ANTIDERIVBENCH: Evaluating language models on indefinite integration

Bartosz Piotrowski
Meta FAIR
bpio@meta.com

Kaiyu Yang
Meta FAIR
kaiyuy@meta.com

Abstract

We present ANTIDERIVBENCH: a benchmark consisting of integration problems extracted from the challenging annual MIT Integration Bee competition. A number of frontier, closed models as well as smaller, open-source models are evaluated on it. Additionally, we create *more challenging* versions of the benchmark by symbolically manipulating the original competition problems. We envision that the benchmark will be useful for evaluating reasoning capabilities of LLMs and for experimenting with post-training LLM pipelines depending on verifiable rewards.

1 Introduction

Recently, there has been substantial progress in enhancing the reasoning capabilities of LLMs. Specifically, this effort includes applying LLMs to solve mathematical problems (Trinh et al., 2024), writing code (Jiang et al., 2025), performing calculations (Charton, 2024), recognizing complex spatial patterns (Chollet et al., 2024), and writing formal proofs (Yang et al., 2024).

An important aspect of developing LLMs for these reasoning-intensive domains is evaluating their performance. However, reliable evaluation is difficult. In particular, it poses the following challenges:

1. **Avoiding data contamination:** It is not unlikely that existing popular benchmarks – e.g., GSM8K (Cobbe et al., 2021) or MATH(Hendrycks et al., 2021) – are unintentionally leaked to the pre-training corpora of LLMs, which makes them less suitable for measuring genuine reasoning skills. Hence, semi-synthetic benchmarks based on the popular ones have been created, e.g., GSM-Symbolic (Mirzadeh et al., 2024) or MATH() (Srivastava et al., 2024).
2. **Reflecting real-world problems:** Synthetic, algorithmically generated benchmarks (like those created by Saxton et al. (2019) or Lample & Charton (2019)) avoid the contamination problems, but, on the other hand, it is less clear to what extent they are representative of practical problems encountered in real scenarios.
3. **Evaluating answers:** The ability to automatically and reliably evaluate the correctness of the LLM-generated answers constitutes a significant limitation in benchmark design. For instance, the frontier LLMs are now able to prove competition-style mathematical theorems, but there is no way to automatically assess the correctness of natural-language proofs.

   Hence, models have been developed for proving in formal mathematical languages (like Lean (de Moura et al., 2015)) where the proof correctness is assessed with absolute certainty.

   In the natural language space, reliable evaluation is limited to problems where the LLM answer can be easily compared to the ground truth – which in practice means it must be just a number or other simple mathematical expression (Glazer et al., 2024).

   Reliable assessment of answer correctness becomes even more important when used as a training signal for fine-tuning models via reinforcement learning.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: MATH-AI.

In this work, we contribute to the landscape of reasoning datasets and introduce ANTIDERIVBENCH:[1] a new benchmark consisting of indefinite integration problems sourced from the challenging annual MIT Bee Integration competition.[2]

ANTIDERIVBENCH attempts to mitigate the challenges (1-3) described above: First, we provide a method to synthetically augment the problems sourced from the competition to obtain complex, unseen problems, but still related to the original ones. This reduces the contamination effects.

Second, our benchmark is based on human-written, competition problems reflecting real-world difficulty of integration. Lample & Charton (2019) generated a large (millions of examples) dataset of randomly sampled mathematical expressions and their integrals. Their dataset, however, is fully synthetic, and as noted above, it may not truly reflect challenging or practical integration problems.

Third, ANTIDERIVBENCH targets evaluating reasoning in natural language, but as opposed to the popular benchmarks with math word problems with numerical answers, here the response may be an arbitrarily mathematical expression with one variable, composed of standard math operations and basic functions (log, exp, trigonometric functions, *etc*). We evaluate the correctness of such answers by implementing an evaluator which parses LaTeX mathematical expressions and performs symbolic and numeric checks leveraging SymPy (Meurer et al., 2017) and SciPy (Virtanen et al., 2020) Python libraries.

Integration of functions is both challenging and related to many problems arising in science and engineering. In principle, there exists a complete algorithm for solving arbitrary indefinite integrals (Risch & Protter, 1970). However, the algorithm is complex (its description takes more than 100 pages), and its implementations in computer algebra systems are typically incomplete (and slow).

In a concurrent work, Tang et al. (2025) develop a similar integration benchmark based on problems sourced from competitions and textbooks. However, they focus on *definite* integration as opposed to *indefinite* ones. Additionally, we introduce symbolic augmentation techniques for generating more challenging, unseen problems.

Simonds & Yoshiyama (2025) develop LADDER, a self-improvement LLM post-training pipeline involving auxiliary question generation and evaluate their models, among others, on integration problems from 2025 edition of MIT Integration Bee. This indicates that the competition is a good testbed for evaluating LLM training pipelines. We therefore collect and process, all the available problems from *all* the years of the competition.

We believe that ANTIDERIVBENCH may be useful to the community for evaluating LLM models. It may also serve as a testbed for RL and other LLM self-improvement pipelines, e.g., involving proposing alternative integration problems. The benchmark, in a sense, fills the gap between the math word problem benchmarks with numerical answers, and the formal math benchmarks, for which training data is limited and whose usage involves engineering overhead.

2 Benchmark

We use the Mathpix[3] software to extract all the integration problems from the original PDFs available on the MIT Integration Bee website and turn them into LaTeX expressions. The problems consist of both definite and indefinite integrals; we select only the indefinite ones. We also filter out expressions involving infinite summations or products. We parse the Latex expressions using `parse_latex` function from the SymPy library and convert them into SymPy format.

This yields in 243 integration problems, which we divide into two partitions:

- `qualifier`: 177 simpler problems from the qualification round.
- `competition`: 66 problems form the proper competition.

[1] `https://github.com/BartoszPiotrowski/antiderivbench`

[2] `https://math.mit.edu/~yyao1/integrationbee.html`

[3] `https://mathpix.com/`

We also prepare synthetic augmentations of the 66 competition problems, based on the linearity of integration:

$$\int a \cdot f(x) + b \cdot g(x)\, dx = a \cdot \int f(x)\, dx + b \cdot \int g(x)\, dx \tag{1}$$

as well as the substitution rule for integration:

$$\int f(g(x)) \cdot g'(x)\, dx = \int f(u)\, du, \tag{2}$$

where $u = g(x)$. The augmented problems form three subsets (each of size $66 \cdot 3 = 198$):

- `lin_comb`: where each of the `competition` problems is linearly combined with three randomly selected `qualifier` problems using random integer coefficients $a, b \in \{-9, -8, \ldots, -1, 1, \ldots, 8, 9\}$.
- `subst_poly`: where for each $f(x) \in$ `competition` we randomly generate three polynomials $g(x) = ax^3 + bx^2 + cx + d$ where $a, b, c, d \in \{-9, -8, \ldots, 8, 9\}$ and construct integrals $\int f(g(x)) \cdot g'(x)\, dx$. To solve it, the model will need to first spot the substitution pattern, which in turn requires to be able to integrate $g'(x)$ (an easy task in case of polynomial $g(x)$).
- `subst_hard`: where the problems are generated in the same way as above, but this time $g(x)$'s are not polynomials, but randomly selected functions from `qualifier`. Intuitively, this should make such problems more difficult than those from `subst_poly`.

For all the above augmentation procedures, once a new function has been created, we run SymPy's `simplify` function in order to potentially change the structure of the expression and make its initial form more difficult to notice.

Below there are examples of problems from ANTIDERIVBENCH: the first from `qualifier`, the second from `competition`, the last from `subst_hard`.

$$\int e^{x+e^x} + e^{x-e^x}\, dx = e^{e^x} - e^{-e^x}$$

$$\int \frac{1}{\sqrt[4]{x^4+1}} dx = \frac{1}{2} \arctan\left(\frac{x}{\sqrt[4]{x^4+1}}\right) + \frac{1}{4} \log\left(\frac{\sqrt[4]{x^4+1} + x}{\sqrt[4]{x^4+1} - x}\right)$$

$$\int \tan(x) \sqrt{2 + \sqrt{4 + \cos(x)}}\, dx = -4\sqrt{2 + \sqrt{4 + \cos(x)}} - 2 \log\left(\frac{\sqrt{2 + \sqrt{4 + \cos(x)}} - 2}{\sqrt{2 + \sqrt{4 + \cos(x)}} + 2}\right)$$

3 Evaluator

Assessing the correctness of the model's answer $F(x)$ for the indefinite integral $\int f(x)\, dx$ boils down to computing $F'(x)$ and comparing it to $f(x)$. However, in general, the functional equivalence is undecidable. We implement an approximate procedure for determining the answer correctness that, in practice, captures the functional equivalence in a vast majority of cases. The steps are as follows:

1. We extract the answer $F(x)$ from the model's generation using SymPy's `parse_latex`.[4]
2. We compute $F'(x)$ (using SymPy's `diff`) and starting with $F'(x) - f(x)$, we execute a chain of SymPy's simplifying procedures. If we obtain 0, the check concludes positively.
3. Otherwise, we attempt to compute the integral $\int f(x)\, dx$ by SymPy's `integrate` and compare it to $F(x)$ as above. If successful, the check concludes positively.
4. Otherwise, we evaluate $|F'(x) - f(x)|$ for $x \in \bigcup_{i \in \{1,\ldots,1000\}} \{i, -i, \frac{1}{i}, \frac{-1}{i}\}$. If there is x_0 in this set for which $|F'(x) - f(x)| > 10^{-8}$, we conclude the check negatively.
5. Otherwise, we attempt to maximize $|F'(x) - f(x)|$ using SciPy's `minimize_scalar`. Again, if the obtained value is $> 10^{-8}$, the check concludes negatively.
6. Otherwise, the check concludes positively.

[4] As `parse_latex` fails to parse some more complicated but correct Latex expression (e.g., those containing less usual trigonometric functions like arctanh), we write additional code to handle (most of) these cases.

Table 1: Performance of LLMs on MIT Integration Bee problems and their synthetic augmentations. We also include the performance of the SymPy's `integrate` procedure (with 10 s time limit).

data partition	model / method	pass@1	pass@16	pass@64
`qualifier`	SymPy	83.1%	–	–
	Gemini 2.5 Flash	92.4%	94.4%	94.9%
	GPT-4o	52.2%	81.5%	88.1%
	Qwen 3 8B (non-thinking mode)	77.7%	93.7%	95.5%
	Qwen 3 8B (thinking mode)	55.5%	77.5%	93.8%
	Llama 3.1 8B	7.0%	39.2%	59.9%
`competition`	SymPy	65.1%	–	–
	Gemini 2.5 Flash	78.7%	86.3%	86.4%
	GPT-4o	24.5%	54.6%	63.6%
	Qwen 3 8B (non-thinking mode)	49.1%	79.2%	84.8%
	Qwen 3 8B (thinking mode)	77.8%	87.8%	87.9%
	Llama 3.1 8B	0.7%	8.3%	15.2%
`lin_comb`	SymPy	53.0%	–	–
	Gemini 2.5 Flash	70.3%	83.8%	84.8%
	GPT-4o	4.1%	15.6%	22.2%
	Qwen 3 8B (non-thinking mode)	18.9 %	49.2%	57.7%
	Qwen 3 8B (thinking mode)	62.6%	83.5%	86.9%
	Llama 3.1 8B	0.0%	0.2%	0.5%
`subst_poly`	SymPy	42.4%	–	–
	Gemini 2.5 Flash	51.4%	65.7%	67.1%
	GPT-4o	4.3%	17.1%	21.2%
	Qwen 3 8B (non-thinking mode)	13.1%	38.5%	48.0%
	Qwen 3 8B (thinking mode)	45.9%	62.8%	66.2%
	Llama 3.1 8B	0.1%	2.6%	3.5%
`subst_hard`	SymPy	16.7%	–	–
	Gemini 2.5 Flash	59.5%	78.1%	81.7%
	GPT-4o	3.4%	15.1%	22.7%
	Qwen 3 8B (non-thinking mode)	15.3%	46.1%	58.1%
	Qwen 3 8B (thinking mode)	56.0%	77.5%	81.3%
	Llama 3.1 8B	0.1%	0.3%	3.0%

4 Experiments

We evaluate the following models on the benchmark: Gemini 2.5 Flash, GPT-4o, Qwen3 8B (including its thinking mode), and Llama 3.1 8B. For each, we set the generation temperature of 1.0 and the maximum generation length of $16,384$ tokens. For each of the problems, we generate 64 samples.

The results for `qualifier` and `competition` benchmark subsets based on the original problems, as well as for the three synthetically augmented datasets are shown in Table 1. In addition to the models' performance, we also evaluate SymPy's `integrate` function, which implements part of the Risch algorithm for integration. We impose a 10-second timeout on the procedure.

Overall, the best-performing models are Gemini 2.5 Flash and Qwen3 8B in thinking mode, which achieved solid pass@64 scores of 86.4% and 87.9% on `competition`, respectively. It is surprising, given Qwen's small size. The third best result was achieved by Qwen3 8B in non-thinking mode, where it used on average about 6.5 times less tokens compared to the thinking mode.

Also, surprisingly, the results of Gemini and Qwen with thinking mode on `subst_hard` and `lin_comb` are not much worse than on `competition`. However, for the other models, the performance differences between these subsets are much more significant (especially for GPT-4o).

Predictably, `qualifier` turned out to be overall simpler than `competition`. However, somewhat counterintuitively, `subst_hard` turned out to be simpler than `subst_poly`.

Llama 3.1 8B, as a model that was not trained for reasoning problems, was the weakest of the evaluated models. Still, it achieved a reasonably good pass@64 on `qualifier`.

SymPy's `integrate` procedure performed weaker that Qwen3 and Gemini 2.5 across all the partitions. Unlike the models, its performance on `subst_hard` was much lower than on `subst_poly`.

5 Conclusion

We believe that ANTIDERIVBENCH is a useful benchmark for evaluating mathematical capabilities of language models, and may also serve as an environment for RL-training pipelines.

The presented way of mechanically augmenting existing integration problems can be extended and used to produce even more challenging problems, if needed.

Acknowledgments

We thank Amitayush Thakur and Zhaoyu Li for useful discussions.

References

François Charton. Learning the greatest common divisor: explaining transformer predictions. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL `https://openreview.net/forum?id=cmcD05NPKa`.

François Chollet, Mike Knoop, Gregory Kamradt, and Bryan Landers. ARC Prize 2024: Technical report. *CoRR*, abs/2412.04604, 2024. doi: 10.48550/ARXIV.2412.04604. URL `https://doi.org/10.48550/arXiv.2412.04604`.

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. *CoRR*, abs/2110.14168, 2021. URL `https://arxiv.org/abs/2110.14168`.

Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp (eds.), *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pp. 378–388. Springer, 2015. doi: 10.1007/978-3-319-21401-6_26. URL `https://doi.org/10.1007/978-3-319-21401-6_26`.

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. *CoRR*, abs/2411.04872, 2024. doi: 10.48550/ARXIV.2411.04872. URL `https://doi.org/10.48550/arXiv.2411.04872`.

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. In Joaquin Vanschoren and Sai-Kit Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL `https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html`.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation. *ACM Trans. Softw. Eng. Methodol.*, July 2025. ISSN 1049-331X. doi: 10.1145/3747588. URL `https://doi.org/10.1145/3747588`.

Guillaume Lample and François Charton. Deep learning for symbolic mathematics. *CoRR*, abs/1912.01412, 2019. URL `http://arxiv.org/abs/1912.01412`.

Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103. URL https://doi.org/10.7717/peerj-cs.103.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models, 2024. URL https://arxiv.org/abs/2410.05229.

Robert H. Risch and Murray H. Protter. The solution of the problem of integration in finite terms. *Bulletin of the American Mathematical Society*, 76:605–608, 1970. URL https://api.semanticscholar.org/CorpusID:46987154.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models, 2019. URL https://arxiv.org/abs/1904.01557.

Toby Simonds and Akira Yoshiyama. LADDER: self-improving llms through recursive problem decomposition. *CoRR*, abs/2503.00735, 2025. doi: 10.48550/ARXIV.2503.00735. URL https://doi.org/10.48550/arXiv.2503.00735.

Saurabh Srivastava, Annarose M. B, Anto P. V, Shashank Menon, Ajay Sukumar, Adwaith Samod T, Alan Philipose, Stevin Prince, and Sooraj Thomas. Functional benchmarks for robust evaluation of reasoning performance, and the reasoning gap. *CoRR*, abs/2402.19450, 2024. URL https://doi.org/10.48550/arXiv.2402.19450.

Bintao Tang, Xin Yang, Yuhao Wang, Zixuan Qiu, Zimo Ji, and Wenyuan Jiang. INTEGRALBENCH: benchmarking llms with definite integral problems. *CoRR*, abs/2507.21130, 2025. doi: 10.48550/ARXIV.2507.21130. URL https://doi.org/10.48550/arXiv.2507.21130.

Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin Lauter, Swarat Chaudhuri, and Dawn Song. Formal mathematical reasoning: A new frontier in AI. *CoRR*, abs/2412.16075, 2024. doi: 10.48550/ARXIV.2412.16075. URL https://doi.org/10.48550/arXiv.2412.16075.

Appendix

Prompts used in experiments

```
Compute the integral of the following function:

{x}

Think step by step. Put the final answer in \boxed{ }
using LaTeX notation.
```

Figure 1: The prompt used for integrating functions with tested LLMs.