10] showed

that converting reasoning into code execution or lightweight snippets can substantially improve math performance, while implementations such as ToRA [8] and Qwen2.5-Math-Instruct-TIR [16] further demonstrated gains from deeper integration. However, prior studies primarily focus on *when* to call tools, overlooking *how* tools are applied once invoked. We address this gap by modeling pattern selection in TIR, introducing a pattern-aware framework that aligns tool-use strategies and yields significant improvements in reasoning accuracy.

3 Method

We train a pattern-aware reasoning model in two stages. For stage 1, we builds code competence by supervising the model on both calculator-style and algorithmic-style solutions. Stage 2 performs pattern preference alignment, using Direct preference optimization (DPO) to learn the pattern preference.

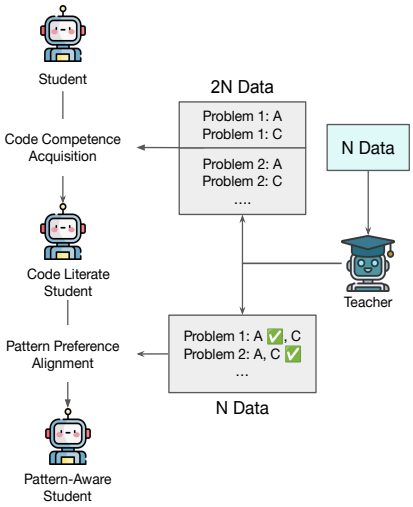

Figure 2: **Two-stage training framework.** In the 1st phase, each problem is expanded into both calculator- and algorithmic-style solutions ($2N$ data) to build code competence, and in the 2nd phase, teacher-provided preferences on N problems guide the student to become pattern-aware.

3.1 Training set construction

We distinguish two usage patterns in tool-integrated reasoning: the calculator-pattern, where code is used only for direct computation or verification, and the algorithmic-pattern, where problems are expressed as full programs. For each problem, we generate a Chosen Solution with the more suitable pattern and a Counter Solution with the alternative one. This dual construction not only equips the model with competence in both styles of code usage, but also provides the supervision necessary to align its pattern selection with teacher preferences, thereby improving robustness and accuracy in downstream reasoning tasks. And We put the complete version is provided in Section A.

3.2 Code Competence Acquisition

For each problem x, we construct two solutions:

- an algorithmic-style solution y^{alg} that converts the problem to an execuable program;
- a calculator-style solution y_{calc} that keeps the reasoing path while using python for arithmetic or verification.

We write both solutions with the same ouput shcema (reasoning $\rightarrow$ python block $\rightarrow$ ouputs $\rightarrow$ final answer), which can simplifies parsing and execution. Let $D_{SFT} = \{(x_i, y_i^{alg}), (x_i, y_i^{calc})\}$ be the set of both patterns, we minimize the negative log-likelihood over response tokens:

$$\mathcal{L}_{\text{SFT}} = -\mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{SFT}}} \sum_{t=1}^{|y|} \log \pi_\theta(y_t \mid x, y_{<t}),$$

3.3 Pattern Preference Alignment

We use DPO [15] to empower our model with the ability to pick the correct pattern to solve math problems. For each problem x, we collect two candidate completions y^{alg}, y^{calc} and label a winner y^+ and loser y^- using the teacher model. For a pair (x, y^+, y^-), we then apply direction preference optimization with a frozen reference policy π_{ref} (our SFT model):

$$\mathcal{L}_{\text{DPO}} = -\log \sigma\Big(\beta\big[\log \pi_\theta(y^+ \mid x) - \log \pi_\theta(y^- \mid x) - \big(\log \pi_{\text{ref}}(y^+ \mid x) - \log \pi_{\text{ref}}(y^- \mid x)\big)\big]\Big),$$

where β scales preference strength and σ is the logistic function. This increases the likelihood of preferred (pattern-appropriate) solutions relative to dispreferred ones, without training a separate reward model.

4 Experiments

4.1 Dataset

We use google Gemini-2.5-flash-lite as our teacher model, using our designed two pattern prompt to solve the problems in OpenR1-Math-220k [14] dataset. From this corpus, we select the first 10,000 problems and divide them into training and validation sets using a 9:1 ratio. To assess complex mathematical reasoning and generalization, we evaluate on a broad set of out-of-domain benchmarks, including MATH500 [9], AIME24 [1], and AMC23 [2].

4.2 Evaluation Metrics

We report **Pass@1** as the primary accuracy metric. For additional insight into reasoning behavior, we also track (i) the proportion of problems where the model's output contains executable code (**Code@1**), and (ii) the proportion of problems where such code is both present and leads to the correct final answer (**Code+Pass@1**).

5 Main Results

Method	R1Math		GSM8K (OOD)		MATH500 (OOD)		AIME24 (OOD)	
	Code@1	Code+Pass@1	Code@1	Code+Pass@1	Code@1	Code+Pass@1	Code@1	Code+Pass@1
R1-Distill-Qwen-1.5B								
Base	3.0%	2.0%	4.2%	4.2%	8.0%	6.0%	0.0%	0.0%
+SFT	66.0%	55.5%	72.0%	70.4%	64.0%	58.2%	26.7%	6.6%
+SFT+DPO	**72.2%**	**70.3%**	**80.1%**	**77.4%**	**70.5%**	**63.2%**	**50.0%**	**26.7%**

Table 1: Pass@1 and Code+Pass@1 accuracy on R1-Distill-Qwen models across R1Math, GSM8K, MATH500, and AIME24.

According to Table 1, the base model achieves near-zero code usage and accuracy. With SFT, the model begins to leverage Python extensively (e.g., R1Math Code@1 rises to 66%), and correctness follows accordingly. Adding DPO yields further substantial improvements, notably pushing GSM8K Code@1 from 72.0% to 80.1% and AIME24 from 26.7% to 50.0%, while also boosting correctness (e.g., AIME24 Code+Pass@1 from 6.6% to 26.7%). These results highlight that SFT is crucial for enabling tool use, whereas DPO markedly enhances both adoption and accuracy.

6 Conclusion

In this work, we studied two complementary tool-use patterns—algorithmic code generation and calculator-style usage—and showed that many failures arise from mismatched pattern choices rather than reasoning ability. Experiments on GSM8K, MATH500, and AIME24 demonstrate that our pattern-aware approach improves strategy selection and accuracy, offering an effective and lightweight way to enhance tool-integrated reasoning.

References

[1] AI-MO Team. AIMO Validation Set - AIME Subset. https://huggingface.co/datasets/AI-MO/aimo-validation-aime, 2024.

[2] AI-MO Team. AIMO Validation Set - AMC Subset. https://huggingface.co/datasets/AI-MO/aimo-validation-amc, 2024.

[3] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

[4] Yongchao Chen, Yueying Liu, Junwei Zhou, Yilun Hao, Jingquan Wang, Yang Zhang, and Chuchu Fan. R1-code-interpreter: Training llms to reason with code via supervised and reinforcement learning. *arXiv preprint arXiv:2505.21668*, 2025.

[5] Guanting Dong, Yifei Chen, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Yutao Zhu, Hangyu Mao, Guorui Zhou, Zhicheng Dou, and Ji-Rong Wen. Tool-star: Empowering llm-brained multi-tool reasoner via reinforcement learning. *arXiv preprint arXiv:2505.16410*, 2025.

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

[7] Mingqi Gao, Jie Ruan, Renliang Sun, Xunjian Yin, Shiping Yang, and Xiaojun Wan. Human-like summarization evaluation with chatgpt. *arXiv preprint arXiv:2304.02554*, 2023.

[8] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. Tora: A tool-integrated reasoning agent for mathematical problem solving. *arXiv preprint arXiv:2309.17452*, 2023.

[9] 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. *arXiv preprint arXiv:2103.03874*, 2021.

[10] Shima Imani, Liang Du, and Harsh Shrivastava. Mathprompter: Mathematical reasoning using large language models. *arXiv preprint arXiv:2303.05398*, 2023.

[11] Yuxuan Jiang, Dawei Li, and Frank Ferraro. Drp: Distilled reasoning pruning with skill-aware step decomposition for efficient large reasoning models. *arXiv preprint arXiv:2505.13975*, 2025.

[12] Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, et al. From generation to judgment: Opportunities and challenges of llm-as-a-judge, 2025. *URL https://arxiv. org/abs/2411.16594*, 2025.

[13] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

[14] OpenR1 Team. Openr1-math-220k. https://huggingface.co/datasets/open-r1/OpenR1-Math-220k, 2025. Accessed: 2025-09-25.

[15] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

[16] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[17] Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

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

[19] Yifan Wei, Xiaoyan Yu, Yixuan Weng, Tengfei Pan, Angsheng Li, and Li Du. Autotir: Autonomous tools integrated reasoning via reinforcement learning. *arXiv preprint arXiv:2507.21836*, 2025.

[20] Fengli Xu, Qianyue Hao, Zefang Zong, Jingwei Wang, Yunke Zhang, Jingyi Wang, Xiaochong Lan, Jiahui Gong, Tianjian Ouyang, Fanjin Meng, et al. Towards large reasoning models: A survey of reinforced reasoning with large language models. *arXiv preprint arXiv:2501.09686*, 2025.

[21] Yiyao Yu, Yuxiang Zhang, Dongdong Zhang, Xiao Liang, Hengyuan Zhang, Xingxing Zhang, Ziyi Yang, Mahmoud Khademi, Hany Awadalla, Junjie Wang, et al. Chain-of-reasoning: Towards unified mathematical reasoning in large language models via a multi-paradigm perspective. *arXiv preprint arXiv:2501.11110*, 2025.

[22] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

[23] Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. *arXiv preprint arXiv:2504.03160*, 2025.

A Complete Prompt

Double Pattern Prompt

Use a python interpreter as a tool to solve the following math problem.
There are two possible usage patterns:

- *Pattern A: Treat the problem as a coding problem, write a complete solution using Python code.*
- *Pattern B: Treat the Python interpreter as a simple calculator, only using it for arithmetic or verification when needed.*

Your task:

1. *Decide which pattern (A or B) is more appropriate for this problem. Call this the* ***Chosen Solution****.*
2. *Provide the* ***Chosen Solution*** *with EXACTLY ONE continuous reasoning paragraph (no lists, no numbering, no bullet points). Then give ONE Python code block, then the code outputs, then the final answer.*
3. *Provide the* ***Counterfactual Solution*** *using the other pattern with the SAME constraints (one continuous reasoning paragraph, one code block, outputs, final answer).*
4. *Strict constraints: - Do NOT restate the problem text. - Do NOT include introductions, apologies, or meta comments. - Do NOT duplicate any content across fields.*

Output format (pure JSON, no backticks, no extra keys):

```
{
  "problem": "<the math problem here>",
  "chosen_pattern": "A or B",
  "chosen_solution": {
     "reasoning": "<one continuous paragraph>",
     "code_blocks": ["<one python block only>"],
     "outputs": ["<stdout lines>"],
     "final_answer": "..."
  },
  "counter_solution": {
     "reasoning": "<one continuous paragraph>",
     "code_blocks": ["<one python block only>"],
     "outputs": ["<stdout lines>"],
     "final_answer": "..."
  }
}
```

B Evaluate Pattern Chosen

To further examine the reliability of Gemini's pattern selection, we asked GPT-5 to independently select patterns without prior exposure to Gemini's choices. Across 100 test cases, GPT-5's selections were consistent with Gemini's in 98 instances, corresponding to a 98%

agreement rate. This high level of concordance provides strong empirical support for the robustness of our approach. This follows LLM as a Judge [12, 7] method.