Test-time Scaling Techniques in Theoretical Physics - A Comparison of Methods on the TPBench Dataset

Zhiqi Gao*
Department of Computer Science
University of Wisconsin-Madison

Tianyi Li*
Department of Physics
University of Wisconsin-Madison

Yurii Kvasiuk
Department of Physics
University of Wisconsin-Madison

Sai Chaitanya Tadepalli
Department of Physics
Indiana University, Bloomington

Daniel J.H. Chung
Department of Physics
University of Wisconsin-Madison

Maja Rudolph
Data Science Institute (DSI)
University of Wisconsin-Madison

Frederic Sala
Department of Computer Science
University of Wisconsin-Madison
`fredsala@cs.wisc.edu`

Moritz Münchmeyer
Department of Physics
University of Wisconsin-Madison
`muenchmeyer@wisc.edu`

Abstract

Large language models (LLMs) have shown strong capabilities in complex reasoning, and test-time scaling techniques can enhance their performance with comparably low cost. Many of these methods have been developed and evaluated on mathematical reasoning benchmarks such as AIME. This paper investigates whether the lessons learned from these benchmarks generalize to the domain of advanced theoretical physics. We evaluate a range of common test-time scaling methods on the TPBench physics dataset and compare their effectiveness with results on AIME. To better leverage the structure of physics problems, we develop a novel, symbolic weak-verifier framework to improve parallel scaling results. Our empirical results demonstrate that this method significantly outperforms existing test-time scaling approaches on TPBench. We also evaluate our method on AIME, confirming its effectiveness in solving advanced mathematical problems. Our findings highlight the power of step-wise symbolic verification for tackling complex scientific problems.

1 Introduction

Large language models (LLMs), while increasingly competent at mathematical reasoning, still face challenges with intricate multi-step derivations. To bridge this gap, a variety of test-time scaling

*Equal contribution.

Machine Learning and the Physical Sciences Workshop, NeurIPS 2025.

techniques have been developed to augment their reasoning capabilities. These techniques, including repeated sampling and self-consistency [4], sampling-plus-verifier pipelines [25, 12], multi-round reflection [20, 13], hierarchical decomposition [19, 16, 23], and RL-based planners [8, 14, 10], have shown improvement on standard mathematical benchmarks [9, 6]. This approach presents a distinct computational paradigm from scaling efforts focused on pretraining or extensive finetuning. While test-time compute strategies require more resources than a single inference pass, they can offer a more compute-efficient path to performance gains on complex tasks, challenging traditional scaling laws that often optimize for training costs over inference-time expenditures [17, 3].

Much of the progress in test-time scaling has been developed for mathematical settings. (Detailed literature survey in Appendix A) However, theoretical physics reasoning presents distinct challenges compared to undergraduate mathematical level reasoning, such as AIME [2], research level mathematical reasoning such as FrontierMath [7], and proof assistant-based mathematics, where results can be auto-verified [1]. For a comparison between the use of math in theoretical physics and in pure math, see [5]. In short, theoretical physics uses direct calculation techniques on mathematical concepts that are not as formally defined as in mathematics.

In this work, we systematically evaluate prominent test-time scaling methods on the TPBench dataset, a collection of advanced physics problems ranging from undergraduate to research level [5]. We find that simple parallel or sequential test time scaling techniques do not perform very well on this dataset. To improve their performance, we develop a novel weak-verifier framework that uses a SymPy-augmented agent to verify the mathematical steps within each candidate solution, providing a robust mechanism for identifying correct reasoning paths. Our contributions are:

- A comprehensive empirical study of existing test-time scaling techniques, including sampling-based search and multi-round reasoning, on the TPBench theoretical physics dataset. Our analysis investigates how the effectiveness of these methods compares between physics and mathematical reasoning domains.
- A novel step-wise weak-verifier pipeline that integrates symbolic computation (SymPy) to grade and select solutions. Our results show this pipeline surpasses other test-time scaling methods by a significant margin on TPBench, achieving up to a 22% improvement and approaching the theoretical Best-of-N performance. We also test our verifier on the AIME benchmark [2], confirming its general effectiveness.

Our work highlights the importance of tailoring inference-time strategies to specific scientific domains and demonstrates the power of symbolic verification in various domains.

2 Methodology

We study two families of Test-Time Scaling strategies under a single, unified pipeline: a *sequential* multi-round reasoning process and several *parallel* sampling-based processes. Regardless of strategy, each run ends by (i) synthesizing a final symbolic mathematical expression, (ii) translating it into executable Python that conforms to TPBench's auto-verification interface, and (iii) checking correctness with our evaluation pipeline. Prompts and agent instructions are listed in Appendix B.

Sequential: Multi-Round Reasoning This strategy employs an iterative process to solve problems. In each round, the model refines its reasoning, building upon its previous line of thought to progressively develop a solution. After a set number of iterations, the accumulated reasoning is synthesized into a final symbolic mathematical expression and then translated into executable Python code for verification. Detailed algorithm can be found at 1 in Appendix C.1.

Parallel: Majority Vote & Best of N We explore several parallel strategies that begin by generating 50 candidate solutions.

1. **Majority Vote** Each solution is evaluated numerically, and the final answer is determined by selecting the numerical result that appears most frequently among the candidates.
2. **Best of N** As an upper bound on what sampling alone could achieve, we mark a problem as solved if *any* of the 50 candidates passes the TPBench auto-verification step. This does not produce a deployable selector but serves as a ceiling for sampling without structured verification.

Parallel: Sampling-Based Search with Weak Verifiers We implement a two-stage selection framework with optional tie-breaking (Algorithm 2).

Stage 1 — Functional de-duplication. From the $N = 50$ initial candidates $\mathcal{S}$, we construct a pruned set $\mathcal{S}'$ of *functionally distinct* solutions by checking their numerical outputs on test-cases. This step reduces superficial diversity and focuses the verifier on genuinely different behaviors.

Stage 2 — Verification and scoring. For each $s_i \in \mathcal{S}'$, we compute an average verification score $\bar{V}_i$ over k_{verif} repetitions (here k_{verif}=10), using one of two verifiers: We implemented two verifier strategies:

1. **Simple Weak Verifier:** The language model is prompted to judge whether s_i solves the problem; each repetition yields a binary decision that we average.
2. **Symbolic Verifier Augmented Strategy:** A more advanced agent validates s_i step-by-step. For every symbolic calculation identified in the derivation, the agent generates and executes a SymPy script and compares results against the claimed step, returning a binary outcome per repetition.

For both strategies, the solution with the highest average score is selected. If multiple solutions are tied for the top score, a final pairwise comparison is performed by the language model to break the tie. The implementation of the symbolic verifier uses an agent framework equipped with a tool to execute SymPy scripts. The detail description is in Appendix C.2.

We also briefly explored tree-based search methods but did not observe significant improvements over the parallel and sequential strategies presented here. A more thorough investigation is therefore left for future work.

3 Experiments

3.1 Experiment Setup

While our primary focus is TP, we also evaluate our method on a widely-used math benchmark. [2]

TPBench. We use problems from levels 3 to 5—graduate and research-level—to test our hypothesis. These levels contain 11, 14, and 11 problems, respectively.

AIME. As AIME problem numbers roughly reflect difficulty, we select problems 11–15 from both AIME 2024 and 2025, forming a 20-problem set.

3.2 Experimental Results

We benchmark methods on TPBench levels 3–5 (difficulties 1–2 are too easy for the Gemini family). Table 1 summarizes the main comparison across models. In brief:

1. LLM self-grading, implemented as a Simple Weak Verifier, does not improve results over the average single-attempt accuracy. This failure mainly originates from low precision in the grading procedure, which is in line with findings that LLMs are not good at spotting their own mistakes [11, 18].
2. Majority vote is generally a good parallel baseline for selecting the correct answer from a large pool.
3. Our SymPy-augmented weak verifier yields large gains and approaches the best-of-N upper bound. For example, with Gemini 2.5 Pro on Level 5 problems, we go from a 29.3% average accuracy to 54.5%, approaching the best-of-N upper bound of 63.6%
4. Sequential scaling only leads to minor improvements, and in some cases even to a decrease in performance, with no clear benefit from additional reasoning rounds.

[2] Due to limited compute and a daily cap on Gemini API calls, we could not scale to larger datasets. Although other physics benchmarks exist, TPBench remains, to our knowledge, the only one featuring research-level symbolic problems. For example, most physics questions in HLE [15] are unsuitable for our symbolic verification pipeline.

Table 1: Comparison of test-time scaling approaches on TPBench with Gemini 2.5 Pro, Gemini 2.0 Flash, and o4-mini-high.

	Gemini 2.5 Pro		Gemini 2.0 Flash			o4-mini-high	
Method	Level 4	Level 5	Level 3	Level 4	Level 5	Level 4	Level 5
Baseline							
Single Attempt	63.3%	29.3%	52.5%	13.1%	1.5%	53.0%	21.1%
Sequential Methods							
1+Round Reasoning	65.0%	26.4%	61.8%	16.4%	2.7%	50.0%	19.1%
2+Round Reasoning	65.0%	26.4%	60.0%	22.9%	0.9%	52.9%	18.2%
4+Round Reasoning	68.6%	28.2%	60.0%	21.4%	1.8%	52.9%	20.0%
Parallel Methods							
Simple Weak Verifier	71.4%	27.3%	18.2%	7.1%	0%	64.3%	18.2%
Majority Vote	**78.6%**	36.4%	72.7%	35.7%	**9.1%**	**71.4%**	27.3%
SymPy Verifier	71.4%	**54.5%**	**81.8%**	**57.1%**	**9.1%**	**71.4%**	**36.4%**
Best of N (Oracle)	85.7%	63.6%	90.9%	85.7%	18.2%	78.6%	54.5%

Note: Single attempts' accuracy is averaged over 50 attempts; multi-round reasoning over 10 attempts; parallel methods use 50 candidates. 'Best of N' is an oracle measure and is excluded from direct comparison.

We compare the same methods on our subset of AIME problems in Table 2. We find similar improvements to those in the TPBench case. However, on this dataset, the SymPy verifier's performance is identical to that of Majority Vote for both models. This could indicate that symbolic verification is more critical in graduate-level problems, which often contain more complicated symbolic calculations compared to the AIME dataset, which focuses more on the complexity of reasoning.

Table 2: Comparison of test-time scaling approaches on AIME-Subset

Model	Gemini-2.0	Gemini-2.5
Baseline		
Single Attempt	11.4%	71.6%
Sequential Methods		
1+Round Reasoning	8.5%	73.0%
2+Round Reasoning	8.5%	73.0%
4+Round Reasoning	8.5%	73.0%
Parallel Methods		
Simple Weak Verifier	20%	60%
Majority Vote	15%	**80%**
SymPy Verifier	**25%**	**80%**
Best of N (Oracle)	35%	100%

Answer Distribution Analysis To gauge the potential of sampling-based approaches, we analyzed distributions from $N = 50$ attempts per Level 5 problem. We observe substantial variation in both unique solutions and correct hits, with frequent redundancy among samples. This motivates filtering for distinct solutions and a verifier to locate correct ones within the set. The full figure and detailed discussion are provided in Appendix D, Fig. 2.

Performance of the Symbolic Verifier Our SymPy-augmented weak verifier substantially improves accuracy by grounding checks in symbolic computation. This pipeline performs a step-by-step verification, meaning that gains in performance come both from re-visiting individual steps with an LLM and from writing SymPy code for individual steps where possible. However, it is not an oracle and has scope limits (e.g., general tensor calculus and path integrals). Operational statistics, capability tables, and case studies are in Appendix E, Tables 3 and 4, with additional examples in AppendixF and G.

3.3 Method Scaling Analysis

Scaling of Parallel Methods To investigate the scalability of parallel test-time scaling approaches, we analyzed the performance of Majority Vote and our SymPy-augmented weak-verifier framework as a function of the number of generated candidate solutions (N). Figure 1 illustrates the accuracy achieved by these two methods, across different problem difficulty levels (Level 3, 4, and 5) from TPBench, using Gemini 2.0 Flash as the base model.

The results show a consistent trend for both evaluated methods: accuracy improves with an increasing number of attempts for all difficulty levels. The SymPy-augmented verifier consistently outperforms Majority Vote across all tested N values, and saturates at a higher value. It appears that with 50 samples, we are approaching a saturation point, indicating a limit on the maximum performance one can reach with these models and approaches. This aligns with the finding from Wu et al. that shows accuracy from inference flops, the majority voting and weighted majority voting methods follows a curve that saturates. [22]

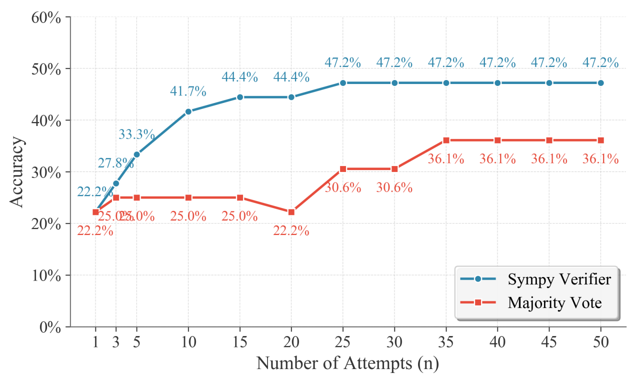

Figure 1: Scaling of accuracy with the number of parallel attempts (N) for Majority Vote and our proposed SymPy-augmented weak-verifier framework. Data is from TPBench problems of difficulty levels 3, 4, and 5, using Gemini 2.0 Flash.

Scaling of Sequential Method We find that sequential scaling only leads to a minor improvement of model performance, and in some cases even to a decrease in performance. Further, there is no clear improvement with additional rounds of reasoning. Our method, loosely inspired by the s1 approach [13], does not work well for this dataset. We plan to examine sequential scaling in more detail in the future, with full plots and a token-cost discussion in Appendix H, Fig. 3.

4 Discussion & Conclusion

In this paper, we address the challenge of enhancing the reasoning capabilities of large language models (LLMs) for advanced theoretical physics problems, particularly where existing test-time scaling techniques have shown limited success. We conducted a comprehensive empirical study of sampling-based parallel scaling and multi-round reasoning on the TPBench, a benchmark for advanced physics problems up to the research level, as well as on AIME. In general, we find that parallel methods work well, in particular when augmented with a step-wise symbolic verification procedure. In future work, we aim to build stronger domain-specific verification tools, for example, for tensor operations in general relativity. It would also be important to use symbolic tools at the time of solution generation rather than only for verification. For sequential scaling, we do not find improvements on hard problems. Improving sequential scaling on theoretical physics problems is a further interesting direction for future research. Finally, parallel and sequential techniques could be combined in compute-optimal ways.

Acknowledgements

M.M. and D.J.H.C. acknowledge the support by the U.S. Department of Energy, Office of Science, Office of High Energy Physics under Award Number DE-SC0017647. M.M. also acknowledges the support by the National Science Foundation (NSF) under Grant Number 2307109 and the Wisconsin Alumni Research Foundation (WARF). F.S. is grateful for the support of the NSF under CCF2106707 and the Wisconsin Alumni Research Foundation (WARF).

References

[1] AlphaProof and AlphaGeometry TEams. Ai achieves silver-medal standard solving international mathematical olympiad problems, Jul 2024.

[2] Art of Problem Solving. Aime problems and solutions. `https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions`. Accessed: 2025-06-19.

[3] Song Bian, Minghao Yan, and Shivaram Venkataraman. Scaling inference-efficient language models, 2025.

[4] Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling, 2024.

[5] Daniel J. H. Chung, Zhiqi Gao, Yurii Kvasiuk, Tianyi Li, Moritz Münchmeyer, Maja Rudolph, Frederic Sala, and Sai Chaitanya Tadepalli. Theoretical physics benchmark (tpbench) – a dataset and study of ai reasoning capabilities in theoretical physics, 2025.

[6] 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. Training verifiers to solve math word problems, 2021.

[7] Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, Caroline Falkman Olsson, Jean-Stanislas Denain, Anson Ho, Emily de Oliveira Santos, Olli Järviniemi, Matthew Barnett, Robert Sandler, Matej Vrzala, Jaime Sevilla, Qiuyu Ren, Elizabeth Pratt, Lionel Levine, Grant Barkley, Natalie Stewart, Bogdan Grechuk, Tetiana Grechuk, Shreepranav Varma Enugandla, and Mark Wildon. Frontiermath: A benchmark for evaluating advanced mathematical reasoning in ai, 2024.

[8] Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. rstar-math: Small llms can master math reasoning with self-evolved deep thinking, 2025.

[9] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021.

[10] Zhenyu Hou, Xin Lv, Rui Lu, Jiajie Zhang, Yujiang Li, Zijun Yao, Juanzi Li, Jie Tang, and Yuxiao Dong. Advancing language model reasoning through reinforcement learning and inference scaling, 2025.

[11] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet, 2024.

[12] Runze Liu, Junqi Gao, Jian Zhao, Kaiyan Zhang, Xiu Li, Biqing Qi, Wanli Ouyang, and Bowen Zhou. Can 1b llm surpass 405b llm? rethinking compute-optimal test-time scaling, 2025.

[13] 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, 2025.

[14] Jianfeng Pan, Senyou Deng, and Shaomang Huang. Coat: Chain-of-associated-thoughts framework for enhancing large language models reasoning, 2025.

[15] Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, Michael Choi, Anish Agrawal, Arnav Chopra, Adam Khoja, Ryan Kim, Richard Ren, Jason Hausenloy, Oliver Zhang, Mantas Mazeika, Dmitry Dodonov, Tung Nguyen, Jaeho Lee, Daron Anderson, Mikhail Doroshenko, Alun Cennyth Stokes, Mobeen Mahmood, Oleksandr Pokutnyi, Oleg Iskra, Jessica P. Wang, John-Clark Levin, Mstyslav Kazakov, Fiona Feng, Steven Y. Feng, Haoran Zhao, Michael Yu, Varun Gangal, Chelsea Zou, Zihan Wang, Serguei Popov, Robert Gerbicz, Geoff Galgon, Johannes Schmitt, Will Yeadon, Yongki Lee, Scott Sauers, Alvaro Sanchez, Fabian Giska, Marc Roth, Søren Riis, Saiteja Utpala, Noah Burns, Gashaw M. Goshu, Mohinder Maheshbhai Naiya, Chidozie Agu, Zachary Giboney, Antrell Cheatom, Francesco Fournier-Facio, Sarah-Jane Crowson, Lennart Finke, Zerui Cheng, Jennifer Zampese, Ryan G. Hoerr, Mark Nandor, Hyunwoo Park, Tim Gehrunger, Jiaqi Cai, Ben McCarty, Alexis C Garretson, Edwin Taylor, Damien Sileo, Qiuyu Ren, Usman Qazi, Lianghui Li, Jungbae Nam, John B. Wydallis, Pavel Arkhipov, Jack Wei Lun Shi, Aras Bacho, Chris G. Willcocks, Hangrui Cao, Sumeet Motwani, Emily de Oliveira Santos, Johannes Veith, Edward Vendrow, Doru Cojoc, Kengo Zenitani, Joshua Robinson, Longke Tang, Yuqi Li, Joshua Vendrow, Natanael Wildner Fraga, Vladyslav Kuchkin, Andrey Pupasov Maksimov, Pierre Marion, Denis Efremov, Jayson Lynch, Kaiqu Liang, Aleksandar Mikov, Andrew Gritsevskiy, Julien Guillod, Gözdenur Demir, Dakotah Martinez, Ben Pageler, Kevin Zhou, Saeed Soori, Ori Press, Henry Tang, Paolo Rissone, Sean R. Green, Lina Brüssel, Moon Twayana, Aymeric Dieuleveut, Joseph Marvin Imperial, Ameya Prabhu, Jinzhou Yang, Nick Crispino, Arun Rao, Dimitri Zvonkine, Gabriel Loiseau, Mikhail Kalinin, Marco Lukas, Ciprian Manolescu, Nate Stambaugh, Subrata Mishra, Tad Hogg, Carlo Bosio, Brian P Coppola, Julian Salazar, Jaehyeok Jin, Rafael Sayous, Stefan Ivanov, Philippe Schwaller, Shaipranesh Senthilkuma, Andres M Bran, Andres Algaba, Kelsey Van den Houte, Lynn Van Der Sypt, Brecht Verbeken, David Noever, Alexei Kopylov, Benjamin Myklebust, Bikun Li, Lisa Schut, Evgenii Zheltonozhskii, Qiaochu Yuan, Derek Lim, Richard Stanley, Tong Yang, John Maar, Julian Wykowski, Martí Oller, Anmol Sahu, Cesare Giulio Ardito, Yuzheng Hu, Ariel Ghislain Kemogne Kamdoum, Alvin Jin, Tobias Garcia Vilchis, Yuexuan Zu, Martin Lackner, James Koppel, Gongbo Sun, Daniil S. Antonenko, Steffi Chern, Bingchen Zhao, Pierrot Arsene, Joseph M Cavanagh, Daofeng Li, Jiawei Shen, Donato Crisostomi, Wenjin Zhang, Ali Dehghan, Sergey Ivanov, David Perrella, Nurdin Kaparov, Allen Zang, Ilia Sucholutsky, Arina Kharlamova, Daniil Orel, Vladislav Poritski, Shalev Ben-David, Zachary Berger, Parker Whitfill, Michael Foster, Daniel Munro, Linh Ho, Shankar Sivarajan, Dan Bar Hava, Aleksey Kuchkin, David Holmes, Alexandra Rodriguez-Romero, Frank Sommerhage, Anji Zhang, Richard Moat, Keith Schneider, Zakayo Kazibwe, Don Clarke, Dae Hyun Kim, Felipe Meneguitti Dias, Sara Fish, Veit Elser, Tobias Kreiman, Victor Efren Guadarrama Vilchis, Immo Klose, Ujjwala Anantheswaran, Adam Zweiger, Kaivalya Rawal, Jeffery Li, Jeremy Nguyen, Nicolas Daans, Haline Heidinger, Maksim Radionov, Václav Rozhoň, Vincent Ginis, Christian Stump, Niv Cohen, Rafał Poświata, Josef Tkadlec, Alan Goldfarb, Chenguang Wang, Piotr Padlewski, Stanislaw Barzowski, Kyle Montgomery, Ryan Stendall, Jamie Tucker-Foltz, Jack Stade, T. Ryan Rogers, Tom Goertzen, Declan Grabb, Abhishek Shukla, Alan Givré, John Arnold Ambay, Archan Sen, Muhammad Fayez Aziz, Mark H Inlow, Hao He, Ling Zhang, Younesse Kaddar, Ivar Ängquist, Yanxu Chen, Harrison K Wang, Kalyan Ramakrishnan, Elliott Thornley, Antonio Terpin, Hailey Schoelkopf, Eric Zheng, Avishy Carmi, Ethan D. L. Brown, Kelin Zhu, Max Bartolo, Richard Wheeler, Martin Stehberger, Peter Bradshaw, JP Heimonen, Kaustubh Sridhar, Ido Akov, Jennifer Sandlin, Yury Makarychev, Joanna Tam, Hieu Hoang, David M. Cunningham, Vladimir Goryachev, Demosthenes Patramanis, Michael Krause, Andrew Redenti, David Aldous, Jesyin Lai, Shannon Coleman, Jiangnan Xu, Sangwon Lee, Ilias Magoulas, Sandy Zhao, Ning Tang, Michael K. Cohen, Orr Paradise, Jan Hendrik Kirchner, Maksym Ovchynnikov, Jason O. Matos, Adithya Shenoy, Michael Wang, Yuzhou Nie, Anna Sztyber-Betley, Paolo Faraboschi, Robin Riblet, Jonathan Crozier, Shiv Halasyamani, Shreyas Verma, Prashant Joshi, Eli Meril, Ziqiao Ma, Jérémy Andréoletti, Raghav Singhal, Jacob Platnick, Volodymyr Nevirkovets, Luke Basler, Alexander Ivanov, Seri Khoury, Nils Gustafsson, Marco Piccardo, Hamid Mostaghimi, Qijia Chen, Virendra Singh, Tran Quoc Khánh, Paul Rosu, Hannah Szlyk, Zachary Brown, Himanshu Narayan, Aline Menezes, Jonathan Roberts, William Alley, Kunyang Sun, Arkil Patel, Max Lamparth, Anka Reuel, Linwei Xin, Hanmeng Xu, Jacob Loader, Freddie Martin, Zixuan Wang, Andrea Achilleos, Thomas Preu, Tomek Korbak, Ida Bosio, Fereshteh Kazemi, Ziye Chen, Biró Bálint, Eve J. Y. Lo, Jiaqi Wang, Maria Inês S. Nunes, Jeremiah Milbauer, M Saiful Bari, Zihao Wang, Behzad Ansarinejad, Yewen Sun, Stephane Durand,

Hossam Elgnainy, Guillaume Douville, Daniel Tordera, George Balabanian, Hew Wolff, Lynna Kvistad, Hsiaoyun Milliron, Ahmad Sakor, Murat Eron, Andrew Favre D. O., Shailesh Shah, Xiaoxiang Zhou, Firuz Kamalov, Sherwin Abdoli, Tim Santens, Shaul Barkan, Allison Tee, Robin Zhang, Alessandro Tomasiello, G. Bruno De Luca, Shi-Zhuo Looi, Vinh-Kha Le, Noam Kolt, Jiayi Pan, Emma Rodman, Jacob Drori, Carl J Fossum, Niklas Muennighoff, Milind Jagota, Ronak Pradeep, Honglu Fan, Jonathan Eicher, Michael Chen, Kushal Thaman, William Merrill, Moritz Firsching, Carter Harris, Stefan Ciobâcă, Jason Gross, Rohan Pandey, Ilya Gusev, Adam Jones, Shashank Agnihotri, Pavel Zhelnov, Mohammadreza Mofayezi, Alexander Piperski, David K. Zhang, Kostiantyn Dobarskyi, Roman Leventov, Ignat Soroko, Joshua Duersch, Vage Taamazyan, Andrew Ho, Wenjie Ma, William Held, Ruicheng Xian, Armel Randy Zebaze, Mohanad Mohamed, Julian Noah Leser, Michelle X Yuan, Laila Yacar, Johannes Lengler, Katarzyna Olszewska, Claudio Di Fratta, Edson Oliveira, Joseph W. Jackson, Andy Zou, Muthu Chidambaram, Timothy Manik, Hector Haffenden, Dashiell Stander, Ali Dasouqi, Alexander Shen, Bita Golshani, David Stap, Egor Kretov, Mikalai Uzhou, Alina Borisovna Zhidkovskaya, Nick Winter, Miguel Orbegozo Rodriguez, Robert Lauff, Dustin Wehr, Colin Tang, Zaki Hossain, Shaun Phillips, Fortuna Samuele, Fredrik Ekström, Angela Hammon, Oam Patel, Faraz Farhidi, George Medley, Forough Mohammadzadeh, Madellene Peñaflor, Haile Kassahun, Alena Friedrich, Rayner Hernandez Perez, Daniel Pyda, Taom Sakal, Omkar Dhamane, Ali Khajegili Mirabadi, Eric Hallman, Kenchi Okutsu, Mike Battaglia, Mohammad Maghsoudimehrabani, Alon Amit, Dave Hulbert, Roberto Pereira, Simon Weber, Handoko, Anton Peristyy, Stephen Malina, Mustafa Mehkary, Rami Aly, Frank Reidegeld, Anna-Katharina Dick, Cary Friday, Mukhwinder Singh, Hassan Shapourian, Wanyoung Kim, Mariana Costa, Hubeyb Gurdogan, Harsh Kumar, Chiara Ceconello, Chao Zhuang, Haon Park, Micah Carroll, Andrew R. Tawfeek, Stefan Steinerberger, Daattavya Aggarwal, Michael Kirchhof, Linjie Dai, Evan Kim, Johan Ferret, Jainam Shah, Yuzhou Wang, Minghao Yan, Krzysztof Burdzy, Lixin Zhang, Antonio Franca, Diana T. Pham, Kang Yong Loh, Joshua Robinson, Abram Jackson, Paolo Giordano, Philipp Petersen, Adrian Cosma, Jesus Colino, Colin White, Jacob Votava, Vladimir Vinnikov, Ethan Delaney, Petr Spelda, Vit Stritecky, Syed M. Shahid, Jean-Christophe Mourrat, Lavr Vetoshkin, Koen Sponselee, Renas Bacho, Zheng-Xin Yong, Florencia de la Rosa, Nathan Cho, Xiuyu Li, Guillaume Malod, Orion Weller, Guglielmo Albani, Leon Lang, Julien Laurendeau, Dmitry Kazakov, Fatimah Adesanya, Julien Portier, Lawrence Hollom, Victor Souza, Yuchen Anna Zhou, Julien Degorre, Yiğit Yalın, Gbenga Daniel Obikoya, Rai, Filippo Bigi, M. C. Boscá, Oleg Shumar, Kaniuar Bacho, Gabriel Recchia, Mara Popescu, Nikita Shulga, Ngefor Mildred Tanwie, Thomas C. H. Lux, Ben Rank, Colin Ni, Matthew Brooks, Alesia Yakimchyk, Huanxu, Liu, Stefano Cavalleri, Olle Häggström, Emil Verkama, Joshua Newbould, Hans Gundlach, Leonor Brito-Santana, Brian Amaro, Vivek Vajipey, Rynaa Grover, Ting Wang, Yosi Kratish, Wen-Ding Li, Sivakanth Gopi, Andrea Caciolai, Christian Schroeder de Witt, Pablo Hernández-Cámara, Emanuele Rodolà, Jules Robins, Dominic Williamson, Vincent Cheng, Brad Raynor, Hao Qi, Ben Segev, Jingxuan Fan, Sarah Martinson, Erik Y. Wang, Kaylie Hausknecht, Michael P. Brenner, Mao Mao, Christoph Demian, Peyman Kassani, Xinyu Zhang, David Avagian, Eshawn Jessica Scipio, Alon Ragoler, Justin Tan, Blake Sims, Rebeka Plecnik, Aaron Kirtland, Omer Faruk Bodur, D. P. Shinde, Yan Carlos Leyva Labrador, Zahra Adoul, Mohamed Zekry, Ali Karakoc, Tania C. B. Santos, Samir Shamseldeen, Loukmane Karim, Anna Liakhovitskaia, Nate Resman, Nicholas Farina, Juan Carlos Gonzalez, Gabe Maayan, Earth Anderson, Rodrigo De Oliveira Pena, Elizabeth Kelley, Hodjat Mariji, Rasoul Pouriamanesh, Wentao Wu, Ross Finocchio, Ismail Alarab, Joshua Cole, Danyelle Ferreira, Bryan Johnson, Mohammad Safdari, Liangti Dai, Siriphan Arthornthurasuk, Isaac C. McAlister, Alejandro José Moyano, Alexey Pronin, Jing Fan, Angel Ramirez-Trinidad, Yana Malysheva, Daphiny Pottmaier, Omid Taheri, Stanley Stepanic, Samuel Perry, Luke Askew, Raúl Adrián Huerta Rodríguez, Ali M. R. Minissi, Ricardo Lorena, Krishnamurthy Iyer, Arshad Anil Fasiludeen, Ronald Clark, Josh Ducey, Matheus Piza, Maja Somrak, Eric Vergo, Juehang Qin, Benjámin Borbás, Eric Chu, Jack Lindsey, Antoine Jallon, I. M. J. McInnis, Evan Chen, Avi Semler, Luk Gloor, Tej Shah, Marc Carauleanu, Pascal Lauer, Tran Đuc Huy, Hossein Shahrtash, Emilien Duc, Lukas Lewark, Assaf Brown, Samuel Albanie, Brian Weber, Warren S. Vaz, Pierre Clavier, Yiyang Fan, Gabriel Poesia Reis e Silva, Long, Lian, Marcus Abramovitch, Xi Jiang, Sandra Mendoza, Murat Islam, Juan Gonzalez, Vasilios Mavroudis, Justin Xu, Pawan Kumar, Laxman Prasad Goswami, Daniel Bugas, Nasser Heydari, Ferenc Jeanplong, Thorben Jansen, Antonella Pinto, Archimedes Apronti, Abdallah Galal, Ng Ze-An, Ankit Singh, Tong Jiang, Joan of Arc Xavier, Kanu Priya Agarwal, Mohammed Berkani, Gang Zhang, Zhehang

Du, Benedito Alves de Oliveira Junior, Dmitry Malishev, Nicolas Remy, Taylor D. Hartman, Tim Tarver, Stephen Mensah, Gautier Abou Loume, Wiktor Morak, Farzad Habibi, Sarah Hoback, Will Cai, Javier Gimenez, Roselynn Grace Montecillo, Jakub Łucki, Russell Campbell, Asankhaya Sharma, Khalida Meer, Shreen Gul, Daniel Espinosa Gonzalez, Xavier Alapont, Alex Hoover, Gunjan Chhablani, Freddie Vargus, Arunim Agarwal, Yibo Jiang, Deepakkumar Patil, David Outevsky, Kevin Joseph Scaria, Rajat Maheshwari, Abdelkader Dendane, Priti Shukla, Ashley Cartwright, Sergei Bogdanov, Niels Mündler, Sören Möller, Luca Arnaboldi, Kunvar Thaman, Muhammad Rehan Siddiqi, Prajvi Saxena, Himanshu Gupta, Tony Fruhauff, Glen Sherman, Mátyás Vincze, Siranut Usawasutsakorn, Dylan Ler, Anil Radhakrishnan, Innocent Enyekwe, Sk Md Salauddin, Jiang Muzhen, Aleksandr Maksapetyan, Vivien Rossbach, Chris Harjadi, Mohsen Bahaloohoreh, Claire Sparrow, Jasdeep Sidhu, Sam Ali, Song Bian, John Lai, Eric Singer, Justine Leon Uro, Greg Bateman, Mohamed Sayed, Ahmed Menshawy, Darling Duclosel, Dario Bezzi, Yashaswini Jain, Ashley Aaron, Murat Tiryakioglu, Sheeshram Siddh, Keith Krenek, Imad Ali Shah, Jun Jin, Scott Creighton, Denis Peskoff, Zienab EL-Wasif, Ragavendran P V, Michael Richmond, Joseph McGowan, Tejal Patwardhan, Hao-Yu Sun, Ting Sun, Nikola Zubić, Samuele Sala, Stephen Ebert, Jean Kaddour, Manuel Schottdorf, Dianzhuo Wang, Gerol Petruzella, Alex Meiburg, Tilen Medved, Ali ElSheikh, S Ashwin Hebbar, Lorenzo Vaquero, Xianjun Yang, Jason Poulos, Vilém Zouhar, Sergey Bogdanik, Mingfang Zhang, Jorge Sanz-Ros, David Anugraha, Yinwei Dai, Anh N. Nhu, Xue Wang, Ali Anil Demircali, Zhibai Jia, Yuyin Zhou, Juncheng Wu, Mike He, Nitin Chandok, Aarush Sinha, Gaoxiang Luo, Long Le, Mickaël Noyé, Michał Perełkiewicz, Ioannis Pantidis, Tianbo Qi, Soham Sachin Purohit, Letitia Parcalabescu, Thai-Hoa Nguyen, Genta Indra Winata, Edoardo M. Ponti, Hanchen Li, Kaustubh Dhole, Jongee Park, Dario Abbondanza, Yuanli Wang, Anupam Nayak, Diogo M. Caetano, Antonio A. W. L. Wong, Maria del Rio-Chanona, Dániel Kondor, Pieter Francois, Ed Chalstrey, Jakob Zsambok, Dan Hoyer, Jenny Reddish, Jakob Hauser, Francisco-Javier Rodrigo-Ginés, Suchandra Datta, Maxwell Shepherd, Thom Kamphuis, Qizheng Zhang, Hyunjun Kim, Ruiji Sun, Jianzhu Yao, Franck Dernoncourt, Satyapriya Krishna, Sina Rismanchian, Bonan Pu, Francesco Pinto, Yingheng Wang, Kumar Shridhar, Kalon J. Overholt, Glib Briia, Hieu Nguyen, David, Soler Bartomeu, Tony CY Pang, Adam Wecker, Yifan Xiong, Fanfei Li, Lukas S. Huber, Joshua Jaeger, Romano De Maddalena, Xing Han Lù, Yuhui Zhang, Claas Beger, Patrick Tser Jern Kon, Sean Li, Vivek Sanker, Ming Yin, Yihao Liang, Xinlu Zhang, Ankit Agrawal, Li S. Yifei, Zechen Zhang, Mu Cai, Yasin Sonmez, Costin Cozianu, Changhao Li, Alex Slen, Shoubin Yu, Hyun Kyu Park, Gabriele Sarti, Marcin Briański, Alessandro Stolfo, Truong An Nguyen, Mike Zhang, Yotam Perlitz, Jose Hernandez-Orallo, Runjia Li, Amin Shabani, Felix Juefei-Xu, Shikhar Dhingra, Orr Zohar, My Chiffon Nguyen, Alexander Pondaven, Abdurrahim Yilmaz, Xuandong Zhao, Chuanyang Jin, Muyan Jiang, Stefan Todoran, Xinyao Han, Jules Kreuer, Brian Rabern, Anna Plassart, Martino Maggetti, Luther Yap, Robert Geirhos, Jonathon Kean, Dingsu Wang, Sina Mollaei, Chenkai Sun, Yifan Yin, Shiqi Wang, Rui Li, Yaowen Chang, Anjiang Wei, Alice Bizeul, Xiaohan Wang, Alexandre Oliveira Arrais, Kushin Mukherjee, Jorge Chamorro-Padial, Jiachen Liu, Xingyu Qu, Junyi Guan, Adam Bouyamourn, Shuyu Wu, Martyna Plomecka, Junda Chen, Mengze Tang, Jiaqi Deng, Shreyas Subramanian, Haocheng Xi, Haoxuan Chen, Weizhi Zhang, Yinuo Ren, Haoqin Tu, Sejong Kim, Yushun Chen, Sara Vera Marjanović, Junwoo Ha, Grzegorz Luczyna, Jeff J. Ma, Zewen Shen, Dawn Song, Cedegao E. Zhang, Zhun Wang, Gaël Gendron, Yunze Xiao, Leo Smucker, Erica Weng, Kwok Hao Lee, Zhe Ye, Stefano Ermon, Ignacio D. Lopez-Miguel, Theo Knights, Anthony Gitter, Namkyu Park, Boyi Wei, Hongzheng Chen, Kunal Pai, Ahmed Elkhanany, Han Lin, Philipp D. Siedler, Jichao Fang, Ritwik Mishra, Károly Zsolnai-Fehér, Xilin Jiang, Shadab Khan, Jun Yuan, Rishab Kumar Jain, Xi Lin, Mike Peterson, Zhe Wang, Aditya Malusare, Maosen Tang, Isha Gupta, Ivan Fosin, Timothy Kang, Barbara Dworakowska, Kazuki Matsumoto, Guangyao Zheng, Gerben Sewuster, Jorge Pretel Villanueva, Ivan Rannev, Igor Chernyavsky, Jiale Chen, Deepayan Banik, Ben Racz, Wenchao Dong, Jianxin Wang, Laila Bashmal, Duarte V. Gonçalves, Wei Hu, Kaushik Bar, Ondrej Bohdal, Atharv Singh Patlan, Shehzaad Dhuliawala, Caroline Geirhos, Julien Wist, Yuval Kansal, Bingsen Chen, Kutay Tire, Atak Talay Yücel, Brandon Christof, Veerupaksh Singla, Zijian Song, Sanxing Chen, Jiaxin Ge, Kaustubh Ponkshe, Isaac Park, Tianneng Shi, Martin Q. Ma, Joshua Mak, Sherwin Lai, Antoine Moulin, Zhuo Cheng, Zhanda Zhu, Ziyi Zhang, Vaidehi Patil, Ketan Jha, Qiutong Men, Jiaxuan Wu, Tianchi Zhang, Bruno Hebling Vieira, Alham Fikri Aji, Jae-Won Chung, Mohammed Mahfoud, Ha Thi Hoang, Marc Sperzel, Wei Hao, Kristof Meding, Sihan Xu, Vassilis Kostakos, Davide Manini, Yueying Liu, Christopher Toukmaji, Jay Paek, Eunmi Yu, Arif Engin Demircali, Zhiyi Sun, Ivan Dewerpe, Hongsen

Qin, Roman Pflugfelder, James Bailey, Johnathan Morris, Ville Heilala, Sybille Rosset, Zishun Yu, Peter E. Chen, Woongyeong Yeo, Eeshaan Jain, Ryan Yang, Sreekar Chigurupati, Julia Chernyavsky, Sai Prajwal Reddy, Subhashini Venugopalan, Hunar Batra, Core Francisco Park, Hieu Tran, Guilherme Maximiano, Genghan Zhang, Yizhuo Liang, Hu Shiyu, Rongwu Xu, Rui Pan, Siddharth Suresh, Ziqi Liu, Samaksh Gulati, Songyang Zhang, Peter Turchin, Christopher W. Bartlett, Christopher R. Scotese, Phuong M. Cao, Aakaash Nattanmai, Gordon McKellips, Anish Cheraku, Asim Suhail, Ethan Luo, Marvin Deng, Jason Luo, Ashley Zhang, Kavin Jindel, Jay Paek, Kasper Halevy, Allen Baranov, Michael Liu, Advaith Avadhanam, David Zhang, Vincent Cheng, Brad Ma, Evan Fu, Liam Do, Joshua Lass, Hubert Yang, Surya Sunkari, Vishruth Bharath, Violet Ai, James Leung, Rishit Agrawal, Alan Zhou, Kevin Chen, Tejas Kalpathi, Ziqi Xu, Gavin Wang, Tyler Xiao, Erik Maung, Sam Lee, Ryan Yang, Roy Yue, Ben Zhao, Julia Yoon, Sunny Sun, Aryan Singh, Ethan Luo, Clark Peng, Tyler Osbey, Taozhi Wang, Daryl Echeazu, Hubert Yang, Timothy Wu, Spandan Patel, Vidhi Kulkarni, Vijaykaarti Sundarapandiyan, Ashley Zhang, Andrew Le, Zafir Nasim, Srikar Yalam, Ritesh Kasamsetty, Soham Samal, Hubert Yang, David Sun, Nihar Shah, Abhijeet Saha, Alex Zhang, Leon Nguyen, Laasya Nagumalli, Kaixin Wang, Alan Zhou, Aidan Wu, Jason Luo, Anwith Telluri, Summer Yue, Alexandr Wang, and Dan Hendrycks. Humanity's last exam, 2025.

[16] Toby Simonds and Akira Yoshiyama. Ladder: Self-improving llms through recursive problem decomposition, 2025.

[17] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024.

[18] Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks, 2024.

[19] Fengwei Teng, Zhaoyang Yu, Quan Shi, Jiayi Zhang, Chenglin Wu, and Yuyu Luo. Atom of thoughts for markov llm test-time scaling, 2025.

[20] Xiaoyu Tian, Sitong Zhao, Haotian Wang, Shuaiting Chen, Yunjie Ji, Yiping Peng, Han Zhao, and Xiangang Li. Think twice: Enhancing llm reasoning by scaling multi-round test-time thinking, 2025.

[21] Yue Wang, Qiuzhi Liu, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Linfeng Song, Dian Yu, Juntao Li, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Thoughts are all over the place: On the underthinking of o1-like llms, 2025.

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

[23] Ling Yang, Zhaochen Yu, Bin Cui, and Mengdi Wang. Reasonflux: Hierarchical llm reasoning via scaling thought templates, 2025.

[24] Xiao-Wen Yang, Xuan-Yi Zhu, Wen-Da Wei, Ding-Chu Zhang, Jie-Jing Shao, Zhi Zhou, Lan-Zhe Guo, and Yu-Feng Li. Step back to leap forward: Self-backtracking for boosting reasoning of language models, 2025.

[25] Eric Zhao, Pranjal Awasthi, and Sreenivas Gollapudi. Sample, scrutinize and scale: Effective inference-time search by scaling verification, 2025.

A Related Work

Test-Time Scaling for Reasoning. Large language models have recently shown that increasing inference-time computation—often called test-time scaling—can significantly boost reasoning performance [17, 3]. OpenAI's proprietary *o1* model exemplified this by using extended "slow thinking" procedures to achieve stronger results than standard one-shot prompting. This has spurred a wave of research into inference-time strategies that improve mathematical problem solving without changing model weights.

Sampling-Based Search and Verification. A common way to leverage more compute at test time is to sample multiple responses in parallel and choose a final answer by some criterion. This approach uses the randomness of LLMs. Brown et al. show that repeated sampling at large scales steadily increases the probability of including a correct solution at least once, boosting solve rates on code and formal proof tasks dramatically [4]. Zhao et al. extend this idea with a "sample, scrutinize and scale" method: generate hundreds of candidate solutions and use the LLM itself as a verifier to score and select the best one [25]. Liu et al. rethink the allocation of inference budget between a solution-proposing policy model and a process reward model, demonstrating that small models with optimal compute splits can rival or beat much larger ones on challenging math benchmarks [12].

Iterative and Multi-Round Reasoning. As a comparison to parallel sampling, sequential approaches such as iterative refinement let the model revisit its answer through multiple rounds. Tian et al. propose *Think Twice*, which feeds the model's initial answer back into itself for a re-answer step, yielding consistent gains on AIME and other math competition tasks [20]. Muennighoff et al. introduce *s1*, using "budget forcing" to extend chain-of-thought generation ("Wait" tokens) and fine-tuning on high-quality reasoning traces, force the answer to have a longer CoT and a better performance [13]. Wang et al. observe that models often underthink—prematurely abandoning reasoning paths—and propose a Thought Importance Penalty decoding tweak to encourage deeper exploration of each line of thought [21]. Yang et al. develop *Step Back to Leap Forward*, endowing the LLM with self-backtracking capabilities to undo and retry flawed steps to enhance mathematical reasoning capabilities. [24].

Hierarchical and Decompositional Strategies. Breaking complex problems into smaller subtasks is another effective paradigm. Teng et al. introduce *Atom of Thoughts* (AoT), treating reasoning as a Markov chain of atomic sub-questions that are solved independently and contracted back into the main problem, preventing distraction by irrelevant context [19]. Simonds and Yoshiyama's *LADDER* framework has models that recursively generate simpler variants of a hard problem, solve them, and use these solutions to tackle the original, achieving dramatic gains on integration calculus questions [16]. Yang et al.'s *ReasonFlux* equips a model with a library of high-level thought templates and trains a hierarchical planner via reinforcement learning to assemble them, pushing state-of-the-art results on MATH and AIME benchmarks [23].

Search-Based Planning and RL-Enhanced Inference. Several works combine LLMs with classical search or reinforcement learning at inference. Guan et al.'s *rStar-Math* pairs a policy model with a process reward model under a Monte Carlo Tree Search framework, enabling a 7B model to match or exceed much larger counterparts on MATH and AIME without distillation [8]. Pan et al.'s *CoAT* (Chain-of-Associated-Thoughts) uses MCTS with an associative memory mechanism for dynamic knowledge integration during the search, improving solution coherence and diversity [14]. Hou et al. train a 32B model via reinforcement learning from synthetic and human-feedback reasoning traces to encourage "slow thinking" behaviors, yielding smooth inference-scaling improvements without external verifiers [10].

TPBench. Theoretical Physics Benchmark (TPBench) [5] is a new benchmark dataset designed to evaluate AI's ability to solve high-level theoretical physics problems, specifically in high-energy theory and cosmology. This benchmark addresses the gap in existing datasets, which primarily focus on high-school-level physics, by including 57 novel problems ranging from undergraduate to research level. The paper also discusses the unique reasoning required for theoretical physics, contrasting it with mathematical reasoning, and highlights the potential of AI as a valuable tool for theoretical physicists by assisting with complex calculations and potentially contributing to the generation of new research ideas.

B All prompts used in this paper

B.1 Prompt for default generating method

```
Problem:
        {Problem Statement}

        IMPORTANT SOLUTION REQUIREMENTS:
        1. You MUST FIRST solve this problem using mathematical reasoning and symbolic calculations:
           - Use proper mathematical notation and symbols
           - Arrive at a final symbolic mathematical expression

        2. ONLY AFTER completing the mathematical solution:
           - Convert your final mathematical expression into Python code
           - The code must satisfy these requirements:
        {Specific Requirements for Python Code Answer}

        Code Format Requirements:
           1. Your solution MUST include the final executable Python code as required by the "Answer
     Requirements"
           2. You MUST wrap the final Python code between ```python and ``` tags
           3. Ensure the code is complete and can run independently
           4. The code should NOT contain ANY externally defined variables, including physical constants.
           5. The code MUST be concise and lightweight, faithfully and directly translating the final
     symbolic mathematical expression derived in step 1. Do NOT perform any further calculations or
     simplifications within the Python code itself.
           6. The code MUST NOT include any redundant elements such as conditional logic (if statements),
     exception handling (try/except blocks), or unnecessary checks.
```

B.2 Prompt for Multi-Round Reasoning

Initial Prompt

```
[Problem Statement]

Provide detailed reasoning to solve this problem step by step.
```

Subsequent Iteration Prompt

```
[General Contextual Instructions]

Original Problem:
[Problem Statement]

Summary from Previous Iteration's Reasoning:
[Previous Iteration's Conclusion]

Rethink the problem from here, then continue with detailed reasoning.
```

Summarization Prompt

```
Conclude the reasoning process above and arrive at 1 final symbolic mathematical expression.

AFTER concluding the reasoning process:
   - Convert the final mathematical expression into Python code
   - The code must satisfy these requirements:
[Specific Requirements for Python Code Answer]

Code Format Requirements:
1. Your solution MUST include the final executable Python code as required by the "Answer Requirements"
2. You MUST wrap the final Python code between ```python and ``` tags
3. Ensure the code is complete and can run independently
4. The code should NOT contain ANY externally defined variables, including physical constants.
5. The code MUST be concise and lightweight, faithfully and directly translating the final symbolic
     mathematical expression derived.
6. The code MUST NOT include any redundant elements such as conditional logic, exception handling, or
     unnecessary checks.
```

B.3 Prompt for Simple Weak Verifier

Verification Prompt (Randomly choose 1 of 3 below)

```
1:      Question: {question}. Answer Requirements: {answer_requirements}.
        Below is a student's solution to the above problem.
        Evaluate whether the student has correctly solved the problem and adheres to the answer
     requirements.
        Consider potential logical flaws, computational errors, or incorrect assumptions in the
     solution.
        Solution: {detailed_solution}.
        Please respond with a JSON object with only the following structure:
        {{ "is_solution_correct": 'yes' or 'no' }}

2:      Question: {question}. Answer Requirements: {answer_requirements}.
        Here is a student's detailed solution to the problem.
        Assess the solution by checking if all necessary steps are followed and if any significant
     errors were made.
        Your task is to determine whether the solution is valid and complete or if it contains major
     flaws.
        Solution: {detailed_solution}.
        Provide your response strictly in the following JSON format:
        {{ "is_solution_correct": 'yes' or 'no' }}

3:      Question: {question}. Answer Requirements: {answer_requirements}.
        I include below a student solution to the above question and answer requirement.
        Determine whether the student solution reaches the correct final answer in a correct fashion;
        e.g., whether the solution makes two major errors that still coincidentally cancel out.
        Please be careful and do not leap to conclusions without first reasoning them through.
        Solution: {detailed_solution}.
        Now summarize your response in a JSON format. Respond in the following format saying nothing
     else:
        {{ "is_solution_correct": 'yes' or 'no' }}
```

Tie-Breaking Prompt

```
  You are an expert physicist and AI assistant. Your task is to evaluate two attempted solutions to a
     given physics problem and determine which one is correct.

**Problem Statement:**
{problem_statement}

**Answer Requirement:**
{answer_requirement}

**Attempt 1:**
{attempt_1}

**Attempt 2:**
{attempt_2}

**Instructions:**

1.  **Analyze Problem and Requirements:** Carefully read the **Problem Statement** and the **Answer
     Requirement**. Understand what is being asked and what constraints or specific formats are
     required for the answer.
2.  **Evaluate Attempt 1:**
    * Assess the physical principles applied in Attempt 1. Are they appropriate for this problem?
    * Check the mathematical calculations and derivations. Are they correct?
    * Does the solution in Attempt 1 fully address the **Problem Statement**?
    * Does the final answer in Attempt 1 meet the **Answer Requirement** (e.g., units, significant
     figures, format)?
3.  **Evaluate Attempt 2:**
    * Assess the physical principles applied in Attempt 2. Are they appropriate for this problem?
    * Check the mathematical calculations and derivations. Are they correct?
    * Does the solution in Attempt 2 fully address the **Problem Statement**?
    * Does the final answer in Attempt 2 meet the **Answer Requirement** (e.g., units, significant
     figures, format)?
4.  **Determine Correct Solution:** Based on your evaluation, identify which attempt is correct.
5.  **Provide Justification:**
    * State clearly which attempt is correct (Attempt 1 or Attempt 2).
    * Provide a step-by-step explanation for why the chosen attempt is correct, referencing the
     physical principles, calculations, and adherence to the problem statement and answer requirements.
    * Explain why the other attempt is incorrect. Point out the specific errors in its application of
     principles, calculations, or its failure to meet the requirements. Be precise in your explanation
     of the flaws.

    Finally, Provide evaluation in compact JSON format with the following structure ONLY,
    remember to format your response using LaTeX notation for mathematical expressions, and follow the
     format for json

    {{"correct_attempt": "enter only number 1 or 2", "analysis": "Detailed explanation of the your
     justification"}}
```

```
Ensure the output is *only* the JSON object, without any introductory text or markdown formatting
  around it.
```

B.4 Prompt for SymPy-Augmented LLM Grading Agent

The following prompt is used to instruct the LLM-based grading agent, which is a key component of our physics-augmented weak verifier pipeline as described in Stage 2 of Section C.2. This agent is responsible for a step-by-step verification of candidate solutions, leveraging SymPy for validating mathematical calculations.

```
You are a grading assistant for theoretical physics problems. Your task is to grade a user-provided
    draft solution.

You need to carefully analyze every logical and computational step in the solution. During the analysis:
1.  Divide the entire mathematical derivation in the solution into sequentially numbered steps.
2.  Identify the key claims and formula usage within each step.
3.  For **each** numbered mathematical calculation step identified in step 1, you **must** construct a
    Python script containing SymPy code to perform that specific calculation. The script must print
    the final result to standard output.
4.  Use the available ‘run_sympy_script‘ tool to execute the Python (SymPy) script you constructed for
    each step.
5.  Carefully examine the Python script’s output (STDOUT) and errors (STDERR) for **each step**. If
    there is an error message (STDERR), rewrite the script for that step and execute it again. If the
    result in STDOUT does not match the one in the solution for that step, this indicates a
    calculation error in that specific step.
6.  Perform the analysis and verification for all steps *before* grading. Based on your step-by-step
    analysis (including the correctness of the logical flow, accuracy of conceptual understanding) and
    the calculation results verified by the SymPy tool for each step, provide the final score.
7.  Ignore any final Python code within the solution; focus only on the preceding mathematical
    derivations and calculations identified in step 1.

Ensure your Python (SymPy) scripts are specific to the calculation, complete, directly runnable, and
    print the final verification result to standard output.
Before calling the tool for a specific step, briefly explain which step number and what calculation you
    intend to verify using SymPy.

A general note on string values within the JSON output:
- For literal backslashes (e.g., in LaTeX commands like ‘\\alpha‘ or ‘\\frac‘), these must be escaped
    in the JSON string. So, ‘\\alpha‘ should be written as ‘\\\\alpha‘ in the JSON string value.
- For newline characters within a string (e.g., in multi-line script content), use ‘\\n‘.
- For double quotes within a string, use ‘\\\"‘.

Your final output **must** be a single JSON object. Do not add any text before or after the JSON object.
The JSON object should have the following structure:
{
  "sympy_verification": [
    // This is a list of objects, each corresponding to a mathematical calculation step that you verify
    using SymPy.
    // If no steps were verified using SymPy (e.g., if the solution has no calculable steps or all
    steps were skipped), this list can be empty.
    {
     "step_number": "integer: The sequence number of the calculation step you identified and
    verified.",
     "calculation_description": "string: A brief description of the calculation performed in this
    step.",
     "sympy_script_content": "string: The exact SymPy script you constructed and executed for this
    step.",
     "script_stdout": "string: The STDOUT captured from executing your SymPy script for this step.",
     "script_stderr": "string: The STDERR captured during execution (if any, otherwise empty string or
    null).",
     "is_correct": "boolean: true if the calculation in this step of the solution matches the SymPy
    output, false otherwise.",
     "error_explanation": "string: If \’is_correct\’ is false, provide a detailed explanation of the
    error found in the solution step based on SymPy verification. If \’is_correct\’ is true, this can
    be a brief confirmation or empty string."
    }
    // ... more objects for other verification steps
  ],
  "overall_score": "integer: Based on your comprehensive analysis, provide a numerical score of 0 or
    1.",
  "general_feedback": "string: An overall assessment of the solution, highlighting its strengths,
    weaknesses, and the reasoning behind your scoring. This should summarize the conclusions drawn
    from your step-by-step analysis."
}

Example of a sympy_verification entry:
```

```
{
  "step_number": 1,
  "calculation_description": "Derivative of z with respect to eta.",
  "sympy_script_content": "import sympy\\n# ... rest of the script ...\\nprint(result)",
  "script_stdout": "z*(H + epsilon_prime/(2*epsilon))",
  "script_stderr": "",
  "is_correct": true,
  "error_explanation": "The calculation for z\\' is correct."
}
```


C Detailed Algorithm in Methodology

C.1 Sequential: Multi-Round Reasoning

This methodology uses a multi-round reasoning strategy to iteratively refine the problem-solving process. This is repeated for a fixed number of iterations, allowing the model to build upon or reconsider its previous lines of thought. To limit context length in multi-round reasoning, we use a summarization technique at each iteration of the process.

The pseudo-code is shown below:

Algorithm 1 Multi-Round Reasoning and Solution Generation

Require: General contextual instructions (GCI), Problem statement (PS), Specific Python code requirements (SCR), Number of iterations (N_{iter})

1: PreviousThinking $\leftarrow$ Null
2: **for** $i \leftarrow 1$ **to** N_{iter} **do**
3:     **if** PreviousThinking is Null **then**
4:         Reasoning $\leftarrow$ LLM(GCI, PS)
5:     **else**
6:         Reasoning $\leftarrow$ LLM($GCI, PS,$ PreviousThinking)
7:     **end if**
8:     PreviousThinking $\leftarrow$ PreviousThinking $+$ Reasoning   ▷ Append new reasoning
9:     CodeAnswer $\leftarrow$ LLM(Reasoning, SCR)   ▷ Generate code for evaluation
10: **end for**

In each iteration, the core task is to generate step-by-step reasoning towards a solution. For the initial iteration, the language model is provided with general contextual instructions and the specific problem statement, guiding it to begin its reasoning using the initial prompt in Appendix B.2. For all subsequent iterations, the process encourages a re-evaluation or continuation of prior efforts. The prompt, which is the subsequent iteration prompt B.2, incorporates the original problem statement and general instructions, along with the summarization by the process above.

Following the generation of detailed reasoning, a separate step is taken to synthesize the results into a final symbolic mathematical expression and then translate that expression into executable Python code (using the Specific Python code requirements), which is needed for the TPBench auto-verification process (step 8 in algorithm 1). This synthesis is guided by a structured prompt that incorporates the accumulated reasoning and any specific constraints or formatting requirements for the Python code by the summarization prompt in B.2. Finally, we evaluate correctness using our pipeline and the generated Python code.

C.2 Parallel: Sampling-Based Search with Weak Verifiers

The sampling-based search methods we discuss adapt the general verification strategies proposed by Zhao *et al.* [25]. This common framework involves generating multiple candidate solutions and then employing a verifier to score and select the best one. We implement this framework using two distinct approaches for Stage 2: alongside a **simple weak verifier**, we introduce a **symbolic verifier augmented strategy**, specifically designed for physics problems, which leverages a SymPy-based agent. Thus, the core difference between our methods lies in the mechanism of the verifier used in Stage 2. Here are the details of this framework. The overall process is described in Algorithm 2.

Stage 1: Identification of Functionally Distinct Solutions. From the initial $N = 50$ candidate solutions, $\mathcal{S} = \{s_1, \ldots, s_{50}\}$ (generated as per 2), we first identify a subset of functionally distinct attempts. This process leverages an auto-verification framework where the final mathematical expression derived from each candidate solution s_i is translated into a programmatic function, denoted f_i. Each such function f_i is then systematically evaluated against a predefined suite of $M = 5$ distinct test case inputs $(x_1, \ldots, x_M)$. This evaluation yields an output vector $\mathbf{o}_i = (f_i(x_1), \ldots, f_i(x_M))$ for each solution s_i. Two solutions, s_i and s_j, are deemed functionally distinct if their respective output vectors $\mathbf{o}_i$ and $\mathbf{o}_j$ are not identical (i.e., $\mathbf{o}_i \neq \mathbf{o}_j$). Solutions that produce identical output vectors across all M test cases are considered functionally equivalent. By selecting one representative from each group of functionally equivalent solutions, we derive a pruned set $\mathcal{S}' \subseteq \mathcal{S}$ containing only functionally distinct candidates for subsequent evaluation stages. This set $\mathcal{S}'$ is used as the input S in Algorithm 2.

Stage 2: Solution Verification and Scoring. For every candidate solution $s_i \in \mathcal{S}'$, a verification score $\bar{V}_i$ is computed. This stage differentiates the two approaches:

Simple Weak Verifier Strategy. In this approach, the language model (LM) itself acts as a simple verifier. For each candidate solution s_i, the LM is queried k_{verif} times (here 10) with a prompt asking whether solution s_i is correct for the given problem Q. Each query j yields a binary score $V_{ij} \in \{0, 1\}$ (1 for correct, 0 for incorrect). The average score $\bar{V}_i = \frac{1}{k_{\text{verif}}} \sum_{j=1}^{k_{\text{verif}}} V_{ij}$ is then used.

Symbolic Verifier Augmented Strategy. This strategy employs a more sophisticated, physics-augmented LLM-based grading agent for verification. For every candidate solution $s_i \in \mathcal{S}'$, this specialized agent conducts a meticulous, step-by-step evaluation. A key feature is its capability for symbolic computation: for each identified mathematical calculation step within the solution's derivation, it automatically generates and executes a Python script leveraging the SymPy library to independently verify the computation. The agent then compares the result from SymPy with the corresponding step in the solution s_i. Based on this comprehensive verification, which encompasses both the logical flow and the correctness of individual calculations confirmed by SymPy, the agent assigns a binary score $V_{ij} \in \{0, 1\}$ to the solution s_i for each of the k_{verif} verification repetitions. The average verification score is $\bar{V}_i = \frac{1}{k_{\text{verif}}} \sum_{j=1}^{k_{\text{verif}}} V_{ij}$.

High-quality solutions are then retained in a set $\mathcal{S}_{\text{best}} = \left\{ s_i \in \mathcal{S}' \,\middle|\, \bar{V}_i \geq \max_{s_j \in \mathcal{S}'} \bar{V}_j - \delta \right\}$, where $\delta = 0.05$ is a tolerance parameter, particularly for the symbolic verifier strategy (for the simple verifier, δ can be set to 0 to select only top-scoring candidates or adjusted as needed).

Stage 3: Pairwise Tie-Breaking. If Stage 2 results in $|\mathcal{S}_{\text{best}}| > 1$ (i.e., multiple solutions are deemed high-quality or are tied), pairwise comparisons are performed as detailed in Algorithm 2, lines 14-20. For each unordered pair $(s_a, s_b) \subset \mathcal{S}_{\text{best}}$, the LLM is prompted k_{tie} times:

> `Given problem Q, which solution is more correct?`
> `A)` s_a `(plausibility` $H_p(s_a)$`)`
> `B)` s_b `(plausibility` $H_p(s_b)$`)`

The solution winning the majority of these k_{tie} comparisons for a given pair is favored in that matchup. The final answer $s^\star$ is the solution that wins the most matchups overall.

Implementation using Agent Framework (for Symbolic Verifier). The SymPy-augmented LLM verification process (used in Stage 2 for the Symbolic Verifier Augmented Strategy) is implemented using the `openai-agents` API. While our implementation relies on this particular framework, a number of other agent-based libraries could be used equivalently to achieve the same functionality. The system employs an LLM-based grading agent governed by a detailed set of instructions (provided in Appendix B.4) and equipped with a list of tools. The key tool for this verification is a custom Python function, `run_sympy_script`. This tool is designed to execute SymPy scripts: the agent identifies mathematical calculation steps within a candidate solution's derivation, formulates a corresponding SymPy script for each, and then invokes the `run_sympy_script` tool. The tool executes the provided script in a sandboxed environment, captures its standard output (STDOUT) and standard error (STDERR), and returns these to the agent. The agent subsequently compares this SymPy output with the assertion made in the solution's step to verify its correctness. This iterative, tool-mediated verification enables a meticulous, step-by-step assessment of the mathematical reasoning, with all

Algorithm 2 Sampling-Based Search with Weak Verifiers

Require: Language model LM, SymPy agent, problem Q, verification count k_{verif}, verification strategy VS

Generate 50 initial samples.
Select distinct solution set S from samples.
for each candidate response $s_i \in S$ **do**
    if VS = Simple **then**
        Score $\bar{V}_i \leftarrow$ fraction of "correct" votes from LM over k_{verif} queries.
    else if VS = Symbolic **then**
        Score $\bar{V}_i \leftarrow$ fraction of "correct" votes from SymPy agent over k_{verif} checks.
    end if
end for
$V_{\max} \leftarrow \max_{s_k \in S}\{\bar{V}_k\}$
$S_{\text{Best}} \leftarrow \{s_i \in S \mid \bar{V}_i = V_{\max}\}$ ▷ Gather all responses with the max score
if $|S_{\text{Best}}| = 1$ **then**
    return the single response in S_{Best}.
else ▷ Tie-breaking with pairwise comparison
    for each pair $(s_i, s_j) \in \binom{S_{\text{Best}}}{2}$ **do**
        $C_{i,j} \leftarrow LM(\text{Evaluate } s_i \text{ vs. } s_j \text{ for } Q)$
    end for
    return response from S_{Best} most frequently selected in all $C_{i,j}$.
end if

findings reported in a structured JSON format alongside an overall binary score for that verification run (V_{ij}), as detailed in Appendix B.4.

D Answer Distribution Analysis

To understand the potential of sampling-based approaches, we first analyzed the distribution of solutions generated by repeatedly sampling the model. For each Level 5 problem in our test set, we generated $N = 50$ attempts. Figure 2 illustrates the outcome of this initial sampling stage.

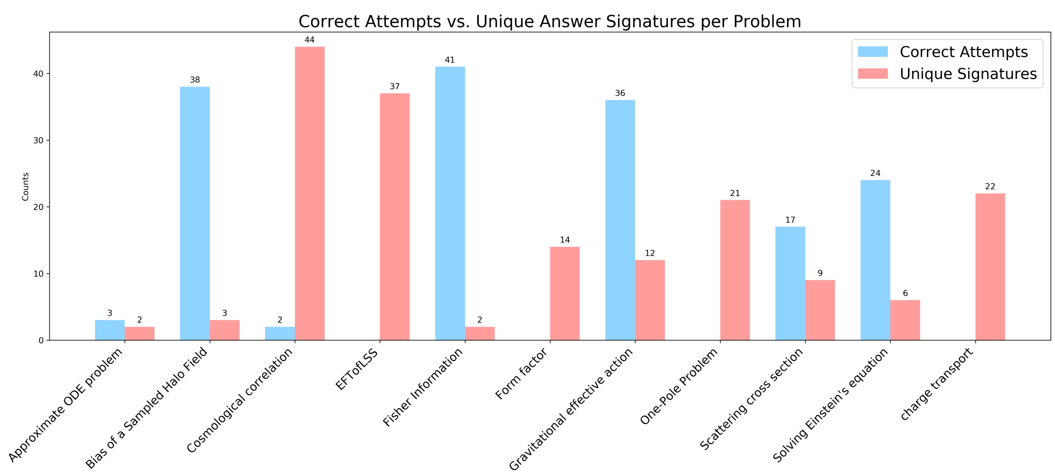

Figure 2: Analysis of 50 attempts per Level 5 TPBench problem by Gemini-2.5 Pro. The red bars show the total number of unique attempts (i.e., inequivalent answers) generated for each problem, while the blue bars indicate the number of correct attempts among those 50 samples. This visualization highlights the raw solution space from which our weak verifier selects.

As depicted in Figure 2, there is considerable variation across problems in both the number of unique solutions and the number of correct solutions found within the 50 attempts. For some problems, a significant fraction of the attempts are correct, providing a rich pool for a verifier. For others, correct solutions are sparse. Notably, the number of unique attempts (blue bars) is often less than 50, indicating that the model frequently generates identical or semantically equivalent incorrect (or correct) reasoning paths. This redundancy underscores the importance of filtering for distinct solutions, as proposed in our methodology.

The key insight from this initial analysis is that even with a fixed number of 50 samples, a correct solution is often present within the generated set. The challenge, which our weak verifier aims to address, is to effectively identify these correct instances from the typically larger set of unique (but not necessarily correct) attempts. The orange bars represent the best-case scenario for a perfect verifier operating on these 50 samples. Our approach aims to bridge the gap between raw sampling and this "oracle" best-of-N performance by leveraging domain-specific checks on the unique solutions.

E Performance of the Symbolic Verifier

While our SymPy-augmented weak verifier outperforms other methods, it is crucial to analyze its performance critically. The verifier's score represents an automated judgment, not an infallible ground truth. This section dissects the verifier's operational statistics, quantifies its accuracy against a manually graded baseline, and outlines its current capabilities and limitations, framing it as a promising but foundational step toward more robust automated scientific reasoning.

Verifier Usage Statistics. First, we examine how the symbolic verifier operates during the evaluation of candidate solutions. As shown in Table 3, the agent's reasoning process is heavily reliant on symbolic computation. On average, each candidate solution is broken down into approximately 8 distinct logical or mathematical steps. Of these, a remarkable 83% involve the generation and execution of a SymPy script, confirming that the agent actively uses its symbolic tool to validate calculations rather than relying solely on its internal knowledge. Furthermore, the tie-breaker stage was activated in 31% of the problems, indicating that it is common for multiple, functionally distinct solutions to receive a perfect score from the verifier, making the pairwise comparison a necessary final step for disambiguation.

Table 3: Operational statistics of the SymPy-augmented verifier on Level 4 and 5 problems. The data shows a deep reliance on symbolic checks and frequent use of the tie-breaking mechanism.

Metric	Value
Avg. verification steps per solution	8.2
Verification steps with SymPy script	83%
Agent-perceived correctness of steps	88%
Problems requiring tie-breaker	31%

Capabilities and Limitations. The verifier's strength lies in its ability to catch concrete algebraic and calculus errors. For instance, in the cosmological halo bias problem detailed in Appendix G, the verifier successfully identified errors in the calculation of truncated Gaussian moments. By symbolically re-computing the integrals, it flagged that the candidate's first moment calculation was incorrect by a factor of the variance σ, a subtle but critical mistake that invalidated the final result.

However, the verifier's capabilities are bound by the scope of its SymPy toolset. As summarized in Table 4, it struggles with mathematics that is highly abstract or lacks a straightforward algorithmic representation.

The agent cannot, for example, robustly verify steps involving tensor calculus in arbitrary spacetimes or abstract path integrals, as these concepts lack direct, general-purpose implementations in SymPy. This limitation explains some of the recall failures, where correct but advanced reasoning steps were flagged as unverifiable and thus penalized.

Our analysis shows that symbolic verification is a powerful and necessary tool for advancing AI in theoretical physics. It provides a substantial boost in identifying correct solutions by grounding the LLM's reasoning in formal computation. Nonetheless, its imperfections and the clear boundaries of its capabilities underscore that this is merely a first step. Future work must focus on expanding the

Table 4: A summary of the symbolic verifier's current capabilities for different types of mathematical operations.

Mathematical Operation	Verifiable?
Polynomial & Rational Function Algebra	Yes
Standard Differentiation & Integration	Yes
Residue Calculation at Poles	Yes
Approximations & Limit-taking	Partially
General Tensor Manipulations (GR)	No
Advanced Path Integrals (QFT)	No

verifier's toolkit and improving its ability to parse and validate more abstract forms of physical and mathematical reasoning.

F Case Study: SymPy-Enhanced Verification of the One-Pole Problem

This appendix provides a detailed analysis of how our SymPy-augmented LLM verification framework systematically identified a critical algebraic error in a candidate solution for the one-pole Bogoliubov coefficient calculation problem, a public level 5 TPBench problem that is generally not solved correctly even by the best models. While the symbolic verifier does identify a mistake, overall, its verification is less convincing than in the algebraically simpler case presented in App. G, indicating potential for future work on symbolic verification of mathematically difficult problems.

F.1 Problem Statement

The problem requires computing the Bogoliubov coefficient for a conformally coupled scalar field with the following specifications:

One-Pole Problem Statement

Consider the conformally coupled scalar field ϕ with Lagrangian density:

$$\mathcal{L} = \frac{1}{2}\left[g^{\mu\nu}\partial_\mu\phi\partial_\nu\phi - \left(m^2 - \frac{1}{6}R\right)\phi^2\right] \tag{1}$$

in curved spacetime

$$ds^2 = a^2(\eta)\left(d\eta^2 - |d\vec{x}|^2\right) \tag{2}$$

where the Ricci scalar is

$$R = -6\frac{a''(\eta)}{a(\eta)} \tag{3}$$

and a satisfies the differential equation

$$\frac{d}{dt}\ln a = \Theta(t_e - t)H_I + \Theta(t - t_e)\frac{H_I}{1 + \frac{3}{2}H_I(t - t_e)} \tag{4}$$

with t_e a finite positive number, the Θ function having the steplike behavior

$$\Theta(t - t_e) \equiv \begin{cases} 1 & t \geq t_e \\ 0 & \text{otherwise} \end{cases}, \tag{5}$$

and t being the comoving proper time related to η through

$$t = t_e + \int_{\eta_e}^{\eta} a(y)dy. \tag{6}$$

The boundary condition for the differential equation is $a|_{t=t_e} = a_e$.

Task: In the limit that $k/(a_e H_I) \to \infty$, using the steepest descent approximation starting from the dominant pole $\tilde{\eta}$ (with $\Re\tilde{\eta} > 0$) of the integrand factor $\omega_k'(\eta)/(2\omega_k(\eta))$, compute the Bogoliubov coefficient magnitude $|\beta(k)|$ approximated as

$$|\beta(k)| \approx \left| \int_{-\infty}^{\infty} d\eta \frac{\omega_k'(\eta)}{2\omega_k(\eta)} e^{-2i\int_{\eta_e}^{\eta} d\eta' \omega_k(\eta')} \right| \tag{7}$$

for particle production, where the dispersion relationship is given by

$$\omega_k^2(\eta) = k^2 + m^2 a^2(\eta) \tag{8}$$

with $0 < m \lesssim H_I$. Use a one-pole approximation, which dominates in this limit.

Answer Format: Provide the answer as a Python function:

```
def abs_beta(k: float, a_e: float, m: float, H_I: float) -> float:
    pass
```

F.2 Candidate Solution Overview

The candidate solution follows a systematic approach through the steepest descent method. We present here the key steps of the derivation that will be verified in detail. We note that the attempted solution is not very close to the correct expert solution which can be found in the TPBench paper.

Candidate Solution Structure

Step 1: Mode Equation
The solution begins by deriving the mode equation for $\chi_k(\eta)$ where $\phi = \chi/a$:

$$\chi_k''(\eta) + \omega_k^2(\eta)\chi_k(\eta) = 0 \tag{9}$$

Step 2: Scale Factor Calculation
Solving the differential equation (4) yields:

$$a(\eta) = \begin{cases} \frac{a_e}{1 - a_e H_I \eta} & \eta < 0 \\ a_e \left(1 + \frac{a_e H_I}{2}\eta\right)^2 & \eta \geq 0 \end{cases} \tag{10}$$

Step 3-4: Integral Setup
The Bogoliubov coefficient integral is expressed as $I = \int_{-\infty}^{\infty} d\eta f(\eta) e^{\phi(\eta)}$ where:

$$f(\eta) = \frac{\omega_k'(\eta)}{2\omega_k(\eta)} \tag{11}$$

$$\phi(\eta) = -2i \int_0^{\eta} d\eta' \omega_k(\eta') \tag{12}$$

Step 5: Pole Identification
Poles occur when $\omega_k^2(\eta) = 0$, yielding:

- From $\eta < 0$ continuation: $\eta_p^{(1)} = \frac{1}{a_e H_I} \mp i \frac{m}{k H_I}$
- From $\eta \geq 0$ continuation: Multiple poles with different imaginary parts

Step 6: Dominant Pole Selection
In the limit $k/(a_e H_I) \to \infty$, the dominant pole is identified as:

$$\tilde{\eta} = \frac{1}{a_e H_I} + i\frac{m}{kH_I} \tag{13}$$

Step 7-8: Residue and Approximation
The residue at the pole is calculated as $\text{Res}[f, \tilde{\eta}] = 1/4$, leading to:

$$|\beta(k)| \approx \frac{\pi}{2}|e^{\phi(\tilde{\eta})}| = \frac{\pi}{2}\exp(2\Im[J]) \tag{14}$$

Step 9: Phase Integral (Critical Step)
The candidate calculates $J = \int_0^{\tilde{\eta}} \omega_k d\eta'$ and claims:

$$\Im[J] = -\frac{1}{H_I}\sqrt{k^2 - (ma_e)^2} \quad \text{(ERROR HERE)} \tag{15}$$

Step 10: Final Result
Using the above, the candidate concludes:

$$|\beta(k)| \approx \frac{\pi}{2}e^{-2k/H_I} \tag{16}$$

Python Implementation:

```python
import math

def abs_beta(k: float, a_e: float, m: float, H_I: float) -> float:
    """
    Computes |beta(k)| using steepest descent approximation
    in the limit k / (a_e * H_I) -> infinity.
    """
    result = (math.pi / 2.0) * math.exp(-2.0 * k / H_I)
    return result
```

F.3 Step-by-Step Verification Process

F.3.1 Steps 1–4: Foundation Verification

The agent successfully verified the initial setup, including the mode equation derivation, scale factor calculation, and the formulation of the Bogoliubov coefficient integral. No errors were detected in these foundational steps.

F.3.2 Step 5: Pole Location Calculation

The agent verified the calculation of poles from the equation $\omega_k^2(\eta) = 0$. For the analytic continuation of $a(\eta) = a_e/(1 - a_e H_I \eta)$ (valid for $\eta < 0$), the poles are correctly identified as:

SymPy Verification of Pole Locations

```python
# Solving (a_e / (1 - a_e * H_I * eta_p))^2 = -k^2 / m^2
# Result: eta_p = 1/(a_e*H_I) +/- I*m/(k*H_I)
```

We note that this is an example where the agent did not write executable Python code but instead performed reasoning in the comments of the script. This sometimes happens and also allows us to spot mistakes.

F.3.3 Step 6-8: Dominant Pole and Residue Analysis

The verification confirmed:

- The dominant pole in the specified limit is $\tilde{\eta} = \frac{1}{a_e H_I} + i\frac{m}{kH_I}$
- The residue calculation yielding $\mathrm{Res}[f, \tilde{\eta}] = 1/4$ is correct
- The integral approximation framework is properly applied

F.3.4 Step 9: Critical Error Detection

The crucial error was discovered during the verification of the imaginary part of the phase integral:

$$J = \int_0^{\tilde{\eta}} \omega_k(\eta')d\eta' \tag{17}$$

The candidate solution claimed:

$$\Im[J] = -\frac{1}{H_I}\sqrt{k^2 - (ma_e)^2} \tag{18}$$

However, the SymPy verification revealed the correct result:

SymPy Script for $\Im[J]$ Verification

```
# Define symbols
a_e, H_I, k, m = symbols('a_e H_I k m', real=True, positive=True)
X = k / (m * a_e)  # Substitution variable, X > 1

# Based on the integral evaluation with sinh substitution:
# J = (m * a_e / H_I) * (sqrt(2) - (log(X) + I*sqrt(X**2 - 1)))
J_derivation = (m * a_e / H_I) * (sqrt(2) - (log(X) + I*sqrt(X**2 - 1)))

# Calculate the imaginary part
Im_J = im(J_derivation)
Im_J_simplified = simplify(Im_J.subs(X, k/(m*a_e)))

print(Im_J_simplified)
# Output: -sqrt(k**2 - a_e**2*m**2)/(H_I*a_e)
```

The SymPy output clearly shows:

$$\Im[J] = -\frac{1}{H_I a_e}\sqrt{k^2 - (ma_e)^2} \tag{19}$$

We note that the presented python script did not use Sympy for symbolic integration, but rather just checked whether a substitution was correctly performed (which it was not). Perhaps the LLM agent correctly recognized that the mistake was made in the substitution and demonstrated this with Python.

F.4 Complete Verification Output

For completeness, we reproduce the agent's final assessment:

Agent's Final Verdict

Overall Score: 0/1

General Feedback: The solution presents a detailed derivation for the Bogoliubov coefficient $|\beta(k)|$ using the steepest descent method. The overall logical flow is mostly correct: identifying the mode equation, the scale factor, setting up the integral for $\beta(k)$, finding poles of the integrand, identifying the dominant pole, calculating the residue, and then approximating the integral.

Key Error: The primary error lies in the calculation of the imaginary part of the phase integral, $\Im[J]$ (Step 9). The solution's Step 9 states $\Im[J] = -\frac{1}{H_I}\sqrt{k^2 - (ma_e)^2}$, which is missing the factor a_e in the denominator. This error propagates to the final expression for $|\beta(k)|$, resulting in an incorrect dependence on the physical parameters.

G Case Study: Stepwise SymPy-Enhanced Verification of the Halo Bias Problem

This appendix provides a detailed analysis of how our stepwise SymPy-augmented LLM verification framework systematically identified critical errors in the calculation of Gaussian moments within a candidate solution for a cosmological halo bias problem. This case study demonstrates the framework's capability to validate complex statistical mechanics derivations in physics by checking the steps with SymPy scripts.

G.1 Problem Statement

The problem requires deriving the effective bias of a clipped halo density field, a common procedure in modeling large-scale structure formation. It is one of the public level 5 problems of TPBench, and is often solved correctly by leading reasoning models.

Halo Bias Problem Statement

In cosmology, large-scale cosmological dark-matter halo fields are biased tracers of the underlying Gaussian matter density δ_m. We simulate a halo number density field by taking $n(\mathbf{x}) = \bar{n}\max(0, 1 + b\delta_m(\mathbf{x}))$, where the bare number density $\bar{n}$ and bare bias b are specified constants, and δ_m is a Gaussian random field with zero mean and variance σ^2 in each pixel.

Task: Derive an equation to evaluate the effective bias b_{eff} of the sampled halo field, defined by the relation $\langle \delta_h \rangle = b_{\text{eff}}\delta_L$ for a long-wavelength background perturbation δ_L. The final expression should depend on the bare bias b and the pixel variance σ^2.

G.2 Candidate Solution Overview

The candidate solution attempts to derive the effective bias using the peak-background split formalism. This involves expressing the mean halo density and its response to a long-wavelength perturbation as expectation values over the Gaussian distribution of the matter density contrast δ_m. The key steps in the candidate's derivation are outlined below.

Candidate Solution Structure

The solution starts by expressing the mean halo density $\bar{n}_h$ as an expectation value over the Gaussian field $X \equiv \delta_m$. The clipping at zero means the integral is non-zero only for $1 + bX > 0$, or $X > -1/b$. Let $\nu = -1/b$.

$$\bar{n}_h = \langle n(X) \rangle = \bar{n}\int_{-1/b}^{\infty}(1 + bX)P(X)dX \tag{20}$$

where $P(X)$ is a Gaussian PDF with mean 0 and variance σ^2. This is expanded into two integrals:

$$\bar{n}_h = \bar{n}\left[\int_{\nu}^{\infty} P(X)dX + b\int_{\nu}^{\infty} XP(X)dX\right] \tag{21}$$

The first term evaluates to $\int_{\nu}^{\infty} P(X)dX = \frac{1}{2}\text{erfc}\left(\frac{\nu}{\sigma\sqrt{2}}\right)$.

The solution then evaluates the first moment integral $I_1 = \int XP(X)dX$ over the truncated domain:

$$I_1 = \int_{-1/b}^{\infty} XP(X)dX = \frac{\sigma^2}{\sqrt{2\pi}} e^{-1/(2b^2\sigma^2)} \quad \text{(ERROR HERE)} \tag{22}$$

Combining these gives the candidate's expression for the mean halo density:

$$\bar{n}_h = \bar{n}\left[\frac{1}{2}\text{erfc}\left(-\frac{1}{b\sigma\sqrt{2}}\right) + b\frac{\sigma^2}{\sqrt{2\pi}} e^{-1/(2b^2\sigma^2)}\right] \tag{23}$$

The effective bias is formulated using the peak-background split argument, which defines b_{eff} as the response of the mean halo number density to a long-wavelength perturbation δ_L:

$$b_{\text{eff}} = \frac{1}{\bar{n}_h} \frac{d\langle n|\delta_L\rangle}{d\delta_L}\Big|_{\delta_L=0} \tag{24}$$

The derivative term is shown to be equivalent to:

$$\frac{d\langle n|\delta_L\rangle}{d\delta_L}\Big|_{\delta_L=0} = \int_{-1/b}^{\infty} \bar{n}(1+bX)\left(P(X)\frac{X}{\sigma^2}\right)dX = \frac{\bar{n}}{\sigma^2}\int_{-1/b}^{\infty}(X+bX^2)P(X)dX \tag{25}$$

To evaluate the above, the solution requires the second moment integral $I_2 = \int X^2 P(X)dX$ over the truncated domain, for which it claims:

$$I_2 = \int_{-1/b}^{\infty} X^2 P(X)dX = \frac{-\sigma^2/b}{\sqrt{2\pi}} e^{-1/(2b^2\sigma^2)} + \sigma^3 \frac{1}{2}\text{erfc}\left(\frac{-1}{b\sigma\sqrt{2}}\right) \quad \text{(ERROR HERE)} \tag{26}$$

Combining the (incorrect) intermediate moment calculations I_1 and I_2, the candidate assembles the numerator for the b_{eff} expression:

$$D(\sigma, b) = \frac{\bar{n}}{\sigma^2}[I_1 + bI_2] = \bar{n}\left[\frac{1+b\nu}{\sqrt{2\pi}} e^{-\nu^2/(2\sigma^2)} + \frac{b\sigma}{2}\text{erfc}\left(\frac{\nu}{\sigma\sqrt{2}}\right)\right] \tag{27}$$

Substituting $\nu = -1/b$ correctly makes the term $(1 + b\nu)$ equal to zero, which simplifies the expression to:

$$D(\sigma, b) = \bar{n}\frac{b\sigma}{2}\text{erfc}\left(-\frac{1}{b\sigma\sqrt{2}}\right) \tag{28}$$

The effective bias is then $b_{\text{eff}} = D(\sigma, b)/\bar{n}_h$. Using the identity $\text{erfc}(-y) = 1 + \text{erf}(y)$ and simplifying, the candidate arrives at the final expression.

Final Mathematical Expression:

$$b_{\text{eff}}(\sigma, b) = \frac{b\sigma\left(1 + \text{erfc}\left(\frac{1}{b\sigma\sqrt{2}}\right)\right)}{\left(1 + \text{erfc}\left(\frac{1}{b\sigma\sqrt{2}}\right)\right) + \sqrt{\frac{2}{\pi}}b\sigma^2 \exp\left(-\frac{1}{2b^2\sigma^2}\right)} \tag{29}$$

G.3 Step-by-Step Verification Process

The following is a detailed verification of the mathematical derivations presented in the solution. Each step was computationally verified using the SymPy symbolic mathematics library in Python. The code used for each verification is provided.

1. **Calculation of $\int_\nu^\infty P(X)dX$:** The solution evaluates the integral of the Gaussian probability density function $P(X) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{X^2}{2\sigma^2}\right)$ from ν to ∞.
   - **Expected Result:** $\frac{1}{2}\,\text{erfc}\left(\frac{\nu}{\sigma\sqrt{2}}\right)$.
   - **SymPy Verification:** The computation confirms the expected result.
   - **Conclusion:** *Correct*.

SymPy Verification of $\int_{\nu}^{\infty} P(X)dX$

```python
import sympy

X, nu = sympy.symbols('X nu', real=True)
sigma = sympy.Symbol('sigma', positive=True)

P_X = (1 / sympy.sqrt(2 * sympy.pi * sigma**2)) * sympy.exp(-X**2 / (2 *
     sigma**2))
integral_val = sympy.integrate(P_X, (X, nu, sympy.oo))

print(integral_val)
```

2. **Calculation of $I_1 = \int_{\nu}^{\infty} XP(X)dX$:** This step calculates the first moment of the truncated Gaussian distribution.
   - **Stated Result:** $\frac{\sigma^2}{\sqrt{2\pi}}e^{-\nu^2/(2\sigma^2)}$.
   - **SymPy Verification:** The computation yields $\frac{\sigma}{\sqrt{2\pi}}e^{-\nu^2/(2\sigma^2)}$.
   - **Conclusion:** *Incorrect*. The result stated in the solution contains an erroneous extra factor of σ.

SymPy Verification of $I_1 = \int_{\nu}^{\infty} XP(X)dX$

```python
import sympy

X, nu = sympy.symbols('X nu', real=True)
sigma = sympy.Symbol('sigma', positive=True)

P_X = (1 / sympy.sqrt(2 * sympy.pi * sigma**2)) * sympy.exp(-X**2 / (2 *
     sigma**2))
integrand = X * P_X
integral_val = sympy.integrate(integrand, (X, nu, sympy.oo))

print(integral_val)
```

3. **Internal Consistency of $\bar{n}_h$ Expression:** This step verifies the algebraic construction of $\bar{n}_h = \bar{n}\left[\int_{-1/b}^{\infty} P(X)dX + b\int_{-1/b}^{\infty} XP(X)dX\right]$ using the solution's flawed intermediate results.
   - **Stated Result:** $\bar{n}\left[\frac{1}{2}\,\mathrm{erfc}\left(-\frac{1}{b\sigma\sqrt{2}}\right) + b\frac{\sigma^2}{\sqrt{2\pi}}e^{-1/(2b^2\sigma^2)}\right]$.
   - **SymPy Verification:** The code correctly reconstructs the stated formula for $\bar{n}_h$ using the flawed value of I_1.
   - **Conclusion:** *Algebraically Consistent*.

SymPy Verification of $\bar{n}_h$ Construction

```python
import sympy

sigma_sym, b_sym, n_bar = sympy.symbols('sigma b n_bar', real=True)
sigma = sympy.Symbol('sigma', positive=True)
b = sympy.Symbol('b', real=True)
nu_val = -1/b

term1_integral_P_X = sympy.Rational(1,2) * sympy.erfc(-1 / (b * sigma *
     sympy.sqrt(2)))
term2_integral_X_P_X_solution =
     (sigma**2/sympy.sqrt(2*sympy.pi))*sympy.exp(-1/(2*b**2*sigma**2))

n_h_constructed = n_bar * (term1_integral_P_X + b *
     term2_integral_X_P_X_solution)
```

```
print(n_h_constructed)
```

4. **Calculation of the Derivative $\frac{dP(X|\delta_L)}{d\delta_L}\Big|_{\delta_L=0}$:** This step evaluates the derivative of a shifted Gaussian distribution.
   - **Stated Result:** $P(X)\frac{X}{\sigma^2}$.
   - **SymPy Verification:** The derivative is calculated as $\frac{X}{\sigma^3\sqrt{2\pi}}\exp\left(-\frac{X^2}{2\sigma^2}\right)$, which is equivalent to $P(X)\frac{X}{\sigma^2}$.
   - **Conclusion:** *Correct*.

SymPy Verification of $\frac{dP(X|\delta_L)}{d\delta_L}|_{\delta_L=0}$

```
import sympy

X, delta_L = sympy.symbols('X␣delta_L', real=True)
sigma = sympy.Symbol('sigma', positive=True)

P_X_cond = (1/sympy.sqrt(2*sympy.pi*sigma**2))*sympy.exp(-(X -
     delta_L)**2/(2*sigma**2))
deriv_P_X_cond = sympy.diff(P_X_cond, delta_L)
deriv_at_0 = deriv_P_X_cond.subs(delta_L, 0)

print(deriv_at_0)
```

5. **Calculation of $I_2 = \int_\nu^\infty X^2 P(X)dX$:** This step calculates the second moment of the truncated Gaussian distribution.
   - **Stated Result:** $\frac{\sigma^2\nu}{\sqrt{2\pi}}e^{-\nu^2/(2\sigma^2)} + \frac{\sigma^3}{2}\,\mathrm{erfc}\left(\frac{\nu}{\sigma\sqrt{2}}\right)$.
   - **SymPy Verification:** The computed integral is $\frac{\sigma\nu}{\sqrt{2\pi}}e^{-\nu^2/(2\sigma^2)} + \frac{\sigma^2}{2}\,\mathrm{erfc}\left(\frac{\nu}{\sigma\sqrt{2}}\right)$.
   - **Conclusion:** *Incorrect*. The solution has errors in the powers of σ for both terms.

SymPy Verification of $I_2 = \int_\nu^\infty X^2 P(X)dX$

```
import sympy

X, nu = sympy.symbols('X␣nu', real=True)
sigma = sympy.Symbol('sigma', positive=True)

P_X = (1 / sympy.sqrt(2 * sympy.pi * sigma**2)) * sympy.exp(-X**2 / (2 *
     sigma**2))
integrand = X**2 * P_X
integral_val = sympy.integrate(integrand, (X, nu, sympy.oo))

print(integral_val)
```

6. **Algebraic Simplification of $D(\sigma, b)$:** This step checks the algebraic simplification of $D(\sigma, b) = \frac{\bar{n}}{\sigma^2}[I_1^{\mathrm{sol}} + bI_2^{\mathrm{sol}}]$ using the solution's incorrect expressions for I_1 and I_2.
   - **Target Result:** $\bar{n}\left[\frac{1}{\sqrt{2\pi}}e^{-\nu^2/(2\sigma^2)}(1+b\nu) + \frac{b\sigma}{2}\,\mathrm{erfc}\left(\frac{\nu}{\sigma\sqrt{2}}\right)\right]$.
   - **SymPy Verification:** Algebraically combining the flawed I_1 and I_2 expressions correctly yields the target expression.
   - **Conclusion:** *Algebraically Consistent*.

SymPy Verification of $D(\sigma, b)$ Simplification

```python
import sympy

nu = sympy.Symbol('nu', real=True)
sigma = sympy.Symbol('sigma', positive=True)
b = sympy.Symbol('b', real=True)
n_bar = sympy.Symbol('n_bar', real=True)

I1_sol = (sigma**2 / sympy.sqrt(2*sympy.pi)) * sympy.exp(-nu**2 /
    (2*sigma**2))
I2_term1_sol = (sigma**2 * nu / sympy.sqrt(2*sympy.pi)) *
    sympy.exp(-nu**2 / (2*sigma**2))
I2_term2_sol = sigma**3 * sympy.Rational(1,2) * sympy.erfc(nu / (sigma *
    sympy.sqrt(2)))
I2_sol = I2_term1_sol + I2_term2_sol

D_constructed = (n_bar / sigma**2) * (I1_sol + b * I2_sol)
D_constructed_simplified = sympy.expand(sympy.simplify(D_constructed))

print(D_constructed_simplified)
```

7. **Substitution of $\nu = -1/b$ into $D(\sigma, b)$:** This step substitutes the lower integration limit $\nu = -1/b$ into the expression for $D(\sigma, b)$.
   - **Expected Result:** $\bar{n}\frac{b\sigma}{2}\operatorname{erfc}\left(-\frac{1}{b\sigma\sqrt{2}}\right)$.
   - **SymPy Verification:** Upon substituting $\nu = -1/b$, the expression simplifies exactly to the expected result.
   - **Conclusion:** *Correct*.

SymPy Verification of $D(\sigma, b)$ with $\nu = -1/b$

```python
import sympy

nu_sym = sympy.Symbol('nu', real=True)
sigma = sympy.Symbol('sigma', positive=True)
b = sympy.Symbol('b', real=True)
n_bar = sympy.Symbol('n_bar', real=True)

term_A = (1 +
    b*nu_sym)/sympy.sqrt(2*sympy.pi)*sympy.exp(-nu_sym**2/(2*sigma**2))
term_B = (b * sigma / 2) * sympy.erfc(nu_sym / (sigma * sympy.sqrt(2)))
D_prev_step_form = n_bar * (term_A + term_B)

D_substituted = D_prev_step_form.subs(nu_sym, -1/b)
D_substituted_simplified = sympy.simplify(D_substituted)

print(D_substituted_simplified)
```

8. **Final Simplification of b_{eff}:** This step verifies the final algebraic manipulation of the b_{eff} expression.
   - **Stated Result:** $\frac{b\sigma(1+\operatorname{erfc}(\frac{1}{b\sigma\sqrt{2}}))}{(1+\operatorname{erfc}(\frac{1}{b\sigma\sqrt{2}}))+\sqrt{\frac{2}{\pi}}b\sigma^2\exp(-\frac{1}{2b^2\sigma^2})}$.
   - **SymPy Verification:** The sequence of substitutions and algebraic simplifications was confirmed to match the solution.
   - **Conclusion:** *Correct*. The algebraic simplification is valid, but it relies on mathematically incorrect intermediate results (I_1, I_2).

SymPy Verification of b_{eff} Final Simplification

```python
import sympy
```

```
sigma = sympy.Symbol('sigma', positive=True)
b = sympy.Symbol('b', real=True)
y_var = sympy.Symbol('y', real=True)

# Starting expression for b_eff, substituting y = 1/(b*sigma*sqrt(2))
# and using erfc(-y) = 1 + erf(y)
num_erf = (b * sigma / 2) * (1 + sympy.erf(y_var))
denom_term1_erf = sympy.Rational(1,2) * (1 + sympy.erf(y_var))
denom_term2 = (b * sigma**2 / sympy.sqrt(2*sympy.pi)) *
     sympy.exp(-y_var**2)

# Multiply numerator and denominator by 2 and simplify
num_final = b * sigma * (1 + sympy.erf(y_var))
denom_final = (1 + sympy.erf(y_var)) +
     (sympy.sqrt(2/sympy.pi)*b*sigma**2)*sympy.exp(-y_var**2)

final_b_eff_expr_in_y = num_final / denom_final

# Substitute y back to get expression in b and sigma
y_definition = 1 / (b * sigma * sympy.sqrt(2))
final_b_eff_expr = final_b_eff_expr_in_y.subs(y_var, y_definition)

print(sympy.simplify(final_b_eff_expr))
```

G.4 Complete Verification Output

The agent's final verdict, based on a comprehensive check of all steps, is reproduced below.

Agent's Final Verdict

Overall Score: 0/1

General Feedback: The solution attempts to derive the effective bias b_{eff}. The overall logical structure of the derivation (setting up integrals for mean density, using peak-background split for the derivative term, and combining them) is generally sound. The initial steps involving the zeroth moment and the PDF derivative were verified as correct.

Key Error: The primary errors lie in the calculation of the first and second moments of the truncated Gaussian distribution. Following the correct foundational steps, the candidate first makes an error in calculating $\int XP(X)dX$, which is incorrect by a factor of σ. This fundamental error is then compounded by similar errors in the second moment calculation. These errors propagate through the entire derivation, leading to an incorrect final expression for the effective bias b_{eff}.

H Scaling of Sequential Method

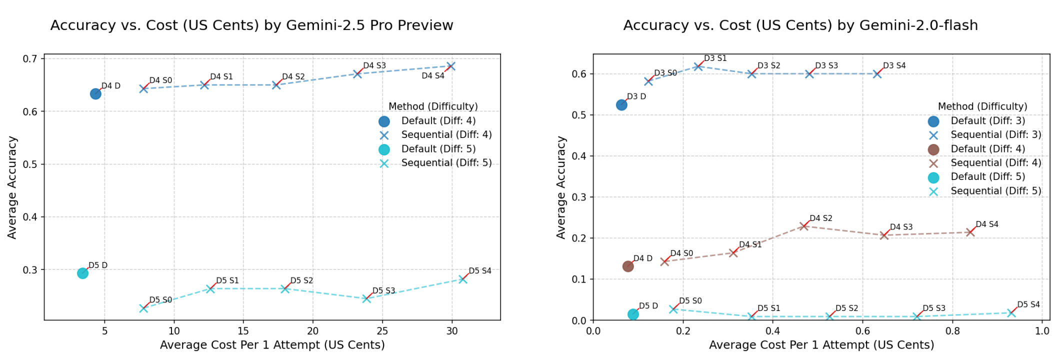

Figure 3: Cost and accuracy comparison for single attempt and sequential scaling method.

We examined the computational cost of sequential scaling compared to the achieved accuracy. In Figure 3, the dots $DnSi$ indicate the average cost for a problem with difficulty level n and in round i of sequential scaling, where round i refers to the number of extra thinking steps the model takes. Note that we include a summarization step for code generation, which makes the cost of $DnS0$ higher than that of a single-pass default reasoning attempt.

In terms of performance, we observe a minor improvement on level 3 and 4 questions, but no or even negative gains on level 5 for both models. This aligns with findings from the S1 paper [13], where s1 and s1.1 models also exhibit negligible improvement from additional “wait" tokens on hard AIME tasks, though they show slight gains on simpler MATH500 problems. These results illustrate the limitations of the sequential scaling approach on more difficult problems.

To better understand the token consumption patterns of this method, we plot in Figure 4 the token usage across different models and difficulty levels.

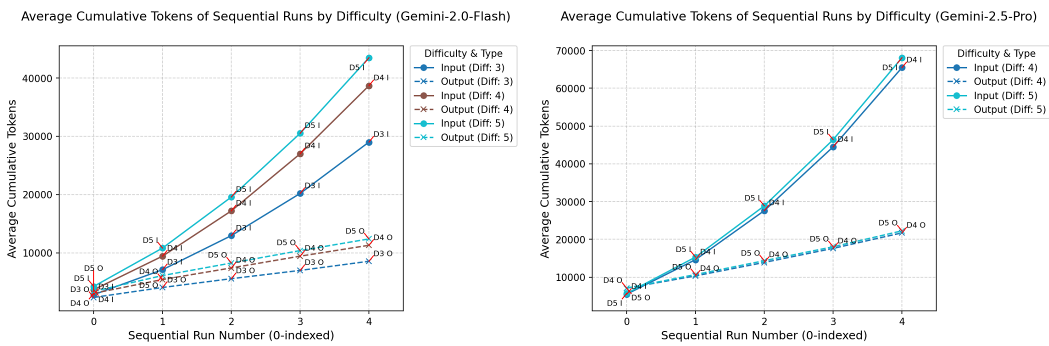

Figure 4: Token consumption across rounds by Gemini 2.0 and 2.5 models on TPBench. More difficult problems require more tokens, and the growth is roughly linear for easy problems but super-linear for harder ones.