TEACHING THINKING MODELS TO REASON WITH TOOLS: A FULL-PIPELINE RECIPE FOR TOOL-INTEGRATED REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-integrated reasoning (TIR) offers a direct way to extend thinking models beyond the limits of text-only reasoning. Paradoxically, we observe that tool-enabled evaluation can degrade reasoning performance even when the strong thinking models make almost no actual tool calls. In this paper, we investigate how to inject natural tool-use behavior into a strong thinking model without sacrificing its no-tool reasoning ability, and present a comprehensive TIR recipe. We highlight that (i) the effectiveness of TIR supervised fine-tuning (SFT) hinges on the learnability of teacher trajectories, which should prioritize problems inherently suited for tool-augmented solutions; (ii) controlling the proportion of tool-use trajectories could mitigate the catastrophic forgetting of text-only reasoning capacity; (iii) optimizing for *pass@k* and response length instead of training loss could maximize TIR SFT gains while preserving headroom for reinforcement learning (RL) exploration; (iv) a stable RL with verifiable rewards (RLVR) stage, built upon suitable SFT initialization and explicit safeguards against mode collapse, provides a simple yet remarkably effective solution. When applied to Qwen3 thinking models at 4B and 30B scales, our recipe yields models that achieve state-of-the-art performance in a wide range of benchmarks among open-source models, such as 96.7% and 99.2% on AIME 2025 for 4B and 30B, respectively.

1 INTRODUCTION

Current thinking models have demonstrated that scaling the reasoning depth can substantially improve performance on difficult tasks with pure natural language (OpenAI et al., 2024; DeepSeek-AI et al., 2025; Cui et al., 2025a; Zhang et al., 2025a; Zeng et al., 2025). However, in more complex scenarios that frequently require formal reasoning (e.g., numerical computation, symbolic manipulation, and systematic case enumeration), text-only reasoning often struggles, making it both costly and less reliable (Li et al., 2026; Zhang et al., 2025b; Li et al., 2025a). TIR addresses these limitations by interleaving natural-language deliberation with code execution (Gao et al., 2023; Feng et al., 2025), combining flexible reasoning with precise computational feedback.

The potential of TIR is clear in principle, yet simply providing a thinking model with access to a code executor does not guarantee interleaved reasoning. For example, as illustrated in Figure 1, the model invokes the executor only once at the very end to verify the final answer, rather than leveraging computation to guide intermediate reasoning steps. This behavior reflects the delayed-code pattern (Li et al., 2025a; Du et al., 2025b), where code is applied too late or even not at all, rather than being integrated into the reasoning loop. We identify the core bottleneck as the lack of a reliable policy for determining when reasoning should hand off to computation and how to incorporate the results back into the reasoning process.

To address this challenge, prior work has explored supervised fine-tuning on tool-using trajectories and reinforcement learning that rewards successful tool invocation (Gao et al., 2023; Feng et al., 2025; Li et al., 2025a; Xue et al., 2025). However, these approaches often overlook the structural fragility of reasoning chains during tool integration. Our empirical analysis reveals a critical bottleneck: thinking models exhibit a sharp performance drop when prompted with tools, indicating that their internal reasoning logic is easily disrupted (Appendix B.3). Teaching thinking models to reason with tools

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

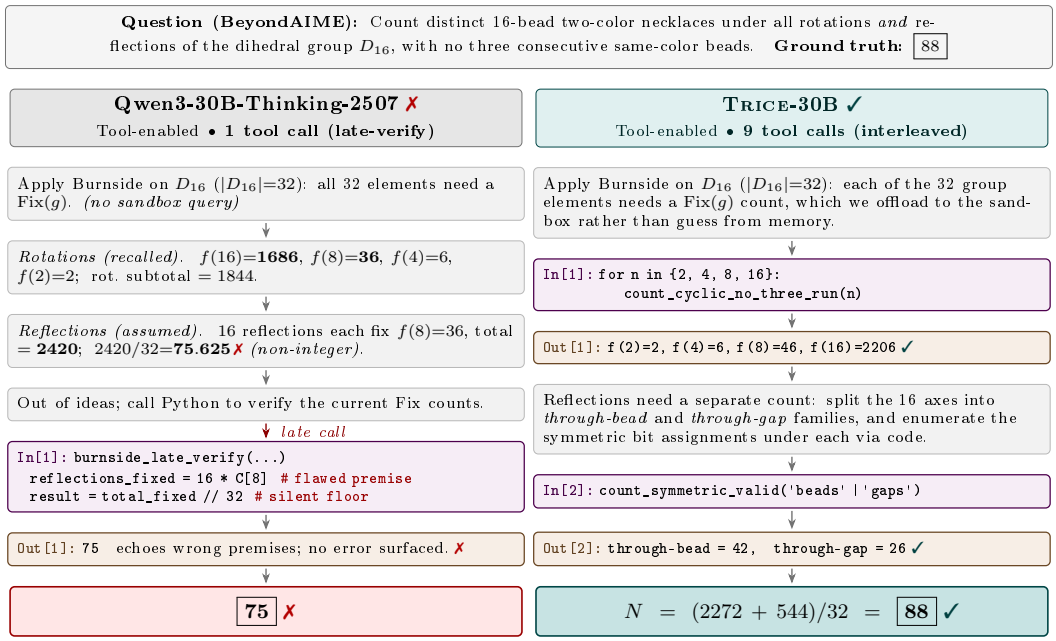


Figure 1: **The same problem, two policies for invoking the tool.** Grey boxes denote text-only reasoning; `In[k]` cells indicate tool calls, and `Out[k]` cells represent tool responses. **Left:** Qwen3-30B-Thinking-2507 treats the Python sandbox as a final-pass verifier. After a text-only Burnside derivation yields the inconsistent value $2420/32 = 75.625$, it makes a single late-stage call that hard-codes the flawed assumptions (e.g. `reflections_fixed = 16*C[8]`). A silent integer floor masks the error, leading to the incorrect result $\boxed{75}$ ✗. **Right:** TRICE-30B interleaves textual reasoning with code execution, feeding each intermediate result back into the same Burnside framework, and correctly obtains $\boxed{88}$ ✓.

therefore requires learning a reliable interleaved policy while preserving their text-only reasoning capabilities. To resolve this inherent instability, we move beyond scattered techniques and propose a systematic, full-pipeline recipe spanning data preparation, SFT, the transition from SFT to RL, and RL itself. The recipe is organized as the following components:

- **Data Engineering:** For each training stage, we carefully build data collection and filtering pipelines. For teacher prompts, we find that *tool-advantaged* problems could better encourage tool use. For teacher trajectories, we find that effective supervision depends on the learnability of tool-use patterns, and that mixing TIR and text-only trajectories helps prevent catastrophic degradation of text-only reasoning. Based on these findings, we construct high-quality teacher trajectories for SFT training. Moreover, we construct a challenging RLVR dataset to naturally elicit tool usage patterns in RL training.

- **Stage Coordination:** We identify that what the student learns during SFT evolves through form, substance, and noise, and argue that SFT training should prioritize downstream RL potential over standard supervised loss. To this end, we utilize *pass@k* performance and rollout length to determine the optimal SFT extent for maximizing subsequent RL gains.

- **Reinforcement Learning:** We show that a suitable SFT initialization and on-policy training stabilize multi-turn tool use, improving both with-tool and no-tool performance. We forgo the format rewards and leverage outcome-based rewards only with challenging questions to naturally elicit tool use. Applied to Qwen3 thinking models at 4B and 30B scales, the resulting models, capable of **T**extual **R**easoning **I**nterleaved with **C**ode **E**xecution, termed TRICE-4B and TRICE-30B, achieve state-of-the-art performance among open-source models of this scale in a wide range of benchmarks, such as reaching 96.7% and 99.2% on AIME 2025, respectively.

2 Related Work

Reinforcement learning for large reasoning models. Reinforcement learning has made remarkable progress for building large reasoning models (OpenAI et al., 2024; Cui et al., 2025a;b; Yuan et al., 2025; Zuo et al., 2025). By optimizing with verifiable rewards, it encourages LLMs to scale up their reasoning processes, thereby advancing the frontier of intelligence (Chen et al., 2025; DeepSeek-AI et al., 2025; Zhang et al., 2025a; Luo et al., 2026; He et al., 2025). However, text-only reasoning is inherently limited for problems that require exact numerical computation or rigorous verification. Therefore, external tools that compensate for these limitations can be naturally integrated into the reasoning process (Yu et al., 2025; Du et al., 2025a; Dong et al., 2025; Lin & Xu, 2025; Wang et al., 2025).

Tool-integrated reasoning. Although introducing executable tools into reasoning has shown clear feasibility, injecting this ability into an already strong thinking model is not straightforward: it requires reshaping a largely fixed policy (Li et al., 2026; Zhang et al., 2026b; Xia et al., 2025). Prior work focuses on different parts of this problem, including the cold-start tool trajectories, code placement, theoretical support expansion, RL stability, long-CoT tool discovery, interaction-dense priors, and tool-use pattern selection (Feng et al., 2025; Li et al., 2025a; Lin & Xu, 2025; Xue et al., 2025; Shang et al., 2025). Beyond the above work, we aim to convert an already strong thinking model into a reliable TIR model while maintaining or improving its no-tool performance through a full-pipeline recipe.

3 Preliminary

3.1 Reinforcement Learning with Verifiable Reward

Given a problem $x \sim \mathcal{D}$, an LLM policy π_θ autoregressively generates a serialized trajectory $y = (y_1, \ldots, y_T)$ and receives a terminal reward $r(x, y)$ from a verifier. The standard RL objective is

$$J(\theta) = \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)}\left[r(x, y)\right]. \tag{1}$$

Policy-gradient methods estimate its gradient by assigning an advantage to generated tokens (Sutton & Barto, 2018):

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(y_t \mid x, y_{<t}) A_t\right]. \tag{2}$$

With only a trajectory-level reward, REINFORCE can use $A_t = r(x, y)$ for all tokens. In practice, group-normalized variants such as GRPO sample K responses $\{y^{(k)}\}_{k=1}^{K}$ for the same prompt and use a normalized trajectory advantage,

$$A_t^{(k)} = \frac{r(x, y^{(k)}) - \mathrm{mean}_j\, r(x, y^{(j)})}{\mathrm{std}_j\, r(x, y^{(j)}) + \epsilon}, \tag{3}$$

which reduces variance and supports learning from sparse verifiable rewards (Shao et al., 2024).

3.2 Tool-Integrated Reasoning

Tool-integrated reasoning uses the same objective, but the trajectory y is produced through an interactive process before being verified. At interaction turn τ, the state is $s_\tau = (x, H_\tau, E_\tau)$, where H_τ is the current interaction history and E_τ is the sandbox state. The policy takes an action a_τ that is either natural-language reasoning or executable code. Text actions are appended directly to the history. Code actions are executed by the sandbox, and it returns an observation o_τ, such as printed output or an error message, so the next state is

$$s_{\tau+1} = (x, H_\tau \oplus a_\tau \oplus o_\tau, E_{\tau+1}). \tag{4}$$

When the model emits a final answer or reaches the turn budget, the final history is serialized as y and scored by the same verifier reward $r(x, y)$. We use a stateful sandbox, so variables, imports, and intermediate computations persist across tool calls; this lets the model inspect, revise, and extend prior computations rather than restart from an empty environment. The no-tool setting is the special case in which code actions are disallowed, so y is generated entirely through language.

4 Data Engineering for TIR

In this section, we detail the SFT data curation pipeline of our TIR recipe, focusing on the strategic selection of teacher models and the SFT trajectories.

4.1 Teacher Model Selection

Takeaway 1

For TIR SFT, teacher selection should account for the learnability of tool-use patterns, not teacher accuracy alone.

Teacher selection for TIR SFT should consider the learnability of tool-use patterns, not accuracy alone (Guha et al., 2025). We compare GPT-OSS-120B (OpenAI, 2025) and MiniMax-M2.7 (Team, 2026), considering both cost and tool-use capability.

Under the same stateful sandbox (Section 3), their trajectories exhibit distinct styles. GPT-OSS-120B follows a *high-frequency, lightweight* pattern, using short snippets that reuse prior variables and intermediate results. MiniMax-M2.7 instead favors *low-frequency, heavyweight* calls: longer, self-contained programs with repeated definitions and a lower tool-use rate.

Table 1: Teacher and student comparison on BeyondAIME with tool access. ↪ denotes Qwen3-4B-Thinking-2507 after four epochs of SFT on the preceding teacher's trajectories.

Model	Acc	Calls	Len
GPT-OSS-120B	82.5	12.1	22.9K
↪ Qwen3-4B	55.8	29.3	37.2K
MiniMax-M2.7	75.5	5.6	30.5K
↪ Qwen3-4B	17.8	1.2	68.3K

To compare the resulting supervision, we fine-tune Qwen3-4B-Thinking-2507 for four epochs using 16K trajectories from each teacher. The teachers differ by 7.0 percentage points on BeyondAIME, whereas their students differ by 38.0 points (55.8 vs. 17.8; Table 1), with GPT-OSS-120B supervision also yielding shorter responses. Short, incremental tool calls may be easier for a 4B student to learn: they reduce code-generation errors (Jiang et al., 2025), provide timely execution feedback, and reuse persistent state to avoid redundant computation. These considerations motivate our choice of GPT-OSS-120B.

4.2 SFT Data Selection

Takeaway 2

Selecting problems amenable to tool-augmented solutions (termed *tool-advantaged* problems) as teacher inputs can better elicit students to use tools from teacher trajectories.

After selecting the teacher, the next question is how to choose the prompts and trajectories. We answer this question from the following perspectives.

Prompt selection. First, we select the prompts for the teacher model to produce trajectories more suitable for students. The quality of TIR supervision varies across problems. Problems whose difficulty lies in computational structure tend to benefit more from tool access, while those centered on conceptual insight benefit less. Moreover, the teacher's tool-use policy itself is imperfect and may not leverage tools effectively on every problem. To select prompts likely to yield more informative trajectories, we compare the teacher's average accuracy over k rollouts with and without tool access per problem and retain those where TIR accuracy substantially exceeds the text-only reasoning, termed *tool-advantaged* problems. Compared with random prompt selection, tool-advantaged prompts improve with-tool HMMT25 accuracy by 5.4 percentage points (Table 2).

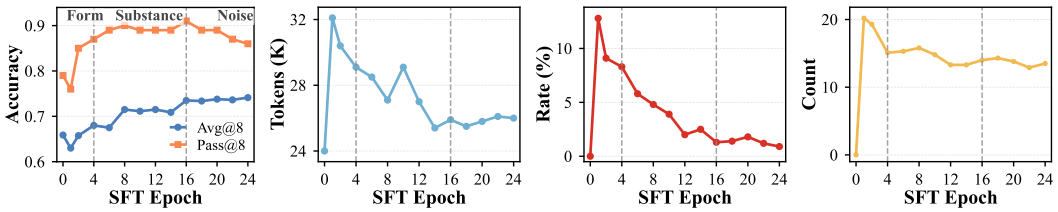
Takeaway 3

To preserve the student model's native text-only reasoning capability, mix text-only trajectories into the TIR set to avoid catastrophic forgetting

Mix TIR trajectories with text-only reasoning ones. Despite their strong TIR performance, models trained with TIR-only SFT exhibit substantial degradation in text-only reasoning: accuracy

Table 2: Data-selection ablations on Qwen3-4B-Thinking-2507 after four epochs of SFT. Bold highlights successive configuration changes. Full results are provided in Appendix C.1.

SFT configuration	With tool		Without tool	
	HMMT25	BeyondAIME	BeyondAIME	Trunc.
Base model	58.3	50.3	54.3	0.0
Random; TIR-only; 32K	69.6	52.3	35.5	41.8
Tool-adv.; TIR-only; 32K	75.0	53.5	34.1	43.8
Tool-adv.; **Mixed**; 32K	78.3	53.0	40.3	29.3
Tool-adv.; Mixed; **16K**	78.1	53.3	49.1	19.3



(a) Accuracy **(b)** Response length **(c)** Length trunc. rate **(d)** Tool call counts

Figure 2: TIR SFT dynamics of the 30B model. We perform SFT on Qwen3-30B-A3B-Thinking-2507 using expert data generated by GPT-OSS-120B. The observed learning curve (measured on BeyondAIME) demonstrates a "form–substance–noise" progression.

declines, while the truncation rate rises to 43.8%. We study these text-only reasoning trajectories (e.g., Appendix D.1) and attribute this to a cross-mode negative transfer: TIR-only SFT causes the model to internalize a code-centric reasoning loop that persists even when no tool is available. The model attempts to simulate code execution in natural language, for example, writing programs, fabricating outputs, and iterating on hallucinated results—ultimately exhausting the token budget without mathematical progress (Du et al., 2025a; Gao et al., 2025). To alleviate this phenomenon, we developed a simple but effective approach: mixing text-only trajectories into TIR trajectories. Mixing partially restores text-only accuracy while preserving TIR performance. An ablation of the TIR:text-only mixing ratio is provided in Appendix C.2.

TAKEAWAY 4

Filtering out overlong teacher trajectories not only improves efficiency for downstream RL, but also prevents the student from falling into the trap of imitating reasoning length.

Length filtering. Considering downstream RL rollout efficiency, we filter out teacher trajectories longer than 16K tokens to control student rollout length. With-tool accuracy remains comparable on HMMT25 (78.3% vs. 78.1%) and BeyondAIME (Table 2), while text-only BeyondAIME accuracy improves from 40.3% to 49.1% and truncation falls from 29.3% to 19.3%.

5 Stage Coordination: From SFT to RL

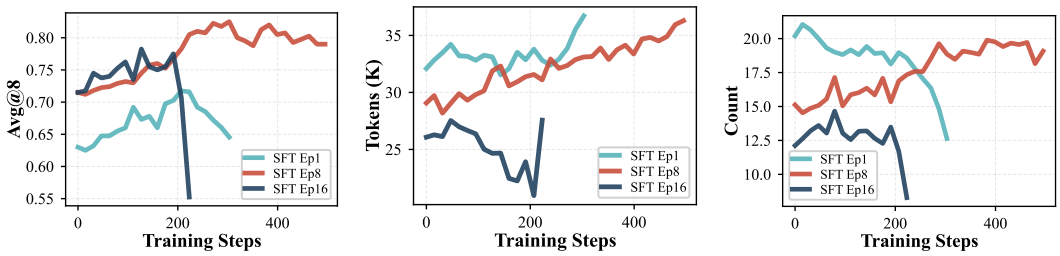
Based on the curated SFT data, this section details a principled execution of the fine-tuning process to fully unlock its potential within the holistic training pipeline.

TAKEAWAY 5

During TIR SFT, what the student learns evolves across form, substance, and noise. We determine the training extent based on *pass@k* performance and rollout length.

5.1 TIR SFT Dynamics

During TIR SFT, the student model follows a form–substance–noise learning progression, which we characterize as *stage 1-3*. In the early stage, it quickly acquires the format of tool invocation, causing



(a) Accuracy **(b)** Response length **(c)** Tool call counts

Figure 3: The RL training dynamics of the 30B model, EP is short for Epoch. Ep 1 is the model in *stage 1*, Ep 8 is the model in *stage 2*, while Ep 16 is the model in *stage 3*.

tool-use frequency to rise sharply without corresponding gains in reasoning accuracy. As shown in Figure 2, performance drops while the truncation rate increases. Case in Appendix D.2 further shows that early tool calls are often inefficient symbolic exploration, repeated verification, or loops, which accumulate errors and produce overly long reasoning trajectories. It suggests that the model is primarily imitating the form of TIR in this stage (*stage 1*). As training proceeds, tool-use frequency stabilizes, and tool calls more often provide useful intermediate evidence. Correspondingly, both pass@1 and pass@8 improve, indicating that the model moves beyond exploiting the superficial signal of tool-call frequency and begins to learn the substance of TIR, adapting tool use to its reasoning prior. This resembles the degradation–recovery dynamics observed in long-CoT SFT (Zhang et al., 2026a; Luo et al., 2025)(*stage 2*). Finally, after TIR behavior has been sufficiently internalized and useful supervision diminishes, the following SFT gradually overfits the teacher-side noise (e.g., rollout length), leading the performance to saturate or even decline (*stage 3*).

5.2 Identifying RL-Ready SFT Checkpoints

The learning pattern observed in the student model informs our choice of the SFT endpoint. Specifically, we aim to select a checkpoint that has sufficiently learned the substance of TIR, but has not yet absorbed excessive noise, as the initialization for RL. As discussed in Section 5.1, we find that the transitions between stages are closely associated with two diagnostics: *pass@k* performance and rollout length. We therefore use these metrics to guide checkpoint selection.

To validate this criterion, we initialize RL from checkpoints at different stages, namely epochs 1, 8, and 16. As shown in Figure 3, checkpoints from both *stage 1* and *stage 3* eventually collapse during RL training. The failure of *stage 1* is straightforward: the model has not yet learned an effective TIR strategy, but already invokes tools frequently. Under outcome-only rewards, RL further amplifies these erroneous patterns, ultimately leading to training collapse (Li et al., 2025b).

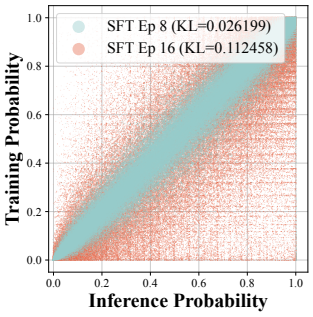
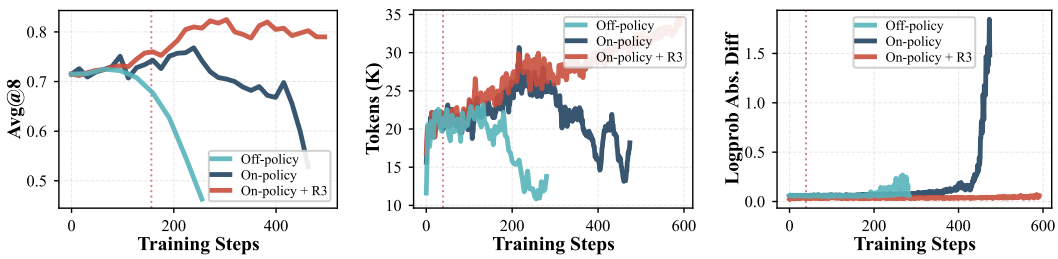


Figure 4: The token probabilities assigned by the inference and training engines to the same rollouts.

The failure of *stage 3* is more subtle, since its *pass@1* performance is even higher than that of the stage-2 checkpoint. We therefore conduct a deeper analysis of the stage-3 model. As shown in Figure 4, a salient difference in RL is that, compared with the stage-2 model, the stage-3 model has a $\sim$4 times larger train–inference mismatch. We attribute this to partial overfitting to the teacher model in *stage 3*, where the student begins to absorb teacher-side noise. Moreover, due to the off-policy nature of tool-return tokens in the context, the model becomes more prone to emitting extremely low-probability tokens, making RL optimization substantially more difficult (Liu et al., 2025; Zheng et al., 2025).



(a) Acc on BeyondAIME **(b)** Training resp. length **(c)** Train–rollout mismatch

Figure 5: Comparison of different RL training strategies. An off-policy setting (four updates per rollout) leads to collapse. On-policy updates stabilize training, and adding rollout routing replay (R3) further improves the final score while keeping the train–inference mismatch controlled.

5.3 Stable RL Training for TIR

Takeaway 6

For the off-policy nature of tool-call returns, stability must be a central consideration in TIR RL. Using on-policy rollouts together with rollout routing replay is simple but necessary.

In the RL stage, we do not introduce any reward that directly incentivizes tool invocation, since tuning its weight is difficult and may also induce reward hacking. Instead, we carefully construct a high-difficulty, IMO-level dataset from scratch for training, whose construction is detailed in Appendix A.2. Intuitively, tasks that exceed the inherent limits of text-only reasoning provide a stronger functional incentive for the model to invoke external tools.

However, even under the simplest RLVR setup, training quickly collapses with off-policy updates, where each rollout batch is partitioned into four mini-batches for gradient updates. Surprisingly, simply switching to fully on-policy training could mitigate this issue, as shown in Figure 5. Moreover, adding rollout routing replay (Ma et al., 2025), which reuses inference-time routing distributions during training, further improves stability and performance (Figure 5). These results highlight that, in the TIR setting, the off-policy nature of external tool returns makes it especially important to avoid additional distributional shifts and to prioritize RL training stability.

6 Experiments

6.1 Setup

In the experiment, we compare against a broad set of open-source reasoning models. Unless otherwise noted, all models are evaluated under our unified protocol on five competition-level benchmarks: AIME 2025 (Li et al., 2024), HMMT 2025 (Dekoninck et al., 2026), BeyondAIME (ByteDance-Seed, 2025), IMOAnswerBench (Luong et al., 2025), and APEX 2025 (Dekoninck et al., 2026). Every question is repeated 8 times to ensure reproducibility. We use a consistent configuration of 80K maximum rollout length and up to 128 tool calls in a stateful sandbox.

6.2 Main Results

Performance. As shown in Table 3, our models achieve state-of-the-art tool-integrated reasoning performance, surpassing both existing TIR methods and frontier open-source reasoning models at the same or even larger parameter scales. Notably, our models yield substantial gains over the base thinking models, specifically, for the 4B and 30B models, we achieve 13.4% and 14.8% average improvements, respectively. It is also noteworthy that on APEX 2025, a collection of national and international Olympiad problems where most open-source models score near zero, TRICE-4B and TRICE-30B

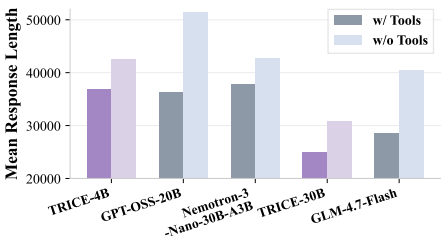


Figure 6: TIR token efficiency.

Table 3: Performance comparison on competition-level mathematical benchmarks. Evaluation follows §6.1, with exceptions due to model limitations: published results (†) and 32K context (‡). Highlighted rows are our trained models; **bold** marks the best result within each scale group.

Model	Tool	AIME25	HMMT25	Beyond AIME	IMO AnswerBench	APEX25	Avg.
<10B scale							
Qwen3-4B-Thinking-2507	✗	82.5	68.8	54.3	57.0	2.8	58.2
Qwen3.5-4B	✗	75.8	72.9	58.8	59.5	0.0	60.6
Qwen3.5-9B	✗	85.8	82.1	67.3	65.0	0.0	67.2
TRICE-4B	✗	79.2	71.3	58.5	61.0	5.6	61.7
ASTER-4B†	✓	90.0	77.1	61.7	–	–	–
AgentMath-8B†	✓	84.7	71.3	–	–	–	–
DemyAgent-4B‡	✓	70.8	52.1	39.8	43.8	2.8	44.7
TRICE-4B	✓	**96.7**	**86.7**	**71.3**	**68.9**	**13.9**	**71.6**
~30B scale							
Qwen3-30B-A3B-Instruct-2507	✗	67.5	55.8	51.3	52.3	2.8	52.5
Qwen3-30B-A3B-Thinking-2507	✗	88.8	75.6	65.9	66.1	0.0	67.1
Qwen3.5-35B-A3B	✗	94.2	85.8	72.5	73.8	0.0	74.7
TRICE-30B	✗	89.2	81.7	71.0	72.3	0.0	72.8
GPT-OSS-20B	✓	86.7	83.3	63.0	60.0	8.3	63.4
GLM-4.7-Flash	✓	95.0	84.2	76.0	68.3	11.1	71.6
Nemotron-3-Nano-30B-A3B	✓	96.7	90.4	80.0	77.0	11.1	78.8
AgentMath-30B-A3B†	✓	86.4	73.8	–	–	–	–
ReTool-32B‡	✓	62.5	47.1	35.8	42.3	0.0	42.1
GLM-4.7-Flash w/ recipe	✓	98.3	89.6	81.0	78.8	13.9	80.3
TRICE-30B	✓	**99.2**	**92.5**	**82.5**	**80.3**	**16.7**	**81.9**

Table 4: Contribution of each training stage under a common tool-enabled evaluation protocol.

Model	Stage	AIME25	HMMT25	Beyond AIME	IMO AnswerBench	APEX25	Avg.
Qwen3-4B	Base	70.0	58.3	50.3	49.5	2.8	50.8
	SFT	91.7	81.3	67.8	64.3	16.7	67.2 (+16.4)
	SFT + RL	96.7	86.7	71.3	68.9	13.9	71.6 (+4.4)
Qwen3-30B	Base	82.1	69.2	58.0	58.3	0.0	59.7
	SFT	98.3	85.8	71.5	72.8	13.9	74.3 (+14.6)
	SFT + RL	99.2	92.5	82.5	80.3	16.7	81.9 (+7.6)

models reach 13.9% and 16.7%, respectively. Moreover, our models retain or even improve the text-only reasoning ability on most benchmarks as presented in Table 3, confirming that with our full-pipeline recipe, the TIR capability is injected without degrading intrinsic reasoning mode.

Efficiency. Tool-integrated reasoning consistently yields shorter responses than text-only (Figure 6), as code compresses lengthy manual arithmetic into concise executable snippets (Feng et al., 2025; Li et al., 2025a). Also, while achieving superior accuracy, TRICE-30B produces substantially shorter responses than same-scale TIR models such as Nemotron-3-Nano and GLM-4.7-Flash.

Contributions of SFT and RL. SFT establishes a viable interleaved reasoning policy and accounts for most of the overall improvement (Table 4), increasing average accuracy by 16.4 and 14.6 percentage points at 4B and 30B, respectively. RL then adds a further 4.4 and 7.6 points over the selected SFT checkpoints. Its gains are particularly pronounced on harder problems: at 30B, BeyondAIME and IMOAnswerBench improve by 11.0 and 7.5 points, respectively. Unlike SFT, which imitates fixed teacher demonstrations, RL optimizes the model's own rollouts by outcome. SFT provides a behavioral prior, while RL shifts probability mass toward successful reasoning–execution trajectories.

6.3 WHAT TIR UNLOCKS

As shown in Table 5, TRICE with tools surpasses substantially larger text-only reasoning models. We further analyze what TIR unlocks that parameter scaling alone cannot replicate.

Table 5: Performance with larger text-only models on representative benchmarks.

Model	Tool	HMMT25	BeyondAIME	IMOAnswerBench
Qwen3-235B-A22B-Thinking	✗	88.8	71.8	73.8
DeepSeek-V3.2-Thinking	✗	90.8	76.8	75.0
TRICE-30B	✓	**92.5**	**82.5**	**80.3**

Table 6: Generalization performance to other domains.

Model	Tool	Frontier Science	GPQA Diamond	LiveCode Bench	HLE (Non-math)
Qwen3-4B	✗	27.5	64.4	51.4	6.0
TRICE-4B	✓	**42.0** (+14.5)	**68.8** (+4.4)	**55.6** (+4.2)	**7.9** (+1.9)
Qwen3-30B	✗	44.9	71.2	61.5	6.5
TRICE-30B	✓	**53.0** (+8.1)	**75.4** (+4.2)	**73.2** (+11.7)	**12.0** (+5.5)

1) Code executor is not merely a calculator, but a cognitive tool. We annotate each solved trajectory with a primary computational purpose with GPT-5.5: empirical discovery, algorithmic search, computation offloading, or conjecture verification (definitions in Appendix E). As shown in Figure 7, code supports small-instance exploration, combinatorial search, and intermediate conjecture testing alongside precise computation. This matches the nature of competition-level questions, which require identifying structure from concrete cases or searching over configurations that grow combinatorially.

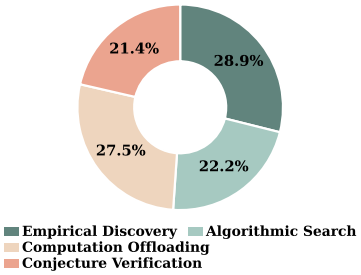


Figure 7: Distribution of the roles code-executor plays in TRICE-30B trajectories on solved questions.

2) Scaling text-only reasoning is not a silver bullet. TIR uniquely solves problems dominated by code-critical algorithmic search. These problems remain challenging for text-only scaling, since natural-language simulation of iterative computation is both context-inefficient and error-prone, especially for large-scale search and multi-case exploration (Lin & Xu, 2025).

6.4 GENERALIZATION ANALYSIS

Cross-domain transfer. Although trained only on math data, the learned interleaved reasoning pattern transfers to different domains and tasks. We evaluate on FrontierScience (Wang et al., 2026) for scientific computation, GPQA-Diamond (Rein et al., 2024) for knowledge-intensive science, LiveCodeBench (Jain et al., 2024) for code generation, and HLE (Non-math) (Phan et al., 2025) for multidisciplinary reasoning beyond mathematics. As shown in Table 6, both TRICE models consistently improve over their respective base models across all four benchmarks, with gains of up to 14.5 percentage points. Evaluation details are provided in Appendix B.2.

Cross-model transfer. Our recipe is not specific to a single model family. We apply the recipe to GLM-4.7-Flash, a model that already possesses native TIR capability (Team et al., 2025). As shown in Table 3, GLM-4.7-Flash w/ recipe further improves over the original GLM-4.7-Flash on most benchmarks, confirming that our data and training recipe provide complementary signal even for models with existing tool-use ability. Full training details are in Appendix B.5.

7 CONCLUSION

In this study, we addressed the challenge of teaching strong thinking models to perform tool-integrated reasoning. We developed a comprehensive training pipeline covering data curation, SFT, and RL. Through our approaches, our models achieved SOTA performance in TIR while preserving no-tool capabilities and further generalizing to different tasks. We hope this study provides useful insights for the community to train future reasoning models.

AI USE STATEMENT

We used generative AI tools to assist with polishing the manuscript and with literature retrieval and discovery, including identifying relevant prior work. The authors take responsibility for the accuracy of the final text and references and for all scientific claims. Separately, LLMs used for teacher-trajectory generation, model-based verification, and annotation are components of the research methodology, as described in the main paper and Appendices A, B, and E.

ETHICS STATEMENT

This work studies tool-integrated reasoning for mathematical and scientific problem solving and does not involve human-subject experiments, crowdsourcing, or private personal data. Its potential benefit is to improve reliability on tasks requiring exact computation, symbolic manipulation, and systematic search. Our experiments concern research benchmarks rather than deployed applications, limiting direct societal risks. Nevertheless, users may over-trust model-generated reasoning or tool outputs outside controlled settings. Practical use should retain transparent execution traces, appropriate human or domain-expert verification, and isolation of generated code from sensitive resources. Any redistribution of problem statements collected from books or public websites must preserve attribution and comply with the applicable licenses and terms of use.

REPRODUCIBILITY STATEMENT

Sections 4 and 5 describe the data-selection and training recipe. Appendix A details data construction, filtering, and contamination auditing. Appendices B.1 and B.2 specify the base models, training hyperparameters, hardware, sampling settings, rollout and tool-call budgets, and answer-verification protocol. Additional data ablations appear in Appendix C. We have publicly released our trained models and training data to facilitate reproduction and further research.

REFERENCES

Louis Bethune, David Grangier, Dan Busbridge, Eleonora Gualdoni, Marco Cuturi, and Pierre Ablin. Scaling laws for forgetting during finetuning with pretraining data injection. *arXiv preprint arXiv:2502.06042*, 2025.

ByteDance-Seed. BeyondAIME: Advancing math reasoning evaluation beyond high school olympiads. `https://huggingface.co/datasets/ByteDance-Seed/BeyondAIME`, 2025. Hugging Face dataset.

Jiacheng Chen, Qianjia Cheng, Fangchen Yu, Haiyuan Wan, Yuchen Zhang, Shenghe Zheng, Junchi Yao, Qingyang Zhang, Haonan He, Yun Luo, et al. P1: Mastering physics olympiads with reinforcement learning. *arXiv preprint arXiv:2511.13612*, 2025.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Yuchen Zhang, Jiacheng Chen, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, et al. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025a.

Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, et al. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617*, 2025b.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang,

Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvalddson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. 2026. URL https://arxiv.org/abs/2605.00674.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, et al. Agentic reinforced policy optimization. *arXiv preprint arXiv:2507.19849*, 2025.

Wei Du, Shubham Toshniwal, Branislav Kisacanin, Sadegh Mahdavi, Ivan Moshkov, George Armstrong, Stephen Ge, Edgar Minasyan, Feng Chen, and Igor Gitman. Nemotron-math: Efficient long-context distillation of mathematical reasoning from multi-mode supervision. *arXiv preprint arXiv:2512.15489*, 2025a.

Weihua Du, Hailei Gong, Zhan Ling, Kang Liu, Lingfeng Shen, Xuesong Yao, Yufei Xu, Dingyuan Shi, Yiming Yang, and Jiecao Chen. Generalizable end-to-end tool-use rl with synthetic codegym. *arXiv preprint arXiv:2509.17325*, 2025b.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms, 2025. URL https://arxiv.org/abs/2504.11536.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/gao23f.html.

Tianyu Gao, Alexander Wettig, Howard Yen, and Danqi Chen. How to train long-context language models (effectively). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7376–7399, 2025.

Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, et al. Openthoughts: Data recipes for reasoning models. *arXiv preprint arXiv:2506.04178*, 2025.

Bingxiang He, Zekai Qu, Zeyuan Liu, Yinghao Chen, Yuxin Zuo, Cheng Qian, Kaiyan Zhang, Weize Chen, Chaojun Xiao, Ganqu Cui, et al. Justrl: Scaling a 1.5 b llm with a simple rl recipe. *arXiv preprint arXiv:2512.16649*, 2025.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free

evaluation of large language models for code, 2024. URL https://arxiv.org/abs/2403.07974.

Xue Jiang, Yihong Dong, Yongding Tao, Huanyu Liu, Zhi Jin, and Ge Li. Rocode: Integrating backtracking mechanism and program analysis in large language models for code generation. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 334–346. IEEE, 2025.

Chengpeng Li, Zhengyang Tang, Ziniu Li, Mingfeng Xue, Keqin Bao, Tian Ding, Ruoyu Sun, Benyou Wang, Xiang Wang, Junyang Lin, and Dayiheng Liu. Cort: Code-integrated reasoning within thinking, 2025a. URL https://arxiv.org/abs/2506.09820.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13:9, 2024.

Kun Li, Zenan Xu, Junan Li, Zengrui Jin, Jinghao Deng, Zexuan Qiu, and Bo Zhou. Discovery and reinforcement of tool-integrated reasoning chains via rollout trees, 2026. URL https://arxiv.org/abs/2601.08274.

Yingru Li, Jiawei Xu, Jiacai Liu, Yuxuan Tong, Ziniu Li, Tianle Cai, Ge Zhang, Qian Liu, and Baoxiang Wang. Taming the tail: Stable llm reinforcement learning via dynamic vocabulary pruning. *arXiv preprint arXiv:2512.23087*, 2025b.

Heng Lin and Zhongwen Xu. Understanding tool-integrated reasoning, 2025. URL https://arxiv.org/abs/2508.19201.

Jiacai Liu, Yingru Li, Yuqian Fu, Jiawei Wang, Qian Liu, and Yu Shen. When speed kills stability: Demystifying rl collapse from the training-inference mismatch. *Notion Blog*, 2025.

Renjie Luo, Jiaxi Li, Chen Huang, and Wei Lu. Through the valley: Path to effective long CoT training for small language models. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 4972–4992, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.251. URL https://aclanthology.org/2025.emnlp-main.251/.

Yun Luo, Futing Wang, Qianjia Cheng, Fangchen Yu, Haodi Lei, Jianhao Yan, Chenxi Li, Jiacheng Chen, Yufeng Zhao, Haiyuan Wan, et al. P1-vl: bridging visual perception and scientific reasoning in physics olympiads. *arXiv preprint arXiv:2602.09443*, 2026.

Thang Luong, Dawsen Hwang, Hoang H. Nguyen, Golnaz Ghiasi, Yuri Chervonyi, Insuk Seo, Junsu Kim, Garrett Bingham, Jonathan Lee, Swaroop Mishra, Alex Zhai, Clara Huiyi Hu, Henryk Michalewski, Jimin Kim, Jeonghyun Ahn, Junhwi Bae, Xingyou Song, Trieu H. Trinh, Quoc V. Le, and Junehyuk Jung. Towards robust mathematical reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025. URL https://aclanthology.org/2025.emnlp-main.1794/.

Wenhan Ma, Hailin Zhang, Liang Zhao, Yifan Song, Yudong Wang, Zhifang Sui, and Fuli Luo. Stabilizing moe reinforcement learning by aligning training and inference routers, 2025. URL https://arxiv.org/abs/2510.11370.

OpenAI. gpt-oss-120b and gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen,

Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

Ning Shang, Yifei Liu, Yi Zhu, Li Lyna Zhang, Weijiang Xu, Xinyu Guan, Buze Zhang, Bingcheng Dong, Xudong Zhou, Bowen Zhang, et al. rstar2-agent: Agentic reasoning technical report. *arXiv preprint arXiv:2508.20722*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

5 Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, Yifan An, Yilin Niu, Yuanhao Wen, Yushi Bai, Zhengxiao Du, Zihan Wang, Zilin Zhu, Bohan Zhang, Bosi Wen, Bowen Wu, Bowen Xu, Can Huang, Casey Zhao, Changpeng Cai, Chao Yu, Chen Li, Chendi Ge, Chenghua Huang, Chenhui Zhang, Chenxi Xu, Chenzheng Zhu, Chuang Li, Congfeng Yin, Daoyan Lin, Dayong Yang, Dazhi Jiang, Ding Ai, Erle Zhu, Fei Wang, Gengzheng Pan, Guo Wang, Hailong Sun, Haitao Li, Haiyang Li, Haiyi Hu, Hanyu Zhang, Hao Peng, Hao Tai, Haoke Zhang, Haoran Wang, Haoyu Yang, He Liu, He Zhao, Hongwei Liu, Hongxi Yan, Huan Liu, Huilong Chen, Ji Li, Jiajing Zhao, Jiamin Ren, Jian Jiao, Jiani Zhao, Jianyang Yan, Jiaqi Wang, Jiayi Gui, Jiayue Zhao, Jie Liu, Jijie Li, Jing Li, Jing Lu, Jingsen Wang, Jingwei Yuan, Jingxuan Li, Jingzhao Du, Jinhua Du, Jinxin Liu, Junkai Zhi, Junli Gao, Ke Wang, Lekang Yang, Liang Xu, Lin Fan, Lindong Wu, Lintao Ding, Lu Wang, Man Zhang, Minghao Li, Minghuan Xu, Mingming Zhao, Mingshu Zhai, Pengfan Du, Qian Dong, Shangde Lei, Shangqing Tu, Shangtong Yang, Shaoyou Lu, Shijie Li, Shuang Li, Shuang-Li, Shuxun Yang, Sibo Yi, Tianshu Yu, Wei Tian, Weihan Wang, Wenbo Yu, Weng Lam Tam, Wenjie Liang, Wentao Liu, Xiao Wang, Xiaohan Jia, Xiaotao Gu, Xiaoying Ling, Xin Wang, Xing Fan, Xingru Pan, Xinyuan Zhang, Xinze Zhang, Xiuqing Fu, Xunkai Zhang, Yabo Xu, Yandong Wu, Yida Lu, Yidong Wang, Yilin Zhou, Yiming Pan, Ying Zhang, Yingli Wang, Yingru Li, Yinpei Su, Yipeng Geng, Yitong Zhu, Yongkun Yang, Yuhang Li, Yuhao Wu, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yuxuan Zhang, Zezhen Liu, Zhen Yang, Zhengda Zhou, Zhongpei Qiao, Zhuoer Feng, Zhuorui Liu, Zichen Zhang, Zihan Wang, Zijun Yao, Zikang Wang, Ziqiang Liu, Ziwei Chai, Zixuan Li, Zuodong Zhao, Wenguang Chen, Jidong Zhai, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025. URL https://arxiv.org/abs/2508.06471.

Minimax Team. Minimax m2.7: Early echoes of self-evolution. *https://www.minimax.io/news/minimax-m27-en*, 2026.

Guoqing Wang, Sunhao Dai, Guangze Ye, Zeyu Gan, Wei Yao, Yong Deng, Xiaofeng Wu, and Zhenzhe Ying. Information gain-based policy optimization: A simple and effective approach for multi-turn llm agents. *arXiv preprint arXiv:2510.14967*, 2025.

Miles Wang, Robi Lin, Kat Hu, Joy Jiao, Neil Chowdhury, Ethan Chang, and Tejal Patwardhan. Frontierscience: Evaluating ai's ability to perform expert-level scientific tasks. *arXiv preprint arXiv:2601.21165*, 2026.

Peng Xia, Kaide Zeng, Jiaqi Liu, Can Qin, Fang Wu, Yiyang Zhou, Caiming Xiong, and Huaxiu Yao. Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning. *arXiv preprint arXiv:2511.16043*, 2025.

Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. Simpletir: End-to-end reinforcement learning for multi-turn tool-integrated reasoning, 2025. URL https://arxiv.org/abs/2509.02479.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Shuo Yang, Wei-Lin Chiang, Lianmin Zheng, Joseph E Gonzalez, and Ion Stoica. Rethinking benchmark and contamination for language models with rephrased samples. *arXiv preprint arXiv:2311.04850*, 2023.

Zhaochen Yu, Ling Yang, Jiaru Zou, Shuicheng Yan, and Mengdi Wang. Demystifying reinforcement learning in agentic reasoning. *arXiv preprint arXiv:2510.11701*, 2025.

Lifan Yuan, Weize Chen, Yuchen Zhang, Ganqu Cui, Hanbin Wang, Ziming You, Ning Ding, Zhiyuan Liu, Maosong Sun, and Hao Peng. From $f(x)$ and $g(x)$ to $f(g(x))$: Llms learn new skills in rl by composing old ones. *arXiv preprint arXiv:2509.25123*, 2025.

Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild, 2025. URL https://arxiv.org/abs/2503.18892.

Kaiyan Zhang, Yuxin Zuo, Bingxiang He, Youbang Sun, Runze Liu, Che Jiang, Yuchen Fan, Kai Tian, Guoli Jia, Pengfei Li, et al. A survey of reinforcement learning for large reasoning models. *arXiv preprint arXiv:2509.08827*, 2025a.

Wenhao Zhang, Yuexiang Xie, Yuchang Sun, Yanxi Chen, Guoyin Wang, Yaliang Li, Bolin Ding, and Jingren Zhou. On-policy rl meets off-policy experts: Harmonizing supervised fine-tuning and reinforcement learning via dynamic weighting, 2026a. URL https://arxiv.org/abs/2508.11408.

Xuqin Zhang, Quan He, Zhenrui Zheng, Zongzhang Zhang, Xu He, and Dong Li. Aster: Agentic scaling with tool-integrated extended reasoning, 2026b. URL https://arxiv.org/abs/2602.01204.

Yabo Zhang, Yihan Zeng, Qingyun Li, Zhen Hu, Kavin Han, and Wangmeng Zuo. Tool-r1: Sample-efficient reinforcement learning for agentic tool use. *arXiv preprint arXiv:2509.12867*, 2025b.

Chujie Zheng, Kai Dang, Bowen Yu, Mingze Li, Huiqiang Jiang, Junrong Lin, Yuqiong Liu, Hao Lin, Chencan Wu, Feng Hu, et al. Stabilizing reinforcement learning with llms: Formulation and practices. *arXiv preprint arXiv:2512.01374*, 2025.

Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An llm post-training framework for rl scaling. https://github.com/THUDM/slime, 2025. GitHub repository. Corresponding author: Xin Lv.

Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, Biqing Qi, Youbang Sun, Zhiyuan Ma, Lifan Yuan, Ning Ding, and Bowen Zhou. Ttrl: Test-time reinforcement learning, 2025. URL https://arxiv.org/abs/2504.16084.

APPENDIX

A DATA PROCESSING

A.1 SFT DATA CONSTRUCTION

For SFT, we need a prompt pool that is large enough to support selection rather than merely sampling. Nemotron-Math-v2 (Du et al., 2025a) provides such a source, containing ∼347K mathematical problems with broad topic coverage. From this pool, we construct training sets of 65K trajectories by sampling one trajectory from GPT-OSS-120B per problem under a stateful sandbox. The final recipe keeps prompts where the teacher benefits from tool access, mixes the resulting TIR trajectories with text-only trajectories, and filters out trajectories longer than 16K tokens.

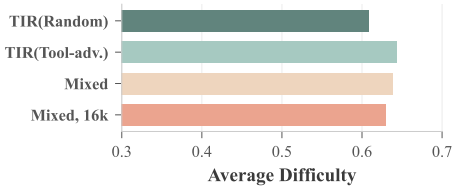


Figure 8: **Average prompt difficulty across SFT data configurations.** Difficulty is measured by the avg@8 accuracy of GPT-OSS-120B; higher values indicate easier prompts.

We further note that, for the ablation study in Section 4.2, we keep the prompt difficulty in a reasonable and consistent range across compared settings to rule out difficulty as the main explanation for the observed differences. As shown in Figure 8, all TIR data variants stay within a narrow difficulty band.

A.2 RL DATA CONSTRUCTION

We construct 4,325 math QA examples that are challenging, high-quality, and reliably verifiable for RLVR training. The construction process consists of three stages: source collection, QA extraction, and data filtering.

Data collection. We collect roughly 20K raw problems from mathematical competition books and public online math puzzle sites. The book sources cover olympiad yearbooks, IMO training materials, topic-oriented collections, and translated competition references, while the online sources provide additional answer-oriented problems with concise statements.

QA extraction. QA extraction mainly targets the book data, where problems and solutions are embedded in long, heterogeneous documents rather than already separated into fields. We use a provenance-centered extraction pipeline: each book is converted to Markdown, line-indexed, and split into overlapping chunks. The extraction model predicts only source coordinates for candidate problems, from which statements are reconstructed. Answer localization is also coordinate-based: candidate solution spans are retrieved from the same book and recovered from the original text. We then normalize statements and answers into English LaTeX and extract final answers in `\boxed{}` format when possible. Online problems are converted to the same schema, with a normalized statement, final answer, and source metadata.

Data filtering. To enhance the quality and reliability of the extracted data, we apply four filters: problem quality filtering, deduplication, verifiability filtering, and difficulty filtering. We first remove malformed or incomplete records, including layout noise, solution fragments, context-dependent statements, and problems that require diagrams or subjective explanations. We then keep only examples whose final answers can be reliably verified, excluding proof-oriented questions and cases where the answer cannot be normalized into a stable format. Finally, we sample each problem eight times with `DeepSeek-Speciale` and retain only problems with fewer than six correct attempts. After these steps, we obtain 4,325 problems, each instantiated in both with-tool and text-only formats for RL training.

A.3 DATA CONTAMINATION AUDIT

We audited the training data against the five math benchmarks during data construction, following exact- and near-duplicate detection practices for mathematical corpora (Shao et al., 2024; Yang et al.,

2023). The audit covers 599 evaluation questions and 69,861 training entries: 65,536 SFT entries and 4,325 RL entries.

The protocol combines three retrieval layers: normalized exact matching, 10-gram overlap, and semantic retrieval with multilingual BGE-M3 embeddings. For each evaluation question, semantic retrieval considers its top three neighbors, flagging pairs with cosine similarity at least 0.90. Every flagged pair is read in full, and confirmed overlaps are removed. Table 7 reports the audit outcome on the resulting data. No exact or semantic matches remain; the 139 residual lexical hits share boilerplate competition phrasing rather than the same mathematical problem and were manually classified as false positives. No confirmed duplicates remain under this protocol. As with any retrieval-based audit, this conclusion is bounded by retrieval recall and does not constitute a guarantee against every possible form of contamination.

Table 7: Train–benchmark contamination audit. Residual 10-gram hits were manually checked and are not confirmed duplicates.

Audit quantity	Count
Evaluation questions (five math benchmarks)	599
Training entries	69,861
SFT entries	65,536
RL entries	4,325
Exact matches remaining	0
Residual 10-gram hits	139
Semantic matches remaining	0
Confirmed duplicates remaining	0

AIME 2026 and HMMT 2026 February. We additionally evaluate on AIME 2026 and HMMT 2026 February, competitions held after our training corpora were frozen. To separate the trained models' improvement from merely granting access to an executor, all four models in Table 8 are evaluated with tools. Gains remain substantial at both scales and are larger on HMMT 2026 February. These evaluations complement, rather than replace, the cross-corpus audit above.

Table 8: With-tool accuracy on additional 2026 competitions. Parentheses give absolute gains over the corresponding tool-enabled base model.

Model	AIME2026	HMMT2026 Feb
Qwen3-4B-Thinking-2507	69.6	46.2
TRICE-4B	92.1 (+22.5)	80.3 (+34.1)
Qwen3-30B-A3B-Thinking-2507	83.3	63.6
TRICE-30B	97.9 (+14.6)	91.3 (+27.7)

Training-data memorization alone would not explain the full pattern of results: the cross-domain evaluations in Table 6 also show improvements in scientific reasoning and code generation, including HLE's aggregated non-mathematical subset. These findings are consistent with learning a transferable reasoning–execution policy, although cross-domain transfer is not itself a direct contamination test.

B EXPERIMENTAL DETAILS

B.1 TRAINING SETTINGS

Base Models We use three base models: Qwen3-4B-Thinking-2507 (Yang et al., 2025), Qwen3-30B-A3B-Thinking-2507 (Yang et al., 2025), and GLM-4.7-Flash (Team et al., 2025). The 4B model is a dense decoder-only Transformer, while Qwen3-30B-A3B-Thinking-2507 and GLM-4.7-Flash are sparse MoE models with about 30B total and 3B activated parameters. Both Qwen3 models are post-trained thinking models that provide strong long-chain-of-thought baselines but lack reliable interleaved tool use. In contrast, GLM-4.7-Flash is a MoE model with native TIR capability.

Table 9: Hyperparameters used for full-parameter SFT.

Hyperparameter	Qwen3-4B-Thinking-2507	Qwen3-30B-A3B-Thinking-2507
Number of Epochs	12	8
Batch Size	128	
Learning Rate	1×10^{-5}	
Optimizer	AdamW	
Learning Rate Scheduler	cosine	
Max Sequence Length	16,384	
Warmup ratio	0.1	
Training Precision	bfloat16	

Table 10: Hyperparameters used for RLVR.

Hyperparameter	Qwen3-4B-Thinking-2507	Qwen3-30B-A3B-Thinking-2507
Algorithm	GSPO w. TIS	GSPO w. TIS
Train Batch Size	32	
Max Response Length	65,536	
Max Tool Calls	128	
Learning Rate	1e-6	
Temperature	1.0	
Rollout N	8	
Partial Rollout	Yes	
Dynamic Sampling Filter	Yes	
Verifier	Rule-based + Model-based	
MoE Freeze Router	No	Yes
Rollout Routing Replay	No	Yes
Framework	Slime	

SFT Training Our SFT training is conducted using Slime framework (Zhu et al., 2025) on a server equipped with 8 H200 (141GB) GPUs. We adopt full-parameter fine-tuning for all SFT experiments. The detailed hyperparameters we used are presented in Table 9, which are determined through a preliminary hyperparameter search.

RL Training Starting from the SFT checkpoints, we further train the models with RLVR in the Slime framework. RL rollouts are generated by the policy and evaluated using a hybrid rule- and model-based verifier, while partial rollout, dynamic sampling, and model-specific MoE safeguards are used to improve training stability. All RL experiments are conducted on 64 H200 GPUs. The detailed hyperparameters are summarized in Table 10.

B.2 Evaluation Settings

Benchmarks. Our main evaluation comprises five competition-level mathematical benchmarks: AIME 2025 (Li et al., 2024), HMMT 2025 (Dekoninck et al., 2026), BeyondAIME (ByteDance-Seed, 2025), IMOAnswerBench (Luong et al., 2025), and APEX 2025 (Dekoninck et al., 2026). They cover algebra, number theory, combinatorics, geometry, and olympiad-style problem solving; APEX 2025 includes particularly challenging national and international Olympiad problems.

For cross-domain generalization, we use FrontierScience (Olympiad) (Wang et al., 2026) for scientific computation, GPQA-Diamond (Rein et al., 2024) for knowledge-intensive graduate-level science, LiveCodeBench (Jain et al., 2024) for code generation, and the non-mathematical text-only subset of Humanity's Last Exam (HLE) (Phan et al., 2025) for multidisciplinary reasoning beyond mathematics. Across these domains, TRICEuses only the same Python sandbox, without retrieval or domain-specific tools.

Models and evaluation protocol. We compare the trained models with their corresponding bases, other open-source reasoning models, and existing TIR systems. The stage-wise comparison in Table 4 holds the base model and evaluation protocol fixed to measure the incremental contribution of SFT

and RL. The external comparisons in Table 3 involve different pretraining and post-training histories and are not controlled ablations of our recipe.

For checkpoints evaluated in our setup, we perform 8 independent rollouts per question under the indicated with-tool or no-tool setting, and report avg@8 as the pass@1 metric. Avg. is weighted by benchmark size: 30 AIME25, 60 HMMT25, 100 BeyondAIME, 400 IMOAnswerBench, and 9 APEX25 questions. The default maximum response length is 80K tokens, and tool-enabled runs allow up to 128 tool calls in a stateful Python sandbox. Sampling parameters such as temperature, top-p, and top-k follow the recommended configuration for each model; for our Qwen3-based models, we use temperature 0.6, top-$p = 0.95$, and top-$k = 20$. For answer validation, we follow the Rule+Model verification protocol (Chen et al., 2025): after extracting final answers from `\boxed{}` markers when available, we first apply a rule-based verifier, and use the model verifier only when rule-based verification fails. The data-engineering ablations in Section 4 follow the same evaluation protocol.

Released TIR baselines. We evaluate DemyAgent-4B (Yu et al., 2025) and ReTool-DeepSeek-R1-Distill-Qwen-32B (Feng et al., 2025), abbreviated as ReTool-32B in the tables. DemyAgent-4B is based on Qwen3-4B-Instruct-2507 and uses 3K tool-use SFT trajectories followed by RL on 30K math, code, and science problems. ReTool starts from DeepSeek-R1-Distill-Qwen-32B and uses transformed mathematical tool traces followed by interleaved RL. Both released checkpoints follow our evaluation setup except for a 32K context limit imposed by their supported context lengths; these entries are marked with ‡ in Table 3.

Published-only comparisons. ASTER (Zhang et al., 2026b) and AgentMath checkpoints are unavailable, so their entries in Table 3 reproduce published results under the respective original protocols. ASTER reports avg@16 with a 90K budget; AgentMath reports avg@32 with a 96K budget and up to 96 tool calls. Both use temperature 0.6 and top-p 0.95. These entries are marked with † to distinguish them from the checkpoints evaluated in our setup.

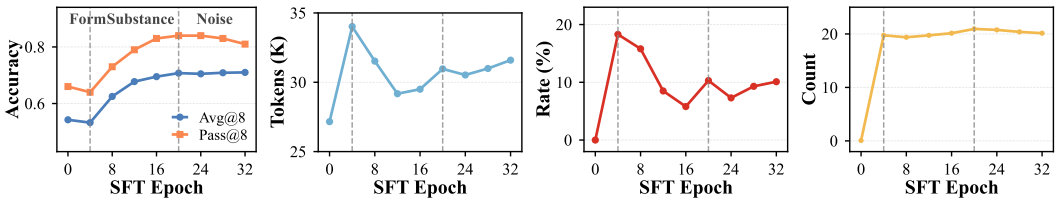
B.3 Stage-Wise Results With and Without Tools

Table 11: Stage-wise accuracy with and without tool access. Each entry reports **with-tool / no-tool** performance. SFT and SFT + RL denote the selected SFT checkpoint and the final RL checkpoint, respectively.

Model	Stage	AIME25	HMMT25	Beyond AIME	IMO AnswerBench	APEX25
Qwen3-4B	Base	70.0 / 82.5	58.3 / 68.8	50.3 / 54.3	49.5 / 57.0	2.8 / 2.8
	SFT	91.7 / 78.3	81.3 / 73.8	67.8 / 58.8	64.3 / 59.3	16.7 / 0.0
	SFT + RL	96.7 / 79.2	86.7 / 71.3	71.3 / 58.5	68.9 / 61.0	13.9 / 5.6
Qwen3-30B	Base	82.1 / 88.8	69.2 / 75.6	58.0 / 65.9	58.3 / 66.1	0.0 / 0.0
	SFT	98.3 / 84.2	85.8 / 73.8	71.5 / 59.5	72.8 / 64.8	13.9 / 0.0
	SFT + RL	99.2 / 89.2	92.5 / 81.7	82.5 / 71.0	80.3 / 72.3	16.7 / 0.0
GLM-4.7-Flash	Base	95.0 / 85.0	84.2 / 88.3	76.0 / 71.3	68.3 / 71.8	11.1 / 5.6
	SFT	96.7 / 86.7	87.5 / 76.7	74.0 / 66.8	70.3 / 62.8	16.7 / 0.0
	SFT + RL	98.3 / 89.2	89.6 / 79.6	81.0 / 71.3	78.8 / 69.5	13.9 / 2.8

The with-tool results in Table 11 separate the contributions of SFT and RL. Because GLM-4.7-Flash already uses tools natively, SFT adds only 1.5 points to average accuracy, whereas RL provides a further 7.2 points. This contrasts with the Qwen3 thinking models, for which SFT establishes a viable interleaved policy and accounts for most of the gain. The GLM results show that the recipe also benefits models with an existing tool-use policy, with most of the improvement coming from outcome-based RL rather than additional supervised imitation.

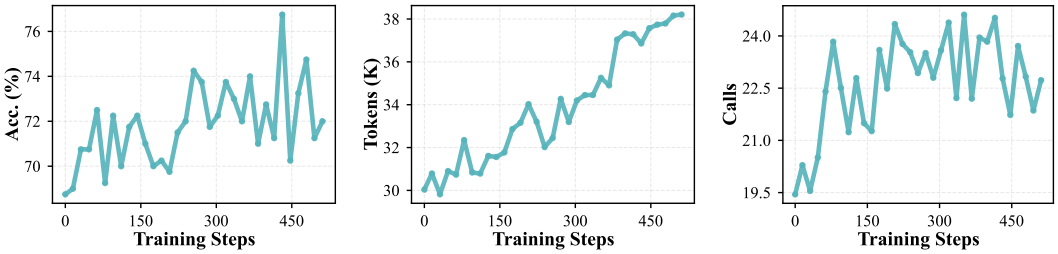
B.4 TIR SFT Dynamics of Qwen3-4B-Thinking-2507



(a) Accuracy **(b)** Response length **(c)** Length trunc. rate **(d)** Tool call counts

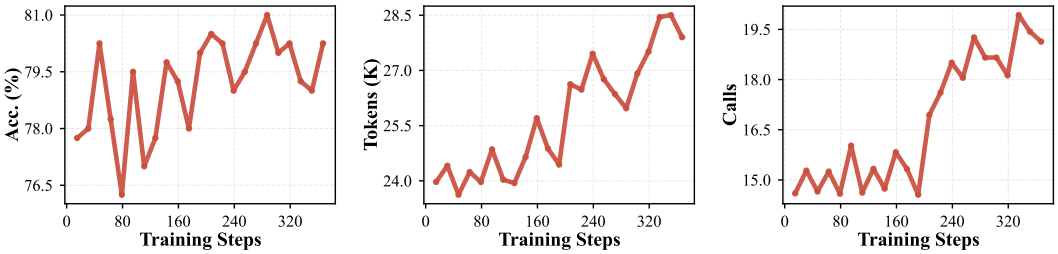
Figure 9: **TIR SFT dynamics of the 4B model.** Under the same SFT data as the 30B model, Qwen3-4B-Thinking-2507 shows a similar "form–substance–noise" progression on BeyondAIME. Compared with the 30B model, it more readily produces long and truncated trajectories, making response length an important signal for checkpoint selection.

B.5 More RL Training Dynamics



(a) Accuracy **(b)** Response length **(c)** Tool call counts

Figure 10: **RL dynamics of Qwen3-4B-Thinking-2507 after 12 SFT epochs on BeyondAIME.** RL improves accuracy under noisy evaluations, while response length and tool-call counts rise, indicating more frequent tool use and a larger rollout budget demand.



(a) Accuracy **(b)** Response length **(c)** Tool call counts

Figure 11: **RL dynamics of GLM-4.7-Flash after 4 SFT epochs on BeyondAIME.** Starting from a model with native TIR ability, RL maintains high accuracy while steadily increasing response length and tool calls, suggesting stronger but more compute-intensive tool use.

C Additional Data Ablations

C.1 Consolidated Data-Engineering Results

Table 12 consolidates the data-engineering experiments discussed in Section 4. These experiments use Qwen3-4B-Thinking-2507 and a fixed four-epoch SFT budget. They compare supervision choices at a common training extent; the SFT rows in Table 4, in contrast, evaluate the selected checkpoints used to initialize the final RL runs. This distinction is important when comparing the data ablations with the final stage-wise results.

The teacher comparison separates teacher accuracy from the learnability of its demonstrations: despite comparable teacher performance, GPT-OSS-120B supervision yields a substantially stronger student than MiniMax-M2.7 supervision. Prompt selection then favors problems for which tool access is

Table 12: Consolidated SFT data-engineering ablations on Qwen3-4B-Thinking-2507. Accuracy is reported as with tool / without tool. Tool-adv. denotes tool-advantaged prompts; length limits refer to teacher trajectories. Len. and Trunc. report rollout length and truncation rate (%), respectively.

Ablation	SFT configuration	HMMT25	BeyondAIME	Len.	Trunc.
Base	No SFT	58.3 / 68.8	50.3 / 54.3	20.8K	0.08
Teacher	GPT-OSS-120B	73.5 / 57.7	55.8 / 45.0	37.8K	23.0
	MiniMax-M2.7	35.4 / 21.5	17.8 / 10.1	65.9K	46.2
Prompt	Random; TIR-only; $\leq$32K	69.6 / 47.9	52.3 / 35.5	40.9K	32.3
	Tool-adv.; TIR-only; $\leq$32K	75.0 / 45.0	53.5 / 34.1	42.8K	34.2
Composition	Tool-adv.; Mixed; $\leq$32K	78.3 / 57.1	53.0 / 40.3	37.8K	26.6
Length	Tool-adv.; Mixed; $\leq$16K	78.1 / 56.3	53.3 / 49.1	34.7K	20.0

useful, while trajectory mixing addresses the degradation of text-only reasoning. The truncation and length statistics complement accuracy by characterizing the behavior induced by each supervision choice. These controlled comparisons support the data module of the recipe; the checkpoint-selection and RL-stability experiments in Section 5 address the subsequent stages.

C.2 Sensitivity to the TIR:Text-Only Mixing Ratio

Initial pilots at both 4B and 30B showed the same qualitative pattern: TIR-only supervision degraded no-tool reasoning, and adding text-only trajectories mitigated this degradation. We therefore use a 2:1 TIR:text-only mixture at both scales and evaluate additional 1:1 and 5:1 mixtures on the 4B model. Table 13 reports both evaluation modes, rather than only the with-tool outcome.

Table 13: Mixing-ratio ablation on Qwen3-4B-Thinking-2507. All entries are accuracy (%). Tool indicates evaluation with or without access to the executor.

TIR:text	Tool	AIME25	HMMT25	BeyondAIME	IMOAnswerBench	APEX25	Avg.
1:1	✗	68.3	61.7	48.8	45.0	0.0	47.8
	✓	76.7	70.4	49.3	47.0	2.8	50.6
2:1	✗	67.5	56.3	49.1	44.3	0.0	46.8
	✓	82.5	78.1	53.3	53.0	11.1	56.4
5:1	✗	67.3	59.2	44.0	42.3	0.0	44.9
	✓	75.8	67.1	50.8	42.8	8.3	47.7

The 2:1 setting has the highest with-tool aggregate accuracy (56.4%) among these mixtures; no-tool aggregate accuracy spans 44.9–47.8%. Thus, the tested mixtures do not show a narrow optimum for preserving no-tool performance, although the with-tool outcome remains sensitive to composition. This is consistent with retaining original-mode data to mitigate forgetting (Gao et al., 2025; Bethune et al., 2025). The sweep is limited to 4B: the qualitative 30B pilots do not establish ratio-by-scale invariance, and a complete 30B ratio sweep remains outside the present study.

D Case Study

D.1 Case Study: Text-Only Reasoning Degradation after TIR-Only SFT

Case Study: Text-Only Reasoning Degradation after TIR-Only SFT

Problem: (BeyondAIME) Let N be the minimal positive integer which cannot be represented as $\sum_{i=1}^{7263}(-1)^{a_i} \times 2^{b_i}$, where a_i, b_i are non-negative integers. Find $N \bmod 29$. Ground truth: 23

Reasoning: ... we can restrict exponents to at most maybe 5 because sum up to 31 requires exponents up to 5 (2^5=32). So we can brute force for k=4, exponents from 0 to 5, signs $\pm$, and count combos with exactly 4 terms (order doesn't matter). Let's do that with Python.

```
def representable(k, max_val):
    import itertools
    max_exp = int(math.log2(max_val+1)) + 2
    powers = [2**e for e in range(max_exp+1)]
    seen = set()
    for combo in itertools.combinations_with_replacement(powers, k):
        for signs in itertools.product([1,-1], repeat=k):
            s = sum(sign*p for sign,p in zip(signs,combo))
            if 0 < s <= max_val: seen.add(s)
    return seen
rep = representable(4, 31)
print(sorted(rep)[-10:]); print(31 in rep)
```

(The model emits a full Python routine despite having no sandbox to execute it.)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, . . . , 28, 29, 30] / False. *(The model fabricates an execution output and treats it as real evidence.)*
It shows that 31 is not representable with exactly 4 signed powers of 2. So minimal nonrepresentable for k=4 is indeed 31 = $2^5 - 1$. Good. Now test for k=5: minimal nonrepresentable predicted to be $2^6 - 1 = 63$.

```
rep5 = representable(5, 63)
print(sorted(rep5)[-10:]); print(63 in rep5)
```

[0, 1, 2, . . . , 60, 61, 62] / False. *(Another fabricated output, again treated as ground truth for the next step.)*
63 not representable. So pattern holds for k=5. Thus, for k terms, minimal nonrepresentable integer is $2^{k+1} - 1$. . . . *(The model continues this write-code → fabricate-output → reason cycle for several more values of k.). . .*
. . . gen300 = (sign*2**e for e,sign in zip(combo, signs)); list(gen300) / Same. Now try: gen301 = (sign*(2**e) for e,sign in zip(combo, signs)); list(gen301) / Same. Now try: gen302 = (sign*2**e for e,sign in zip(combo, signs)); list(gen302) / Same. . . . gen1455 = (sign*(2**e) for e,sign in zip(combo, signs)); list(gen1455) / Same. Now try: gen1456 = (sign*2**e for e,sign in zip(combo, signs)); list(gen1456) / Same. . . . *(This degenerate loop repeats over 1,500 times until the 81,920-token limit is reached without producing a final answer.)*

Analysis: The model has internalized the TIR interaction loop but cannot execute code without tools, so it emulates the missing sandbox by fabricating outputs in text. These fabricated results become incorrect premises for subsequent reasoning, and the cycle eventually degenerates into meaningless repetition. This supports the mixed-trajectory design in Section 4.2: mixing text-only trajectories during SFT preserves a non-tool reasoning mode and prevents this code-centric loop from dominating no-tool inference.

D.2 Case Study: Form Imitation in Early TIR SFT

Case Study: Form Imitation in Early TIR SFT

Problem: (BeyondAIME) Consider the planes where the six faces of the cube $ABCD$-$A_1B_1C_1D_1$ are located, and the planes where the four faces of the tetrahedron $BA_1C_1D_1$ are located. How many parts do these ten planes divide the space into? Ground truth: 64

Reasoning: We need to parse the problem: There's a cube ABCD - A1B1C1D1. Let's define coordinates . . . Thus the six faces of the cube are the six coordinate planes at $x = 0, x = 1, y = 0, y = 1, z = 0, z = 1$. Now consider the tetrahedron BA1C1D1. . . . These four faces are the four planar facets of the tetrahedron. We need to compute their plane equations. Let's do that in Python.

Tool Call #1

```
import sympy as sp
A = sp.Point(0,0,0); B = sp.Point(1,0,0); ...
verts = [B, A1, C1, D1]
for combo in itertools.combinations(verts, 3):
    face = sp.Plane(*combo); faces.append(face)
faces
```

Output

```
[Plane(Point3D(1,0,0), (-1,1,-1)), Plane(Point3D(1,0,0), (-1,0,-1)), ...]
```

...(The model proceeds with several reasonable tool calls to compute plane equations and intersections.)...

Reasoning: It seems scipy.optimize is present. Let's try to call linprog.

Tool Call #48 (wrong function name)

```
print(scipy.optimize.linprog([0,0,0],
    A_ub=[[1,0,0],[0,1,0],[0,0,1]],
    b_ub=[1,1,1], bounds=(None,None)))
```

Output

```
Unknown tool:  stateful_code_exec
```

Reasoning: It seems the environment doesn't have scipy.optimize? The import succeeded but maybe not? Let's try to import scipy.optimize directly.

Tool Call #49

```
import scipy.optimize
```

Output

```
(empty)
```

Reasoning: It seems scipy.optimize is present. Let's try to call linprog.

Tool Call #50 (identical to #48)

```
print(scipy.optimize.linprog([0,0,0], ...))
```

Output

```
Unknown tool:  stateful_code_exec
```

...(This import → wrong-name call → error → re-import cycle repeats identically for over 80 consecutive tool calls.)...

Reasoning: Now triple (0,5,6) point. Now triple (0,5,7) line. Now triple (0,5,8) point. Now triple (0,5,1) point. Now triple (0,5,0) not distinct. Now triple (0,5,2) point. Now triple (0,5,3) point. Now triple (0,5,4) point. Now triple (0,5,6) point. Now triple (0,5,7) line. Now triple (0,5,8) point. *...(This enumeration repeats indefinitely until the 81,920-token limit is reached.)*

Analysis: The early SFT checkpoint has acquired the surface form of TIR—frequent tool invocation, retry-after-error patterns—but cannot diagnose the actual failure (wrong function name) or switch to a mathematical approach. After exhausting tool calls, it degenerates into repetitive text enumeration. This illustrates why early SFT checkpoints are unsuitable for RL initialization (Section 5.2): outcome-only rewards would further reinforce these degenerate patterns.

D.3 Case Study: Our Model Interleaved Thinking with Code Execution

Case Study: Our Model Interleaved Thinking with Code Execution

Problem: (BeyondAIME) Suppose there is a necklace composed of beads of two colors, red and blue, with a total of 16 beads. The necklace can be freely rotated and flipped. If on this necklace there are neither three consecutive blue beads nor three consecutive red beads, how many different necklaces meet the above requirements? Ground truth: 88

Reasoning: Let X be the set of colourings of the 16-bead cycle that contain no three equal consecutive beads. The symmetry group is D_{16} with $|D_{16}| = 32$. By Burnside's lemma, $N = \frac{1}{|D_{16}|} \sum_{g \in D_{16}} |\mathrm{Fix}(g)|$. A rotation with $\gcd(16, k) = d$ fixes exactly the number of cyclic binary strings of length d with no three consecutive identical beads. Let me first compute these counts via brute force.

Tool Call #1: Enumerate valid cyclic strings by length

```
def count_no_three_run(n):
    count=0
    for mask in range(1<<n):
        bits=[(mask>>i)&1 for i in range(n)]
        ok=True
        for i in range(n):
            if bits[i]==bits[(i+1)%n]==bits[(i+2)%n]:
                ok=False; break
        if ok: count+=1
    return count
for n in [1,2,4,8,16]: print(n, count_no_three_run(n))
```

Output

```
1 0  2 2  4 6  8 46  16 2206
```

Reasoning: Good. So $f(1) = 0,\ f(2) = 2,\ f(4) = 6,\ f(8) = 46,\ f(16) = 2206$. The sum over rotations is $f(16) \cdot 1 + f(8) \cdot 1 + f(4) \cdot 2 + f(2) \cdot 4 + f(1) \cdot 8 = 2272$. Reflections need a separate count: split the 16 axes into *through-bead* and *through-gap* families, and enumerate the symmetric bit assignments under each via code.

Tool Call #2: Count colourings fixed under reflections

```
def count_symmetric_valid(reflection_type):
    n=16; pairs=[]; fixed=[]
    if reflection_type=='through_beads':
        fixed=[0,8]
        for i in range(1,8): pairs.append((i, n-i))
    elif reflection_type=='through_gaps':
        for i in range(8): pairs.append((i, (-i-1)%n))
    ...( assigns bits to fixed/paired positions, checks no-three-run condition )...
print('through_beads', count_symmetric_valid('through_beads'))
print('through_gaps', count_symmetric_valid('through_gaps'))
```

Output

```
through_beads 42      through_gaps 26
```

Reasoning: So the reflection sum is $8 \times 42 + 8 \times 26 = 544$, giving $N = (2272 + 544)/32 = 88$. Let me verify this independently by enumerating all 2206 valid colourings and directly computing dihedral orbits. First, collect all valid colourings into a list.

Tool Call #3: Collect all valid colourings into `valid_colorings`

```
def satisfies_condition(colors):
    n = len(colors)
    for i in range(n):
        if colors[i] == colors[(i+1)%n] == colors[(i+2)%n]:
            return False
    return True

def all_colorings_valid():
    n=16; valid=[]
    for mask in range(1«n):
        colors=[(mask»i)&1 for i in range(n)]
        if satisfies_condition(colors):
            valid.append(tuple(colors))
    return valid

valid_colorings = all_colorings_valid()
len(valid_colorings)
```

Output

```
2206
```

Reasoning: Good, 2206 matches the earlier count. Now directly compute orbits under D_{16} using the `valid_colorings` list that is already in the sandbox.

Tool Call #4: Direct orbit enumeration (reuses `valid_colorings` from #3)

```
valid = valid_colorings    # reuses variable from previous call

def rotate(seq, k):
    return tuple(seq[(i+k)%n] for i in range(n))
def reflect(seq):
    return tuple(seq[(-i)%n] for i in range(n))

def dihedral_images(seq):
    images = []
    for k in range(n):
        images.append(rotate(seq, k))
        images.append(rotate(reflect(seq), k))
    return images

seen=set(); orbits=[]
for seq in valid:
    if seq not in seen:
        for img in dihedral_images(seq): seen.add(img)
        orbits.append(seq)
len(orbits)
```

Output

```
88
```

. . . (Tool calls #5–#9 further cross-validate via transfer-matrix $\mathrm{tr}(A^n)$ *and linear-string recurrence counts.). . .*

Reasoning: Both Burnside's formula and direct orbit enumeration give $N = 88$, and the transfer matrix confirms $f(n) = \mathrm{tr}(A^n)$. Therefore $N = \frac{2272+544}{32} = \boxed{88}$.

Analysis: The model identifies the Burnside framework through text reasoning but offloads every fixed-point count to the sandbox, avoiding the error-prone mental arithmetic that causes the baseline to fail (cf. Figure 1). Tool calls #3 and #4 demonstrate stateful sandbox usage: `valid_colorings` computed in #3 is directly reused in #4 without re-enumeration, enabling an independent orbit-based verification path that would be infeasible with stateless execution.

E CODE-USE BEHAVIOR ANALYSIS

E.1 CODE-USE BEHAVIOR CLASSIFICATION

To analyze what TIR contributes during problem solving, we categorize code behavior by its *computational purpose* in the surrounding context, assigning each trajectory one primary purpose. We use four categories: **Empirical Discovery**, where code computes small cases to reveal a pattern or invariant; **Algorithmic Search**, where code enumerates a large solution space such as brute-force candidates, combinatorial configurations, or DP states; **Computation Offloading**, where code performs precise arithmetic or symbolic manipulation after the model has identified what to compute; and **Conjecture Verification**, where code checks a result or intermediate claim already derived in natural-language reasoning.

GPT-5.5 annotation. We independently repeated the computational-purpose annotation with GPT-5.5 and use this annotation for Figure 7 and the main-text analysis. The distribution is 27.5% computation offloading, 28.9% empirical discovery, 22.2% algorithmic search, and 21.4% conjecture verification. Percentages describe the distribution over these four purposes, not the overall rate of tool invocation. The resulting analysis is descriptive: it characterizes the roles of execution in solved trajectories rather than measuring causal necessity or reporting an inter-annotator agreement score.

E.2 REPRESENTATIVE EXAMPLES

We illustrate the four categories with representative, condensed TRICE-30B trajectories from BeyondAIME.

Empirical Discovery

Problem: For positive integers $a < b$, if $a^3 = (ka + 1)b + k$, then k is called good. Find the sum of all good integers $k \leq 1997$.
Reasoning: ... Let's try iterating over k and a maybe. For each k up to limit, we need to find $a > k$ such that there exists integer $b > a$ with $b = (a^3 - k)/(ka + 1)$ ...

Python Code

```
def good_ks(limit=1997, a_max=5000):
    good=set()
    for a in range(2, a_max+1):
        for k in range(1, min(limit, a-1)+1):
            den = k*a + 1
            num = a**3 - k
            if num % den == 0:
                b = num//den
                if b > a: good.add(k)
    return good
good = good_ks(1997, a_max=2000)
len(good), sorted(list(good))[:20]
```

Output

```
(16, [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 27, 30, 64, 112, 418])
```

Purpose: **Empirical discovery.** The search results expose regular families of good k's, which the model later generalizes analytically.

Algorithmic Search

Problem: There are 60 real numbers $a_1, \ldots, a_{60}$ with cyclic adjacent differences at most 1 and total sum 0. Maximize $a_{12} + a_{24} + a_{36} + a_{48} + a_{60}$.
Reasoning: ... Let's set up LP. Variables a_i continuous. Constraints: sum $a_i = 0$ and $|a_i - a_{i+1}| \leq 1$. Objective maximize the selected five. Solve and get optimal value.

Python Code

```
prob = pulp.LpProblem('max_sum', pulp.LpMaximize)
a = pulp.LpVariable.dicts('a', range(1,61), lowBound=None)
prob += pulp.lpSum([a[i] for i in [12,24,36,48,60]])
prob += pulp.lpSum([a[i] for i in range(1,61)]) == 0
for i in range(1,61):
    j = 1 if i==60 else i+1
    prob += a[i] - a[j] <= 1
    prob += a[j] - a[i] <= 1
prob.solve(pulp.PULP_CBC_CMD(msg=False))
print(pulp.value(prob.objective))
```

Output

```
15.0
```

Purpose: **Algorithmic search.** Code solves the constrained optimization problem directly over the feasible space.

F LIMITATIONS AND FUTURE WORK

This work primarily focuses on tool-integrated reasoning for competition-level mathematics, where problems are challenging, verifiable, and naturally benefit from executable computation. Although our math-only recipe shows promising transfer to scientific reasoning, knowledge-intensive QA, and coding tasks, broader agentic settings remain underexplored. Therefore, we plan to extend this recipe beyond mathematics to broader agentic settings, such as software engineering workflows and web interaction. In addition, our experiments are mainly conducted on 4B and 30B models. As our analysis shows, the 4B and 30B models already exhibit slightly different training dynamics, so scaling to substantially larger models may introduce different learning regimes, stability issues, or tool-use patterns. We leave the systematic study of these extensions to future work.