LLM vs ITP

S. Frieder*
Department of Computer Science
University of Oxford
simon.frieder@cs.ox.ac.uk

M. Trimmel
RISE Research Institutes of Sweden
martin.trimmel@ri.se

R. Alawadhi
r.alawadhi@qmul.ac.uk

K. Gy
klausgy@gmail.com

Abstract

Wiedijk's list of 100 theorems provides a benchmark for comparing interactive theorem provers (ITPs) and their mathematics libraries. As shown by the GHOSTS dataset, large language models (LLMs) can also serve as searchable libraries of mathematics, given their capacity to ingest vast amounts of mathematical literature during their pre-training or finetuning phases. ITP libraries are the only other repositories of comparable size and range of mathematical intricacy. This paper presents the first comparison between these two unique mathematical resources, centered on Wiedijk's list. Beyond the intrinsic interest of such a comparison, we discuss the importance of analyzing whether knowledge contained in LLMs (represented by GPT-4 and Claude 2) matches that encoded in ITPs. This analysis contributes thus further to advance the intersection between LLM and ITP technology (examples being tasks like autoformalization, LLM-guided proof generation, or proof completion) by ensuring LLMs possess, beyond ITP code generation capabilities, sufficient mathematical knowledge to carry out the desired formalization. The dataset with our findings, called "LLMKNOW", is made available to the public.

https://llmknow.friederrr.org

1 Introduction

Interactive theorem provers (ITPs), such as Lean (introduced in [16], currently at version 4 [30]) or Isabelle (introduced originally in [32, 33] subsequently updated and expanded [34, 20]), which are some of the most well-known examples of ITPs, have large libraries of formal proofs of mathematical theorems associated to them. E.g., in the case of Isabelle, these are core libraries included in the main distribution as well as external proof developments contained in the Archive of Formal Proof[2]; in the case of Lean's latest version, the Lean Mathematical Library [28], `mathlib4`[3], contains all the pertinent mathematics.

Progress in ITPs has been steady, and as the burden on the person converting a natural-language proof to a proof in the formal system of an ITP was alleviated in time [20], the libraries of formally verified proofs grew to the point where a sufficient amount of mathematics is encoded that allows ITPs to be used as a teaching support for undergraduate curricula (and beyond [7]). For example, undergraduate mathematical textbooks exist where taught mathematics is formalized as it is developed [27].

In some instances, formalization has advanced to cover research-level mathematics. Some of the notable examples are Szemerédi's Regularity Lemma in extremal graph theory [17], hyper-dual

*Corresponding author. The other authors are listed in random order.

[2] https://www.isa-afp.org/

[3] https://github.com/leanprover-community/mathlib4

37th Conference on Neural Information Processing Systems (NeurIPS 2023) Workshop on MATH-AI.

numbers in second-order automatic differentiation [37], schemes in algebraic geometry [9, 5] or the Liquid Tensor Experiment [36, 10].

With the introduction of the first version of ChatGPT in November, which was widely adopted, investigating LLMs mathematical abilities has received renewed impetus, e.g. [6, 19, 26, 3]. All of these works have focused on the abilities of ChatGPT (also known as "GPT-3.5") or GPT-4, which were shown to trump all other models at the time.

In [19], it has been noted that LLMs are able to function well as mathematical search engines. This begs the question: Where do they search? LLMs obtain their knowledge during pre-training and finetuning - and it is plausible that LLMs have been trained on the entirety of public sources of mathematical data, such as the `arxiv.org` preprint repository, or various digitized books. For the most prominent LLMs, e.g. GPT-4 [31], the precise collections of training data has not been revealed. LLMs, therefore, can be assumed to have encoded most of the digitally accessible books on mathematics in their architectures and weights.

Whereas for ITPs the mathematical knowledge they encode is completely transparent (even if it may be hard to parse their formal language), for LLMs it is unknown what and how much mathematics they have seen, which raises the question of whether the (implicit) mathematical knowledge base of an LLM can (at least) match the (explicit) knowledge base of ITP.

Aside from an academic interest in comparing the knowledge bases, there is a second reason, more pertinent reason for such comparison, coming from the growing trend of merging LLM technology with ITP technology: LLMs have been used to perform *autoformalization* (converting a natural-language proof to a formal proof that an ITP can ingest) as well as *proof completion and generation* (suggesting the next step in a partially elaborated proof/generation an entire formal proof from a formal statement). It is unreasonable to expect LLMs to carry out these tasks successfully if an LLM does not have a sufficiently advanced level of mathematics encoded to comprehend its formalization task. E.g., if an LLM does not have any understanding of the notions such as "*compact*" and "*closed*", it is unreasonable to assume it will be able to autoformalize the natural-language statement: "*A subset of a compact metric space is compact if and only if it is closed.*". We elaborate further on these matters in Appendix D.

The fairest comparison of the knowledge contained in LLMs vs ITPs would be a brute-force comparison, where their entire knowledge is compared. Because evaluating LLM output that consists of university-level mathematics is difficult, it cannot be outsourced to some of the many commercial crowdsourcing services. Brute-force comparison, by comparing long lists of statements and proofs sourced from books, is thus not possible. To keep the evaluation effort reasonable, we, therefore, focus on a proxy benchmark for assessing knowledge: Freek Wiedijk's list, *Formalizing 100 Theorems*[4]. We motivate this choice and elaborate on how we test for knowledge inclusion in Section 3 and Appendix B, where we present a more detailed motivation for the use of Wiedijk's list.

In summary, our contributions are:

- A first analysis of how an LLM approach that digitizes and distills knowledge from many textbooks in an opaque manner compares to the largest formal libraries of mathematics;
- The first evaluation of LLMs on university-level mathematics using best-practice prompt engineering;
- A dataset that accompanies this LLM evaluation, where we collect information on how well the LLM did on each ITP-related item.

2 Related Work

LLMs are typically evaluated on high school-level or lower undergraduate-level mathematics [14, 21, 42]. Few articles evaluate LLMs on graduate-level mathematics [19]. Recently, there have been attempts to integrate LLMs with ITPs by designing datasets that contain formal and natural-language proofs side-by-side [1], respectively, to use LLMs to complete formal proofs [18]. Turning ITP systems into "dojos", e.g. [46], that allow for convenient extraction of proof traces (as well as other metrics that can help guide proof search) will further accelerate the convergence of LLMs and ITPs to automate mathematics.

[4] `https://www.cs.ru.nl/%7Efreek/100`

3 Methodology

3.1 ITP Proof Sources

The best way to compare the knowledge encoded in an LLM to the knowledge encoded in libraries accompanying an ITP would be, as mentioned, a complete comparison: For each item in an ITP library, a series of prompts are submitted to the LLMs of choice, testing whether the LLM is knowledgable about that item.

Human evaluation of advanced mathematics that approaches research level is expensive. We, therefore, focus on evaluating a smaller dataset of ITP library items to lower the evaluation effort: Our approach centers on Freek Wiedijk's list, *Formalizing 100 Theorems*[5], which is a popular progress tracker for formalization progress. This benchmark contains both well-known and difficult theorems, making their formalization highly non-trivial: at the time of writing, only 88 theorems from the list have been proven with the best-ranked ITP, Isabelle. (However, all ITPs taken together have proved 99 out of the 100 formalized theorems. The only exception is Fermat's Last Theorem[6].) Some of the theorems from this list, even though they are of low mathematical difficulty and can be found in undergraduate textbooks, have only recently been formalized, which highlights the importance of this benchmark. Examples of such easy theorems formalized late are Ptolemy's theorem or Stirling's formula[7]. From this set of theorems, we select 50 theorems randomly (see Figure 2 for a list of the precise theorems that were selected).

3.2 LLM Evaluation Protocol

The evaluation of the output of the language model is performed by the authors, who are all mathematicians. 10% of data points were randomly checked and verified in order to make sure that the evaluators' grade agrees with the verifiers' grade. We use GPT-4, via the API with temperature 0.7, as the LLM of choice since it has the best-reported performance among LLMs [19, 6], and Claude 2 (via the web[8]) as a fallback.

For each theorem, GPT-4 was asked, using standardized prompt templates, to complete the following three tasks:

1. **Statement.** State the full theorem (given only the name of the theorem from Wiedijk's list).
2. **Items.** Explain and define all constituent items from the theorems, i.e., all definitions that go beyond foundational mathematical objects like numbers or functions (for example, the concept of a derivative), which are non-trivial to formulate in ITP. We have noted a priori in our dataset what we understand to be non-foundational items that we expect to be given.
3. **Proof.** Prove the theorem. Then, reflect back if the proof was correct; otherwise, make corrections.

These are wrapped in the following prompt engineering approaches, which comprises some of best, recommended practices, as recommended in OpenAI's cookbook[9], OpenAI's GPT practices[10], see also [24], that we detail below in the complete pipeline:

A. **Impersonate.** At initialization time, we ask it to impersonate a professional mathematician.

B. **Proceed step by step.** We instruct the model twice to proceed step by step: The first time, when initializing it in the API (only available GPT-4), it should proceed in a stepwise manner throughout its entire interaction. We reinforce this by asking it for a second time when

[5] `https://www.cs.ru.nl/%7Efreek/100`

[6] Which is in the process of being formalized and was arguably added to this list merely as a joke [8].

[7] In case of Stirling's formula, in descending order of recency, a formalization was added to Lean's mathlib4 in 2023, to MetaMath in 2017, to Isabelle's Archive of Formal Proofs in 2016, and to the Coqtail library associated to Coq and to HOL Light in 2010, respectively – according to GitHub commit history of the relevant formalization file (except MetaMath, where the formalization of Stirling's formula is located at `https://us.metamath.org/mpeuni/stirling.html`.

[8] At the time of writing an API for Claude 2 is not publicly available.

[9] `https://github.com/openai/openai-cookbook`

[10] `https://platform.openai.com/docs/guides/gpt-best-practices`

we present the proof to it. We further reinforce it by asking it to present a skeleton of the theorem before producing the actual theorem.

C. **Reflect on the output.** After the LLM was asked to carry out the proof, we asked it to reflect on it in order to allow it to make any amendments. Asking the model to review its output has been shown to be a strategy that, in some cases, leads to correct answers on a second try [22]. Our rating protocol was such that a second, corrected attempt was rated as a 1.

We refer to Appendix C for a diagram illustrating the full pipeline outlined above, as well as the full prompts submitted to the LLM(s).

For each of these three sections (statement, items, proof[11]), we rate each output on a binary 0-1 scale, representing incorrect-correct. The reason for this choice of course rating is that either a piece of knowledge is in the LLM's mathematical library or it is not. Therefore, a fine-grained scale, such as that introduced in [19, 43] to rate how well an LLM responds to a prompt, is not required. We have evaluated 50 randomly selected theorems from the list of 100 theorems by F. Wiedijk in this way. For each of the 50 theorems, four prompts were provided to the LLM, with additional follow-up questions. In total, we, therefore, have rated more than 200 outputs of GPT. Additionally, if the output was not convincing, we followed up with questions as described in the next paragraph. Unless the response was perfect, in order to make sure authors engaged with the task, a short justification was asked from raters for the reason why a 0 or 1 was given.

Suppose the output is rated as 0 on one of the three sections (statement, item, proof[11]). In that case, our policy is to ask the prompt corresponding to that section again at most three times until it gets it right: We ask two times using ChatGPT, increasing temperature by 0.1 each time, preserving the previous chat interaction so that the LLM can make use of all the previous interactions. We customize these subsequent prompts to the error that was made and provide individualized feedback to assess whether we can steer the LLM in the right direction. On the last attempt, if no sufficiently good answer was given so far, our policy is to use Claude 2 to ask for a new generation of all the sections, as described above, until the desired one (because API access is not available at the time of writing for Claude 2, and we do not have a separate way to initialize it, we simply ask the "impersonation" as the first prompt, before we start prompting the LLM for the full statement etc.).

We elaborate in Appendix A on why we have not used techniques such as Chain-of-Thoughts nor why we haven't implemented voting strategies to generate proofs. See Appendix F for a specific example how a datapoint from the LLMKNOW dataset looks like.

We note that formal proofs are always larger than non-formalized proofs (which is captured by the de Bruijn factor [15, 44]). Moreover, during the process of formalization, small omissions from the original are regularly observed. In particular, this happens if the result is new or complex, as in the case of the more recent formalizations within the Archive of Formal Proofs:

- For undergraduate-level mathematics: For the proof of Gödel's Incompleteness Theorem, L.C. Paulson mentions in [35]: "[...] other technical problems had to be solved in order to complete the argument".
- For research-level mathematics: The authors mention in [17]: "Much of the effort in this project had not to do with the formalization itself but with ascertaining precisely what to formalize. Although this material is considered mathematics of central importance, sources are conflicting about the basic definitions".) One of the major benefits of using ITPs is to uncover such omissions and fix them.

Because the pen-and-paper proofs on which LLMs are trained also suffer from such omissions, an approach where an LLM is asked to construct a proof at a similar level of detail to that of an ITP, which contains various fixed omissions, would not be a fair comparison. The level of detail in proofs we aim for should, therefore, be similar to the level of detail in a pen-and-paper proof.

The total score per item is the mean of the individual scores. We note that it is necessary to collect all these scores independently: A correct response regarding, e.g., the proof of a theorem, does not –perhaps counterintuitively– a priori imply that the subsequent, easier questions will be answered correctly as well. The reason is that LLMs are known to act as stochastic parrots occasionally, which

[11]We rate the combined effect of both the first prompt asking for a proof, as well the second prompt, asking it to reflect on the proof, which sometimes leads to new, improved proofs.

in this case would mean they have simply memorized a proof but contain no "working knowledge" about the item that makes up that proof. (This is unlike the case for humans, where, if they quote the proof of a theorem correctly, in all likelihood, they will be familiar with mathematical concepts that that proof employs.)

We do not penalize an LLM for dubious outputs such as:

- returning more information than what was asked for (including if that information is wrong).
- getting minor details wrong, such as describing a counterexample that is, in effect, not a true counterexample.
- Being slightly vague at some stages in the proof.

The reason for this lower bar of tolerance is that our aim is not to have an LLM devise a perfect proof with logically flawless chains of reasoning. Rather, we allow it to get some of the reasoning wrong *as long as it can convince us that it has a good grasp of the knowledge of the mathematical objects involved in the statement of the theorem and its proof*. If we are confronted with proofs that aren't a clear 0 or 1 because some mistakes are present, we adopt the criterion of marking it as a 1 if we believe that progress was sufficiently good and understanding of the mathematical object sufficiently deep that by having longer interactions with the LLM we could ultimately get it to output a longer proof. What matters to us is that the right mathematical information can be elicited.

4 Results and Conclusion

Because both LLMs and ITPs are being rapidly developed, we do not focus on evaluating a single (LLM, ITP) pair but investigate whether, for each datapoint in the ITP library (specifically, the portion that overlaps with the scope of our benchmark), some LLM exists that matches it. In mathematical terms, we are interested in clarifying whether the following holds: $\forall\,\text{ITP}\ \exists\,\text{LLM}$: $\text{knowledge}(\text{LLM}) \approx \text{knowledge}(\text{ITP})$.[12]

Please see the end of the Appendix for an overview of the performance of GPT-4 on each of the 50 theorems after the first proof, which was the most challenging section. The average score on the proof section was 0.68, i.e., 68% of all the proofs were satisfactory to our standards. On the statements and items sections, scores were 94% and 98%, respectively. We have followed the majority in this case. We refer to Appendix E for further noteworthy observations about how well the LLM proved theorems.

Some theorems of the list of 100 theorems are actually collections of theorems, for example, in the case of "Lebesgue measure and integration". In this case, one ITP, HOL, seems to prove a specific lemma about the open halfplane[13], whereas the other ITPs define lebesgue measure and integral (as expected).

On Wiedijk's list, we are satisfied that the mathematical knowledge encoded in LLMs matches that of ITP libraries. Nonetheless, further investigations need to be carried out to ensure that LLMs possess sufficient mathematical knowledge (in order to carry out autoformalization, for example): Beyond for raw knowledge, it is important to assess whether LLMs are also capable of applying *proof techniques*. This is particularly relevant for the miniF2F dataset [48], which is relevant for autoformalization and uses problems from mathematical olympiads, on which it is known that LLMs struggle [19].

We conclude that future work is required to assess how strongly an LLM needs to be conditioned on mathematical knowledge in order to be able carry out various formalization tasks (autoformalization, proof generation, proof completion) successfully.

[12]The converse analysis, whether $\forall\,\text{LLM}\ \exists\,\text{ITP} : \text{knowledge}(\text{LLM}) \approx \text{knowledge}(\text{ITP})$ is not relevant for autoformalization, but would be appropriate for an effort of what could be called "autonaturalization", which would investigate whether an ITP can match the level of knowledge of an LLM. This is of inferior interest because almost all formal ITP theorems in existence are of human invention and thus are represented in natural language in books or texts. (Exceptions to this exist in the form of theorems of combinatorial flavor that are well-suited for automated theorem provers and have been discovered with their use, such as the solution to the Robbins Conjecture [29].) Since the informal, natural-language representation of a theorem is already available somewhere, it does not need to be deduced from formal material. Furthermore, some ITPs, such as Mizar, already have some support for presenting formal mathematics in a language similar to natural-language [2].

[13]`https://www.cs.ru.nl/~freek/100/hol.html#86`

References

[1] Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W Ayers, Dragomir Radev, and Jeremy Avigad. ProofNet: Autoformalizing and formally proving undergraduate-level mathematics. *arXiv preprint arXiv:2302.12433*, 2023.

[2] Grzegorz Bancerek, Czesław Byliński, Adam Grabowski, Artur Korniłowicz, Roman Matuszewski, Adam Naumowicz, and Karol Pąk. The role of the Mizar Mathematical Library for interactive proof development in Mizar. *Journal of Automated Reasoning*, 61:9–32, 2018.

[3] Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, et al. A multitask, multilingual, multimodal evaluation of ChatGPT on reasoning, hallucination, and interactivity. *arXiv preprint arXiv:2302.04023*, 2023.

[4] Alexander Bogomolny. Pythagorean theorem, Retrieved 2023-08-10. `https://www.cut-the-knot.org/pythagoras`.

[5] Anthony Bordg, Lawrence Paulson, and Wenda Li. Simple type theory is not too simple: Grothendieck's schemes without dependent types. *Experimental Mathematics*, 31(2):364–382, 2022.

[6] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with GPT-4. *arXiv preprint arXiv:2303.12712*, 2023.

[7] Kevin Buzzard. The future of mathematics?, 2019. Accessed on September 28, 2023.

[8] Kevin Buzzard. Formalising fermat, 2022. Accessed on October 28, 2023.

[9] Kevin Buzzard, Chris Hughes, Kenny Lau, Amelia Livingston, Ramon Fernández Mir, and Scott Morrison. Schemes in lean. *Experimental Mathematics*, 31(2):355–363, 2022.

[10] Davide Castelvecchi et al. Mathematicians welcome computer-assisted proof in 'grand unification'theory. *Nature*, 595(7865):18–19, 2021.

[11] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

[12] Wenhu Chen, Ming Yin, Max Ku, Elaine Wan, Xueguang Ma, Jianyu Xu, Tony Xia, Xinyi Wang, and Pan Lu. Theoremqa: A theorem-driven question answering dataset. *arXiv preprint arXiv:2305.12524*, 2023.

[13] Elizabeth Clark, Tal August, Sofia Serrano, Nikita Haduong, Suchin Gururangan, and Noah A Smith. All that's' human'is not gold: Evaluating human evaluation of generated text. *arXiv preprint arXiv:2107.00061*, 2021.

[14] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[15] N.G. de Bruijn. A survey of the Project Automath. reprinted from: Seldin, j. p. and hindley, j. r., eds., to h. b. curry: Essays on combinatory logic, lambda calculus and formalism, p. 579-606, by courtesy of academic press inc., orlando. In R.P. Nederpelt, J.H. Geuvers, and R.C. de Vrijer, editors, *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*, pages 141–161. Elsevier, 1994.

[16] Leonardo 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, editors, *Automated Deduction - CADE-25*, pages 378–388, Cham, 2015. Springer International Publishing.

[17] Chelsea Edmonds, Angeliki Koutsoukou-Argyraki, and Lawrence C Paulson. Formalising Szemerédi's regularity lemma and Roth's theorem on arithmetic progressions in Isabelle/HOL. *Journal of Automated Reasoning*, 67(1):2, 2023.

[18] Emily First, Markus N Rabe, Talia Ringer, and Yuriy Brun. Baldur: whole-proof generation and repair with large language models. *arXiv preprint arXiv:2303.04910*, 2023.

[19] Simon Frieder, Luca Pinchetti, Ryan-Rhys Griffiths, Tommaso Salvatori, Thomas Lukasiewicz, Philipp Christian Petersen, Alexis Chevalier, and Julius Berner. Mathematical capabilities of ChatGPT. *arXiv preprint arXiv:2301.13867*, 2023.

[20] John Harrison, Josef Urban, and Freek Wiedijk. History of interactive theorem proving. In *Computational Logic*, volume 9, pages 135–214, 2014.

[21] 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 J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1. Curran, 2021.

[22] 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. *arXiv preprint arXiv:2310.01798*, 2023.

[23] Albert Q Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. *arXiv preprint arXiv:2210.12283*, 2022.

[24] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in Neural Information Processing Systems*, 35:22199–22213, 2022.

[25] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857, 2022.

[26] Hanmeng Liu, Ruoxi Ning, Zhiyang Teng, Jian Liu, Qiji Zhou, and Yue Zhang. Evaluating the logical reasoning ability of ChatGPT and GPT-4. *arXiv preprint arXiv:2304.03439*, 2023.

[27] Patrick Massot and Jeremy Avigad. Mathematics in Lean. *GitHub repository*, 2023.

[28] The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, page 367–381, New York, NY, USA, 2020. Association for Computing Machinery.

[29] William Mccune. Solution of the Robbins Problem. *Journal of Automated Reasoning*, 19(3):263–276, Dec 1997.

[30] Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction – CADE 28*, pages 625–635, Cham, 2021. Springer International Publishing.

[31] OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[32] Lawrence C Paulson. Natural deduction as higher-order resolution. *The Journal of Logic Programming*, 3(3):237–258, 1986.

[33] Lawrence C. Paulson. A preliminary user's manual for Isabelle. *Computer Laboratory, University of Cambridge*, Report 133, 1988.

[34] Lawrence C Paulson. Isabelle: The next 700 theorem provers. In *Logic and computer science*, volume 31, pages 361–386. Citeseer, 1990.

[35] Lawrence C. Paulson. Gödel's incompleteness theorems. *Archive of Formal Proofs*, November 2013. `https://isa-afp.org/entries/Incompleteness.html`, Formal proof development.

[36] Peter Scholze. Half a year of the Liquid Tensor Experiment: Amazing developments, 2022. Accessed on October 3, 2023.

[37] Filip Smola and Jacques D. Fleuriot. Hyperdual numbers and forward differentiation. *Archive of Formal Proofs*, December 2021. `https://isa-afp.org/entries/Hyperdual.html`, Formal proof development.

[38] Christian Szegedy. A promising path towards autoformalization and general artificial intelligence. In Christoph Benzmüller and Bruce Miller, editors, *Intelligent Computer Mathematics*, pages 3–20, Cham, 2020. Springer International Publishing.

[39] Qingxiang Wang, Cezary Kaliszyk, and Josef Urban. First experiments with neural translation of informal to formal mathematics. In *Intelligent Computer Mathematics: 11th International Conference, CICM 2018, Hagenberg, Austria, August 13-17, 2018, Proceedings 11*, pages 255–270. Springer, 2018.

[40] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

[41] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-Thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc., 2022.

[42] Sean Welleck, Jiacheng Liu, Ronan Le Bras, Hannaneh Hajishirzi, Yejin Choi, and Kyunghyun Cho. NaturalProofs: Mathematical theorem proving in natural language. *arXiv preprint arXiv:2104.01112*, 2021.

[43] Sean Welleck, Jiacheng Liu, Ximing Lu, Hannaneh Hajishirzi, and Yejin Choi. NaturalProver: Grounded mathematical proof generation with language models. *Advances in Neural Information Processing Systems*, 35:4913–4927, 2022.

[44] Freek Wiedijk. The "de bruijn factor", 2012. Accessed on September 28, 2023.

[45] Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. *Advances in Neural Information Processing Systems*, 35:32353–32368, 2022.

[46] Kaiyu Yang, Aidan M Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. *arXiv preprint arXiv:2306.15626*, 2023.

[47] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

[48] Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021.

A Issues with In-Context Learning and Voting for Open-Ended LLM Outputs

In our evaluation pipeline, we ask the model to output a statement of the theorem, its constituent items, and a proof (see Figure 1 from Section C. Here, we indicate why in-context learning and voting do not apply to enhancing proof performance.

Various prompt-engineering techniques have emerged in the past year, some the most effective being in-context learning ones such as Chain-of-Thoughts (CoT) [41], or Tree-of-Thoughts (ToT) [47] that have established themselves as performance-enhancing techniques for various tasks, including math word problems [41] from the GSM8K dataset [14], games based on reasoning such as the 24 puzzle[14] [47], or numerical reasoning tasks [11]. These in-context learning techniques all revolve around letting a model elaborate longer on a task by making its reasoning explicit and using examples. This approach, therefore, works best when *problem categories* are large / *task instances* are many so that the model can be given a few examples of (annotated) problem-solution pairs before being prompted with another problem for which a solution is sought. Implementing such prompt-enhancing techniques by letting the model make its reasoning more explicit is difficult in a setting like ours, where *open-ended* outputs are needed. In particular, when outputting proofs, it is not straightforward how one could give the LLM an "example proof", as required by such an in-context learning technique, that does not yet contain all the essential ideas of the proof. Supposing this were possible, the diversity of proofs that the same statement can have[15] complicates matters further since it is not clear which proof to aim for when learning it in-context.

Voting [40, 25] has shown to offer performance benefits if an LLM has difficulties with a specific type of task. This technique works best where there is an easy criterion to judge whether the output in one run was similar to the output in another run, as aggregating them and establishing a majority automatically is straightforward. This is difficult for *open-ended* approaches like ours since LLM outputs are highly heterogeneous and cannot be reduced to a single answer. Executing such a voting technique would require significant manual inspection of the output: In two different runs, proofs might be produced that are both correct but different, and therefore, an automatic aggregation procedure will not work. Nonetheless, repeating a prompt can help elicit a good response. Therefore, we have used the approach to allow the LLM to reflect on its output, as this does not require comparisons of outputs.

We note that mathematical problems from datasets like MATH [21] or TheoremQA [12] have a much more constrained scope; in particular, both support the concept of a "final answer", which can easily be compared to a ground truth, which does not exist in the present article.

B Motivating Wiedijk's List of 100 Theorems

We argue on the following grounds that using this list of theorems is a reasonable test of whether LLMs' breadth of mathematical knowledge rivals that of ITPs:

1. To formulate each theorem in an ITP, all the concepts appearing in the proof statement need to be formalized first (and various properties about each of these concepts, as well as relations between them, need to be known as well, as they are used in the proof of the theorem[16]). We, therefore, require our assessment (outlined below, in Section 3.2) to recursively enquire the LLM about all concepts required to formulate the statement, similar to what a library of an ITP would contain. Because this list touches diverse areas of mathematics, it highlights the coverage of mathematics of ITP libraries well.
2. Because these 100 theorems are not tied to any specific ITP, we get a sense of how an LLM compares against all of the ITP libraries (Archive of Formal Proofs, mathlib4, etc.) combined. Furthermore, they are an entrenched metric that is often used by the community.
3. Previous research on the mathematical capabilities of LLMs involved creating datasets of mathematics that span high-school (GSM8K), undergraduate-level (MATH), and graduate-level or olympiad-style mathematics (GHOSTS [19]). From these, it is known that LLMs are

[14] `https://en.wikipedia.org/wiki/24_(puzzle)`

[15] An extreme example is the Pythagorean theorem with hundreds of known proofs [4].

[16] For example, in order to prove that a function $f : K \to \mathbf{R}$ on a compact, topological space K attains a minimum and a maximum, properties about compact spaces need to be known.

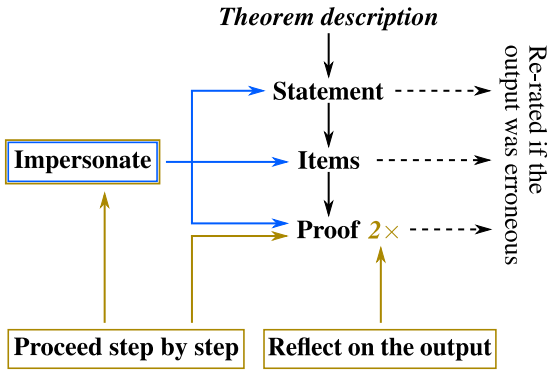

Figure 1: This figure shows the pipeline that was executed to evaluate the LLM output. The yellow boxes show which type of prompt engineering was performed and where it was applied. The blue boxes show which of these methods made use of API initialization. The "theorem description" is the description as taken from Wiedijk's list. The "2×" emphasizes that a second proof output is generated after asking the LLM to reflect on the proof. The dotted arrows indicate that, as described in Section 3, this part was re-rated by human assessors if the output was not satisfactory.

fairly proficient at reproducing undergraduate mathematics: E.g., the MATH and GHOSTS datasets cover a significant part of mathematical objects that typically appear in an undergraduate curriculum. This alleviates us of the duty to re-check whether LLMs possess such knowledge of undergraduate mathematics encoded by ITPs, which can be assumed to be present. It frees us to investigate major theorems that are present in Wiedijk's list (which are not present in MATH or GHOSTS, that focus mostly on LLMs proving much smaller, "unnamed" theorem).

C Prompt Templates and Data Generation Pipeline

Figure 1 shows the entire pipeline that was used to obtain the LLM output, starting from Wiedijk's list, as well as where prompt engineering was performed and of which type it was. The human component in this pipeline solely pertains to rating the output.

The following prompt templates were used for each of the three sections, statement-items-proof (where for the proof section, we use two prompts in order to allow the LLM to reflect on the output):

Statement-prompt (where `{theorem}` is the theorem name from Wiedjik's list):

```
The following is a well-known statement or theorem: "{theorem}". Provide
a full, complete, and explicit formulation of it, of the form "*Theorem.
...*", as you would encounter it in a textbook.
```

Items-prompt:

```
Please explain all the individual, constituent concepts that make up
this complete and explicit formulation of the theorem. If all of the
constituent concepts are basic, such as numbers of functions, do not
elaborate on them.
```

Proof-prompt-main:

```
Now provide a proof of the statement. First, describe in a single
paragraph a skeleton of the proof. Then try to create a proof step by step
until you either succeed or have no idea how to proceed further.
```

Proof-prompt-reflect:

```
Now, take stock of what you have proved previously, reflect on it and make
any potential corrections.
```

D On the Importance of Assessing LLMs' Ability to Autoformalize

Autoformalization [39, 38] has seen a big push in the last years, and this trend is expected to accelerate, as LLMs have recently been used to perform autoformalization [45, 23]. We recall:

1. The goal of autoformalization is to turn a mathematical text written in the usual, informal style of natural language into a formal text that is parseable by an ITP;
2. Mathematical scenarios frequently arise in which humans make implicit operations (e.g., type conversions between the same number that is both a natural and a real number) that have to be made explicitly for ITP (which is only possible if the proof that is to be formalized is well understood in the first place).

Therefore, it is highly plausible that any LLM-like model, were it able to autoformalize a natural-language statement/proof in a target ITP, would need to be intimately familiar[17] with the mathematical concepts and objects that are involved in that statement/proof in order to fill in the various gaps that any natural-language statement/proof invariable contains and to connect the mathematical objects presented in natural language with the corresponding formal ones, already present in the (library associated to) an ITP.

To the best of our knowledge, existing research works on LLM-guided autoformalization and proof completion/generation have operated under the assumption that LLMs possess sufficient mathematics to carry out the assigned tasks. Nonetheless, even advanced models such as GPT-4 have been shown to perform poorly on olympiad-level mathematical problems (e.g., see the GHOSTS dataset [19]), so it is not plausible that older LLMs, such as PaLM and Codex, which are used by [45] as baselines, have a deep understanding of the natural-language version of the proofs of the statements from the miniF2F dataset, which deals with problems sourced from mathematical competitions. Therefore, it is unclear whether the generally modest state-of-the-art performance that has been achieved so far (e.g., autoformalization success rates were less than 35% on the miniF2F dataset, see Table 3 from [45]) is due to the inherent complexity related to formalization, or whether LLMs simply do not encode a sufficiently advanced body of mathematics (definitions, proof techniques, etc.), on top of which various formalization-related tasks are to be carried out.

E Noteworthy Observations

In just three cases, GPT-4 wasn't able to generate a convincing formulation of the theorem directly from the theorem name, as given on the list of 100 theorems. In just one case, it was not able to explain all constituent items from a theorem formulation. The fault in this case was the ambiguous name of the theorem, *Ascending or Descending Sequences*, which, after inspecting the ITP source, revealed itself to be actually the Erdős–Szekeres theorem[18]

Even though ChatGPT does well on some more complicated theorems, it could not produce a satisfactory proof for the Pythagorean theorem. It was very close on its second try, though it did not produce the correct picture from its own instruction. Interestingly, on the third try, it attempted to mix two different methods of proving the theorem but failed in the end.

The Königsberg problem can be understood as showing that no Eulerian path exists for the concrete example of Königsberg or that an Eulerian path exists whenever the path is connected and every vertex has an even degree. ChatGPT tried the latter, but the second part of the proof was incomplete.

We also noted that sometimes it did not went into sufficient detail in proofs, such as for the *The Area of a Circle* theorem, where it initially gave only an intuitive argument from which we were not able to assess whether it had some operational knowledge of the concepts involved in proving what the area of the circle was (see Appendix F).

[17] We note that the wording that an LLM is "familiar" with a piece of mathematics is, strictly speaking, a case of anthropomorphization. Nonetheless, we feel this abuse of language is acceptable since 1) it is difficult to avoid it, as even the expression that an LLM "learns" is a case of it, and 2) it is clear what is meant: It is plausible that if the training data contains sufficiently many cases where the involved mathematical objects appear, the LLM will learn some form of the true mathematical relationships between the objects and be able to manipulate them in a somewhat mathematically consistent way—similar to how sufficiently large and advanced LLMs learn to use correct grammar [13].

[18] `https://en.wikipedia.org/wiki/ErdÅŚsâĂŞSzekeres_theorem`

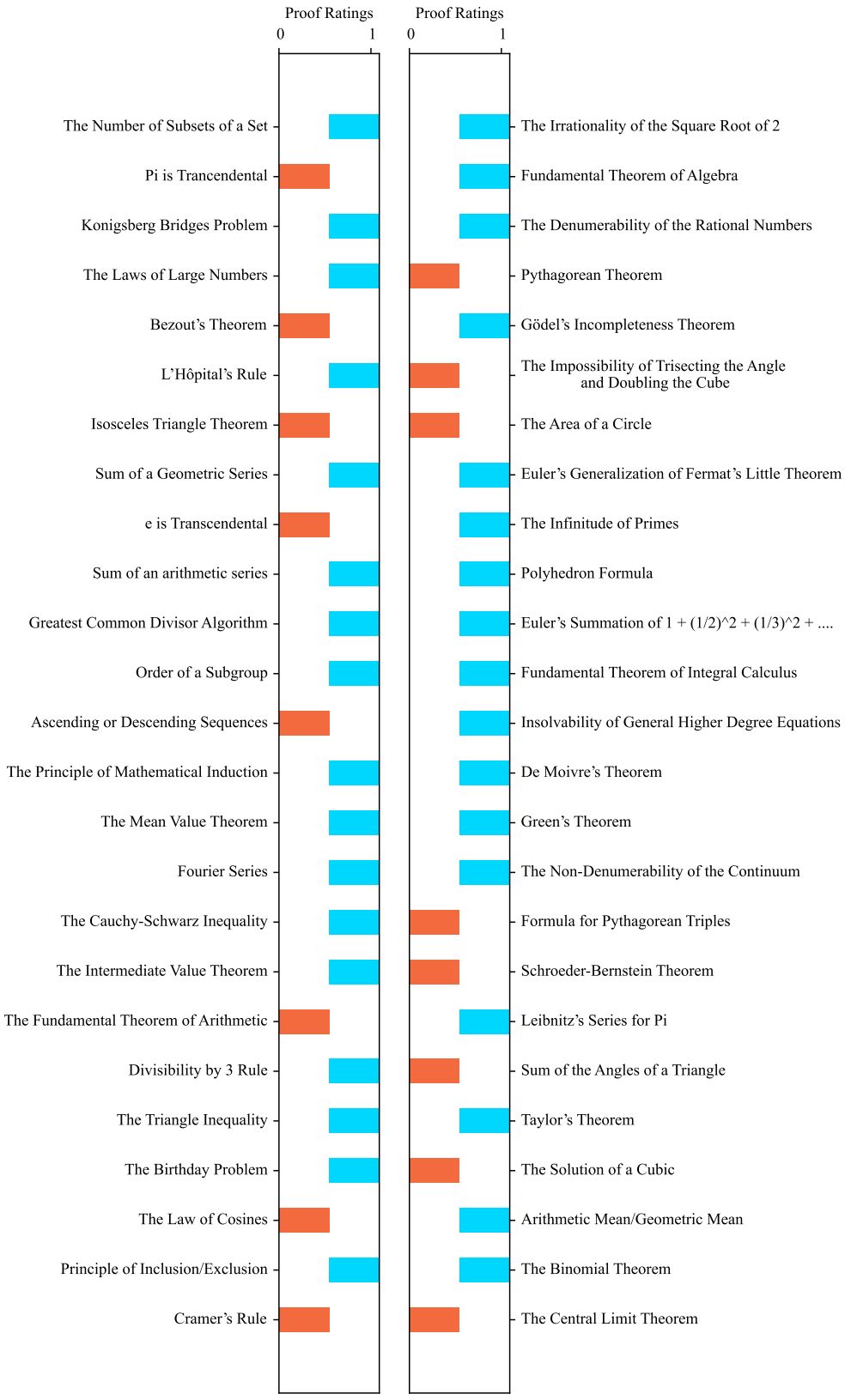

Figure 2: The ratings of all 50 theorems after the first proof attempt, carried out on GPT-4. An incorrect proof is denoted by 0, while a proof that is convincing enough that the LLM understood the concepts it was talking about and combined them in a meaningful manner (even if the proof may be slightly erroneous) is denoted by 1.

F Dataset

We highlight how one datapoint from the LLMKNOW dataset looks like:

- The value of the `theorem_name` key is the name of the theorem from Wiedijk's list.
- The value of the `expected_items` key is a list of items the raters expected the LLM to find.
- The value of `statement` is a list of list, where the first entry is `1` if the statement was correct, `0` otherwise. The second entry is a comment the rater may or may not make. There may be, in total, at most four lists within this list, one for each attempt. If a `1` is encountered, then no further lists should be present.
  Analogous considerations hold for the `items` and `proof` key.

Here is how a single JSON datapoint related to the theorem *The Area of a Circle* looks like from our dataset.

```
"theorem_name": "The Area of a Circle",
"expected_items": ["real numbers", "measure"],
"statement": [[1,""]],
"items": [[1, ""]],
"proof": [
[0, "it gives the intuitive proof, not the formal proof"],
[0, "it claims that small section of the area has certain form (r^2 *
    d\theta) without justification (also incorrectly), then
proceeds to integrate and changes the result once again to land on
    the correct result in the end. Just integrate 2*sqrt(r^2 - t^2)
    dt from -r to r without polar coordinates and stuff, you can do
    this!"],
[1, ""]]
```