ThinkBooster: A Unified Framework for Seamless Test-Time Scaling of LLM Reasoning

Vladislav Smirnov[1] **Chieu Nguyen**[1] **Sergey Senichev**[8] **Minh Ngoc Ta**[1] **Ekaterina Fadeeva**[2]
Artem Vazhentsev[1] **Daria Galimzianova**[1] **Nikolai Rozanov**[1,3] **Viktor Mazanov**[8]
Jingwei Ni[2] **Tianyi Wu**[5] **Igor Kiselev**[7] **Mrinmaya Sachan**[2] **Iryna Gurevych**[1,4,6]
Preslav Nakov[1] **Timothy Baldwin**[1] **Artem Shelmanov**[1]
[1]MBZUAI [2]ETH Zürich [3]Imperial College London
[4]TU Darmstadt [5]NUS [6]INSAIT [7]Accenture [8]Independent Researcher
{vladislav.smirnov, timothy.baldwin, artem.shelmanov}@mbzuai.ac.ae

Abstract

Test-time compute (TTC) scaling has emerged as a powerful paradigm for improving large language model (LLM) reasoning by allocating additional compute during inference, e.g., via multi-sample generation and verifier-based reranking. Existing TTC scaling strategies and reasoning scorers remain fragmented, evaluated under inconsistent protocols, and are rarely analyzed through the lens of quality-cost trade-offs. We introduce **THINKBOOSTER**, a unified framework for seamless test-time scaling of LLM reasoning, which consists of (i) a modular Python library implementing state-of-the-art TTC scaling strategy and scorer families, (ii) a benchmark that jointly evaluates performance and compute efficiency, and (iii) a deployable OpenAI-compatible proxy service that enables drop-in integration of adaptive reasoning into real-world applications. We further provide a demo visual debugger for inspecting the reasoning trajectories and escalation decisions.[1,2] Empirical results on mathematical and coding tasks reveal the performance-compute trade-offs of TTC scaling strategies and scoring methods, and demonstrate that THINKBOOSTER provides practical gains in real-world tasks. The code is available online under an MIT license.[3]

1 Introduction

Chain-of-thought (CoT) prompting (Wei et al., 2022; Kojima et al., 2022) and, more recently, fine-tuning LLMs to natively produce intermediate reasoning before a final answer (Lightman et al., 2023; DeepSeek-AI, 2025) have unlocked powerful capabilities in LLMs to solve complex tasks in mathematics, programming, and even scientific research (Wei et al., 2022; Chen et al., 2021; Lu et al., 2024).

It has also been shown that increasing compute at test time, such as by generating longer reasoning chains or sampling multiple solutions and selecting the best ones, can significantly improve performance on challenging problems. This approach is known as *test-time compute (TTC) scaling* (Snell et al., 2025; Muennighoff et al., 2025). It is especially effective when a state-of-the-art LLM fails to solve a challenging problem in a single pass and cannot be further fine-tuned. Moreover, recent work shows that TTC scaling can provide a better performance to efficiency trade-off than simply increasing model size (Snell et al., 2025).

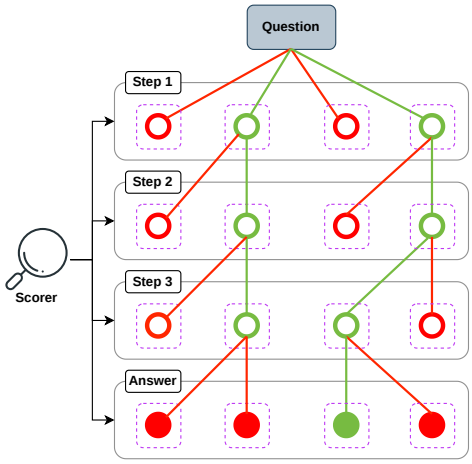


Figure 1: Illustration of the common reasoning test-time compute scaling strategy – beam search (tree of thought with breadth first search).

Common TTC scaling strategies include best-of-N (BoN: Cobbe et al. (2021); Snell et al. (2025)), tree-of-thought (ToT: Yao et al. (2023), see Figure 1), and self-consistency (also known as majority voting: Wang et al. (2023)). Recently, there have been a number of proposed dynamic TTC scaling strategies that adapt the amount of compute spent based on confidence or uncertainty estimated at the level of individual reasoning steps (Zhang et al., 2025a; Fu et al., 2025; Yan et al., 2025).

Another line of research develops step- and

[1] http://demo-thinkbooster.nlpresearch.group
[2] http://video-thinkbooster.nlpresearch.group
[3] http://thinkbooster.nlpresearch.group

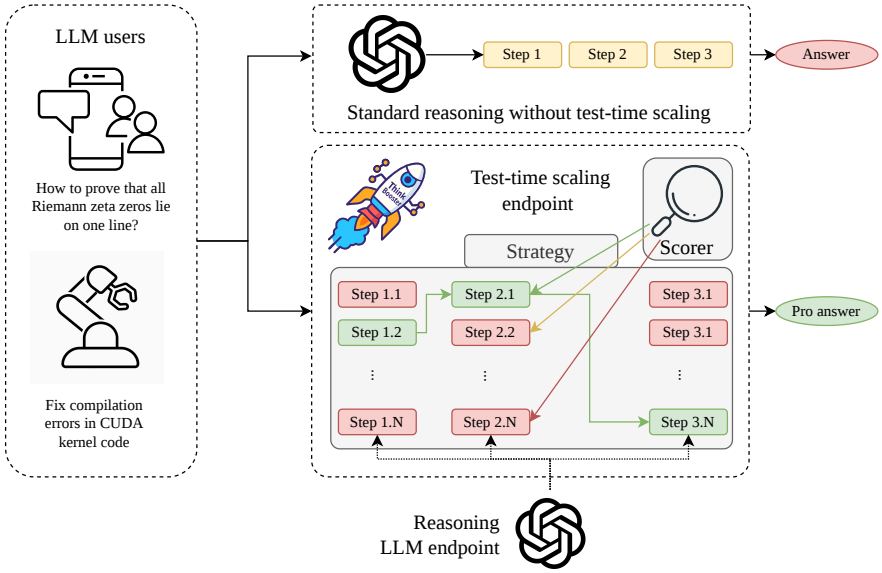


Figure 2: An illustration of the ThinkBooster endpoint gateway for test-time compute scaling.

trajectory-level scorers to select the most prominent reasoning path from multiple candidates. Common approaches include verification via process reward models (PRMs) (Uesato et al., 2022; Lightman et al., 2023; Li et al., 2023; Luo et al., 2024; Zhang et al., 2025b) and self-verification via the same LLM (Xie et al., 2023; Weng et al., 2023). Recent work has also proposed unsupervised and supervised uncertainty or confidence scores to assess the reliability of the reasoning steps (Kadavath et al., 2022; Zhu et al., 2025; Ni et al., 2025).

While TTC scaling has seen rapid advancements, the literature on this topic remains highly fragmented. Methods are typically evaluated under different experimental protocols, model configurations (e.g., structured CoT vs. native unstructured thinking), and compute budgets, which makes direct comparison difficult. Moreover, many studies focus primarily on accuracy gains while overlooking the associated computational costs, latency, and efficiency trade-offs. As a result, it is difficult to identify which methods truly offer the best performance–compute trade-off. Moreover, most published research typically presents implementations of only a single proposed method, along with a limited set of baselines. Finally, the released code often serves merely as a proof of concept, exhibiting efficiency limitations and lacking support for practical deployment. The lack of standardized, reliable, and efficient implementations creates an additional challenge for practitioners deploying TTC scaling in real-world applications.

In this work, we bridge these gaps by presenting THINKBOOSTER, a unified framework for seamless test-time scaling of LLM reasoning. THINKBOOSTER targets NLP researchers studying reasoning and test-time compute scaling as well as practitioners deploying LLM-based applications. Our goals are two-fold: (i) to provide a unified framework for principled benchmarking and research on TTC scaling and reasoning in general, and (ii) to provide a practical developer-oriented integration layer that supports easy deployment of TTC scaling in real-world applications through a unified API and clear abstractions. The framework combines a modular Python library, a benchmark, and a deployable TTC scaling endpoint gateway (see Figure 2) that provides test-time compute scaling as a service and can be used as a drop-in replacement for an OpenAI-compatible LLM. Acting as a transparent proxy to the underlying LLM, the THINKBOOSTER endpoint gateway seamlessly applies test-time compute scaling on top of the underlying LLM generations, thus enhancing the quality of the reasoning trajectories and the final answers without the need for any modifications to the existing application logic. It effectively equips the underlying LLM with a "Pro" reasoning mode. By improving the quality of the final LLM answers, THINKBOOSTER directly enhances the reliability and the effectiveness of downstream applications built on top of LLMs, including AI agents. Finally, we provide a demo visual debugger for reasoning trajectories that can be used for inspecting the intermediate steps and selection decisions.

THINKBOOSTER lowers the barrier for adopting

TTC scaling for both researchers and practitioners. The contributions of this work are as follows:

- A Python library that implements state-of-the-art TTC scaling strategies and scorers behind a unified, consistent programming API.
- A practical TTC scaling endpoint gateway with an OpenAI-compatible remote API that can be used as a drop-in replacement for the original LLM endpoint in various applications, including AI agents. We show that THINKBOOSTER can improve the downstream performance in real-world tasks, using the CUDA kernel optimization as an example.
- A benchmark for conducting research in reasoning and test-time compute scaling with both performance and compute metrics. We also conduct a pilot study of implemented strategies and scorers, and provide insights into their performance–efficiency tradeoffs.
- Finally, we create a demo visual debugger for LLM reasoning trajectories during TTC scaling, which enables interactive inspection of the intermediate reasoning steps and facilitates the systematic analysis of LLM's errors.

2 Core Library

THINKBOOSTER, at its core, is a lightweight, modular, and extendable Python library that implements the key components for TTC scaling. These include scaling strategies, scorers, reasoning generators, and reasoning step boundary detectors.

Scaling strategies include implementations of nine state-of-the-art algorithms (see Table 1), which cover more than twenty recent publications on test-time compute scaling and reasoning. Typically, the strategies in the library define the high-level scaling algorithm and do not prescribe low-level implementation details, such as how the reasoning steps or the trajectories are scored. Some strategies operate offline, meaning that they enable scoring trajectories after the final solution is obtained, while others perform scoring online as part of the reasoning process. They also differ by the level of access to LLM outputs and internal states. *White-box* strategies require access to logits or internal states, which limits their applicability in certain LLM deployment settings. *Black-box* strategies do not need anything except the generated tokens. Finally, some strategies require the *Prefill* option to be enabled in the provider's API – it allows populating the textual prefix with previously generated reasoning steps, so that the LLM does not start reasoning from scratch.

Scorers specify the way in which individual reasoning steps or entire trajectories are assessed. We implement four major approaches (see Table 2): (1) PRMs (Lightman et al., 2023), (2) uncertainty scores and confidence scores based on the LM-Polygraph library (Fadeeva et al., 2023; Vashurin et al., 2025), (3) LLM-as-a-judge assessment (including self-assessment: Yao et al. (2023)), and (4) ReProbes (Ni et al., 2025). The scorers also can be white-box or black-box.

Reasoning generators are wrappers around LLM deployments. The library supports LLMs deployed locally via the Hugging Face Transformer library and vLLM (Kwon et al., 2023). An LLM could also be deployed as a service via vLLM, OpenRouter, ChatGPT, or by any other provider with an OpenAI compatible API. Usually, the most flexible white-box access can be obtained through the Transformers and vLLM API. The majority of providers expose only generated tokens, but some give access to the logits (e.g., gpt-4o in OpenAI, some LLMs in OpenRouter, and the DeepSeek API). The prefill option is also available for vLLM deployments and some providers, such as DeepSeek and Anthropic.

Reasoning step extractors enable the decomposition of the reasoning trajectory into atomic segments and allow controlled pauses in generation for processing individual steps. For non-thinking LLMs, it is possible to specify a system prompt that will facilitate CoT in a certain format available for parsing. For large reasoning language models (LRLMs) with a native thinking mode, such as DeepSeek R1 or Qwen 3, it is not possible to control the format of the thoughts generated inside `<think>...</think>` tags, which makes reliable step extraction more challenging. We support the extraction of reasoning steps from both structured generation facilitated by the system prompt and unstructured thinking in LRLMs.

The library enables flexible combinations of scaling strategies, scoring mechanisms, LLM backends, and reasoning step extractors, allowing developers and researchers to tailor configurations to their specific needs. Previous related work has focused on non-thinking LLMs, did not allow combinations of different strategies and scorers, and primarily focused on evaluating reasoning chains (Hao et al., 2024). A key requirement for the practical adoption

Method	Key idea	Offline / Online	Level of access to LLM	Needs prefill
Best of N (Cobbe et al., 2021)	Sample N solutions and select the best one	Offline	Black-box	No
Majority voting (Wang et al., 2023)	Sample N solutions and select answer by majority vote	Offline	Black-box	No
Beam search (ToT) (Yao et al., 2023; Xie et al., 2023)	Explore tree of reasoning paths and select best	Online	Black-box	Yes
Extended thinking (Muennighoff et al., 2025)	Control reasoning budget to force longer CoT	Online	Black-box	Yes
Dynamic exploration, MUR (Zhang et al., 2025a; Yan et al., 2025)	Only allocate more compute on uncertain steps	Online	White-box	Yes
DeepConf online (Fu et al., 2025)	Steer generation toward high-confidence tokens	Online	White-box	Yes
DeepConf offline (Fu et al., 2025)	Rerank candidate solutions by model confidence scores	Offline	White-box	No
Phi-decoding (Xu et al., 2025)	Foresight sampling and adaptive pruning based on an uncertainty signal	Online	White-box	Yes
Uncertainty CoT (Zhu et al., 2025)	Generate multiple trajectories when uncertain	Online	White-box	Yes

Table 1: Test-time scaling strategies implemented in THINKBOOSTER.

Method	Key idea	Level of access to LLM
Process reward models (Lightman et al., 2023)	Train a separate critic LLM to score reasoning steps and trajectories	Black-box
Self-verification (Yao et al., 2023; Weng et al., 2023)	Ask the same LLM to evaluate the reasoning step / trajectory	Black-box
Uncertainty and confidence (Zhang et al., 2025a; Fu et al., 2025)	Confident step options are higher priority	White-box
ReProbes (Ni et al., 2025)	A supervised regressor on top of the LLM internal states	White-box

Table 2: Reasoning step and trajectory scorers implemented in THINKBOOSTER.

```python
from openai import OpenAI

client = OpenAI(
    base_url="<THINKBOOSTER_ENDPOINT>/v1/beam_search/prm",
    api_key="<YOUR_API_KEY>",
)
response = client.chat.completions.create(
    model="Qwen/Qwen3-30B-A3B",
    messages=[{"role": "user", "content":
        "Find the number of ordered pairs (x, y) of "
        "positive integers satisfying x + 2y = 2xy."}],
    extra_body={ # Optional parameters
        "model_base_url": "https://openrouter.ai/api/v1",
        "max_tokens": 8192, "tts_beam_size": 4,
    },
)
print(response.choices[0].message.content)
# reasoning path with answer
```

Figure 3: Accessing the THINKBOOSTER endpoint gateway through the OpenAI Python SDK. Parameter `base_url` specifies the THINKBOOSTER endpoint URL, which encodes scaling strategy and scorer.

of TTC scaling is the efficiency and reliability of its implementation. We address this by engineering optimized components that fully exploit provider APIs. For example, we implement custom wrappers around vLLM that expose the hidden states during inference, thus enabling efficient integration of internal-state-based scoring without requiring modifications to the underlying model.

3 Endpoint Gateway for Scaling

For developers and end-users, we built a dedicated endpoint gateway that provides test-time compute scaling as a service. It enhances model outputs without requiring any modifications to existing application logic (see Figure 2). The service operates as a proxy in front of the underlying LLM, applying scaling strategies before returning the final response. To enable TTC scaling, the user simply needs to replace the original LLM endpoint URL with the URL of the THINKBOOSTER endpoint (see Figure 3).

The THINKBOOSTER endpoint supports the configuration of test-time compute scaling parameters directly through a URL or an API, allowing users to control compute budgets, reasoning strategies, and selection policies. As the endpoint preserves the standard LLM interface, it can be integrated into downstream systems such as AI agents, code assistants, or enterprise copilots without any refactoring. Endpoint URLs are easily configured in the code or even by specifying the environment variables. This allows developers to enable a "Pro reasoning mode" for any OpenAI-compatible LLM deployment, improving reliability and quality of final answers while maintaining explicit control over computational costs.

4 Demo Visual Debugger for Reasoning

To support the analysis of TTC scaling algorithms and scorers, we provide an interactive Visual Debugger for Reasoning. The tool allows users to inspect intermediate trajectories and reasoning steps. At each reasoning step, it exposes scores (e.g. uncertainty / confidence / PRM scores), making it possible to trace escalation decisions and understand why a particular trajectory is selected.

Compared to prior work that visualizes reasoning traces (Li et al., 2025), our debugger is natively integrated with a modular TTC scaling framework that supports diverse strategies and scorers, enabling side-by-side comparison of reasoning trajectories under identical conditions (see Appendix A).

5 Experiments

5.1 Experimental Setup

We developed a benchmarking script for the systematic evaluation of scaling strategies and scoring mechanisms implemented in THINKBOOSTER, as well as custom user-defined methods across diverse datasets and LLMs. Using the developed benchmark, we conducted a pilot study on the performance efficiency trade-off of popular methods.

LLMs for reasoning. We experimented with three state-of-the-art LLMs: Qwen2.5-Math-7B-Instruct (Yang et al., 2024) without thinking mode, Qwen3-8B in native thinking mode (Yang et al., 2025), and a large GPT-OSS-120B (OpenAI et al., 2025). The selected models span distinct categories: non-thinking, small-thinking, and large-thinking LLMs, and achieve state-of-the-art performance within their respective groups.

Datasets. We select challenging datasets from three categories: *mathematics*: MATH-500 (Hendrycks et al., 2021), OlympiadBench (He et al., 2024), GaoKao23EN (Zhang et al., 2023), AIME-2024, and AIME-2025 (Mathematical Association of America, 2024); *scientific QA*: GPQA-Diamond (Rein et al., 2023); and *coding*: HumanEval+, MBPP+ (Liu et al., 2023), and KernelBench (Ouyang et al., 2025). The datasets preprocessed in the required format are available in our repository.[4] Some benchmarks appear to be saturated for particular LLMs. Therefore, we report results for a given model-dataset pair only when the dataset remains sufficiently challenging for that model. The model-dataset mapping, dataset statistics, and further details are in Section B.

Performance metrics and result parsing procedures are dataset-specific. Their implementations are taken from the corresponding papers for consistency. For mathematical and scientific datasets, usually, we can determine whether the LLM response matches the gold answer exactly (exact match = EM). Since parsing can be imperfect, the benchmark can provide LLM-as-a-judge scores, which compare model responses against the gold answers.

For HumanEval+ and MBPP+, we report the pass@1. Evaluation is performed using the *EvalPlus* package, which runs each generated solution against a set of tests. A solution is considered correct only if it passes all tests in EvalPlus.

For KernelBench, we adopt three measures: (i) Syntax Check, which verifies that the generated code is syntactically valid, (ii) Compilation Check, which ensures that the code compiles, and (iii) Correctness Check, which evaluates whether the custom operand produces outputs consistent with the corresponding PyTorch implementation within a specified floating-point tolerance. Each measure is a fraction of successful cases.

Efficiency metrics in the THINKBOOSTER benchmark include: (1) inference cost in TFLOPs (Kaplan et al., 2020) and (2) the number of generated tokens. The amount of TFLOPs is estimated according to Hoffmann et al. (2022). We account for the fact that processing prompt tokens can leverage the KV cache, in contrast to generating new tokens in multiple different samples. We also include the TFLOPs associated with running PRMs. The computational overhead of information-theoretic uncertainty scorers (e.g., perplexity, sequence probability, and entropy) is considered negligible. A detailed description of the efficiency metric computation is provided in Appendix B.

Note that FLOP estimates reflect *theoretical* throughput and do not account for runtime optimizations such as vLLM prefix caching, which can substantially reduce wall-clock time by reusing KV-cache across prompts that share a common prefix. We report theoretical FLOPs to ensure reproducibility and hardware-independent comparisons.

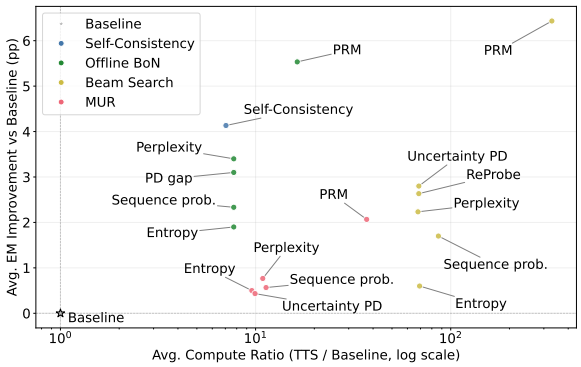
Details on scorer and strategy configurations (aggregation functions and scoring windows) are provided in Appendix B and C.

5.2 Results

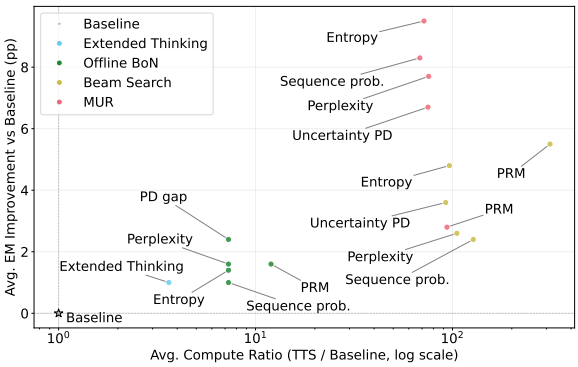
Trade-off between performance and compute. The results for non-thinking Qwen2.5-Math-7B are presented in Figure 4a, for Qwen3-8B on HumanEval-Plus in Figure 4b, and additional results on AIME are in Figure 7 in Appendix C.

On mathematical benchmarks, the strongest results are obtained with PRM-based scorers. In contrast, on coding tasks (Figure 4b), PRMs do not consistently outperform lightweight uncertainty-based scorers. In particular, the combination of the uncertainty-based scorer with the MUR strategy substantially surpasses all baselines on HumanEval-Plus. A plausible explanation is that the PRM used in our experiments is predominantly trained on mathematical data and thus overfits to that domain. In contrast, uncertainty-based scorers are domain-agnostic and generalize better to code generation, which constitutes an out-of-distribution setting for

[4] https://huggingface.co/test-time-compute



(a) Qwen2.5-Math-7B (aggregate over MATH-500, OlympiadBench, Gaokao 2023 EN)



(b) Qwen3-8B (HumanEval-Plus)

Figure 4: Accuracy improvement vs. compute ratio for different TTC scaling methods. Each point represents a strategy–scorer combination; the x-axis shows the compute ratio relative to the baseline (log scale).

Strategies	GPQA-Diamond	HumanEval+	MBPP+	KernelBench		
	EM	pass@1	pass@1	Syntax	Compilation	Correctness
Raw CoT	70.3	83.5	76.0	82.0	**65.0**	26.0
Offline BoN	72.2	85.4	**78.8**	**87.0**	64.0	**30.0**
Beam Search	**73.2**	**87.8**	78.4	–	–	–

Table 3: Evaluation results for `gpt-oss-120b` using a PRM-based scorer across QA (GPQA-Diamond), coding (HumanEval+, MBPP+), and CUDA kernel generation (KernelBench). The best results are highlighted in **bold**.

the PRM. Overall, uncertainty emerges as a robust and competitive scoring signal across tasks. Given their simplicity, near-zero computational overhead, and domain independence, uncertainty-based scorers offer a practical alternative in real-world deployments. At the same time, our findings highlight the need for coding-specific PRMs, which remain largely underexplored.

Across strategies, beam search often underperforms compared to BoN and even the self-consistency baseline, despite requiring substantially more compute. Nevertheless, when paired with PRM-based scoring, it can achieve the highest absolute performance on mathematical benchmarks. Dynamic compute scaling (MUR) offers a more efficient alternative, with the potential to reduce computational costs relative to beam search. It delivers the strongest results on HumanEval+, but on mathematical datasets, even when combined with PRMs, it still lags behind BoN.

Real world application: optimization of CUDA kernels and programming. Table 3 presents the results for coding tasks with GPT-OSS-120B. CUDA kernels generated with Offline BoN guided by a PRM have 5% fewer syntax errors. The compilation rate is slightly lower as PRM tends to select more sophisticated code produced by the LLM, which can be more effective but also more prone to compilation failures. Importantly, the overall correctness is 4% higher, i.e., THINKBOOSTER improves end-to-end CUDA kernel quality.

For the coding tasks, we observe that the Offline BoN strategy improves the baseline LLM performance by 1.9% on HumanEval+ and by 2.8% on MBPP+ across seeds. Moreover, Beam Search further improves performance by 4.3% on HumanEval+. Overall, these results show that THINKBOOSTER can help in real-world tasks.

6 Conclusion

We introduce THINKBOOSTER, a unified framework for test-time scaling of LLM reasoning that bridges research and deployment. By combining a modular library of TTC scaling strategies and scorers, a joint performance–compute benchmark, and an OpenAI-compatible endpoint that provides compute scaling as a service, THINKBOOSTER enables principled comparison and seamless integration of adaptive reasoning into real-world systems. Our results show that uncertainty can be used for both scoring and dynamic scaling of compute, achieving in some cases better results than PRMs and static methods. We hope THINKBOOSTER will facilitate more systematic, compute-aware research and practical adoption of test-time compute scaling.

Limitations

THINKBOOSTER provides a unified implementation and benchmark for test-time compute scaling, but several components depend on deployment-specific capabilities. In particular, some strategies and scorers require white-box signals (e.g., logits/hidden states) or API features such as prefill-style continuation, which are not available for many black-box hosted models. Moreover, reliable step boundary extraction remains challenging for models with native, unstructured "thinking" traces, potentially affecting online scoring and escalation. Our empirical study currently covers a limited set of models and datasets (primarily math and coding), so the observed quality–cost trade-offs may not generalize to other domains (e.g., long-context QA or tool-using agents). Finally, proxy-based TTC scaling can introduce variable latency and provider-dependent cost behavior, and judge-based evaluation may add noise in correctness estimates.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Ekaterina Fadeeva, Roman Vashurin, Akim Tsvigun, Artem Vazhentsev, Sergey Petrakov, Kirill Fedyanin, Daniil Vasilev, Elizaveta Goncharova, Alexander Panchenko, Maxim Panov, and 1 others. 2023. Lm-polygraph: Uncertainty estimation for language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 446–461.

Yichao Fu, Xuewei Wang, Yuandong Tian, and Jiawei Zhao. 2025. Deep think with confidence. *Preprint*, arXiv:2508.15260.

Shibo Hao, Yi Gu, Haotian Luo, Tianyang Liu, Xiyan Shao, Xinyuan Wang, Shuhua Xie, Haodi Ma, Adithya Samavedhi, Qiyue Gao, and 1 others. 2024. Llm reasoners: New evaluation, library, and analysis of step-by-step reasoning with large language models. *arXiv preprint arXiv:2404.05221*.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, and 1 others. 2024. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3828–3850.

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 *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, and 1 others. 2022. Training compute-optimal large language models. *Advances in Neural Information Processing Systems*, 35:30016–30030.

Saurav Kadavath, Eric Tang, Toby Shevlane, Cameron McKinnon, and et al. 2022. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

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*.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2023. Making language models better reasoners with step-aware verifier. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5315–5333, Toronto, Canada. Association for Computational Linguistics.

Zongqian Li, Ehsan Shareghi, and Nigel Collier. 2025. ReasonGraph: Visualization of reasoning methods and extended inference paths. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 140–147, Vienna, Austria. 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. 2023. Let's verify step by step. In *The twelfth international conference on learning representations*.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in neural information processing systems*, 36:21558–21572.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. 2024. The ai scientist: Toward fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, and 1 others. 2024. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*.

Mathematical Association of America. 2024. American Invitational Mathematics Examination (AIME). `https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions`. Administered by the Mathematical Association of America (MAA).

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. 2025. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20286–20332.

Jingwei Ni, Ekaterina Fadeeva, Tianyi Wu, Mubashara Akhtar, Jiaheng Zhang, Elliott Ash, Markus Leippold, Timothy Baldwin, See-Kiong Ng, Artem Shelmanov, and 1 others. 2025. Efficient test-time scaling of multi-step reasoning by probing internal states of large language models. *arXiv preprint arXiv:2511.06209*.

OpenAI, :, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, and 108 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

Anne Ouyang, Simon Guo, Simran Arora, Alex L Zhang, William Hu, Christopher Re, and Azalia Mirhoseini. 2025. Kernelbench: Can LLMs write efficient GPU kernels? In *Forty-second International Conference on Machine Learning*.

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.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, volume 2025, pages 10131–10165.

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

Roman Vashurin, Ekaterina Fadeeva, Artem Vazhentsev, Lyudmila Rvanova, Daniil Vasilev, Akim Tsvigun, Sergey Petrakov, Rui Xing, Abdelrahman Sadallah, Kirill Grishchenkov, Alexander Panchenko, Timothy Baldwin, Preslav Nakov, Maxim Panov, and Artem Shelmanov. 2025. Benchmarking uncertainty quantification methods for large language models with LM-polygraph. *Transactions of the Association for Computational Linguistics*, 13:220–248.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. 2023. Large language models are better reasoners with self-verification. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2550–2575, Singapore. Association for Computational Linguistics.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, James Xu Zhao, Min-Yen Kan, Junxian He, and Michael Xie. 2023. Self-evaluation guided beam search for reasoning. In *Advances in Neural Information Processing Systems*, volume 36, pages 41618–41650. Curran Associates, Inc.

Fangzhi Xu, Hang Yan, Chang Ma, Haiteng Zhao, Jun Liu, Qika Lin, and Zhiyong Wu. 2025. ϕ-decoding: Adaptive foresight sampling for balanced inference-time exploration and exploitation. *Preprint*, arXiv:2503.13288.

Hang Yan, Fangzhi Xu, Rongman Xu, Yifei Li, Jian Zhang, Haoran Luo, Xiaobao Wu, Luu Anh Tuan, Haiteng Zhao, Qika Lin, and Jun Liu. 2025. Mur: Momentum uncertainty guided reasoning for large language models. *Preprint*, arXiv:2507.14958.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, and 1 others. 2024. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822. Curran Associates, Inc.

Jinghan Zhang, Xiting Wang, Fengran Mo, Yeyang Zhou, Wanfu Gao, and Kunpeng Liu. 2025a. Entropy-based exploration conduction for multi-step reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 3895–3906, Vienna, Austria. Association for Computational Linguistics.

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

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025b. The lessons of developing process reward models in mathematical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 10495–10516, Vienna, Austria. Association for Computational Linguistics.

Yuqi Zhu, Ge Li, Xue Jiang, Jia Li, Hong Mei, Zhi Jin, and Yihong Dong. 2025. Uncertainty-guided chain-of-thought for code generation with llms. *arXiv preprint arXiv:2503.15341*.

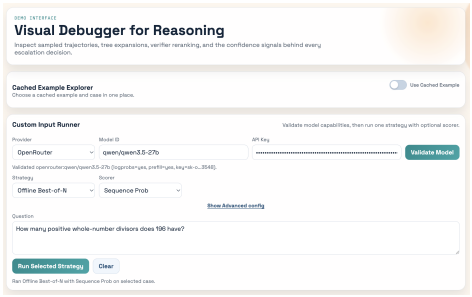
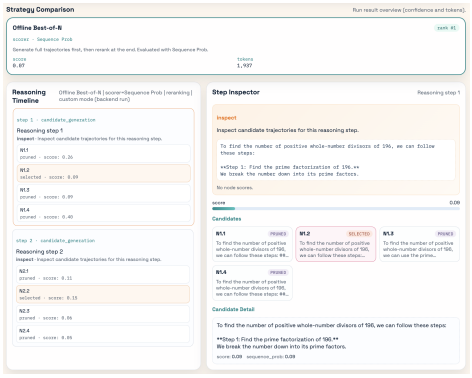
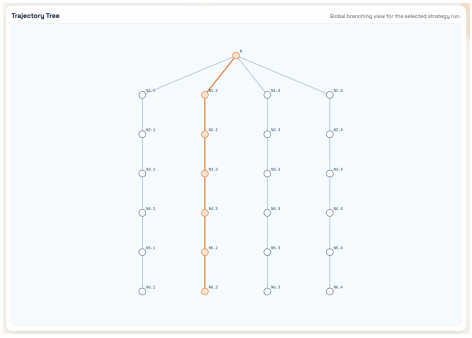


Figure 5: Visual Debugger Demo. In this example, the user runs an Offline Best-of-N strategy with a Sequence Probability scorer on a mathematical question.



(a) Reasoning Trace Inspector



(b) Tree View.

Figure 6: Visual Debugger Demo – Run Results.

Appendix

A Visual Debugger UI

We present a web-based Visual Debugger for inspecting test-time compute scaling strategies, sampled trajectories, and scoring signals. It reveals the search process, including candidate generation, pruning, reranking, and final selection.

Figure 5 shows the configuration interface for selecting models and strategies. After execution, the interface summarizes performance and token usage, and displays step-level candidates and scores for inspection (Figure 6a). Figure 6b visualizes the full trajectory tree, highlighting the selected path. Debugger enables compact, systematic analysis of TTC scaling beyond single-pass outputs.

B Details on the Experimental Setup

Model–dataset mapping. Qwen2.5-Math-7B is evaluated on three mathematical datasets (MATH-500, OlympiadBench, Gaokao 2023 EN). Qwen3-8B is evaluated on mathematical (AIME 2024, AIME 2025) and coding (HumanEval+) tasks. GPT-OSS-120B is evaluated on scientific QA (GPQA-Diamond) and coding tasks (HumanEval+, MBPP+, KernelBench). All experiments use vLLM (Kwon et al., 2023) as the inference backend. Results are averaged over 3 seeds with the standard deviation reported where applicable.

Hyperparameters. Generation settings vary by model. **Qwen2.5-Math-7B**: temperature 0.7, top-p 0.8, top-k 20, max tokens 4096. **Qwen3-8B**: temperature 0.6, top-p 0.95, top-k 20, max tokens 32768, with native thinking mode enabled. **GPT-OSS-120B**: temperature 0.6, top-p 0.95, top-k 20, max tokens 65536.

For beam search, we use beam size 5 with 8 candidates per beam, and 30 max steps for Qwen2.5; and beam size 3 with 5 candidates per beam, and 250 max steps for Qwen3. Offline BoN and self-consistency use $N{=}8$ samples. MUR uses 8 candidates per step, momentum 0.9, and up to 50 steps for Qwen2.5 or 250 for Qwen3. Extended thinking for Qwen3 allows up to 3 continuations.

Datasets. Table 4 reports the number of test samples for each dataset used in our experiments. MATH-500 is a representative subset of the MATH benchmark (Hendrycks et al., 2021) used for symbolic or numeric answer checking. OlympiadBench (He et al., 2024) is an olympiad-level math and physics benchmark. GaoKao23EN is an English math QA compilation centered on 2023 exam-style items (Zhang et al., 2023). AIME-2024 and AIME-2025 (Mathematical Association of America, 2024) are problems from the American Invitational Mathematics Examination.

For *scientific QA*, we use GPQA-Diamond (Rein et al., 2023), a subset of GPQA with graduate-level MCQs in biology, chemistry, and physics that require expert-level reasoning.

For *coding* datasets, we use HumanEval+ and MBPP+ (Liu et al., 2023), which are extensions of the original HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) bench-

Task	Dataset	# test samples
Math	MATH-500	500
	OlympiadBench	675
	GaoKao23EN	385
	AIME-2024	30
	AIME-2025	30
Scientific QA	GPQA-Diamond	198
Coding	HumanEval+	164
	MBPP+	378
	KernelBench	100

Table 4: Dataset statistics for the benchmarks used in our experiments.

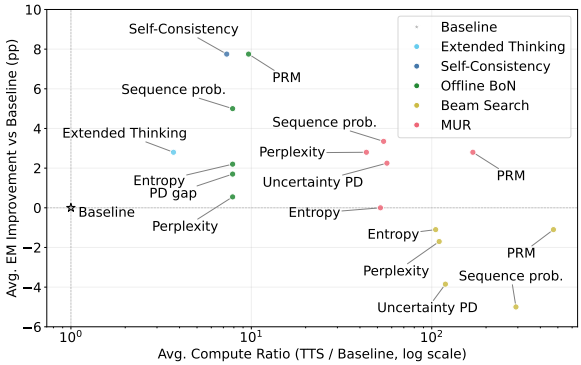


Figure 7: Accuracy improvement vs. compute ratio for Qwen3-8B on AIME (aggregate of AIME 2024 and AIME 2025). Each point represents a strategy–scorer combination; the x-axis shows the compute ratio relative to the baseline (log scale).

marks. MBPP+ consists of entry-level Python tasks, while HumanEval+ contains moderately challenging function-level Python problems. KernelBench (Ouyang et al., 2025) poses the real-world problem of writing GPU kernels: producing CUDA kernel code that outperforms PyTorch's native implementations. We focus on foundational Level-1 operators.

TFLOPs metric. For each forward pass, we compute FLOPs $= 2 \times N \times T$, where N is the number of model parameters and T is the total number of tokens (input context plus generated output). This approximation counts one multiply-add per weight per token, and is the standard metric used in scaling-law analysis (Hoffmann et al., 2022). We track token counts separately for the reasoning generator and, when applicable, for PRMs. The generator tracker records the context tokens once per prompt, even when the LLM generates multiple candidates from a shared prefix, and output tokens summed over all candidates. The PRM tracker records the number of input tokens for each scoring call; because PRM is a reward model that only performs a forward pass without generation, its cost is $2 \times N_{\text{PRM}} \times T_{\text{input}}$. Finally, $\text{TFLOPs}_{\text{total}} = \text{TFLOPs}_{\text{gen}} + \text{TFLOPs}_{\text{PRM}}$. Overhead for computing uncertainty-based scorers is considered negligible.

Reasoning step extraction. All models are prompted with a simple instruction, for example, "Please reason step by step and put your final answer within \boxed{}", without additional formatting constraints. Step boundaries are detected post hoc using a `ThinkingMarkerDetector` that identifies linguistic cues across several categories, such as sequence indicators, conclusion signals, verification phrases, self-corrections, and structural patterns. Splits occur only at sentence boundaries to avoid mid-sentence breaks. The resulting segments are then normalized by merging short fragments and splitting overly long ones. For thinking-mode models (Qwen3-8B), the same detector operates on the content inside `<think></think>` tags.

C Additional Experimental Results

Scorer details. As the PRM scorer, we use Qwen2.5-Math-PRM-7B (Yang et al., 2024), a process reward model trained on mathematical reasoning data and deployed on a separate GPU. Since no coding PRM is released, we use the same math-trained PRM as a proxy for coding tasks. For uncertainty-based scorers, we use entropy, perplexity, sequence probability, and probability differential, all computed from token-level log-probabilities produced during generation with negligible overhead. We also evaluate ReProbe (Ni et al., 2025), a lightweight linear probe (<10M parameters) trained on internal model representations.

Scorer configurations. For each in Figures 4 and 7, we select the best step-level aggregation and scoring window via grid search.

For **offline BoN**, PRM uses product aggregation over all steps (w=all), while other scorers use a w=5 window with their best aggregation.

For **beam search**, we report mean and min aggregation with w=5, and include ReProbe (Ni et al., 2025) with mean aggregation over all steps. ReProbe is a lightweight linear probe head with negligible overhead.

MUR uses default scorer settings. Compute, measured in TFLOPs, includes both generation and scoring costs.