Agentic-R1: Distilled Dual-Strategy Reasoning

Anonymous ACL submission

Abstract

Current long chain-of-thought (long-CoT) models excel at mathematical reasoning but rely on slow and error-prone natural language traces. Tool-augmented agents address arithmetic via code execution, but often falter on complex logical tasks. We introduce a fine-tuning framework, **DualDistill**, that distills complementary reasoning strategies from multiple teachers into a unified student model. Using this approach, we train **Agentic-R1**, which dynamically selects the optimal strategy for each query, invoking tools for arithmetic and algorithmic problems and using text-based reasoning for abstract ones. Our method improves accuracy on computation-intensive tasks and reduces inference latency on standard benchmarks, demonstrating the promise of multi-strategy distillation for robust and efficient reasoning.

1 Introduction

A recently proposed reasoning paradigm for language models, long chain-of-thought (long-CoT) reasoning, has achieved state-of-the-art performance on challenging tasks such as mathematical problem solving (Guo et al., 2025; Jaech et al., 2024). By allocating a large inference budget, these models generate reasoning trajectories with iterative self-verification and refinement. Despite this progress, open-source long-CoT models remain limited: their reasoning traces rely solely on natural language, which is both computationally expensive and error-prone without explicit verification.

In contrast, tool-aided reasoning provides greater efficiency and reliability, particularly for large-scale numerical computations and tasks that require rigorous verification (Gao et al., 2023). Advanced agent frameworks, such as OpenHands (Wang et al., 2024), place language models in a multi-turn environment with a code interpreter and other tools. The resulting agentic trajectories are effective for tool-intensive tasks but often fall short on abstract or conceptually complex reasoning problems (Duan et al., 2024).

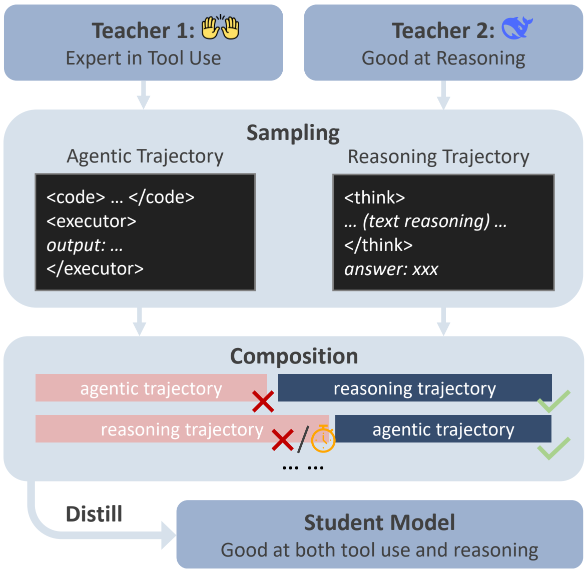


Figure 1: **Overview of *DualDistill*.** We distill knowledge from two complementary teacher models. Trajectories from both teachers are sampled and composed based on correctness, enabling the student model to learn when and how to elect the most appropriate one for each problem.

To leverage the strengths of both reasoning and tool-based strategies, we introduce **DualDistill**, a novel distillation framework (Fig. 1) that combines trajectories from two complementary teachers: one reasoning-oriented, the other tool-augmented, in a unified student. The resulting model, **Agentic-R1**, learns to mix both strategies and dynamically selects the most appropriate one for each problem, executing code for arithmetic and algorithmic tasks and reasoning in natural language for abstract ones. Our contributions are as follows:

- **DualDistill**, a distillation method that enables a language model to learn from multiple teacher models with complementary capabilities via trajectory composition.

- **Agentic-R1**, a distilled model that achieves strong performance in mathematical tasks requiring both tool use and reasoning, while maintaining competitive accuracy on tasks best handled by a single strategy.

2 Related Work

While prior efforts have integrated external tools into language models (Gao et al., 2023; Schick et al., 2023; Nakano et al., 2022), they are often specialized to either non-math domains or are confined to shorter reasoning chains. Concurrently, the paradigm of long chain-of-thought (long-CoT) reasoning or inference-time compute has demonstrated significant improvements (Guo et al., 2025; OpenAI et al., 2024). However, these approaches can be difficult to control and may suffer from 'overthinking', particularly when applied to tool-use scenarios (Cuadron et al., 2025). Some recent works have combined tool use with long reasoning (Feng et al., 2025; Song et al., 2025), but these are often applied to different domains or rely on reinforcement learning, which can be less stable than our proposed distillation method. To the best of our knowledge, **DualDistill** is the first framework to employ distillation with trajectory composition from two heterogeneous teacher models, one specializing in agentic tool-use and the other in pure textual reasoning, creating a unified student model capable of adaptively leveraging both strategies. See Appendix B for a more detailed discussion.

3 Method

As illustrated in Fig. 1, **DualDistill** uses trajectory composition to distill the knowledge of the complementary teachers to the student model.

3.1 Trajectory Composition

Let $\mathcal{D} = \{(x_i, a_i)\}$ be a training set where x_i denotes the i-th problem and a_i is its ground-truth solution, and let π_A and π_R be two distinct teacher policies, where the former is for the *agentic* teacher and the latter is for a *reasoning* teacher. For each training instance (x, a), we randomly select the initial teacher by sampling a binary indicator $z \sim \text{Bernoulli}(0.5)$ and then produce solutions y_1 and y_2 as follows:

$$y_1 \sim z\pi_A(\cdot \mid x) + (1-z)\pi_R(\cdot \mid x),$$
$$y_2 \sim (1-z)\pi_A(\cdot \mid x, y_1) + z\pi_R(\cdot \mid x, y_1).$$

That is, after one teacher generates the initial solution y_1, the other teacher subsequently generates the second solution y_2, conditioned on both the original problem x and the preceding solution y_1.

We evaluate the correctness of each solution using a rule-based grader, assigning binary correctness scores $s_1, s_2 \in \{0, 1\}$ to y_1 and y_2, respectively. The distilled training trajectories are then composed based on these correctness scores.

- $s_1 = 0, s_2 = 1$: The first teacher produces an incorrect solution, and the second teacher successfully corrects it. The composed trajectory is structured as $y_1 \oplus t_{-+} \oplus y_2$. (Here $\oplus$ donates concatenation and t_{-+} is a transition segment, described later).

- $s_1 = 1, s_2 = 1$: Both teachers provide correct solutions. We create a trajectory $y_1 \oplus t_{++} \oplus y_2$ to reflect complementary correct strategies.

- $s_1 = 1, s_2 = 0$: Only the initial teacher provides a correct solution. In this scenario, the composed trajectory includes only y_1.

- $s_1 = 0$, $s_2 = 0$: Both teachers do not solve the problem correctly. In this case, we just discard the problem without composing any trajectory.

The transition segments t_{-+} and t_{++} are predefined sentences indicating strategy shifts (e.g., *"Wait, using text reasoning is too tedious, let us try code reasoning."*). More examples and detailed descriptions can be found in Appendix A.4.1.

3.2 Training Instance Selection

We curate a training set with the instances for which one strategy has a clear advantage over the other in performance. Using an existing data set such as GSM8K (Cobbe et al., 2021) would be insufficient in this sense as most of the problems are relatively simple and can be solved by either strategy without a significant performance difference. Instead, we construct two contrasting subsets of math problems: one can benefit more from tool-assisted reasoning, while the other can benefit more from pure text-based reasoning. After composition, we apply additional filtering to balance the training dataset, resulting in 2.6k distilled trajectories. Detailed statistics can be found in Appendix A.3.2. Further filtering details are provided in the Appendix A.3.1.

3.3 Teacher and Student Models

As the teacher of agentic reasoning, we utilize OpenHands (Wang et al., 2024), a tool-assisted agent built upon *Claude-3.5-Sonnet* (Anthropic, 2024) to employ human-designed problem-solving pipelines. As the teacher for text-based reasoning, we adopt *Deepseek-R1*. The details can be found in Appendix A.4.2.

As for the student model, we adopt *Deepseek-R1-Distill-7b*, which has been fine-tuned on pure text-based reasoning trajectories and has also been exposed to code-related data during pre-training. We deliberately choose a model already familiar with both modalities to minimize the amount of training data required for the strategic composition. We want to examine whether it can effectively learn multiple problem-solving strategies.

4 Experiments

4.1 Benchmarks

We evaluate our method on several benchmarks that test different aspects of mathematical reasoning, including tasks where tool-aided calculation is hypothesized to provide a significant advantage.

DeepMath-Large. DeepMath (He et al., 2025) is a comprehensive dataset of mathematical and STEM problems compiled from various benchmarks. To evaluate the effectiveness of our method on numerically intensive tasks, we curate a subset of 87 problems where the answers are large integers (absolute value greater than 10^5). These problems are excluded from our fine-tuning data, though they may appear in pretraining corpora. We refer to this evaluation set as *DeepMath-Large*, with the assumption that code-aided computation is more effective in solving such problems.

Combinatorics300. This benchmark consists of 300 combinatorics problems aggregated from diverse math test sets. Each problem yields an answer larger than 10^4, reflecting the factorial growth in combinatorial counts. We hypothesize that tool-aided reasoning is important for handling the enumeration and sampling required in such tasks.

Standard Mathematical Benchmarks. To assess the generalizability of our approach, we further evaluate on widely-used mathematical reasoning tasks, including MATH500 (Lightman et al., 2023), AMC (AI-MO, 2024), and AIME (2025 Parts I and II) (AIME, 2025).

4.2 Baselines

We compare against the following strong baselines:

- **DeepSeek-R1-Distill.** A distilled version of DeepSeek-R1 fine-tuned on long chain-of-thought trajectories, representing a strong baseline for pure language-based reasoning.

- **Qwen-2.5-Instruct (w./w.o. tools)** (Yang et al., 2024). A general-purpose short-CoT model with optional tool-use capabilities. The tool-augmented variant serves as a competitive baseline for tool-aided strategies.

The training configuration details are provided in Appendix A.2.

4.3 Evaluation Metrics

To evaluate both reasoning quality and computational efficiency, we adopt the *Accuracy at Budget* metric. Given the model output $\mathbf{O} = (O_0, O_1, \ldots, O_L)$ and a reference answer A. We define accuracy under the budget b as:

$$\mathrm{Acc}(b) = \mathrm{G}\big(O_{0\cdots\min(b,L)}, A\big),$$

where G is a binary grader that checks whether the output matches the ground truth. We report results under three budgets: *Small (S, 2048)*, which reflects the typical context limit of short-CoT models; *Medium (M, 4096)*, a moderate budget for long-CoT reasoning; and *Large (L, 32768)*, which approximates an unbounded budget and allows the model to reason adequately. Inference details can be found in Appendix A.6.

4.4 Results

As shown in Table 1, our student model, *Agentic-R1*, demonstrates substantial performance improvements on *DeepMath-Large* and *Combinatorics300*, two challenging datasets that benefit from both agentic and reasoning strategies. Specifically, our model outperforms two similarly sized models, each specializing exclusively in tool-assisted (*Qwen2.5-7b-Instruct*) or pure reasoning (*Deepseek-R1-distill-7b*) strategies. *Agentic-R1* surpasses tool-based models by intelligently adopting reasoning strategies when appropriate, while maintaining greater efficiency compared to pure reasoning models on standard mathematical tasks. However, we note a slight performance decrease in relatively simpler benchmarks (*MATH500*) compared to the pure text-reasoning model, and a detailed discussion is provided in the limitations section.

	Budget	DeepMath-L	Combinatorics300	MATH500	AIME	AMC	avg.
Qwen2.5-7b (w.o. tool)	S	17.2	21.5	75.1	8.0	42.7	32.9
	M	17.2	21.8	75.1	8.0	42.9	33.0
	L	17.5	21.8	75.2	8.0	42.9	33.1
Qwen2.5-7b (w. tool)	S	**34.5**	28.7	68.3	15.3	**51.6**	39.7
	M	34.7	28.9	68.4	14.7	50.8	39.5
	L	34.5	28.9	68.4	14.7	50.8	39.5
DeepSeek-R1-Distill-7b	S	20.5	26.5	**75.9**	9.3	44.1	35.3
	M	34.7	34.5	**82.8**	23.3	61.2	47.3
	L	56.3	44.7	**88.8**	**40.7**	**84.8**	63.1
Agentic-R1-7b (ours)	S	30.8	**31.5**	**75.9**	**17.3**	51.1	**41.3**
	M	**37.5**	**37.1**	80.2	**29.3**	**66.7**	**50.2**
	L	**61.8**	**50.9**	82.5	**40.7**	82.2	**63.6**

Table 1: **Main Results.** We evaluate performance on five benchmarks under three budgets: **S** (2048), **M** (4096), and **L** (32768). The results are averaged over 5 random seeds with $T = 0.6$. *Agentic-R1* demonstrates significant gains on *DeepMath-Large* and *Combinatorics300*, where both complex reasoning and tool use are crucial. It also exhibits efficiency on competitive math tasks, outperforming baselines under smaller budgets (S and M), while maintaining comparable performance under the large-budget setting (L).

Qualitative Examples. We provide illustrative trajectories demonstrating *Agentic-R1*'s adaptive strategy-switching capability: (1) initially using the tool-assisted strategy and then switching to textual reasoning to correct an incorrect initial solution (Fig. 6); and (2) starting with textual reasoning and then switching to the tool-assisted strategy to bypass tedious manual calculations (Fig. 7).

Agentic-R1 Knows When to Use Tools. An intriguing observation is that *Agentic-R1* learns when to appropriately invoke code tools purely through supervised fine-tuning. For instance, the dataset *Combinatorics300* contains problems involving large numerical computations, making tools particularly beneficial. Consequently, *Agentic-R1* activates at least one code execution tool in 79.2% of *Combinatorics300* problems, whereas the usage of the tool drops to only 52.0% in the relatively simpler *AMC* dataset.

Agentic-R1 Learns from Imperfect Teachers. Although OpenHands, based on *Claude-3.5-Sonnet*, is not a strong standalone reasoning agent and sometimes performs worse than the student's initial model (*R1-Distill*), the student model still effectively acquires valuable agentic strategies through distillation. For example, the agentic strategy teacher achieves only 48.4% accuracy on *Combinatorics300*, yet after training, the student's performance improves significantly from 44.7% to 50.9%, surpassing the teacher. This shows that demonstrations from an imperfect agentic teacher can still yield meaningful gains in the student.

4.5 Ablation Study

Dataset	DeepMath-Large	AIME	AMC
w/o. composition	40.0%	34.0%	50.8%
w. composition	**61.8%**	**40.7%**	**82.2%**

Table 2: **Trajectory Composition**. We compare performance between composition and non-composition in the large budget setting; composition is always better.

Trajectory Composition. To verify the effectiveness of our data composition strategy, we compare it with a training strategy that does not use composition, meaning that each student trajectory is either fully generated by the agentic teacher or fully generated by the reasoning teacher. As shown in Table 2, our composition strategy consistently surpasses its non-composition counterpart.

5 Conclusion

We propose **DualDistill**, an efficient distillation framework based on data composition, allowing a student model to learn from multiple teacher models specialized in different domains of problem solving. Using the appropriate strategy for each problem, our trained model, *Agentic-R1*, achieves superior performance in benchmarks that require both reasoning and tool-assisted capabilities. This approach demonstrates the potential for unifying diverse problem-solving strategies within a single model, opening new directions for building versatile and adaptive language agents.

Limitations

While our approach demonstrates strong overall performance, several limitations remain that suggest avenues for future work. First, we observe a slight performance decrease in relatively simple reasoning tasks such as *MATH500* and *AMC*, compared to the baseline of pure text reasoning baseline (*Deepseek-r1-distill*). We hypothesize two main reasons. These tasks are well-suited for pure text-based reasoning, with the baseline achieving over 84% accuracy. In such cases, tool-aided reasoning offers limited additional benefit. For example, when using tool assistance, *Qwen-2.5-7b-Instruct* achieves only around 68.4% accuracy on *MATH500*—a performance drop compared to the tool-free baseline, indicating that code execution may be less useful for relatively simple problems. (2) Our current training scheme relies solely on strategy distillation. Combining preference-based learning methods, such as expert iteration (Polu et al., 2022) or DPO (Rafailov et al., 2023) afterwards, may help the student model better select the appropriate reasoning strategy.

Second, our training dataset contains approximately 2.6k trajectories. While this appears sufficient to teach a model that has been pre-trained on both text reasoning and code generation (e.g., *Deepseek-R1-Distill-7b*) to choose between strategies, it is likely insufficient for training a model to learn a new reasoning strategy from scratch. For example, *Deepseek-R1-Distill* was fine-tuned on over 800k distilled examples to acquire long CoT reasoning capabilities. Expanding the dataset and covering a wider range of strategies will be an important direction for future research.

References

Pranjal Aggarwal and Sean Welleck. 2025. L1: Controlling how long a reasoning model thinks with reinforcement learning. *arXiv preprint arXiv:2503.04697*.

AI-MO. 2024. AIMO Validation AMC. Dataset, Apache-2.0 licence. Accessed 2025-05-17.

AIME. 2025. American invitational mathematics examination (aime). `https://www.maa.org/math-competitions/aime`. Organized by Mathematical Association of America (MAA).

Anthropic. 2024. Claude 3.5 sonnet model card addendum. Technical report, Anthropic. Accessed 16 May 2025.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Preprint*, arXiv:2211.12588.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Alejandro Cuadron, Dacheng Li, Wenjie Ma, Xingyao Wang, Yichuan Wang, Siyuan Zhuang, Shu Liu, Luis Gaspar Schroeder, Tian Xia, Huanzhi Mao, Nicholas Thumiger, Aditya Desai, Ion Stoica, Ana Klimovic, Graham Neubig, and Joseph E. Gonzalez. 2025. The danger of overthinking: Examining the reasoning-action dilemma in agentic tasks. *Preprint*, arXiv:2502.08235.

Jinhao Duan, Renming Zhang, James Diffenderfer, Bhavya Kailkhura, Lichao Sun, Elias Stengel-Eskin, Mohit Bansal, Tianlong Chen, and Kaidi Xu. 2024. Gtbench: Uncovering the strategic reasoning limitations of llms via game-theoretic evaluations. *arXiv preprint arXiv:2402.12348*.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025. Retool: Reinforcement learning for strategic tool use in llms. *Preprint*, arXiv:2504.11536.

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

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

Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, and 1 others. 2025. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. *arXiv preprint arXiv:2504.11456*.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. *Preprint*, arXiv:2305.02301.

HuggingFace. 2025. Math-verify: A robust mathematical expression evaluation system.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *Preprint*, arXiv:2503.09516.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Xiang Li, Shizhu He, Jiayu Wu, Zhao Yang, Yao Xu, Yang jun Jun, Haifeng Liu, Kang Liu, and Jun Zhao. 2024. MoDE-CoTD: Chain-of-thought distillation for complex reasoning tasks with mixture of decoupled LoRA-experts. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 11475–11485, Torino, Italia. ELRA and ICCL.

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

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *Preprint*, arXiv:2501.19393.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2022. Webgpt: Browser-assisted question-answering with human feedback. *Preprint*, arXiv:2112.09332.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, and 244 others. 2024. Openai o1 system card. *Preprint*, arXiv:2412.16720.

Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. 2022. Formal mathematics statement curriculum learning. *arXiv preprint arXiv:2202.01344*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Preprint*, arXiv:2302.04761.

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. 2025. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *Preprint*, arXiv:2503.05592.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, and 1 others. 2024. Openhands: An open platform for ai software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2025. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *Preprint*, arXiv:2408.00724.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

A Appendix

A.1 Code and Dataset

Our training data and code are available in the Supplementary Material.

A.2 Training Configuration

Loss Masking. To prevent the student model from learning incorrect patterns from unsuccessful attempts, we exclude specific segments of trajectories from the loss calculation. Specifically, trajectory segments occurring before a transition from incorrect to correct reasoning (*i.e.,* t_{-+}) are omitted. Additionally, any code blocks resulting in execution errors are also excluded from influencing the loss computation.

Inference Prompt

System: A conversation between User and Assistant. The user asks a question, and the Assistant solves it. The assistant first thinks about the reasoning process in the mind and then provides the user with the answer. The reasoning process and answer are enclosed within <think> </think> and <answer> </answer> tags, respectively, i.e., <think> reasoning process here </think> <answer> answer here </answer>.
The final answer should be enclosed within boxed tags, i.e., $\boxed{\text{answer here}}$.
Meanwhile, you can use Python code to help you reasoning. The code should be enclosed within <code> </code> tags. For example, <code> code here </code>.
A executor will run the code and provide feedback immediately after the code. The executor feedback should be enclosed within <executor> </executor> tags.
You can use the executor feedback to improve your reasoning.

Figure 2: **Inference Prompt.** The system prompt used to guide the model during inference. Instructions highlighted in brown indicate guidance specific to tool usage.

Hyperparameters. For training the student model **Agentic-R1**, we use 4 × A6000 GPUs over a total of 12.7 hours. The model is trained for 4 epochs using the AdamW optimizer (Loshchilov and Hutter, 2017) with a learning rate of 1×10^{-5}. We set the maximum context length to 16,384 tokens and discard any training examples that exceed this limit.

A.3 Dataset Details

A.3.1 Problem Filtering Heuristics

To curate a training dataset that can guide a student model in learning when to apply agentic versus pure text-based reasoning, we construct two subsets of mathematical problems.

Agentic-Favored Subset. We identify problems where tool use is highly beneficial using two heuristics:

- **Numerical Scale:** Problems whose final integer answers exceed an absolute value of $1,000$ often require non-trivial arithmetic operations or algorithms that are more suitable for tool-assisted computation.
- **Difficulty Under Constraints:** We use a baseline text reasoning-only model, *Deepseek-r1-Distill-7b*, with a limited context length (4096 tokens). Problems unsolvable under these constraints are deemed more difficult and suitable for agentic strategies.

Pure Reasoning-Favored Subset. To balance the dataset, we include problems in which agent execution is error-prone. These are selected by identifying the cases where the tool-assisted method fails and produces incorrect output.

We apply this selection process to DeepMath-103K (He et al., 2025) and balance the two subsets to ensure that the model sees roughly equal representation from both strategies during training.

A.3.2 Dataset Scale

type	$s_1, s_2 = 1, 1$	$s_1, s_2 = 1, 0$	$s_1, s_2 = 0, 1$
number	685	600	1393

Table 3: **Dataset Scale**. We report the number of training examples in each correctness category.

After running the two teachers on the filtered subset and compositing the trajectories, the final distilled dataset contains $2,678$ examples. The detailed number for each correctness category is listed in Tab. 3.

A.3.3 License

Our training dataset is constructed based on existing datasets, language models, and software. The following lists the relevant resources and their corresponding licenses:

- Openhands: An open-source agent framework under the MIT License;
- Deepseek-R1: An open-source language model under the MIT License;
- Claude-3-5-Sonnet: A commercial language model under a proprietary license; accessible via the Anthropic API and supported cloud platforms;
- MATH500: An open-source math dataset under the MIT License;
- DeepMath: An open-source math dataset under the MIT License.

All third-party resources were used in accordance with their licenses and intended use, as publicly specified. Our own model and dataset will be released under the MIT License.

A.4 Composition Trajectory

A.4.1 Transition Segment

When the teacher changes, a hand-designed transition segment is added to signify and point out the meaning of the transition. There are three typical transition segments t, which are shown in Table 4.

Meaning	Content
tool (×) → text (✓)	`Wait, the code is not correct, let's try text reasoning.`
text (×) → tool (✓)	`Wait, use text reasoning is too tedious, let's try code reasoning.`
A (✓) → B (✓)	`Wait, we can also use {B}-reasoning as an alternative way to verify the solution.`

Table 4: **Transition Segment.** The transition segments are used to connect trajectories from different teachers. 'Tool' and 'text' in the table represent agentic and pure text reasoning strategies, respectively. ✓ and × mean whether the trajectory is correct or not.

A.4.2 Trajectory Composition Implementation

To transform multiturn agentic trajectories from OpenHands logs into a suitable training format, we extract content from log fields labeled '*thought*', '*code*', and '*final thought*' along with their associated executor feedback if any. Each extracted segment is then enclosed within distinct resource tags—*<think></think>*, *<code></code>*, *<answer></answer>* or *<executor></executor>*—and concatenated sequentially. For reasoning trajectories from *Deepseek-R1*, we specifically apply the *<answer></answer>* tag to segments that lie outside the long-chain-of-thought reasoning portion (i.e., beyond the *<think></think>* segment).

We aim for the student model to inherently select the most efficient strategy, and we adopt the average number of inference tokens consumed for the efficiency measure. Thus, we enforce a token budget on the first teacher's inference: if y_1 does not complete within a randomly determined inference budget L_0 (sampled between 3072 and $+\infty$), the inference is stopped and labeled unsuccessful. Conversely, we do not impose any token budget constraint on y_2.

During preliminary experiments, we observed substantial differences in the distributional characteristics between OpenHands trajectories (π_A) and *Deepseek-r1* trajectories (π_R). To avoid performance degradation of y_2 due to potential contamination from combined inputs, we assume conditional independence and explicitly define the teacher model inference policy as $\pi(\cdot \mid x, y_1) = \pi(\cdot \mid x)$.

A.5 Qualitative Example

We observed that **Agentic-R1** shows several promising behaviors: (1) The model initially adopts tool-aided reasoning, but results in wrong results after several attempts, and then the model automatically switches to text reasoning and finally derives the correct answer (Fig. 6); (2) The model initially tries to apply text reasoning for a combinatorial problem, and then change to tool-aided reasoning to reduce computational complexity (Fig. 7).

A.6 Inference Details

For all evaluation experiments, we use the VLLM framework (Kwon et al., 2023) to enable fast inference via prefix caching, which significantly accelerates multiturn tool calling. In the tool-augmented setting, the language model is allowed to invoke a Python executor up to 10 times per problem, with each execution capped at 3 seconds. During inference, whenever the model outputs the special token *</code>*, the generation process is temporarily paused, the preceding code block is executed, and the resulting feedback is appended to the ongoing generation enclosed with *<executor> </executor>* before resuming inference. Although tool execution introduces up to 30 seconds of additional runtime per query, this cost is relatively small compared to the time-intensive pure text reasoning process, which can take several minutes to reach a conclusion using *Deepseek-r1-Distill-7b* on 2×A6000 GPUs. Additionally, the prompt template is listed in Fig 2.

For evaluation, we adopt *Math-Verify* (HuggingFace, 2025) to judge the correctness of models' outputs.

A.7 Full Results

We report the performance trend of different models tested in various budgets. Please refer to Fig. 4 for individual benchmarks and Fig. 3 for the average.

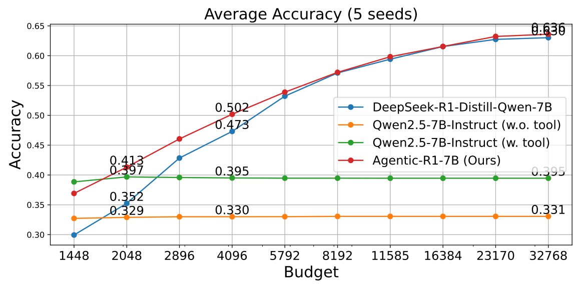


Figure 3: Average accuracy across benchmarks under various budgets.

B Related Work

Tool-Augmented Reasoning. Integrating external tools into language model chain-of-thoughts has substantially improved accuracy on numerical and factual tasks. Early program-aided methods such as PaL (Gao et al., 2023) and PoT (Chen et al., 2023) demonstrated significant gains by converting reasoning steps into executable programs, delegating precise computations to code interpreters. Other lines of work, including WebGPT (Nakano et al., 2022) and ReAct (Yao et al., 2023), introduced agent-like reasoning frameworks that interleave tool invocation (e.g., web searches or API calls) within multi-step reasoning. Toolformer (Schick et al., 2023) further generalized this approach by training language models to self-supervise API calls across various tasks such as arithmetic, translation, and retrieval. However, unlike *DualDistill*, these methods typically rely on short chain-of-thoughts primarily using prompting or heuristic-based tool invocation, lacking mechanisms for automatically balancing pure long reasoning against tool use based on task complexity.

Long Chain-of-Thought Reasoning. Recent approaches have highlighted significant performance improvements by scaling inference-time chain-of-thought length. OpenAI's O1 (OpenAI et al., 2024) and DeepSeek-R1 (Guo et al., 2025) used outcome-driven reinforcement learning to generate extensive reasoning trajectories, substantially outperforming shorter-CoT baselines on complex math and reasoning benchmarks. Similarly, S1 (Muennighoff et al., 2025) and L1 (Aggarwal and Welleck, 2025) demonstrated scaling curves showcasing a log-linear relationship between performance and inference compute. Empirical evidence supports that increased inference compute can often yield more cost-effective gains than increased model size alone (Wu et al., 2025). Nonetheless, long-CoT models frequently encounter overthinking i.e, generating overly long reasoning that leads to redundant or incorrect outcomes, especially in tool use scenarios, in a phenomenon known as reasoning-action dilemma (Cuadron et al., 2025). Our work addresses these issues by teaching a student model when to switch between internal reasoning and tool-based execution adaptively.

Reasoning Models with Tool-Calling. Recently, some works have explored the idea of combining long-form reasoning with explicit tool invocation. R1-Searcher (Song et al., 2025) and Search-R1 (Jin et al., 2025) introduced reinforcement-learning-based retrieval policies within reasoning loops, achieving substantial performance improvements on open-domain question answering tasks. However, unlike these methods, *DualDistill* is specifically tailored for math tasks. Similarly, ReTool (Feng et al., 2025) trained a reasoning model with tool-calling for math tasks. However, unlike these approaches that rely on expensive and unstable reinforcement learning techniques, *DualDistill* is a simple distillation approach, leading to a more data efficient and practical training setup.

Distillation in Large Language Models. Knowledge distillation is widely used to transfer capabilities from larger models to smaller, more efficient ones (Sanh et al., 2019; Hsieh et al., 2023). Recent extensions include multi-teacher distillation frameworks, which aggregate knowledge from multiple similar-structured teachers (Li et al., 2024). Nevertheless, existing distillation works typically assume homogeneous teacher models or single-modal reasoning paradigms. In contrast, our proposed *DualDistill* explicitly utilizes heterogeneous teacher models—one specialized in agentic tool-use and another in pure textual reasoning—and proposes an innovative way to compose trajectories that guide the student to effectively learn and combine from both of the two strategies.

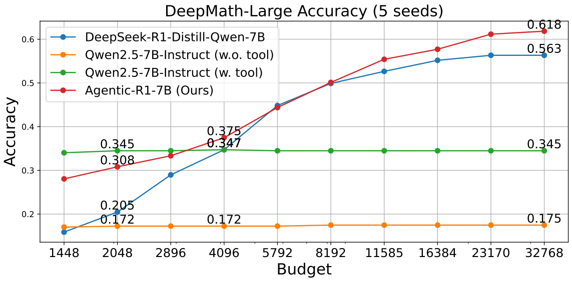
C Use of AI Assistants

For this project, we use AI assistants for paper editing (e.g., grammar, spelling, word choice) and to assist with the running of experiments (e.g., scripting, automation support).

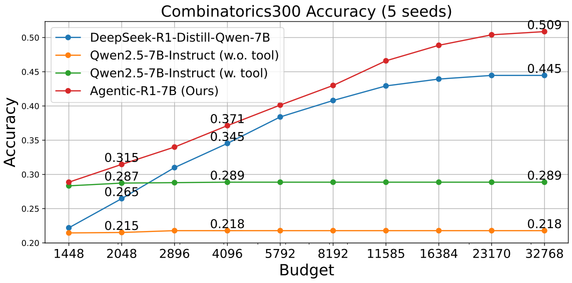
D Potential Risks

Our work focuses on a model distillation framework for mathematical reasoning tasks, using synthetic and publicly available data. It does not involve deployment or the use of personal data. Therefore, we do not identify significant risks related to safety, privacy, fairness, or security within the current scope.

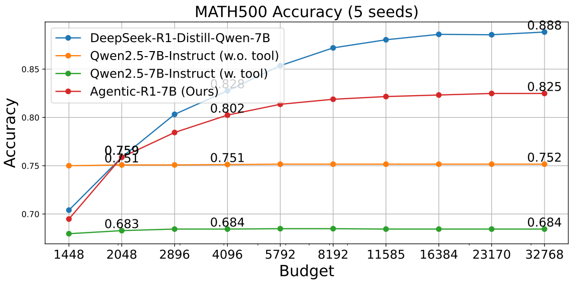
One potential data-related risk arises from the distillation of the teacher models. Although our training data is public, trajectory distillation may produce misleading or hallucinatory content. Such distilled data may require filtering if applied to sensitive domains in future work.



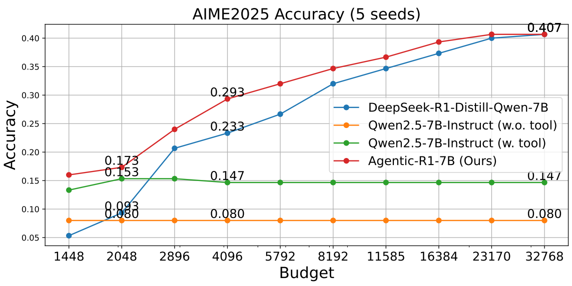
(a) Results on DeepMath-Large.



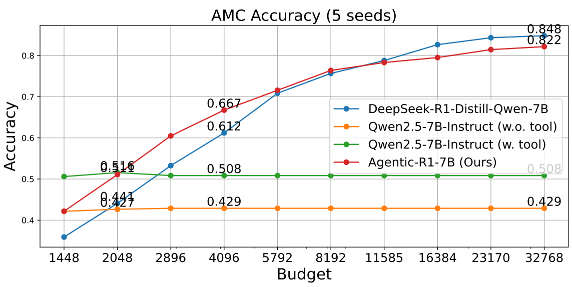
(b) Results on Combinatorics300.



(c) Results on MATH500.



(d) Results on AIME.



(e) Results on AMC.

Figure 4: Performance trend of different tested models under various budgets on the five benchmarks.

Agentic Trajectory Example

User: For real numbers x and y, define $\nabla(x, y) = x - \frac{1}{y}$. Evaluate $\underbrace{\nabla(2, \nabla(2, \nabla(2, \ldots \nabla(2, \nabla(2, 2)) \ldots)))}_{2016\ \nabla\text{s}} = \frac{m}{n}$ for relatively prime positive integers m and n, and compute $100m + n$.

Model:
<think>
Let me help solve this step by step.
1) Let's understand what we're calculating:
* We have a function $\nabla(x, y) = x - \frac{1}{y}$
* We're nesting this function 2016 times
* Each time, the first argument is 2
* The innermost y starts with 2
2) Let's try to see if there's a pattern by calculating the first few terms: Let's write a Python program to help us:
</think>
<code>

```
def nabla(x, y):
    return x - 1/y
# Let's calculate first several terms
result = 2  # Starting value
for i in range(10):  # First 10 iterations
    result = nabla(2, result)
    print(f\"After {i+1} iterations: {result}\")
```

</code>
<executor>

```
After 1 iterations: 1.5
... (omitted) ...
After 10 iterations: 1.0909090909090908
```

</executor>
<think>
Looking at the pattern, it seems the values are converging. Let's try to understand what's happening:
1) Let's call the sequence a_n where $a_1 = 2$ and $a_{n+1} = \nabla(2, a_n)$
2) For any term after the first: $a_{n+1} = 2 - \frac{1}{a_n}$
3) If this sequence converges to some value L, then: $L = 2 - \frac{1}{L}$
4) This gives us: $L^2 = 2L - 1$, $L^2 - 2L + 1 = 0$, $(L - 1)^2 = 0$, $L = 1$
5) Let's verify this is indeed what we get after 2016 iterations:
</think>
<code>

```
result = 2  # Starting value
for i in range(2016):
    result = nabla(2, result)
print(f\"After 2016 iterations: {result}\")
# Let's convert to fraction
from fractions import Fraction
final = Fraction(result).limit_denominator()
print(f\"\
As a fraction: {final}\")
# Calculate 100m + n
m = final.numerator
n = final.denominator
print(f\"\
100m + n = {100*m + n}\")
```

</code>
<executor>

```
After 2016 iterations: 1.0004957858205237
As a fraction: 2018/2017
100m + n = 203817
```

</executor>
<think>
After 2016 iterations, we get $\frac{2018}{2017}$, and $100m + n = 203817$. Let's verify these numbers are correct:
1) 2018 and 2017 are indeed relatively prime (2017 is prime)
2) The value is very close to but slightly larger than 1, which matches our convergence analysis
3) For $100m + n$: $100(2018) + 2017 = 203817$

Therefore, $\boxed{203817}$
</think>
<answer> $\boxed{203817}$ *</answer>*

Figure 5: **An example agentic trajectory derived from Openhands.**

Agentic-R1 Behavior I

User: Evaluate the limit:

$$\lim_{n\to\infty} \sqrt[n]{n^{(4n)} + (4n)^n} \left[\left(2 + \frac{1}{n^2}\right)^{18} - \left(4 + \frac{4}{n^2}\right)^9 \right]$$

Model:
<think>
Let me break this down and solve it step by step.
... (Omitted) ...
Let me write some code to help verify these observations.
</think>
<code>

```
import numpy as np
def first_expr(n):
    return (n**(4*n) + (4*n)**n)**(1/n)
def second_expr(n):
    return (2 + 1/n**2)**18 - (4 + 4/n**2)**9
# Let's test for large n
n = 1000
print(f"First expression for n={n}: {first_expr(n)}")
print(f"Second expression for n={n}: {second_expr(n)}")
```

</code>
<executor>

```
OverflowError: int too large to convert to float
```

</executor>
<think>
Let me modify the code to use logarithms to better understand the growth:
</think>
<code>

```
... (Omitted) ...
print(f"Log of first expression grows like {log_first_expr(1000)/1000}")
print(f"Log of second expression grows like {log_second_expr(1000)}")
```

</code>
<executor>

```
Log of first expression grows like 10.907755278982137
Log of second expression grows like 0.0
```

</executor>
<think>
Let me analyze this:
... (Omitted) ...
Let me verify this with very large n:
</think>
<code>

```
n = 10**10
first = (n**(4*n) + (4*n)**n)**(1/n)
second = (2 + 1/n**2)**18 - (4 + 4/n**2)**9
print(f"First expression = {first}") # This will show as inf, but we know it grows like n^4
print(f"Second expression = {second}")  # Should approach 0
```

</code>
<executor>

```
Error: Code execution timed out.
```

</executor>
<think>
... (Omitted) ...
Wait, the code is not correct, let's try text reasoning.
Okay, so I need to evaluate this limit:
... (Omitted) ...
However, 2^16 is 65536, and 65536 * 9 is indeed 589,824. So, perhaps the answer is 589,824.
</think>
<answer> $\boxed{589,824}$ *</answer>*

Figure 6: **Example Behavior I of *Agentic-R1*.** The model initially attempts to solve the problem using the tool to compute a limit expression, but fails after several attempts. Notably, it then transitions to a pure text-based reasoning strategy and successfully arrives at the correct solution.

Agentic-R1 Behavior II

User: Determine the number of tuples $(k_1, k_2, \ldots, k_{26})$ such that each k_i is either 0, 1, or 3, and the sum $k_1 + k_2 + \ldots + k_{26} = 15$.

Model:

<think>

Okay, so I need to figure out how many 26-tuples there are where each element is either 0, 1, or 3, and the total sum is 15.

... (Omitted) ...

Let me compute each term step by step.

First, y=0:

x=15, y=0. Then the number of tuples is C(26,15)*C(11,0). C(26,15) is the same as C(26,11) because C(n,k) = C(n, n-k). So C(26,11). C(11,0) is 1. So total is C(26,11).

Similarly, y=1:

x=12, y=1. Number of tuples is C(26,12)*C(14,1). C(26,12) is the same as C(26,14). C(14,1) is 14. So total is 14*C(26,14).

Wait, use text reasoning is too tedious, let's try code reasoning.

</think>

<think>

Let me think about this problem systematically.

Let's break this down:

1) We need to find the number of 26-tuples where each element is 0, 1, or 3
2) The sum of all elements must be 15
3) This is a constrained counting problem that can be solved using dynamic programming
4) Let's use Python to implement a solution
5) We'll need to:
   - Create a 2D DP table where dp[i][j] represents the number of ways to make sum j using first i numbers
   - For each position i, we can use 0, 1, or 3
   - We'll build this up from 0 to 26 numbers
   - The final answer will be in dp[26][15]

</think>

<code>

```
def count_tuples():
    # dp[i][j] represents number of ways to make sum j using first i numbers
    dp = [[0] * 16 for _ in range(27)]  # 27 numbers (0-26) and sums 0-15
    # Base case: 0 numbers can only make sum 0
    dp[0][0] = 1
    for i in range(1, 27):
        for j in range(16):
            if j >= 0: dp[i][j] += dp[i-1][j] # Try adding 0
            if j >= 1: dp[i][j] += dp[i-1][j-1] # Try adding 1
            if j >= 3: dp[i][j] += dp[i-1][j-3] # Try adding 3
    return dp[26][15]
result = count_tuples()
print(f"The number of valid 26-tuples is: {result}")
```

</code>

<executor>

```
The number of valid 26-tuples is: 853423740
```

</executor>

<think>

Let me verify this result with another approach to make sure it's correct:

</think>

<code>

```
... (Omitted) ...
verified_result = verify_result()
print(f"Verified result: {verified_result}")
```

</code>

<executor>

```
Verified result: 853423740
```

</executor>

<think>

I have verified the result using two different approaches and both give the same answer.

</think>

<answer> $\boxed{853,423,740}$ </answer>

Figure 7: **Example Behavior II of *Agentic-R1*.** The model first attempts to solve the combinatorial problem via text-based reasoning, but switches to tool-assisted reasoning due to computational complexity. It then implements a dynamic programming algorithm to solve the problem efficiently and verify the result.