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

Vladislav Smirnov[1] **Chieu Nguyen**[1] **Sergey Senichev**[7] **Minh Ngoc Ta**[1]
Ekaterina Fadeeva[2] **Artem Vazhentsev**[1] **Daria Galimzianova**[1] **Nikolai Rozanov**[1,3]
Viktor Mazanov[6] **Jingwei Ni**[2] **Tianyi Wu**[4] **Igor Kiselev**[5] **Mrinmaya Sachan**[2]
Iryna Gurevych[1] **Preslav Nakov**[1] **Timothy Baldwin**[1] **Artem Shelmanov**[1]
[1]MBZUAI [2]ETH Zürich [3]Imperial College London [4]NUS
[5]Accenture [6]Innopolis University [7]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 compute 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 computational 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, intermediate selection decisions, and alternative reasoning paths.[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 Large Language Models (LLMs) to natively produce intermediate reasoning before a final answer (Lightman et al., 2024; Guo et al., 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., 2026).

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

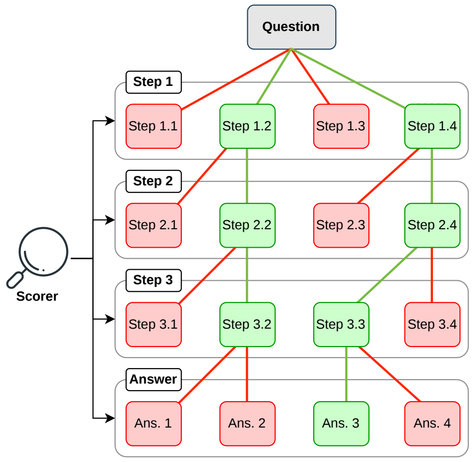


Figure 1: Illustration of the common reasoning test-time compute scaling strategy – beam search (tree of thought with breadth first search).

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).

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 compute spent based on confidence or uncertainty estimated at the level of individual reasoning steps (Zhang et al., 2025a; Fu et al., 2026; Yan et al., 2025).

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), pages 715–727
July 2-7, 2026 ©2026 Association for Computational Linguistics

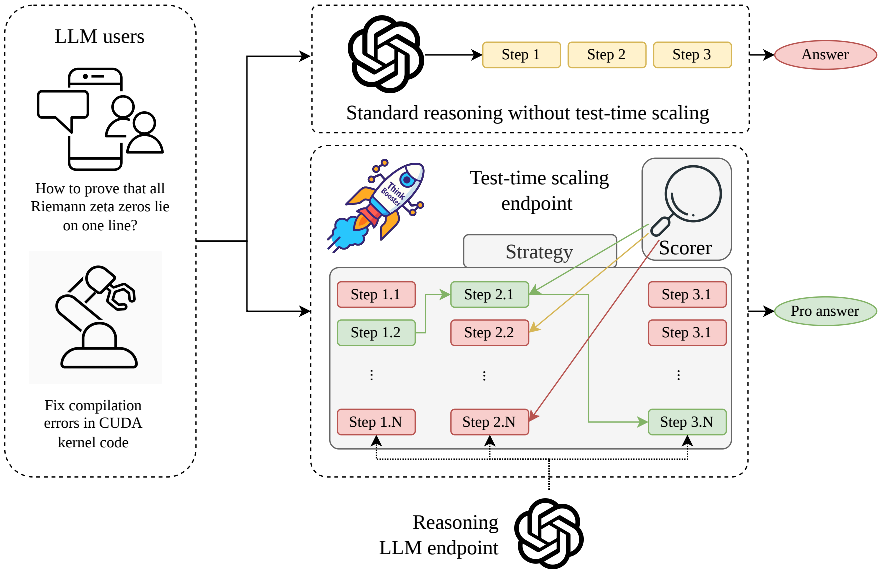


Figure 2: An illustration of the THINKBOOSTER endpoint gateway for test-time compute scaling.

Another line of research develops step- and trajectory-level scorers to select the most promising reasoning path from multiple candidates. Common approaches include verification via process reward models (PRMs) (Uesato et al., 2022; Lightman et al., 2024; 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., 2026).

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 compute 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 TTC scaling as a service and can be used as a drop-in replacement for an OpenAI-compatible LLM endpoint. 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 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 to inspect the intermediate steps, selection decisions, and alternative paths.

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., 2026)	Steer generation toward high-confidence tokens	Online	White-box	Yes
DeepConf offline (Fu et al., 2026)	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 compute scaling strategies implemented in THINKBOOSTER.

Method	Key idea	Level of access to LLM
Process reward models (Lightman et al., 2024)	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., 2026)	Confident step options are higher priority	White-box
ReProbes (Ni et al., 2026; Shelmanov 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.

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 and consistent programming API.
- A practical TTC scaling endpoint 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 and assistants. We show that THINKBOOSTER can improve downstream performance in real-world tasks, using 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 define the high-level scaling algorithm for test-time compute scaling, but typically do not prescribe low-level details, such as how exactly the reasoning steps or trajectories are scored. Our library includes implementations of nine state-of-the-art algorithms (see Table 1), which cover more than twenty recent publications on TTC scaling and reasoning. 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 their 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 for the generated tokens. Finally, some strategies require the *Prefill* option to be enabled in the provider's API – it allows to populate 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., 2024), (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., 2026). The scorers can differ in their required level of access to the internal states or outputs of the underlying LLM: white-box vs. 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 token log-probabilities (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, one can specify a system prompt to 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 inside <think>...</think> tags, which complicates reliable step extraction. 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, tailored to specific needs. A key requirement for the practical adoption 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 TTC Scaling

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

```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.

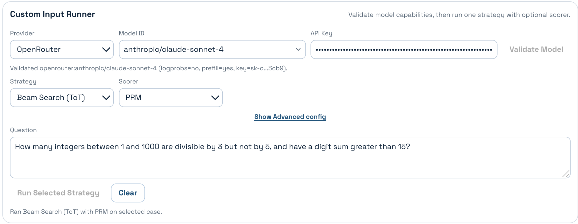
The THINKBOOSTER endpoint supports the configuration of TTC scaling parameters directly through a URL or an API, allowing users to control compute budgets, reasoning strategies, and scorers. As the endpoint preserves the standard LLM interface, it can be integrated into downstream systems such as AI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

4 Demo Visual Debugger for Reasoning

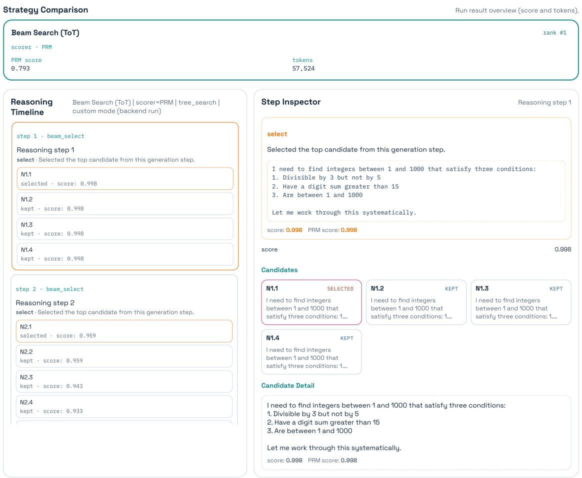
To support the analysis of TTC scaling algorithms and scorers, we provide an interactive Visual Debugger for Reasoning. The tool allows users to inspect intermediate trajectories and reasoning steps. At each reasoning step, it exposes scores (e.g. uncertainty / confidence / PRM scores), making it possible to trace decisions, alternative reasoning paths, and understand why a particular trajectory is selected (see Figure 4).

5 Related Work

Comparison with other open-source TTC scaling frameworks is presented in Table 3. OptiLLM (Sharma, 2024) is the closest in scope – an OpenAI-compatible proxy that implements several TTC scaling techniques. However, it lacks PRM scorers, FLOPs-level compute accounting, and a visual reasoning debugger.

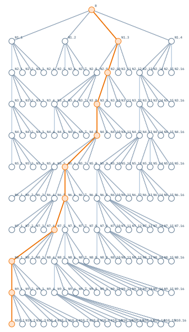


(a) Customization of provider, model, strategy, and scorer.



(b) Reasoning timeline and step inspector with per-candidate score breakdowns.

Figure 4: Visual debugger for reasoning.



(c) Trajectory tree: orange path = selected run, siblings = pruned candidates.

LLM Reasoners (Hao et al., 2024) provides a modular library with implementations of search algorithms, reward functions, methods for reasoning correction, and a reasoning tree visualizer. However, it has no OpenAI-compatible endpoint, no native vLLM backend, and no joint performance-compute benchmark. OpenR (Wang et al., 2024) implements two TTC scaling strategies: beam search and best-of-N with PRM scorers and is coupled with the FastChat inference framework (vLLM analog). search-and-learn (Beeching et al., 2024) is a limited set of TTC scaling methods for reproducing the experiments from (Snell et al., 2025) on the MATH-500 dataset. TreeQuest (Inoue et al., 2025) implements adaptive branching MCTS over user-defined state generators with visualization utilities. ReasonGraph (Li et al., 2025) provides basic implementations of TTC scaling techniques, but its primary focus is on visualizing reasoning paths. We deliberately omit self-correction and evolutionary search methods (Romera-Paredes et al., 2024; Kumar et al., 2025; Liu et al., 2025; Madaan et al., 2023), as they present an orthogonal line of work, where the TTC scaling considered in this work could be plugged-in for better performance.

ThinkBooster stands out as a uniquely feature-complete framework. It provides a comprehensive suite of TTC scaling strategies and scorers, seamless compatibility with efficient LLM-serving frameworks like vLLM, and a benchmark comprising nine bundled math, coding, and scientific datasets with joint TFLOPs-and-tokens compute accounting. Additionally, the framework equips practitioners with an OpenAI-compatible API endpoint and an interactive visual reasoning debugger.

6 Experiments

6.1 Experimental Setup

We introduce a benchmarking tool designed for the systematic evaluation of ThinkBooster's built-in scaling strategies and scorers, as well as custom user-defined methods, across a diverse range of datasets and LLMs. Using this tool, 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 (Agarwal et al., 2025). The selected models span distinct categories: non-thinking, small-thinking, and large-thinking LLMs, achieving 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 (Art of Problem Solving, 2026); *scientific QA*: GPQA-Diamond (Rein et al., 2024); and *coding*: HumanEval+, MBPP+ (Liu et al., 2023), and KernelBench (Ouyang et al., 2025).

Feature	ThinkBooster	OptiLLM	LLM Reasoners	OpenR	S&L	TreeQuest	ReasonGraph
A. Strategy taxonomy breadth	✓ (9)	○ (7)	○ (5)	○ (5)	× (3)	× (1)	× (0)
B. Scorer family breadth	✓ (4)	○ (2)	○ (2)	× (1)	× (1)	○ (1)	× (0)
C. Supports TTC methods up to year	**2026**	2026	2024	2024	2024	2026	2023
D. Supports uncertainty-based scorers	✓	×	×	×	×	×	×
E. Joint perf–compute benchmarks (TFLOPs + tokens)	✓	○	○	○	○	×	×
F1. Bundled math benchmarks (count)	**5**	0	0	2	1	0	0
F2. Bundled coding benchmarks (count)	**3**	0	0	0	0	0	0
G. Backends supported	vLLM+HF+API	API+HF+MLX	HF+SGLang+API	vLLM+HF	vLLM	agnostic	API
H. OpenAI-compatible REST gateway	✓	✓	×	×	×	×	×
I. Visual debugger	✓	×	○	○	×	×	✓
J. Modular architecture	✓	○	○	○	×	○	○

Table 3: Comparison of ThinkBooster with other TTC scaling libraries and reasoning-analysis frameworks: OptiLLM (Sharma, 2024), LLM Reasoners (Hao et al., 2024), OpenR (Wang et al., 2024), search-and-learn (S&L) (Beeching et al., 2024), TreeQuest (Inoue et al., 2025), and ReasonGraph (Li et al., 2025). Symbols: ✓ supported, ○ partial / limited, × not supported. "API" denotes any OpenAI-compatible HTTP endpoint (OpenAI, Anthropic, Gemini, OpenRouter, etc.). Strategy counts include genuine search/scoring/decoding-intervention methods only (prompt-engineering scaffolds excluded). Bundled benchmark counts include only datasets with prewired prompts, answer extraction, and judging.

We provide the appropriately formatted datasets via our repository.[4] Because certain benchmarks have reached saturation for specific LLMs, we report results only for model-dataset pairs where the task remains sufficiently challenging. Detailed dataset statistics and exact model-dataset mappings can be found in Section B.

Performance metrics and the result parsing procedures are dataset-specific. We take their implementations from the corresponding papers for consistency and more direct comparison. For mathematical and scientific datasets, usually, we can determine whether the LLM response matches the gold answer exactly (exact match = EM). The benchmark tool can also use LLM-as-a-judge to compare the model responses to the gold answers for cases where parsing is imperfect.

For the coding benchmarks HumanEval+ and MBPP+, we report pass@1 accuracy. Solutions are evaluated using the *EvalPlus* package; a generated response is classified as correct strictly when it successfully passes the original base tests alongside the expanded test cases introduced by EvalPlus.

For KernelBench, we employ three metrics: (i) Syntax Check, which verifies that the generated code is syntactically valid, (ii) Compilation Check, which ensures that code compiles successfully, and (iii) Correctness Check, which evaluates whether the generated kernel produces outputs consistent with the native PyTorch implementation within a specified floating-point tolerance. All metrics are reported as the proportions of successful cases.

Efficiency metrics used: (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 calculate the TFLOPs for prompt processing only once per input, reflecting KV cache reuse, while tracking generation costs across all individual samples. The computational overhead of running PRMs is also included in the total. The computational overhead of information-theoretic uncertainty scorers (e.g., perplexity, sequence probability, and entropy) is considered negligible. A detailed description of metric computation is provided in Appendix B.

Note that our TFLOPs estimates reflect *theoretical* compute costs and do not account for systems-level optimizations, such as vLLM prefix caching, which can substantially reduce wall-clock latency by reusing the KV cache across generations that share a common prefix. We report theoretical TFLOPs to ensure reproducibility and hardware-independent comparisons.

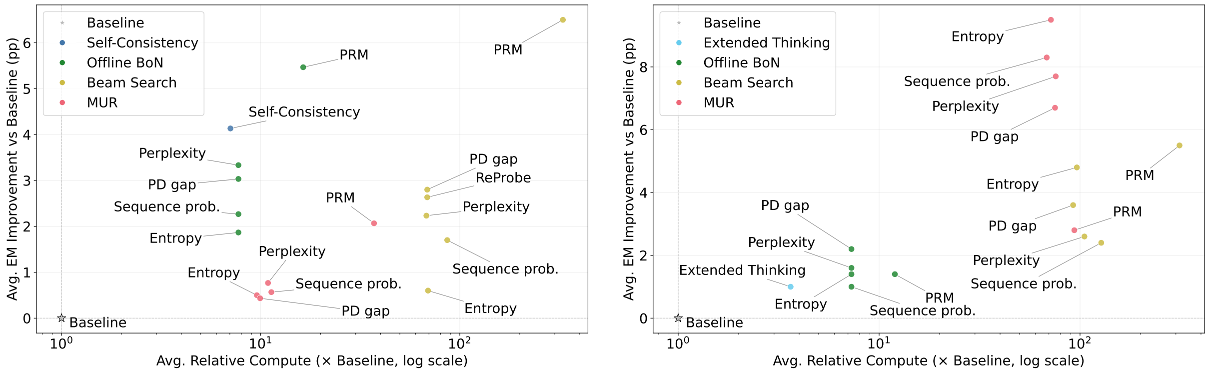
Details on scorer and strategy configurations (aggregation functions and scoring windows) are provided in Appendix B and C.

6.2 Results

Trade-off between performance and compute. The results for non-thinking Qwen2.5-Math-7B are presented in Figure 5a, results for Qwen3-8B with native thinking on HumanEval-Plus in Figure 5b, and additional results on AIME can be found in Figure 6 in Appendix C.

On mathematical benchmarks, the best results are obtained using PRM-based scorers. In contrast, on coding tasks (Figure 5b), PRMs do not consistently outperform lightweight uncertainty-based scorers. In particular, the combination of the uncertainty-based scorer with the MUR strategy

[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 5: 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 4: 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**.

substantially surpasses all baselines on HumanEval-Plus. A plausible explanation is that the PRM in our experiments is trained predominantly on mathematical data and thus could overfit to it. In contrast, uncertainty-based scorers are domain-agnostic and generalize more effectively to code generation, which constitutes an out-of-distribution setting for the PRM.

Overall, uncertainty emerges as a robust and competitive scoring signal. Given their simplicity, near-zero computational overhead, and domain independence, uncertainty-based scorers offer a practical alternative in real-world deployments. However, our findings also underscore a clear need for coding-specific PRMs, a research area that remains largely underexplored.

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

Real world application: optimization of CUDA kernels and programming. Table 4 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 4pp higher, i.e., THINKBOOSTER improves end-to-end CUDA kernel quality.

7 Conclusion

We introduced THINKBOOSTER, a unified framework for TTC 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 the practical adoption of test-time compute scaling.

Limitations

Several components of our proposed THINKBOOSTER framework depend on deployment-specific capabilities. Some strategies and scorers require *white-box* signals such as logits or hidden states, or API features such as prefill-style continuation, which are not available for fully hosted, black-box commercial models. THINKBOOSTER offers its full range of dynamic, uncertainty-driven strategies only against open-weight or self-hosted LLM; for closed APIs, it supports a black-box subset (best-of-N, majority voting, extended thinking with optional logits, and LLM-as-a-judge scoring). Reliable step-boundary extraction also remains challenging for large reasoning models with native, unstructured "thinking" traces, which can affect online scoring and escalation.

Our empirical study focuses on a relatively narrow set of tasks – primarily math, coding, and graduate-level scientific QA; therefore, the observed quality-cost trade-offs may not generalize to other settings such as long-context question answering, open-ended generation, or tool-augmented agents. For efficiency, we report *theoretical* TFLOPs and generated token counts to ensure reproducibility and hardware-independent comparisons. Wall-clock latency is highly sensitive to batching, KV-cache reuse, hardware, and provider-side serving optimizations (e.g., vLLM prefix caching), so we leave a wall-clock study under a fixed serving configuration to future work and instead expose wall-clock measurement. as a first-class field in the benchmark's per-request log.

Acknowledgements

We thank the anonymous reviewers for their feedback, which helped improve the paper.

References

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, Che Chang, Kai Chen, Mark Chen, Enoch Cheung, Aidan Clark, Dan Cook, Marat Dukhan, Casey Dvorak, Kevin Fives, Vlad Fomenko, Timur Garipov, Kristian Georgiev, Mia Glaese, Tarun Gogineni, Adam Goucher, Lukas Gross, Katia Gil Guzman, John Hallman, Jackie Hehir, Johannes Heidecke, Alec Helyar, Haitang Hu, Romain Huet, Jacob Huh, Saachi Jain, Zach Johnson, Chris Koch, Irina Kofman, Dominik Kundel, Jason Kwon, Volodymyr Kyrylov, Elaine Ya Le, Guillaume Leclerc, James Park Lennon, Scott Lessans, Mario Lezcano-Casado, Yuanzhi Li, Zhuohan Li, Ji Lin, Jordan Liss, Lily (Xiaoxuan) Liu, Jiancheng Liu, Kevin Lu, Chris Lu, Zoran Martinovic, Lindsay McCallum, Josh McGrath, Scott McKinney, Aidan McLaughlin, Song Mei, Steve Mostovoy, Tong Mu, Gideon Myles, Alexander Neitz, Alex Nichol, Jakub Pachocki, Alex Paino, Dana Palmie, Ashley Pantuliano, Giambattista Parascandolo, Jongsoo Park, Leher Pathak, Carolina Paz, Ludovic Peran, and Dmitry Pimenov et al. 2025. gpt-oss-120b & gpt-oss-20b model card. *ArXiv preprint*, arXiv:2508.10925.

Art of Problem Solving. 2026. AIME problems and solutions. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions. Accessed 2026-05-13. General index containing the 2024 and 2025 AIME pages.

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

Edward Beeching, Lewis Tunstall, and Sasha Rush. 2024. Scaling test-time compute with open models. https://huggingface.co/spaces/HuggingFaceH4/blogpost-scaling-test-time-compute.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *ArXiv preprint*, arXiv:2107.03374.

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

Ekaterina Fadeeva, Roman Vashurin, Akim Tsvigun, Artem Vazhentsev, Sergey Petrakov, Kirill Fedyanin, Daniil Vasilev, Elizaveta Goncharova, Alexander

Panchenko, Maxim Panov, Timothy Baldwin, and Artem Shelmanov. 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, Singapore. Association for Computational Linguistics.

Yichao Fu, Xuewei Wang, Hao Zhang, Yuandong Tian, and Jiawei Zhao. 2026. Deep think with confidence. In *The Fourteenth International Conference on Learning Representations*, ICLR '26, Rio de Janeiro, Brazil.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, and Qiushi Du et al. 2025. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638.

Shibo Hao, Yi Gu, Haotian Luo, Tianyang Liu, Xiyan Shao, Xinyuan Wang, Shuhua Xie, Haodi Ma, Adithya Samavedhi, Qiyue Gao, Zhen Wang, and Zhiting Hu. 2024. LLM reasoners: New evaluation, library, and analysis of step-by-step reasoning with large language models. In *Proceedings of the First Conference on Language Modeling*, COLM '24, Philadelphia, PA, USA. OpenReview.net.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. 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, Bangkok, Thailand. Association for Computational Linguistics.

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 *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, Virtual Conference.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Thomas Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karén Simonyan, Erich Elsen, Oriol Vinyals, Jack Rae, and Laurent Sifre. 2022. An empirical analysis of compute-optimal large language model training. In *Advances in Neural Information Processing Systems*, volume 35 of *NIPS '22*, pages 30016–30030, New Orleans, LA, USA. Curran Associates, Inc.

Yuichi Inoue, Kou Misaki, Yuki Imajuku, So Kuroki, Taishi Nakamura, and Takuya Akiba. 2025. Wider or deeper? scaling llm inference-time compute with adaptive branching tree search. In *Advances in Neural Information Processing Systems*, volume 38, pages 35448–35484, San Diego, CA, USA. Curran Associates, Inc.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. 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. In *Advances in Neural Information Processing Systems*, volume 35, pages 22199–22213, New Orleans, LA, USA. Curran Associates, Inc.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, JD Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. 2025. Training language models to self-correct via reinforcement learning. In *International Conference on Learning Representations*, volume 2025 of *ICLR '25*, pages 54523–54549, Singapore.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, pages 611–626, Koblenz, Germany. Association for Computing Machinery.

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*, 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. 2024. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024 of *ICLR '24*, pages 39578–39601, Vienna, Austria.

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. In *Advances in Neural Information Processing Systems*, volume 36, pages 21558–21572, New Orleans, Louisiana, USA. Curran Associates, Inc.

Xiaoyuan Liu, Tian Liang, Zhiwei He, Jiahao Xu, Wenxuan Wang, Pinjia He, Zhaopeng Tu, Haitao Mi, and Dong Yu. 2025. Trust, but verify: A self-verification approach to reinforcement learning with verifiable rewards. In *Advances in Neural Information Processing Systems*, volume 38 of *NeurIPS '25*, pages 130475–130501, San Diego, CA, USA. Curran Associates, Inc.

Chris Lu, Cong Lu, Robert Tjarko Lange, Yutaro Yamada, Shengran Hu, Jakob Foerster, David Ha, and Jeff Clune. 2026. Towards end-to-end automation of AI research. *Nature*, 651(8107):914–919.

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

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. SELF-REFINE: iterative refinement with self-feedback. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, New Orleans, LA, USA. Curran Associates Inc.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, EMNLP '25, pages 20275–20321, Suzhou, China. Association for Computational Linguistics.

Jingwei Ni, Ekaterina Fadeeva, Tianyi Wu, Mubashara Akhtar, Jiaheng Zhang, Elliott Ash, Markus Leippold, Timothy Baldwin, See-Kiong Ng, Artem Shelmanov, and Mrinmaya Sachan. 2026. ReProbe: Efficient test-time scaling of multi-step reasoning by probing internal states of large language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, San Diego, CA, USA. Association for Computational Linguistics.

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 *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 47356–47415, Vancouver, Canada. PMLR.

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

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. 2024. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475.

Asankhaya Sharma. 2024. OptiLLM: Optimizing inference proxy for LLMs. Accessed: 2026-05-16.

Artem Shelmanov, Ekaterina Fadeeva, Akim Tsvigun, Ivan Tsvigun, Zhuohan Xie, Igor Kiselev, Nico Daheim, Caiqi Zhang, Artem Vazhentsev, Mrinmaya Sachan, Preslav Nakov, and Timothy Baldwin. 2025. A head to predict and a head to question: Pre-trained uncertainty quantification heads for hallucination detection in LLM outputs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 35712–35731, Suzhou, China. Association for Computational Linguistics.

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, Singapore.

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.

Jun Wang, Meng Fang, Ziyu Wan, Muning Wen, Jiachen Zhu, Anjie Liu, Ziqin Gong, Yan Song, Lei Chen, Lionel M. Ni, Linyi Yang, Ying Wen, and Weinan Zhang. 2024. OpenR: An open source framework for advanced reasoning with large language models. *ArXiv preprint*, arXiv:2410.09671.

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. OpenReview.net.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837, New Orleans, LA, 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, New Orleans, LA, USA. 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. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13214–13227, Vienna, Austria. Association for Computational Linguistics.

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. *ArXiv preprint, arXiv:2507.14958*.

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

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.

Appendix

A Visual Debugger UI

We provide a web-based Visual Debugger for inspecting test-time compute scaling strategies, sampled trajectories, and scoring signals. It exposes the search process step by step, including candidate generation, pruning, reranking, and final selection. Figure 4 shows the reasoning timeline, step inspector, and trajectory tree. The tool also provides a configuration interface for selecting models, strategies, and scorers; a screencast and live demo are available at the URLs in the footnotes on p.1.

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 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.

We use beam size 5 with 8 candidates, and 30 max steps for Qwen2.5; and beam size 3 with 5 candidates, and 250 max steps for Qwen3. Offline BoN and self-consistency use 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 6 gives statistics for each dataset 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 (Art of Problem Solving, 2026) are problems from the American Invitational Mathematics Examination.

For *scientific QA*, we use GPQA-Diamond (Rein et al., 2024), a subset of GPQA with graduate-level MCQs in biology, chemistry, and physics that require expert-level reasoning.

For *coding*, we use HumanEval+ and MBPP+ (Liu et al., 2023), which are extensions of HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021). 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 standard 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, $\text{TFLOPs}_{\text{total}} = \text{TFLOPs}_{\text{gen}} + \text{TFLOPs}_{\text{PRM}}$. The 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 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.

Strategy	Scorer	AIME 2024 EM (%)	AIME 2024 Δ	AIME 2024 TFLOPs	AIME 2025 EM (%)	AIME 2025 Δ	AIME 2025 TFLOPs	HumanEval-Plus Score (%)	HumanEval-Plus Δ	HumanEval-Plus TFLOPs
Baseline	—	75.6 ± 2.7	0.0	6626.9 ± 153.1	64.4 ± 6.2	0.0	8083.6 ± 305.2	79.3 ± 0.6	0.0	10094.9 ± 468.1
Extended Thinking	—	78.9 ± 5.1	3.3	26477.7 ± 1594.4	66.7 ± 3.3	2.3	27573.9 ± 1140.5	80.3 ± 2.3	1.0	36625.5 ± 504.3
Self-Consistency	—	82.2 ± 1.9	6.6	52797.2 ± 816.3	73.3 ± 5.8	8.9	53684.9 ± 1499.8	—	—	—
Offline BoN	prm (min, w=15)	81.1 ± 3.8	5.5	65399.1 ± 421.2	74.4 ± 3.9	10.0	76286.1 ± 350.1	80.9 ± 0.9	1.6	120938.7 ± 1038.0
Offline BoN	entropy (max, w=all)	74.4 ± 5.1	-1.2	52455.6 ± 392.2	70.0 ± 3.3	5.6	63349.7 ± 286.3	80.7 ± 3.4	1.4	73546.4 ± 950.6
Offline BoN	perplexity (max, w=all)	72.2 ± 6.9	-3.4	52455.6 ± 392.2	68.9 ± 5.1	4.5	63349.7 ± 286.3	80.9 ± 1.5	1.6	73546.4 ± 950.6
Offline BoN	sequence_prob (mean, w=1)	75.6 ± 1.9	0.0	52455.6 ± 392.2	74.4 ± 3.9	10.0	63349.7 ± 286.3	80.3 ± 0.7	1.0	73546.4 ± 950.6
Offline BoN	pd_gap (max, w=all)	75.6 ± 1.9	0.0	52455.6 ± 392.2	67.8 ± 6.9	3.4	63349.7 ± 286.3	81.7 ± 1.6	2.4	73546.4 ± 950.6
Beam Search (min, w=5.0)	entropy	75.6 ± 1.9	0.0	658505.8 ± 89881.0	62.2 ± 3.8	-2.2	896577.1 ± 52575.6	84.1 ± 1.6	4.8	973513.9 ± 47317.0
Beam Search (min, w=5.0)	perplexity	72.2 ± 3.8	-3.4	745916.4 ± 85315.2	64.4 ± 7.7	0.0	868007.9 ± 14121.5	81.9 ± 0.7	2.6	1061892.2 ± 101885.9
Beam Search (min, w=5.0)	prm	71.1 ± 1.9	-4.5	3294504.2 ± 204100.1	66.7 ± 9.4	2.3	3625247.2 ± 285892.8	84.8 ± 1.8	5.5	3157463.9 ± 265871.8
Beam Search (min, w=5.0)	sequence_prob	68.9 ± 5.1	-6.7	1809340.9 ± 122115.7	61.1 ± 5.1	-3.3	2528486.9 ± 148198.6	81.7 ± 2.8	2.4	1287438.5 ± 211202.5
Beam Search (min, w=5.0)	pd_gap	66.7 ± 3.3	-8.9	777367.6 ± 128986.7	65.6 ± 8.4	1.2	976551.8 ± 80244.4	82.9 ± 0.0	3.6	932738.7 ± 113300.2
MUR	entropy	74.4 ± 3.8	-1.2	313539.7 ± 45205.0	65.6 ± 5.1	1.2	457755.2 ± 20973.2	88.8 ± 0.9	9.5	723296.3 ± 17034.3
MUR	perplexity	76.7 ± 0.0	1.1	312986.7 ± 13905.3	68.9 ± 5.1	4.5	320316.1 ± 9509.8	87.0 ± 1.9	7.7	764281.0 ± 65294.1
MUR	prm	77.8 ± 3.8	2.2	1446222.0 ± 129645.7	67.8 ± 5.1	3.4	968297.5 ± 66626.3	82.1 ± 3.6	2.8	945664.2 ± 575351.3
MUR	sequence_prob	76.7 ± 3.3	1.1	340592.8 ± 42368.6	70.0 ± 5.8	5.6	459197.6 ± 148899.7	87.6 ± 0.9	8.3	689156.8 ± 14247.9
MUR	pd_gap	78.9 ± 5.1	3.3	312766.3 ± 18273.8	65.6 ± 3.8	1.2	531023.4 ± 41281.8	86.0 ± 1.6	6.7	758145.9 ± 51357.1

Table 5: Full results for Qwen3-8B across mathematical and coding benchmarks. For each strategy–scorer pair, we report accuracy together with standard deviation, improvement over the baseline (Δ), as well as compute cost in terms of TFLOPs.

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 6: Statistics of the datasets used in the experiments.

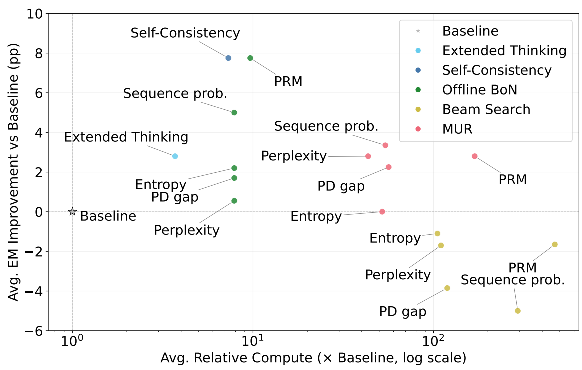


Figure 6: Accuracy improvement vs. compute ratio for Qwen3-8B on AIME (aggregate of AIME 2024 and AIME 2025). Each point represents a strategy–scorer combination; the x-axis shows the compute ratio relative to the baseline (log scale).

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; the scorer abstraction supports per-domain PRM swap via a config flag. For uncertainty-based scorers, we use entropy, perplexity, sequence probability, and probability differential, all computed on token-level log-probabilities produced during generation with negligible overhead. We also evaluate ReProbe (Ni et al., 2026), a lightweight linear probe (<10M parameters) trained on internal model representations.

Scorer configurations. For the results reported in Figure 5 in the main text and in Figure 6, we select the best combination of step-level aggregation function and scoring window through a grid search over the corresponding hyperparameter space. This is done independently for each scorer and strategy configuration for a fair comparison across methods.

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 the mean and the minimum aggregation with w=5, and we further include ReProbe (Ni et al., 2026) with mean aggregation over all steps. ReProbe is a lightweight linear probe head with negligible overhead.

MUR uses default settings. Compute is measured in TFLOPs for generation plus scoring.

Detailed per-dataset results. Table 5 reports the full per-dataset results for Qwen3-8B across the mathematical (AIME 2024, AIME 2025) and coding (HumanEval+) benchmarks, which are shown in aggregate in Figures 5b and 6. For each strategy–scorer pair, we report accuracy with standard deviation, improvement over the baseline (Δ), as well as compute cost in TFLOPs.