START: Self-taught Reasoner with Tool

Anonymous ACL submission

Abstract

Large reasoning models (LRMs) like OpenAI-o1 and DeepSeek-R1 have demonstrated remarkable capabilities in complex reasoning tasks through the utilization of long Chain-of-thought (CoT). However, these models often suffer from hallucinations and inefficiencies due to their reliance solely on internal reasoning processes. In this paper, we introduce START (**S**elf-**Ta**ught **R**easoner with **T**ool), a novel tool-integrated long CoT reasoning LLM that significantly enhances reasoning capabilities by leveraging external tools. Through code execution, START is capable of performing complex computations , self-checking, exploring diverse methods, and self-debugging, thereby addressing limitations of LRMs. The core innovation of START lies in its self-learning framework, which comprises two key techniques: 1) Hint-infer: We demonstrate that inserting artificially designed hints (e.g., "Wait, maybe using Python here is a good idea.") during the inference process of a LRM effectively stimulates its ability to utilize external tools without the need for any demonstration data. Hint-infer can also serve as a simple and effective sequential test-time scaling method; 2) Hint Rejection Sampling Fine-Tuning (Hint-RFT): Hint-RFT combines Hint-infer and RFT by scoring, filtering, and modifying the reasoning trajectories with tool invocation generated by a LRM via Hint-infer, followed by fine-tuning the LRM. Through this framework, we have fine-tuned the QwQ-32B model to achieve the START. On PhD-level science QA (GPQA), competition-level math benchmarks (AMC23, AIME24, AIME25), and the competition-level code benchmark (LiveCodeBench), START achieves accuracy rates of 63.6%, 95.0%, 66.7%, 47.8%, and 47.3%, respectively. It significantly outperforms the base QwQ-32B and achieves performance comparable to the state-of-the-art open-weight model R1-Distill-Qwen-32B and the proprietary model o1-Preview.

1 Introduction

The evolution of reasoning capabilities in large language models (LLMs) has followed a paradigm shift marked by increasingly sophisticated thinking patterns. The chain-of-thought (CoT) approach (Wei et al., 2022) pioneers this progression by enabling models to decompose problems through explicit intermediate reasoning steps. Then, the breakthrough comes with reinforcement learning exemplified by OpenAI-o1 (OpenAI, 2024b) and DeepSeek-R1 (DeepSeek-AI, 2025), establishing a new paradigm termed long CoT, which emulates human-like cognitive strategies including self-refine, self-reflection, multi-strategy exploration and so on. Despite these advancements, fundamental limitations persist in long Chain-of-thought (CoT) approaches, such as hallucinations when facing complex computations or simulations, primarily due to their exclusive dependence on internal reasoning mechanisms. Tool-integrated reasoning (TIR) (Gou et al., 2024), another approach that improves traditional CoT through tool invocation(typically code interpreter), can effectively mitigate the issues that arise in Long CoT. Therefore, a straightforward yet important question arises: *How can we synergistically combine long CoT with TIR?*

In this paper, we focus exclusively on Python interpreter invocation, as it is both important and representative of many reasoning tasks (Shao et al., 2024; Yang et al., 2024). The fundamental challenge lies in synthesizing data that includes calls to a Python interpreter within Long Chain-of-thought (CoT). We have tried using direct prompt, well-designed prompt (Li et al., 2025), and in-context prompt (Gou et al., 2024; Schick et al., 2023) on AIME24 and LivecodeBench with QwQ (Qwen Team, 2024) and DeepSeek-R1, but none were successful in prompting the model to invoke the Python tool during the long CoT(see more in Ap-

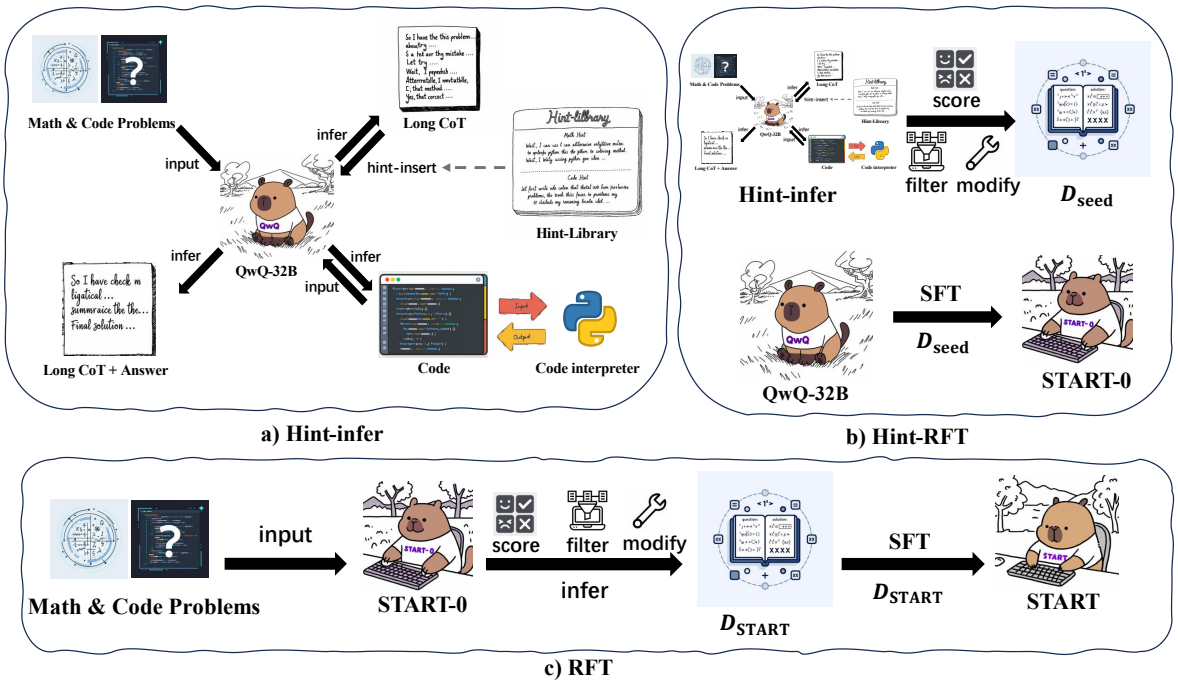

Figure 1: **Training framework for START.** Training Framework for START. START's training involves two phases: Hint-RFT followed by RFT. a) Hint-infer: code/math data is processed by QwQ, with responses truncated at predefined terminators. Context-aware hints from a Hint-Library are injected at truncation points (including endpoints), and QwQ resumes inference using a code interpreter for Python execution feedback. b) Hint-RFT: Hint-infer outputs undergo rule-based scoring, filtering, and content modification to create D_{seed} . QwQ is then fine-tuned on D_{seed} to produce START-0, enabling self-aware tool usage. c) RFT: START-0 generates self-distilled trajectories to build D_{START} (enhancing diversity/tool-use patterns), followed by fine-tuning to produce START.

pendix A.3). A possible reason is that large reasoning models(LRMs) typically focus solely on problem-solving during training for complex reasoning tasks, resulting in a loss of generalization in instruction following. Considering the nature of next-token prediction in LRMs (LLMs), we attempt to insert some hints directly during or at the end of the LRMs' reasoning process, aiming to directly prompt the model to write code and invoke code interpreter. We are surprised to discover that LLMs indeed possess the corresponding capabilities. For mathematical tasks, simply inserting basic hints along with Python identifiers enables the LLM to follow the hints and write the appropriate code. In contrast, for coding generation tasks, carefully designed hints and code templates are necessary to activate the model's ability to execute candidate code on test cases on its own during the long CoT. We refer to the paradigm of LRM inference aided by hints as **Hint-infer**.

Based on above Hint-infer, we present START: **S**elf-**Ta**ught **R**easoner with **T**ool, a LRM that synergizes Long CoT and TIR, which we refer to as Long TIR. The whole traing framework is illustrated in Figure1. First, we design a set of hints with different functionalities based on the cognitive characteristics of LLMs, which we refer to as the Hint-Library. Figure 3 presents some representative hints from the Hint Library. Second, these hints are randomly inserted after certain high-frequency conjunctions, such as "Alternatively" and "Wait" (see more in Figure 4), because these words typically indicate that the model begins to introspect or seek new solutions to the problem (Li et al., 2025). Additionally, we also add hints before the stop token of long CoT, as this approach provides the LRM with more time to think without disrupting its original reasoning process. We find it intriguing that when hints are added before the stop token of Long Chain-of-thought (CoT), the model exhibits a sequential test time scaling effect; that is, as the thinking time increases, the success rate of problem-solving also gets higher(see more in 4.5.2). Through a series of data scoring, filtering, modifications, and rejection sampling fine-tuning, we eventually obtain our START from QwQ. We do not choose the DeepSeek-R1-Distill-Qwen series as the input models, as we find that they seem

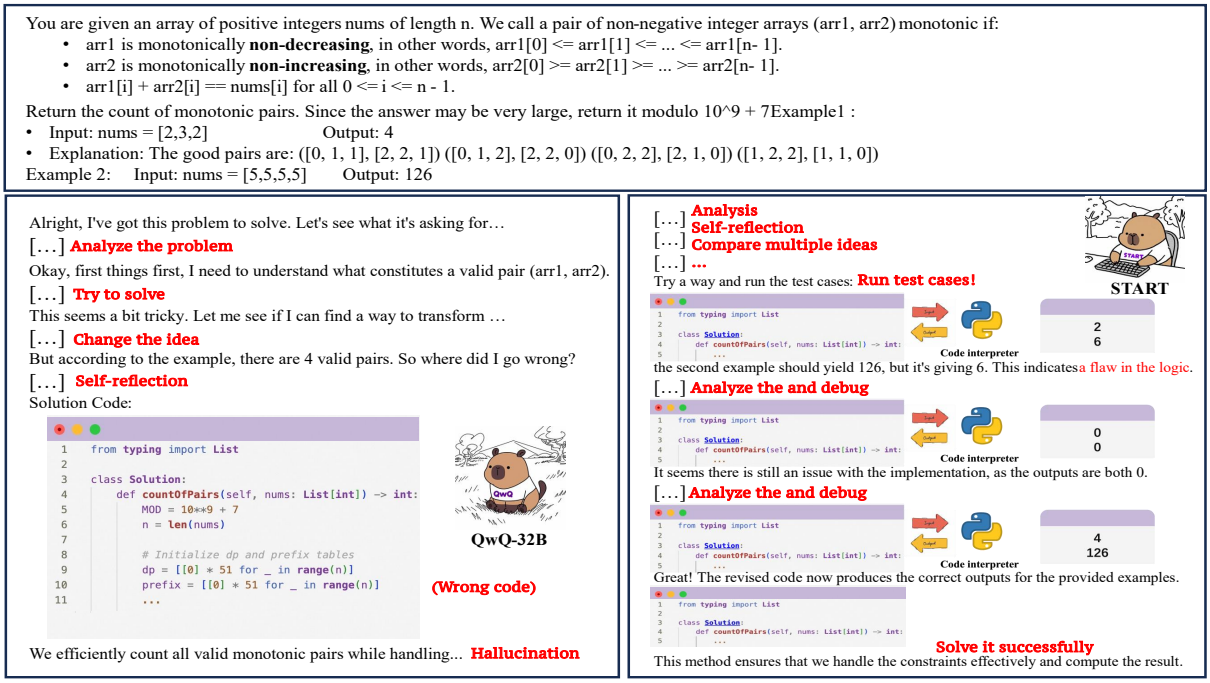

Figure 2: Comparison between the responses generated by QwQ and START. This is a question from LiveCodeBench with a difficulty level of "hard". QwQ employs long-chain CoT with self-reflection and trying different approaches, yet hallucinates during complex test case analysis, leading to flawed solutions. START retains QwQ's cognitive framework but integrates code execution: (1) Runs code via interpreter, (2) Detects output mismatch, (3) Iteratively analyzes and debugs (2 cycles), and (4) Gives the final solution.

to be prone to generating repetitive outputs and not good as a base model(The output is easy to be produce repetitive content). From Figure 2, it can be seen that there is a comparison between the reasoning of QwQ and START. When encountering a complex case analysis, QwQ-32 generates hallucinations and provides incorrect answers, while START utilizes a code interpreter to self-debug, delivering the correct answer.

Empirical evaluations across a suite of benchmarks encompassing mathematical problem-solving, scientific inquiries, coding challenges, and GPQA tasks demonstrate that START-RL markedly surpasses existing tool-integrated and long CoT models, including QwQ, o1-mini, and o1-preview. These results underscore the efficacy of integrating external tools into the long CoT framework, highlighting START-RL as the first open-source tool-integrated long CoT reasoning model that sets a new standard for LLM performance in complex reasoning domains.

In summary, our contributions are threefold:

- We introduce Hint-infer, a simple and effective sequential test-time scaling method for .
- We introduce Hint-RFT, a self-training framework that enables a large language model (LRM) to teach itself how to utilize code interpreter.
- We present START, the first open-source LRM that utilizes long CoT and code interpreter to address complex reasoning tasks.

2 Related Work

Large Language Models have demonstrated remarkable dominance across numerous Natural Language Processing tasks. To enhance the complex reasoning capabilities of LLMs, Wei et al. (2022) introduce Chain-of-Thought (CoT), which incorporates multi-step intermediate reasoning before arriving at final conclusions. CoT exhibits significant advantages across multiple domains, including mathematics, science, and programming. Subsequently, OpenAI (2023) further explore CoT and propose the Long Chain-of-Thought framework. In Long CoT, LLMs demonstrate advanced cognitive behaviors such as reflection, verification, correction, and multi-path exploration, thereby further enhancing their problem-solving capabilities in complex reasoning tasks. Moreover, Long

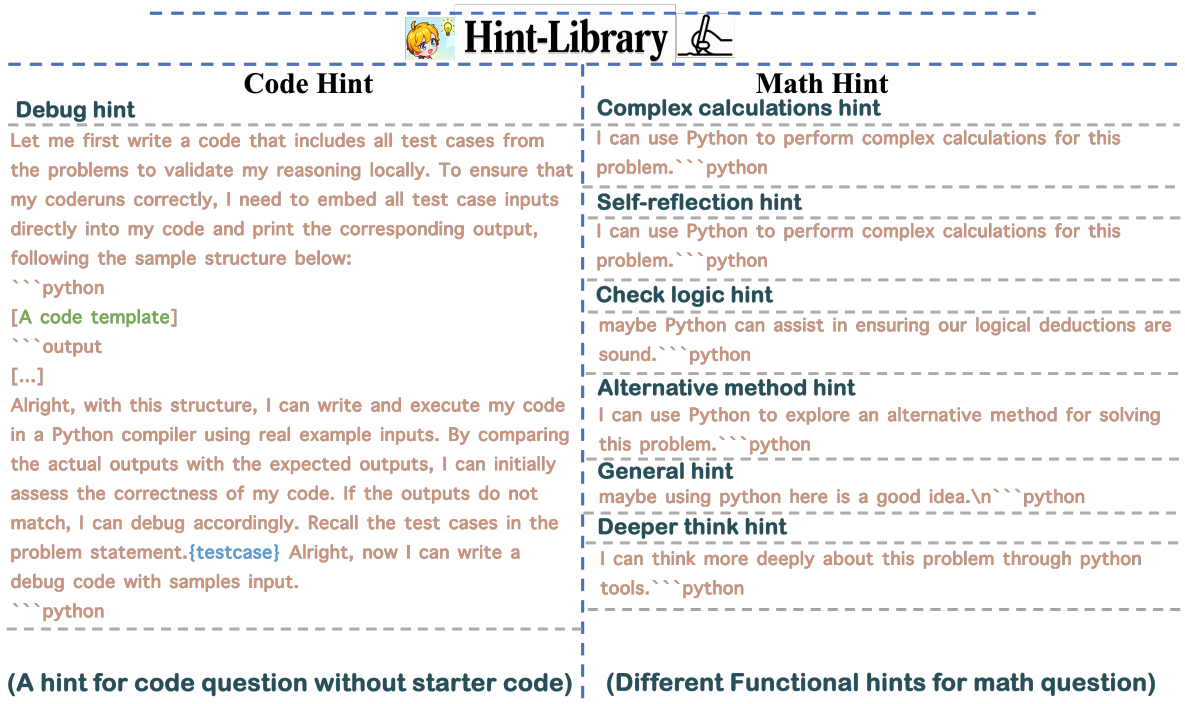

Figure 3: **Hint-Library**. Code generation tasks: Debug hint guides test case review and local code validation. The code template is in A.4. Math reasoning: Domain-specific hints (e.g., Complex Calculations, Self-Reflection, Logic Check, Alternative Methods) steer code-aided reasoning behaviors.

Figure 4: Word cloud of conjunction frequency statistics from QwQ infering on D_{seed}.

CoT exhibits excellent test-time scaling properties, where increased computational resources correlate with improved reasoning outcomes. Models like QwQ (Qwen Team, 2024), DeepSeek-R1 (DeepSeek-AI, 2025), and InternThinker (Cai et al., 2024) have successfully experimented with Long CoT for enhanced reasoning, combining fine-tuning and Reinforcement Learning to elevate the performance of open-source reasoning models to unprecedented levels. Notably, subsequent models such as Open-R1 (Huggingface, 2025) and S1 (Muennighoff et al., 2025a) observes significant benefits from Long CoT even in smaller models through simple distillation.

Nevertheless, significant challenges persist, particularly in addressing hallucination phenomena and computational inaccuracies that impede optimal performance. Drawing parallels with human cognition, where external aids such as scratch paper and calculators substantially mitigate computational errors, LLMs can similarly benefit from the integration of auxiliary tools. Research by Shao et al. (2024) demonstrates that code-based pre-training protocols significantly augment LLMs' mathematical reasoning proficiency. Various works successfully implemente Python-based computational tools to enhance model performance (Chen et al., 2023; Gou et al., 2024; Liao et al., 2024; Li et al., 2024). In the domain of mathematical proof verification, the incorporation of Lean yield notable advancements (Xin et al., 2024; Wu et al., 2024).

This study synthesizes the advantages of Python-based tools and long CoT methodologies, advancing QwQ-type long CoT models through the integration of tool utilization capabilities. This integrated approach yields improved performance metrics across mathematical and coding benchmarks.

3 Methodology

3.1 Training data

Our training data comprises two parts: one consists of math data sourced from previous AIME

problems [1](before 2024), MATH (Hendrycks et al., 2021), and Numina-MATH (LI et al., 2024), while the other includes code data from Codeforces [2], code contests [3] and LiveCodeBench(before July 2024) (Jain et al., 2024). We apply the same decontamination method as described in (Yang et al., 2024) to the training set in order to minimize potential test data leakage risks. There are a total of 40K math problems and 10K code problems, and the specific quantity distribution can be referred to in Appendix A.1.

3.2 Hint-RFT

Construct Hint we have designed a series of hints(Hint-Library) tailored to the various scenarios that may arise during LLM reasoning. Since mathematical reasoning with tools can be quite complex, we develop different hints focused on reflection, logical verification, exploring new methods, and more. These diverse hints enable the model to adopt different strategies based on the specific situation it encounters. For coding tasks, we concentrate on designing hints that promote the model's self-debugging capabilities. By encouraging the model to check its code against test cases, it can verify the correctness of its solutions and make necessary adjustments as needed. We find that adding code template to the hint can effectively prompt the model to generate desired debugging code. Figure 3 show the hints.

Hint-infer For mathematical reasoning, we strategically insert hints after specific conjunction tokens, such as *Alternatively* and *Wait* as these tokens typically indicate that the model may be questioning its own reasoning or considering alternative approaches. It is important to note that after the hints are inserted, the model continues its reasoning process. The generated code is then sent to a Python interpreter for execution, and upon obtaining the results, the model proceeds to generate further outputs based on that information. Similarly, we can insert hints before the stop token to encourage the model to engage in deeper reasoning based on its existing reasoning. By inserting hints at these critical junctures at random, we encourage the model to explore a broader reasoning space. For code reasoning, we primarily concentrate on code generation tasks. We insert hints that prompt the model to test its own code right before the model generates the final code solution. This strategic placement encourages the model to engage in self-assessment, thereby enhancing the accuracy and reliability of the generated code. A more intuitive description is in Figure 1.

Data process and model fine-tuning Inspired by (Lightman et al., 2024), we adopt an active learning method, where we perform greedy inference and hint inference using QwQ on all training data, and we recall data from reasoning tasks where QwQ would not succeed without tools, but succeeded with hint inference. This is incorporated into our startup data D_{seed} with 10K math data and 2K code data. It is important to note that, in addition to scoring the generated reasoning trajectories based on the rules, we also filter out responses that contain repetitive patterns. Additionally, we modify the Python identifiers in the code data hints to *"Debug Code Template"* and remove the output placeholders. We fine-tune QwQ based on D_{seed} to obtain START-0. The purpose of this fine-tuning step is to enable the model to learn the response paradigm for utilizing tools.

3.3 RFT

To further enhance the diversity and quantity of the training data, as illustrated in Figure 1, we utilize the obtained START-0 to perform rejection sampling fine-tuning on all training data. Specifically, we use sampling parameters of temperature 0.6 and top-p 0.95 with START-0 to perform 16 rounds of sampling. We score the sampled long TIR data, filter out responses with repetitive patterns, and manually modify any unreasonable content. We retain a maximum of one response per question, resulting in our dataset D_{START}. Using the 40,000 math data entries and 10,000 code data entries from D_{START}, we fine-tune QwQ once again, resulting in our final LRM named START.

4 Experiment

4.1 Benchmarks

In this work, we primarily focus on integrating Python tools into long CoT reasoning. Given Python's effectiveness in enhancing computational and programming aspects of reasoning, we select several representative and challenging reasoning benchmarks to validate our methodology.

[1] https://huggingface.co/datasets/gneubig/aime-1983-2024

[2] https://codeforces.com/problemset

[3] https://github.com/google-deepmind/code_contests

Table 1: Main results on challenging reasoning tasks, including PhD-level science QA, math, and code benchmarks. We report Pass@1 metric for all tasks. For models with 32B parameters, the best results are in **bold** and the second-best are underlined. Symbol "†" indicates results from their official releases.

Method	GPQA	MATH500	AMC23	AIME24	AIME25	LiveCodeBench
			General LLMs			
Qwen2.5-32B	46.4	75.8	57.5	23.3	-	22.3
Qwen2.5-Coder-32B	33.8	71.2	67.5	20.0	-	25.0
Llama3.3-70B	43.4	70.8	47.5	36.7	-	34.8
DeepSeek-V3-671B	59.1	90.2	-	39.2	-	40.5
GPT-4o†	50.6	60.3	-	9.3	-	33.4
			Reasoning LLMs			
API Only						
o1-preview†	73.3	85.5	81.8	44.6	37.5	53.6
o1-mini†	-	90.0	-	63.6	50.8	-
o1†	77.3	94.8	-	74.4	-	63.4
o3-mini(low)†	70.6	95.8	-	60.0	44.2	75.6
Open weights						
R1-Distill-Qwen-32B†	62.1	94.3	93.8	**72.6**	**46.7**	**57.2**
s1-32B†	59.6	93.0	-	50.0	33.3	-
Search-o1-32B†	**63.6**	86.4	85.0	56.7	-	33.0
QwQ	58.1	90.6	80.0	50.0	40.0	41.4
START	**63.6**(+5.5)	**94.4**(+3.8)	**95.0**(+15.0)	66.7(+16.7)	**47.8**(+7.8)	47.3(+5.9)

GPQA: This benchmark comprises 448 graduate-level multiple-choice questions authored by experts in biology, physics, and chemistry (Rein et al., 2023). These questions present significant challenges, as even domain experts achieved less than 75% accuracy in testing (OpenAI, 2024b).

Math Benchmarks: Mathematical performance of LLMs remains a focal point for researchers. In the mathematical domain, we select MATH500 (Lightman et al., 2024) at the high school level, along with competition-level AMC23 [4], AIME24 [5] and AIME25 [6] as our evaluation datasets. These datasets encompass various mathematical question types, including algebra, calculus, number theory, probability, and geometry, enabling a comprehensive assessment of LLMs' mathematical problem-solving capabilities.

LiveCodeBench: This benchmark evaluates LLMs' programming capabilities, with test cases categorized into easy, medium, and difficult levels (Jain et al., 2024). We choose 112 problems from August 2024 to November 2024 as the code benchmark. These questions are categorized as hard, medium, and easy based on difficulty.

[4] https://huggingface.co/datasets/AI-MO/aimo-validation-amc

[5] https://huggingface.co/datasets/AI-MO/aimo-validation-aime

[6] https://huggingface.co/datasets/TIGER-Lab/AIME25

4.2 Baselines

We evaluate our approach against the following baseline methods:

General LLMs: These methods are general LLMs without the long CoT reasoning. The open-source models include Qwen2.5-32B-Instruct (Yang et al., 2025), Qwen2.5-Coder-32B-Instruct (Hui et al., 2024),DeepSeek-V3-671B (DeepSeek-AI et al., 2024), Llama3.3-70B-Instruct (Dubey et al., 2024) and GPT-4o (OpenAI, 2024a).

LRMs: These methods are equipped with long CoT reasoning. **(1) API only:** These models can only be accessed through the API, including o1-series (OpenAI, 2024b) and o3-mini (OpenAI, 2025). **(2) Open weights:** we compare with some open weights LLMs, including DeepSeek-r1 series (DeepSeek-AI, 2025), QwQ (Qwen Team, 2024), s1 (Muennighoff et al., 2025b) and Search-o1 (Li et al., 2025).

4.3 Implementation Details

Responses are generated using greedy decoding with a maximum sequence length of 32,768 and a limit of 6 maximum tool uses. Checkpoints are not selected with early stops. The training pro-

cess employs full-parameter fine-tuning with DeepSpeed ZeRO-3 (Rajbhandari et al., 2020) optimization and FlashAttention2 (Dao et al., 2022). For MATH500,GPQA and LiveCodeBench, we report the greedy accuracy. For AMC23, AIME24 and AIME25, we report the average accuracy with temperature 0.5, top-p 0.95 and sample time 16. The hardware setup involves 32 NVIDIA A100 GPUs.

Table 2: Scores on GPQA in various subjects.

Model	Physics	Chemistry	Biology
QwQ	73.8	41.9	68.4
Search-o1	77.9	**47.3**	**78.9**
START	**80.0**	**47.3**	68.4

Table 3: Scores on questions of different difficulty levels on LiveCodeBench.

Model	Easy	Medium	Hard
QwQ	**92.3**	46.0	10.2
START	**92.3**	**84.6**	**12.2**

4.4 Main Results

Table 1 presents the evaluation results of START across various benchmarks, demonstrating its superior reasoning performance in scientific, mathematical, and coding domains compared to other open-source models. Overall, general LLMs, even domain-specific LLMs, are difficult to compete with LRMs in complex tasks.

PHD-level Science QA Performance It can be observed that on the ScienceQA benchmark, START demonstrates an absolute improvement of 5.5% over QwQ, achieving the same score as the state-of-the-art model, search-o1-32B. Table 3 presents the scores of QwQ, Search-o1, and START across three subjects of GPQA: Physics, Chemistry, and Biology. Specifically, START achieves the highest score in Physics, while Search-o1 outperforms QwQ significantly in Biology. This discrepancy can be attributed to the fact that Physics often necessitate extensive computational reasoning, whereas Biology primarily relies on knowledge-based reasoning. Consequently, the utilization of Python-based tools(START) yields more pronounced efficacy in the former disciplines, while the utilization of internet knowledge(search-o1-32B) works better on the latter.

MATH Benchmarks Performance On the MATH benchmarks, START also demonstrates considerable advantages over QwQ. Specifically, it achieves absolute improvements of 3.8%, 15.0%, 16.7% and 7.8% on the MATH500, AMC23, AIME24 and AIME25, respectively. The performance of START is comparable to that of R1-Distill-Qwen-32B, which is distilled from 671B DeepSeek-R1, and overall it exceeds o1-preview. These results highlight the significant role of Python-based tools in enhancing mathematical reasoning capabilities.

LiveCodeBench Performace On the LiveCodeBench, START, by equipping the model with the capability to invoke debugging tools, achieves an absolute improvement of 5.9% over QwQ. We find that START improves the most compared to QwQ on moderately difficult questions. The possible reason is that for easy questions, QwQ can generate the correct answers with high probability without debugging, and for hard questions, based on the current capabilities of the model, a limited number of debugs is also difficult to solve.

4.5 Analysis

4.5.1 Long CoT vs Long TIR

To ascertain whether our performance gains stem from the additional training questions or from the tool invocation capability, we conduct an experiment using the same set of queries from D_{START} but only apply RFT with QwQ. For each query, we sampled 32 responses with a temperature of 0.7 and a top-p value of 0.95. After filtering out incorrect responses and responses with repeating strings, we retain at most one response per question and get the long CoT dataset D_{RFT}. Based on D_{RFT}, we fine-tune QwQ, yielding QwQ-RFT. This methodological approach allows us to isolate the impact of tool invocation from that of the expanded training dataset.

The results presented in Table 4 indicate that the performance of QwQ-RFT is nearly on par with that of QwQ. Therefore, the observed performance advantage of START is likely predominantly driven by its tool invocation capability, suggesting that this feature plays a critical role in enhancing its effectiveness.

4.5.2 Analysis of Hint-infer

Compare QwQ with Hint-infer and START Through Hint-RFT, we discover that QwQ in-

Table 4: Compare long cot with long tir on challenging reasoning tasks, including PhD-level science QA, math, and code benchmarks. We report Pass@1 metric for all tasks.

Method	GPQA	MATH500	AMC23	AIME24	AIME25	LiveCodeBench
QwQ	58.1	90.6	80.0	50.0	40.0	41.4
QWQ-RFT	58.5	91.8	82.5	53.3	33.3	42.1
START (Ours)	**63.6**(+5.5)	**94.4**(+3.8)	**95.0**(+15.0)	**66.7**(+16.7)	**47.8**(+7.8)	**47.3**(+5.9)

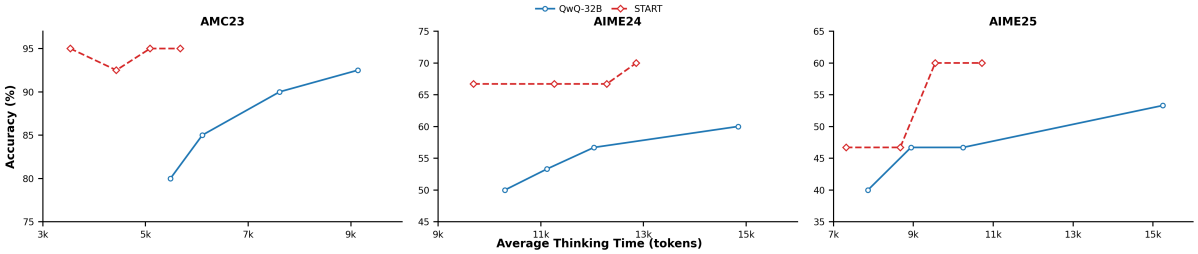

Figure 5: Test time scaling for QwQ and START on challenge math bench marks via Hint-infer.

herently possesses the potential to invoke tools, although this capability is challenging to elicit through prompting alone and instead requires explicit hints to activate. START, which is fine-tuned from QwQ using Hint-RFT, allows us to directly compare the performance of QwQ with Hint-infer against that of START. To avoid interrupting QwQ's reasoning process, we only insert hints before the stop token of QwQ(see more in Appendix A.4). This comparison provides a basis for evaluating the necessity of fine-tuning, as it helps to determine whether the enhanced performance of START is primarily due to the fine-tuning process or can be sufficiently achieved through hint-based prompting alone.

From Table 6, it is evident that incorporating hints during the inference process of QwQ leads to improvements across all benchmarks. However, these improvements are relatively modest compared to the gains achieved by START. Consequently, START, through Hint-RFT, significantly enhances QwQ's tool invocation capabilities, demonstrating the effectiveness of fine-tuning in unlocking the model's latent potential.

Test-time scaling via Hint By inserting hints at the end of QwQ's inference process, we can simultaneously increase both the model's thinking time and its accuracy(see Figure 5) by multiple rounds of inserting hint before stop token. It indicates that Hint-infer is a simple yet effective method for achieving sequential test-time scaling. Unlike the strategy outlined in (Muennighoff et al., 2025b), which merely increases the number of "wait" tokens, our method augments the number of tool invocation. The use of Hint-infer for START, on the other hand, does not work as well as on QwQ, and the reason behind this may be that those hints we inserted were already available during the reasoning process between STARTs, reducing the amount of information in the added hints. More results and analysis are list in A.2.

5 Conclusion

this paper presents START, a groundbreaking tool-integrated long Chain-of-Thought reasoning model that effectively mitigates the limitations of existing large reasoning models (LRMs) through the innovative integration of external tools and self-learning techniques. Our contributions, namely Hint-infer and Hint-RFT, showcase a novel approach to enhance reasoning capabilities by enabling LRM to leverage coding interpreters for complex computations and self-debugging. The empirical results demonstrate significant improvements in performance across a range of challenging benchmarks, establishing START as a leading open-source solution for advanced reasoning tasks. By combining long CoT with tool integration, START sets a new standard for the future development of LLMs, paving the way for more reliable and efficient reasoning in higher-level cognitive tasks.

6 Limitations

While our work on START demonstrates significant advancements in tool-integrated long Chain-of-Thought reasoning, it is essential to acknowledge several limitations inherent in our approach.

Firstly, our research exclusively focuses on the integration of a Python interpreter as the sole external tool. Although this choice was made for its relevance to many reasoning tasks, we believe that incorporating a wider variety of tools—such as search engines, specialized libraries, or different computational resources—could potentially enhance the model's performance and versatility. Future work could explore how diverse toolsets might contribute to more robust reasoning across various domains.

Secondly, the manual design of hints for insertion into the long CoT reasoning process may inadvertently disrupt the model's original flow of thought. While we aimed to strategically position these hints to optimize performance, the effectiveness of hint positioning and selection could vary based on the specific task or context. More nuanced criteria for determining the most effective types and placement of hints might result in further improvements in reasoning fluidity and accuracy.

Additionally, our empirical evaluations were conducted on a limited set of benchmarks. Although results reported demonstrate promising outcomes, the generalizability of our findings remains to be established across broader and more diverse datasets. The performance of START may be sensitive to variations in task complexity, domain specificity, and the characteristics of the input data.

Lastly, potential risks associated with the misuse of the technology must be considered. The ability of our model to generate code or suggest problem-solving strategies could be inadvertently leveraged for malicious purposes, such as crafting disinformation or automating harmful tasks. It is crucial to implement safeguards and establish ethical guidelines to monitor and mitigate such risks.

In summary, while our research provides a significant step forward, acknowledging these limitations is essential to paving the way for future improvements and ensuring the responsible development and application of tool-integrated reasoning models.

References

Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, Xiaoyi Dong, Haodong Duan, Qi Fan, Zhaoye Fei, Yang Gao, Jiaye Ge, Chenya Gu, Yuzhe Gu, Tao Gui, Aijia Guo, Qipeng Guo, Conghui He, Yingfan Hu, Ting Huang, Tao Jiang, Penglong Jiao, Zhenjiang Jin, Zhikai Lei, Jiaxing Li, Jingwen Li, Linyang Li, Shuaibin Li, Wei Li, Yining Li, Hongwei Liu, Jiangning Liu, Jiawei Hong, Kaiwen Liu, Kuikun Liu, Xiaoran Liu, Chengqi Lv, Haijun Lv, Kai Lv, Li Ma, Runyuan Ma, Zerun Ma, Wenchang Ning, Linke Ouyang, Jiantao Qiu, Yuan Qu, Fukai Shang, Yunfan Shao, Demin Song, Zifan Song, Zhihao Sui, Peng Sun, Yu Sun, Huanze Tang, Bin Wang, Guoteng Wang, Jiaqi Wang, Jiayu Wang, Rui Wang, Yudong Wang, Ziyi Wang, Xingjian Wei, Qizhen Weng, Fan Wu, Yingtong Xiong, Chao Xu, Ruiliang Xu, Hang Yan, Yirong Yan, Xiaogui Yang, Haochen Ye, Huaiyuan Ying, Jia Yu, Jing Yu, Yuhang Zang, Chuyu Zhang, Li Zhang, Pan Zhang, Peng Zhang, Ruijie Zhang, Shuo Zhang, Songyang Zhang, Wenjian Zhang, Wenwei Zhang, Xingcheng Zhang, Xinyue Zhang, Hui Zhao, Qian Zhao, Xiaomeng Zhao, Fengzhe Zhou, Zaida Zhou, Jingming Zhuo, Yicheng Zou, Xipeng Qiu, Yu Qiao, and Dahua Lin. 2024. Internlm2 technical report. *Preprint*, arXiv:2403.17297.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Trans. Mach. Learn. Res.*, 2023.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and memory-efficient exact attention with io-awareness. In *NeurIPS*.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, 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, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen,

S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. 2024. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. 2024. The Llama 3 herd of models. *CoRR*, abs/2407.21783.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2024. Tora: A tool-integrated reasoning agent for mathematical problem solving. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *NeurIPS Datasets and Benchmarks*.

Huggingface. 2025. Open r1.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. 2024. Qwen2.5-Coder technical report. *CoRR*, abs/2409.12186.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. *CoRR*, abs/2403.07974.

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

Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. 2024. Numinamath. [https://github.com/project-numina/aimo-progress-prize](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf).

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025. Search-o1: Agentic search-enhanced large reasoning models. *Preprint*, arXiv:2501.05366.

Minpeng Liao, Chengxi Li, Wei Luo, Jing Wu, and Kai Fan. 2024. MARIO: math reasoning with code interpreter output - A reproducible pipeline. In *ACL (Findings)*, pages 905–924. Association for Computational Linguistics.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*.

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

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

OpenAI. 2023. Learning to reason with llms.

OpenAI. 2024a. Hello GPT-4o.

OpenAI. 2024b. Learning to reason with LLMs.

OpenAI. 2025. Openai o3-mini.

Qwen Team. 2024. QwQ: Reflect deeply on the boundaries of the unknown.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. GPQA: A graduate-level Google-proof Q&A benchmark. *CoRR*, abs/2311.12022.

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. *CoRR*, abs/2302.04761.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Zijian Wu, Suozhi Huang, Zhejian Zhou, Huaiyuan Ying, Jiayu Wang, Dahua Lin, and Kai Chen. 2024. Internlm2.5-stepprover: Advancing automated theorem proving via expert iteration on large-scale LEAN problems. *CoRR*, abs/2410.15700.

Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. 2024. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data. *CoRR*, abs/2405.14333.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. 2024. Qwen2.5-Math technical report: Toward mathematical expert model via self-improvement. *CoRR*, abs/2409.12122.

A Appendix

A.1 Training set of START

A.2 More results about Hint-infer

Building upon the observed trends, the detailed results in Table A.2 further underscore the efficacy of the QWQ-Hint-infer method across diverse challenging reasoning tasks. Specifically, for datasets such as aime24, aime25, gpqa, amc23, MATH500, and LiveCodeBench, QWQ consistently demonstrates performance enhancements with each subsequent round of hint insertion. For instance, aime24 improves from 50.0% in Round 0 to 60.0% in Round 3, and MATH500 shows a marginal yet steady increase from 90.6% to 92.4% over the same rounds. This consistent upward trend highlights the method's ability to incrementally refine the model's reasoning capabilities through iterative hint integration.

In contrast, the START-Hint-infer approach exhibits a more varied performance across different datasets. While there are improvements in some areas, such as amc23, where the Pass@1 metric reaches 95.0% by Round 3, other datasets like gpqa and LiveCodeBench show relatively modest gains. Notably, LiveCodeBench sees an increase from 41.4% to 50.0%, which, although positive, does not match the consistency observed with QWQ. This disparity suggests that the effectiveness of Hint-infer may be contingent on the inherent characteristics of the dataset and the nature of the reasoning tasks involved.

Moreover, the plateauing of performance in some START-Hint-infer scenarios indicates potential limitations in the method's applicability when certain types of information are already encapsulated within the model's existing reasoning processes. As mentioned earlier, the redundancy of hints in the START framework could dilute the incremental benefits typically associated with additional hint layers.

Overall, the comparative analysis affirms that while Hint-infer is a robust strategy for enhancing model performance in the QWQ framework, its benefits are less pronounced in the START framework. Future work may explore adaptive hint insertion strategies tailored to specific model architectures or task types to optimize the advantages of Hint-infer across different reasoning paradigms. Additional insights and comprehensive evaluations are provided in Appendix A.2.

A.3 Prompting Methods for Data annotation

We investigated three common methods to trigger existing reasoning LLMs to generate long CoT with Python tool calls in mathematical reasoning tasks. The first method is "direct prompt," which instructs the model to directly use Python tools during reasoning. The second method, "well-designed prompt," is derived from search-o1 (Li et al., 2025) and provides detailed instructions on how to use the tools; this prompt successfully triggers the model to generate special tokens for browser calls in search-o1. The third method is "in-context prompt," which leverages examples to guide the model in generating data in the same format. We do not use general LLMs, as they typically cannot produce long CoTs. For the O1 series, we can only assess whether the summary includes Python tool invocation. As a result, we found that neither QwQ, DeepSeek-R1, nor o1-mini could successfully generate long CoTs with tool calls using the three prompt-based methods. In contrast, the hint-infer method was able to trigger the model to produce Python code with 100% success.

A.4 Hint-infer for test time scaling

The three rounds hints of GPQA and MATH for Hint-infer are: *Wait, I can use Python to check if my approach is correct and refine it, if necessary.“‘python*, *Wait, I need to utilize Python code again to meticulously check to make sure I understand the question correctly as well as reasoning correctly.“‘python* and *Wait, I can think more deeply about this problem through python tools.“‘python*. Hints of LivecodeBench is the same in Hint-Library. For code problem with starter code, the code template is

```
{startcoder}
# Test the example inputs
solution = Solution()
# Example input1
test_input1 = ...
# Example input2
test_input2 = ...
# Print output
print(solution.function_name(test_input1))
print(solution.function_name(test_input2))
# Check the output
'''
'''output
[...]
'''
```

Table 5: Sources of Dataset D

Source	Quantity
AIME problems (before 2024)	890
MATH (Hendrycks et al., 2021)	7500
Numina-MATH (LI et al., 2024)	28505
Code Data	
Codeforces	7505
Code contests	2011
LiveCodeBench (before July 2024) (Jain et al., 2024)	558
Total	49969

Table 6: Comparison of QWQ-Hint-infer and START-Hint-infer on challenging reasoning tasks, including PhD-level science QA, math, and code benchmarks. We report Pass@1 metric for all tasks.

Dataset	**QWQ**				**START**			
	Round 0	**Round 1**	**Round 2**	**Round 3**	**Round 0**	**Round 1**	**Round 2**	**Round 3**
aime24	50.0%	53.3%	56.7%	60.0%	66.7%	66.7%	66.7%	66.7%
aime25	40.0%	47.8%	47.8%	53.3%	47.8%	47.8%	60.0%	60.0%
gpqa	58.5%	58.6%	59.6%	59.6%	63.6%	61.6%	60.6%	61.6%
amc23	80.0%	85.0%	90.0%	92.5%	95.0%	92.5%	95.0%	95.0%
MATH500	90.6%	92.0%	92.0%	92.4%	94.4%	95.0%	95.6%	95.2%
LiveCodeBench	41.4%	42.0%	42.0%	42.0%	47.3%	48.2%	50.0%	50.0%

For code problem without starter code, the code template is

```
def function_name(parameters):
#Implementation\n
# Test the example inputs
solution = Solution()
# Example input1
test_input1 = ...
# Example input2
test_input2 = ...
# Print output
print(solution.function_name(test_input1))
print(solution.function_name(test_input2))
# Check the output
'''
'''output
[...]
'''
```