# The Conductor and the Engine: A Path Towards Co-Designed Reasoning

**Yuanxin Wang**
michael.wang@cerebras.net

**Pawel Filipczuk**
pawel.filipczuk@cerebras.net

**Anisha Garg**
anisha.garg@cerebras.net

**Amaan Dhada**
amaan.dhada@cerebras.net

**Mohammad Hassanpour**
mohammad.hassanpour@cerebras.net

**David Bick**
david.bick@cerebras.net

**Ganesh Venkatesh**
ganesh.venkatesh@cerebras.net

APPLIED AI RESEARCH, CEREBRAS

## Abstract

Modern LLM reasoning relies on extensive test-time computation, driven by internal model training and external agentic orchestration. However, this synergy is often inefficient, as model verbosity and poor instruction following lead to wasted compute. We analyze this capability-cost trade-off and introduce an optimized reasoning workflow (CEPO) that empowers smaller open-source models to outperform models multiple times their size. Our work demonstrates a clear path toward co-designing orchestration frameworks with the underlying model capabilities to unlock powerful reasoning in small-to-medium sized models. Our work is open-sourced at https://github.com/codelion/optillm/tree/main/optillm/cepo.

## 1 Introduction

The ability of Large Language Models (LLMs) to solve exceptionally complex problems in domains like mathematics, software development, and strategic games is increasingly unlocked by substantial computation at inference time [9, 8, 32, 4]. Progress in this domain is advancing along two parallel fronts: i) **Reasoning Engine:** training models with techniques like reinforcement learning to cultivate implicit, "internal" reasoning capabilities [22, 20], ii) **Conductor**: external, agentic workflows that decompose tasks into iterative, verifiable steps [4, 18]. While the combination of these internal model capabilities and external frameworks holds enormous promise, it also introduces critical new challenges in efficiency and scalability which we evaluate and address in this paper.

The first of these challenges is effort duplication. When both the external orchestration framework (the "Conductor") and the internal model (the "Engine") attempt to perform high-level reasoning, they can produce redundant or conflicting operations, leading to wasted compute. This inefficiency is compounded by a second, pervasive challenge: weak instruction following [3, 7]. These powerful reasoning models often struggle to adhere perfectly to complex instructions, causing the agentic flow

Accepted to Workshop on Efficient Reasoning @ 39th Conference on Neural Information Processing Systems (NeurIPS 2025).

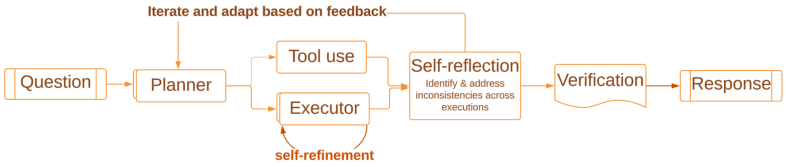


Figure 1: **Overview of CEPO Framework**. Simplified view of our orchestration framework

to deviate, consume excess resources, and ultimately fail. Together, these issues create a significant barrier to deploying robust and capable reasoning systems at scale.

To overcome these obstacles, this paper introduces **C**onductor for **E**fficient **P**lanning and **O**rchestration (CEPO), an adaptive framework that actively orchestrates the reasoning Engine. Our approach amplifies model capabilities, enabling medium-sized models to achieve state-of-the-art results that surpass models multiple times their size across challenging tasks in scientific reasoning, math and code generation. This provides a clear and efficient path towards the next generation of powerful AI systems - ones that deliver elite performance and can be deployed on systems with practical memory budgets.

## 2 The Conductor: A Framework for State-of-the-Art Reasoning

CEPO (simplified view in Figure 1) is composed of several key components that work in concert. It begins with an **adaptive planner** that assesses a given problem and can map out multiple distinct execution paths to find a solution. The framework then passes these plans to **executor**, which attempts to solve the problem. A crucial element of executor is **iterative self-refinement**, where the system can analyze feedback from its own solution attempts such as the output from a code execution and adapt its approach (such as **problem reformulation**) accordingly. The executor phase produces multiple possible executions which is then passed onto **self-reflection** phase to synthesize insights from multiple attempts and facilitate "cross-pollination" of ideas by identifying and addressing inconsistencies across different executions. Final step of our orchestration is **Verification** which can invoke multiple methods such as majority vote or LLM-as-a-judge based on the problem setting.

The above formulation (Algorithm 1, Appendix C) enables us to dynamically allocate more compute to more difficult problems by invoking multiple stages of problem reformulation and self-refinement while the simple queries can pass through without invoking all the stages of the pipeline. This "right-tool-for-the-job" approach allows the framework to invoke a variety of external tools as needed including Python interpreter, code compilation and verification.

The effectiveness of CEPO is demonstrated in Table 1 and Table 2 by its ability to elevate strong open-source models to the top of leaderboards, often outperforming much larger commercial models. For Qwen [32] family of models, applying our orchestration allows Qwen3 32B to outperform much larger models such as DeepSeek R1, Qwen3 235B and even closed-source models such as OpenAI O3-mini [16]. To demonstrate the generality of our approach, we also applied it to the recently released GPT-OSS [17] models in a manner compliant with the Artificial Analysis [1] benchmark suite. On various tasks[1], our framework boosts these models to the top tier of leaderboards[2].

The rest of the paper will delve deeper into the importance of these individual components, with detailed analysis and ablations. We will conclude with a discussion of future work focused on further enhancing the capabilities of the orchestration via co-design with next-generation reasoning models.

## 3 Deconstructing the Conductor: A Component-wise Analysis

We systematically evaluate the impact of the core components – planning, self-reflection and cross-solution verification – on a challenging reasoning benchmarks. The results, detailed in Table 3 and Table 4, reveal the importance of this multi-component orchestration in CEPO for achieving high performance on diverse reasoning tasks. Our analysis begins to uncover the critical interplay

[1]GPQA results are averaged across three runs.

[2]The AIME benchmark was excluded for this model as its baseline performance already approached 95%.

Table 1: Our framework, CEPO, elevates the medium-sized Qwen3 model to achieve state-of-the-art (SOTA) performance, outperforming much larger open-source models as well as leading closed-source reasoning systems on math and coding benchmarks. Evaluation setup details in Appendix B.

| Benchmark | Qwen3 8B | Qwen3 8B + CEPO | Qwen3 32B | Qwen3 32B + CEPO | Qwen3 235B | DeepSeek R1 [8] | o3-mini med. [16] | Grok3 Think [30] |
|---|---|---|---|---|---|---|---|---|
| AIME 2024 [2] | 74.00 | 86.70 | 81.40 | 90.70 | 85.70 | 79.80 | 79.60 | 83.90 |
| AIME 2025 [2] | 68.33 | 80.00 | 72.90 | 83.30 | 81.50 | 70.00 | 74.80 | 77.30 |
| GPQA [19] | 59.25 | 62.45 | 66.83 | 70.03 | 71.10 | 71.50 | 76.80 | 80.20 |
| LIVECODEBENCH [10] | 55.69 | 60.48 | 65.70 | 71.86 | 70.70 | 64.30 | 66.30 | 70.60 |

Table 2: CEPO turbocharges recent GPT-OSS [17] models to top positions of Artificial Analysis leaderboard for multiple tasks [1]. Evaluation setup details in Appendix B.

| Benchmark | gpt-oss-20b | gpt-oss-20b + CEPO | gpt-oss-120b | gpt-oss-120b + CEPO | Qwen3 235B 2507 [32] | Gemini 2.5 Pro [4] | Grok4 [31] |
|---|---|---|---|---|---|---|---|
| LIVECODEBENCH [10] | 72.10 | 82.01 | 76.82 | 87.51 | 79.00 | 80.00 | 82.00 |
| SCICODE [24] | 35.40 | 40.10 | 36.20 | 41.00 | 42.00 | 43.00 | 46.00 |
| GPQA [19] | 70.70 | 76.01 | 76.50 | 82.57 | 79.00 | 84.40 | 87.70 |

between the CEPO strategy and the nature of the task, highlighting the Conductor's role in maximally leveraging the reasoning Engine to solve challenging problems.

We explore this dynamic by analyzing the framework's impact on different model families and reasoning tasks, which reveals several key observations:

**Planning** The utility of the planning component appears to be highly task-dependent. We observe significant benefits on scientific reasoning benchmarks such as GPQA. However, its direct impact on improving coding performance was more muted in our experiments. This suggests that a reasoning model specifically post-trained to utilize an explicit planning phase could more fully exploit this component.

**Self-Reflection** The iterative self-reflection component provides a consistent and strong accuracy advantage across the models and tasks we evaluated. This benefit likely stems from two sources: the model's intrinsic ability to critique and refine its own work, and the framework's process of synthesizing insights from multiple execution attempts, effectively cross-pollinating ideas to produce a superior final output.

**Verification** The verification step presents a significant opportunity for future improvement. We observe a notable gap between the final performance of our Conductor and the theoretical maximum achievable (i.e., the recall at best-of-N across multiple attempts), indicating that a more accurate verifier could unlock substantial gains. This points to a promising direction for co-design: explicitly

Table 3: Performance comparison of Qwen3 [32] models, with and without components of our proposed CEPO framework. Evaluation configuration details shown at Appendix B.

| Config | AIME 2025 | GPQA | LiveCodeBench | SciCode |
|---|---|---|---|---|
| Qwen3 8B | 68.33 | 59.25 | 58.05 | 31.70 |
| CEPO | 72.66 | 62.45 | 61.58 | 37.10 |
| w/o planner | 74.66 | 59.42 | 60.63 | 37.00 |
| w/o self-reflection | 69.72 | 60.65 | 59.61 | 19.60 |
| recall@best_of_N | 79.33 | 71.21 | 64.76 | 40.10 |
| Qwen3 32B | 74.00 | 66.83 | 66.77 | 35.40 |
| CEPO | 78.00 | 70.03 | 71.21 | 40.90 |
| w/o planner | 79.33 | 66.83 | 71.21 | 42.80 |
| w/o self-reflection | 76.00 | 67.27 | 67.40 | 33.40 |
| recall@best_of_N | 83.33 | 76.43 | 72.69 | 46.10 |

Table 4: Performance of GPT-OSS [17] benefits from all the components of our CEPO framework while showing room for accuracy boost by improving model capabilities in planning and verification.

| Config | GPQA | LiveCodeBench | SciCode |
|---|---|---|---|
| gpt-oss-20b | 70.70 | 72.10 | 35.40 |
| CEPO | 76.01 | 82.01 | 40.10 |
| w/o planner | 72.89 | 81.48 | 42.90 |
| w/o self-reflection | 73.57 | 77.40 | 38.20 |
| recall@best_of_N | 81.31 | 83.17 | 45.50 |
| gpt-oss-120b | 76.50 | 76.82 | 36.20 |
| CEPO | 82.57 | 87.51 | 41.00 |
| w/o planner | 76.76 | 87.19 | 41.00 |
| w/o self-reflection | 78.84 | 82.79 | 38.80 |
| recall@best_of_N | 87.62 | 88.25 | 44.10 |

training the reasoning Engine to act as a verifier, for instance through reinforcement learning, to more effectively guide the test-time computation.

## 4 Dynamic Problem Reformulation for Efficient Reasoning

CEPO employs dynamic problem reformulation, adapting its strategy based on the problem's complexity and the specific strengths of the Engine model. This allows the system to select the most efficient and reliable path to a solution. We illustrate this capability with two examples below.

**Adaptive Path Selection** We observe that models have strong implicit preferences for solving problems in a certain way (Table 5); for instance, some mathematical problems are solved more reliably through direct textual reasoning ("mental math"), while others are better suited for code generation and execution. Our orchestration flow (Algorithm 2, Appendix C) captures this insight by exploring multiple solution paths and prioritize the one that is most likely to succeed for that specific model. This adaptive approach yields solid gains, boosting the performance of both the Qwen3 8B and Qwen3 32B models on the AIME 2024 and AIME 2025 math benchmarks as shown in Table 6, beating out larger, math-specific models [21]. More qualitative examples are at Appendix D.

Table 5: **Adaptive Path Selection Maximizes Performance** The Pass@10 success rate is highest when the system can choose the optimal solution path, as direct reasoning ("mental math") and code generation excel on different subsets of problems. Dataset here is 15 hard questions from Numina Math dataset [12].

| Mental Math only | Coding only | Mental Math or Code |
|---|---|---|
| 93% | 93% | 100% |

Table 6: Adaptive path selection via code generation provides a significant performance boost for mathematical reasoning. The table shows ablation results where adding the adaptive coding reformulation to CEPO yields large gains on the AIME benchmarks.

| Model | AIME 2024 | AIME 2025 | Model | AIME 2024 | AIME 2025 |
|---|---|---|---|---|---|
| Qwen3 8B | 74.00 | 68.33 | Qwen3 32B | 81.40 | 74.00 |
| CEPO *w/o coding* | 83.33 | 72.66 | CEPO *w/o coding* | 84.00 | 78.00 |
| CEPO | 86.67 | 80.00 | CEPO | 90.70 | 83.30 |

**Iterative Refinement with Rich Feedback** In complex tasks like code generation, a model's first attempt may not be perfect. When our framework executes a generated piece of code, it captures the execution feedback (e.g., interpreter error messages or incorrect output). The problem is then reformulated and presented back to the model, including the original prompt along with this rich feedback and an instruction to correct its previous mistakes. This iterative refinement loop, where the task is progressively clarified based on execution results, leads to significant improvements in code generation accuracy (Table 7).

Table 7: Performance gains for the Qwen3 8B and Qwen3 32B models on the LiveCodeBench benchmark using iterative self-refinement. Evaluation configuration details shown at Appendix B.

| Model | Baseline | CEPO | CEPO with Tests |
|---|---|---|---|
| Qwen3 8B | 55.69 | 56.29 | 60.48 |
| Qwen3 32B | 65.70 | 67.07 | 71.86 |

## 5 Conclusion

In this work, we demonstrated that intelligent orchestration is a parameter-efficient path to elite performance. Our framework propelled medium-sized open-source models, such as the Qwen3 32B and GPT-OSS, to the top of competitive leaderboards, allowing them to outperform models multiple times their size. These findings point towards a future of co-designing LLMs and their reasoning frameworks. Instead of relying on rigid, fixed templates, orchestration should be dynamically guided by model capabilities and problem complexity. The next frontier is to move beyond monolithic reasoning engines and forge models with a toolkit of capabilities - such as planning, task decomposition, verification, reflection - enabling the conductor to transform from a mere prompter into a true director of computational thought.

## References


[1] Artificial analysis: Ai model & api providers analysis. `https://artificialanalysis.ai/`. Accessed: 2025-09-05.

[2] Aime problems and solutions. AoPS Wiki, 2025. Accessed: 2025-09-05.

[3] Andong Chen, Yuchen Song, Wenxin Zhu, Kehai Chen, Muyun Yang, Tiejun Zhao, and Min zhang. Evaluating o1-like llms: Unlocking reasoning for translation through comprehensive analysis, 2025.

[4] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, July 2025.

[5] Mehdi Fatemi, Banafsheh Rafiee, Mingjie Tang, and Kartik Talamadupula. Concise reasoning via reinforcement learning. *arXiv preprint arXiv:2504.05185*, 2025.

[6] Pawel Filipczuk, Vithursan Thangarasa, Eric Huang, Amaan Dhada, Michael Wang, Rohan Deshpande, Emma Call, and Ganesh Venkatesh. Cepo: Empowering llama with reasoning using test-time compute. `https://cerebras.ai/blog/cepo`,, 2024.

[7] Tingchen Fu, Jiawei Gu, Yafu Li, Xiaoye Qu, and Yu Cheng. Scaling reasoning, losing control: Evaluating instruction following in large reasoning models, 2025.

[8] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and et al. Deepseek-R1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

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

[10] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, March 2024.

[11] Ivan Lazarevich, Mohammad Hassanpour, and Ganesh Venkatesh. Longcepo: Empowering llms to efficiently leverage infinite context. `https://cerebras.ai/blog/longcepo`,, 2025.

[12] Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. Numinamath: The largest public dataset in ai4maths with 860 k competition math problem–solution pairs, 2024.

[13] Zhenghao Lin, Zhibin Gou, Yeyun Gong, Xiao Liu, Yelong Shen, Ruochen Xu, Chen Lin, Yujiu Yang, Jian Jiao, Nan Duan, et al. Rho-1: Not all tokens are what you need. *arXiv preprint arXiv:2404.07965*, 2024.

[14] Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. `https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303` 2025. Notion Blog.

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

[16] OpenAI. Openai o3-mini system card. System card, published February 10, 2025, February 2025.

[17] OpenAI et al. gpt-oss-120b & gpt-oss-20b Model Card. arXiv preprint arXiv:2508.10925, 2025.

[18] Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. Mutual reasoning makes smaller llms stronger problem-solvers. *arXiv preprint arXiv:2408.06195*, August 2024.

[19] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, November 2023.

[20] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, July 2017.

[21] Ning Shang, Yifei Liu, Yi Zhu, Li Lyna Zhang, Weijiang Xu, Xinyu Guan, Buze Zhang, Bingcheng Dong, Xudong Zhou, Bowen Zhang, Ying Xin, Ziming Miao, Scarlett Li, Fan Yang, and Mao Yang. rstar2-agent: Agentic reasoning technical report, 2025.

[22] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y.K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, February 2024.

[23] Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Hanjie Chen, et al. Stop overthinking: A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*, 2025.

[24] Minyang Tian, Luyu Gao, Shizhuo Dylan Zhang, Xinan Chen, Cunwei Fan, Xuefei Guo, Roland Haas, Pan Ji, Kittithat Krongchon, Yao Li, Shengyan Liu, Di Luo, Yutao Ma, Hao Tong, Kha Trinh, Chenyu Tian, Zihan Wang, Bohao Wu, Yanyu Xiong, Shengzhu Yin, Minhui Zhu, Kilian Lieret, Yanxin Lu, Genglin Liu, Yufeng Du, Tianhua Tao, Ofir Press, Jamie Callan, Eliu Huerta, and Hao Peng. Scicode: A research coding benchmark curated by scientists. *arXiv preprint arXiv:2407.13168*, July 2024.

[25] Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692*, 2024.

[26] Michael Wang, Anisha Garg, Pawel Filipczuk, David Bick, Akil Pathiranage, and Ganesh Venkatesh. Cepo update: Turbocharging reasoning models' capability using test-time planning. `https://cerebras.ai/blog/cepo-update-turbocharging-reasoning-models-capability-using-test-time-planning`,, 2025.

[27] Xuezhi Wang and Denny Zhou. Chain-of-thought reasoning without prompting. *Advances in Neural Information Processing Systems*, 37:66383–66409, 2024.

[28] Yuanxin Wang and Ganesh Venkatesh. Read quietly, think aloud: Decoupling comprehension and reasoning in llms. *arXiv preprint arXiv:2507.03327*, 2025.

[29] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[30] xAI. Grok 3 beta — the age of reasoning agents. `https://x.ai/news/grok-3`, February 2025. Accessed: 2025-09-05.

[31] xAI. Grok 4 announcement. `https://x.ai/news/grok-4`, July 2025. Accessed: 2025-09-05.

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

[33] Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. Limo: Less is more for reasoning. *arXiv preprint arXiv:2502.03387*, 2025.

[34] Zhenrui Yue, Honglei Zhuang, Aijun Bai, Kai Hui, Rolf Jagerman, Hansi Zeng, Zhen Qin, Dong Wang, Xuanhui Wang, and Michael Bendersky. Inference scaling for long-context retrieval augmented generation. *arXiv preprint arXiv:2410.04343*, 2024.

[35] Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

## A Related Work

Many prior work on reasoning models has largely focused on instilling advanced thinking capabilities into model weights through multiple stages of training — from pretraining [28, 13], supervised fine-tuning [33, 15], to pruning [23] and reinforcement learning [20, 22, 35, 5, 14].

In contrast to these training-based approaches, an emerging line of research investigates training-free algorithms that operate purely at inference time. State-of-the-art reasoning systems such as OpenAI's o1 [9] and o3-mini [16], DeepSeek R1 [8], and Gemini 2.5 [4] achieve their performance largely by allocating substantial computation at inference. Techniques like Chain-of-Thought prompting [29], Monte Carlo Tree Search [18], and CoT-decoding [27] improve reasoning by broadening exploration of intermediate thinking trajectories and systematically breaking down complex problems into clear sub-tasks.

Foundational test-time strategies, such as Chain-of-Thought and Monte Carlo Tree Search, emerged in the era of large-scale LLMs and were often designed to leverage their massive parameter counts, creating a high barrier to entry. However, the recent advent of powerful, parameter-efficient reasoning models challenges this dependency on scale and warrants new research into orchestration techniques built to maximize their unique capabilities. Therefore, recent research has moved toward building multi-agent, iterative frameworks on top of **smaller LLMs** [6, 26, 25], with support of ultra long-context, [34, 11], and integrating planning, decomposition, self-reflection, and verification into unified orchestration pipelines.

Our work builds on this trajectory by demonstrating that carefully designed test-time orchestration can amplify small- and medium-scale models to outperform models many times their size, offering a practical path toward scalable and cost-efficient reasoning systems.

## B Evaluations Details

For Qwen3 family results in Table 1 and 7, we use the evaluation setup in Qwen3 Technical Report [32] and reach parity with the reported scores for both Qwen3-8B and Qwen3-32B models. In our orchestration framework, for AIME, we adopt our adaptive path selection algorithm which allows the model to solve the problem using code; for LiveCodeBench, we utilize public tests and execution feedback for iterative self-refinement, with the same date range and release version used in [32]: release v5 from 2024-08-01 to 2025-02-01.

For gpt-oss family results in Table 2 and all component-wise analysis in Table 3 and 4, we run the benchmarks in Artificial Analysis [1] compliant way and reach parity for both gpt-oss-20B and gpt-oss-120B models. For LiveCodeBench, we didn't use public tests and execution feedback for LiveCodeBench since this is not used by Artificial Analysis; and use the same date range and release version in [1]: release v6 from 2024-07-01 to 2025-01-01.

## C CEPO and Adaptive Path Selection Algorithmic Illustration

**Algorithm 1:** CEPO-Simple (w/o Adaptive Path Selection)

**Input :** Question $\mathcal{Q}$
Planning Agent $\theta_{\text{plan}}$
Execution Agent $\theta_{\text{execute}}$
Self-Reflection Agent $\theta_{\text{reflection}}$
Verification Agent $\theta_{\text{verify}}$

**Parameters :** `best_of_n`: number of attempts
`n_plans`: number of plans and executions per attempt

**Output :** Final answer $\hat{A}$

$\mathcal{S} \leftarrow [\,]$
**for** $i \leftarrow 1$ **to** *best_of_n* **do**
  $\mathcal{E} \leftarrow [\,]$
  **for** $j \leftarrow 1$ **to** *n_plans* **do**
    $\pi_j \leftarrow \theta_{\text{plan}}(\mathcal{Q})$
    $e_j \leftarrow \theta_{\text{execute}}(\mathcal{Q}, \pi_j)$ `// optionally apply self-refinement here`
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{e_j\}$
  $s_i \leftarrow \theta_{\text{reflection}}(\mathcal{E})$
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{s_i\}$
$\hat{A} \leftarrow \theta_{\text{verify}}(\mathcal{Q}, \mathcal{S})$
**return** $\hat{A}$

**Algorithm 2:** CEPO-Adaptive for Math Problems

**Input :** Question $\mathcal{Q}$
Planning Agent $\theta_{\text{plan}}$
Execution Agent $\theta_{\text{execute}}$
Self-Refinement Agent $\theta_{\text{reflection}}$
Verification Agent $\theta_{\text{verify}}$
Coding Agent $\theta_{\text{code}}$

**Parameters :** `num_attempts_CEPO`: number of attempts to use CEPO-Simple flow
`num_attempts_baseline`: number of attempts to use the executor agent directly
`num_attempts_coding`: number of attempts to use the coding agent
`n_plans`: number of plans and executions per attempt inside CEPO-Simple
`best_of_n`: number of attempts inside CEPO-Simple

**Output :** Final answer $\hat{A}$

**Helper routines:**

`StrictMajority`$(\mathcal{A})$: returns (has_maj, $a^\star$) where $a^\star$ occurs $> \frac{|\mathcal{A}|}{2}$ times; otherwise $(\text{false}, \perp)$.

`PluralityVote`$(\mathcal{A})$: returns the mode(s) $\mathcal{M}$ with maximal frequency (may contain ties).

$\mathcal{S} \leftarrow [\,]$ `// collector for all candidate answers`

**Baseline (direct executor)**

$\mathcal{B} \leftarrow [\,]$
**for** $i \leftarrow 1$ **to** *num_attempts_baseline* **do**
  $b_i \leftarrow \theta_{\text{execute}}(\mathcal{Q})$
  $\mathcal{B} \leftarrow \mathcal{B} \cup \{b_i\}$
$\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{B}$
**if** $|$*PluralityVote*$(\mathcal{B})| = 1$ ***and*** *StrictMajority*$(\mathcal{B})$*.has_maj* $=$ *true* **then**
  **return** the unique $b^\star$ from `PluralityVote`$(\mathcal{B})$

**Coding agent pass**

$\mathcal{C} \leftarrow [\,]$
**for** $i \leftarrow 1$ **to** *num_attempts_coding* **do**
  $c_i \leftarrow \theta_{\text{code}}(\mathcal{Q})$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{c_i\}$
$\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{C}$
**if** *StrictMajority*$(\mathcal{C})$*.has_maj* $=$ *true* **then**
  **return** `StrictMajority`$(\mathcal{C}).a^\star$

**CEPO-Simple passes**

$\mathcal{D} \leftarrow [\,]$ `// answers from CEPO-Simple`
**for** $i \leftarrow 1$ **to** *num_attempts_* `CEPO` **do**
  $d_i \leftarrow$ CEPO-Simple$(\mathcal{Q}, \theta_{\text{plan}}, \theta_{\text{execute}}, \theta_{\text{reflection}}, \theta_{\text{verify}};$ `best_of_n`; `n_plans`$)$
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{d_i\}$
$\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{D}$

**Global aggregation**

$\mathcal{M} \leftarrow$ `PluralityVote`$(\mathcal{S})$
**return** the modes $a^\star \in \mathcal{M}$

## D Examples for Adaptive Path Selection

**Example 1: AIME 2024 question where CEPO w/o code fails for Qwen3-8B**

**Question:**
Each vertex of a regular octagon is independently colored either red or blue with equal probability. The probability that the octagon can then be rotated so that all of the blue vertices end up at positions where there were originally red vertices is $\frac{m}{n}$, where $m$ and $n$ are relatively prime positive integers. What is $m+n$?

---

**Ground Truth Answer:**
371

---

**Output from Qwen3-8B + CEPO w/o coding (Truncated due to space):**
To determine the probability that a randomly colored octagon (with each vertex independently colored red or blue with equal probability) can be rotated so that all blue vertices end up on positions that were originally red, we analyze the combinatorial structure of the problem.
**Understanding the Problem**
We are given an octagon with 8 vertices, each independently colored .....
**Strategy and Key Observations**
Let define $A_k$ as the set of all colorings where $ B + k, cap B = emptyset, .....
**Final Answer**
The probability that the octagon can be rotated so that all blue vertices end up on red positions is $\frac{121}{256}$, where $m = 121$ and $n = 256$ are relatively prime.Thus, the final result is:

$$\boxed{377}$$

---

**Output from Qwen3-8B + CEPO with coding:**

```
import math
count = 0
for num in range(256):
    color = [(num >> i) & 1 for i in range(8)]
    blue = [i for i in range(8) if color[i] == 1]
    found = False
    for k in range(8):
        valid = True
        for b in blue:
            pos = (b + k) % 8
            if color[pos] != 0:
                valid = False
                break
        if valid:
            found = True
            break
    if found:
        count += 1
g = math.gcd(count, 256)
m = count // g
n = 256 // g
print(m + n)
```

**Code Execution Result:** 371

---

This is a geometry problem that requires complex reasoning and very long chain-of-thought, for which Qwen3-8B might struggle. However, the small-size model is able to solve this problem using just a few lines of code.

Figure 2: Example of AIME 2024 question where "mental math" prompt fails but code generation succeeds

**Example 2: AIME 2025 question where CEPO w/o code fails for Qwen3-8B**

**Question:**
Let $N$ denote the number of ordered triples of positive integers $(a, b, c)$ such that $a, b, c \leq 3^6$ and $a^3 + b^3 + c^3$ is a multiple of $3^7$. Find the remainder when $N$ is divided by 1000.

---

**Ground Truth Answer:**
735

---

**Output from Qwen3-8B + CEPO w/o coding (Truncated due to space):**
To solve the problem, we are asked to count the number of ordered triples $(a, b, c)$ of positive integers such that:- $1 \leq a, b, c \leq 3^6 = 729$- $a^3 + b^3 + c^3$ is divisible by $3^7 = 2187$. Then, we are to compute this number modulo 1000.
Step 1: Total Number of Triples.Each of $a, b, c$ can independently range from 1 to $729 = 3^6$, so the total number of ordered triples is...

$$N_{\text{total}} = (3^6)^3 = 3^{18}$$

Step 2: Modulo Condition. We are interested in the number of such triples $(a, b, c)$...

$$a^3 + b^3 + c^3 \equiv 0 \pmod{3^7}$$

Step 3: Uniform Distribution AssumptionA key insight is that **modulo $3^k$**, the function $x \mapsto x^3$ is **not uniformly distributed**, as shown in small cases (e.g., modulo 9, where cube residues are not all distinct or evenly distributed)...
Step 4: Final Computation: Now, compute:

$$3^{11} = 177147$$

Now take modulo 1000:

$$177147 \mod 1000 = 147$$

### Final Answer

$$\boxed{147}$$

---

**Output from Qwen3-8B + CEPO with coding:**

```
m = 3**7
max_a = 3**6
freq = [0] * m
for a in range(1, max_a + 1):
    r = pow(a, 3, m)
    freq[r] += 1
C = [0] * m
for r1 in range(m):
    for r2 in range(m):
        s = (r1 + r2) % m
        C[s] += freq[r1] * freq[r2]
total = 0
for s in range(m):
    target = (-s) % m
    total += C[s] * freq[target]
print(total % 1000)
```

**Code Execution Result:** 735

---

This is a hard combinatorial number theory problem which requires a lot of sequential reasoning and memory, and it is very difficult to fit all these thinking into a 40960 context window of Qwen-3 Model. However, when we ask the model to solve it with code, it only takes a few tokens to reach the correct result.

Figure 3: Example of AIME 2025 question where "mental math" prompt fails but code generation succeeds

**Example 3: AIME 2024 question where CEPO w/o code fails for Qwen3-32B**

**Question:**
Define $f(x) = ||x| - \frac{1}{2}|$ and $g(x) = ||x| - \frac{1}{4}|$. Find number of intersections of the graphs of

$$y = 4g(f(\sin(2\pi x))) \quad \text{and} \quad x = 4g(f(\cos(3\pi y))).$$

---

**Ground Truth Answer:**
385

---

**Output from Qwen3-32B + CEPO w/o coding (Truncated due to space):**
To determine the number of intersections between the graphs of the functions ... we begin by analyzing the structure and behavior of the component functions involved.
**Step 1: Understanding the Base Functions...This function is piecewise linear with a W̆shape over the interval $[-1, 1]$. It has peaks at $x = 0$ and $x = \pm 1$ (value $\frac{1}{2}$) and valleys at ...
**Step 2: Mapping the Composites**#### **Function 1: $y = 4g(f(\sin(2\pi x)))$** The inner function has a period of 1 and oscillates between $-1$ and 1. Taking the absolute value gives $|\sin(2\pi x)|$, which oscillates between ...
**Step 3: Counting Valid Intersections**We now consider the number of **valid intersections** between the graphs. The graphs are defined by:- The first function has **16 segments** with slopes... Thus, the total number of valid intersections is: 96 + 96 = $\boxed{192}$

---

**Output from Qwen3-32B + CEPO with coding:**

```
import math
from scipy.optimize import root
import numpy as np
def h(z):
    fz = abs(abs(z) - 0.5)
    return 4 * abs(fz - 0.25)
def equations(vars):
    x, y = vars
    eq1 = y - h(math.sin(2 * math.pi * x))
    eq2 = x - h(math.cos(3 * math.pi * y))
    return [eq1, eq2]
x_values = np.linspace(0, 1, 100)
y_values = np.linspace(0, 1, 100)
solutions = set()
for x0 in x_values:
    for y0 in y_values:
        sol = root(equations, [x0, y0], method='hybr')
        x_sol, y_sol = sol.x
        if 0 <= x_sol <= 1 and 0 <= y_sol <= 1:
        rounded_sol = (round(x_sol, 6), round(y_sol, 6))
        y_check = h(math.sin(2 * math.pi * x_sol))
        x_check = h(math.cos(3 * math.pi * y_sol))
        if abs(y_check - y_sol) < 1e-5 \
        and abs(x_check - x_sol) < 1e-5:
        solutions.add(rounded_sol)
print(len(solutions))
```

**Code Execution Result: 385**

---

This is a complex problem which mixes analysis of piecewise functions, trigonometric composition, and geometric interpretation. Even a medium size Qwen3-32B model struggles to get it correct using CEPOwithout code. Code generation takes the model out of the local optimal in an innovative way.

Figure 4: Another Example of AIME 2024 question where "mental math" prompt fails but code generation succeeds

**Example 4: AIME 2025 question where CEPO w/o code fails for Qwen3-32B**

**Question:**
Alex divides a disk into four quadrants with two perpendicular diameters intersecting at the center of the disk. He draws 25 more line segments through the disk, drawing each segment by selecting two points at random on the perimeter of the disk in different quadrants and connecting those two points. Find the expected number of regions into which these 27 line segments divide the disk.

---

**Ground Truth Answer:**
204

---

**Output from Qwen3-32B + CEPO w/o coding (Truncated due to space):**
Final Review and Error Analysis: The initial solution incorrectly applied the general formula for the expected number of regions formed by random chords:

$$E(R) = 1 + \binom{n}{2} + \binom{n}{4}$$

This formula assumes **no constraints** on the endpoints of the chords. However, in the problem, the chords are **constrained** to ... Correct Approach: To compute the expected number of regions formed by 27 line segments (2 fixed diameters and 25 random chords connecting points in different quadrants), we proceed in steps: 1. **Initial Setup**- The disk is first divided into 4 quadrants by two perpendicular diameters. These diameters intersect at the center, forming **4 initial regions**.- Alex then draws **25 additional chords**, where each chord connects two points on the circle perimeter in **different quadrants**. The total number of line segments is therefore **27**. 2. **Expected Intersections** ...
Final Answer

$$\boxed{\frac{287}{3}}$$

This is the **expected number of regions** into which the 27 line segments divide the disk.

---

**Output from Qwen3-32B + CEPO with coding:**

```
import fractions
# Calculate each part of the expected intersections
I_1 = 1  # Intersection between the two original diameters
I_2 = 25 * 2 * fractions.Fraction(2, 3)  # Intersections between
    original diameters and added chords
# Number of pairs of added chords
num_pairs_added = (25 * 24) // 2
I_3 = num_pairs_added * fractions.Fraction(17, 36)  #
   Intersections between pairs of added chords
# Total expected intersections
E_I = I_1 + I_2 + I_3
E_R = 1 + 27 + E_I
print(int(E_R))
```

**Code Execution Result:** 204

---

This is a geometric probability / combinatorial geometry problem. As shown above, CEPO without coding struggles to get the correct answer even after extensive self-reflection. In the code generation output, the model calculates some magic numbers in <think> section (too long to fit in) and finish the problem utilizing those numbers precomputed.

Figure 5: Another Example of AIME 2025 question where "mental math" prompt fails but code generation succeeds