AI-Assisted Autoformalization of Combinatorics Problems in Proof Assistants

Long Doan
Department of Computer Science
George Mason University
Fairfax, VA, USA

ThanhVu Nguyen
Department of Computer Science
George Mason University
Fairfax, VA, USA

***Abstract*—Proof assistants such as Coq and LEAN have been increasingly used by renowned mathematicians to formalize and prove mathematical theorems. Despite their growing use, writing formal proofs is challenging, and even the first step of stating the problem formally is difficult as it requires a deep understanding of these systems' languages. Recent advancements in AI, especially large language models (LLMs), have shown promise in automating this formalization task. However, domains such as combinatorics pose significant challenges for AI-assisted proof assistant systems due to their cryptic nature and the lack of existing data to train AI models. We introduce AutoForm4Lean, a system designed to leverage LLMs to aid in formalizing combinatorics problems for LEAN. By combining LLMs with techniques from software engineering and formal methods such as validation and synthesis, AutoForm4Lean generates formalizations of combinatorics problems more effectively than the current state-of-the-art LLMs. Moreover, this project seeks to provide a comprehensive collection of formalized combinatorics problems, theorems, and lemmas, which would enrich the LEAN library and provide valuable training data for LLMs. Preliminary results demonstrate the effectiveness of AutoForm4Lean in formalizing combinatorics problems in LEAN, making a step forward in AI-based theorem proving.**

***Index Terms*—Proof assistants, Autoformalization, Combinatorics, AI, LLM, Lean**

I. Introduction

Proof assistants such as Coq [1] and LEAN [2] have been used to formalize and verify many well-known mathematical theorems. Celebrated achievements include using Coq to establish the classical Four Color Theorem [3] and more recently using LEAN to formalize and prove the Polynomial Freiman-Ruzsa conjecture [4], specialized theorems in the Liquid tensor experiment [5], and many more [6]. Once mainly developed for program reasoning, proof assistants—the poster child of formal methods, software engineering, and programming languages—have now been adopted and encouraged by the mathematical community, with many users being Fields Medalists, e.g., Terence Tao, Vladimir Voevodsky, Peter Scholze, and Kevin Buzzard.

However, similarly to writing programs, writing formal proofs can be challenging as it requires a deep understanding of the proof assistant's language, which can be cryptic and unfamiliar to many users. In recent years, the advent of large language models (LLMs) in AI/ML has led to significant advancements in software engineering (SE) and formal method (FM) techniques, e.g., aiding program analysis, verification, and synthesis [7]–[10]. Unsurprisingly, researchers have leveraged LLMs to enhance proof assistants, e.g., by automating proof generation [11], [12]. Various projects [13]–[16] have integrated AI, e.g., as "copilot", to assist mathematicians in using LEAN. Most recently, the new AlphaProof and AlphaGeometry systems from Google DeepMind combine LEAN with reinforcement learning to solve four out of six problems at the 2024 International Math Olympiad (IMO) [17], becoming a major milestone in the field of AI-assisted theorem proving.

Despite much progress in using AI+SE/FM in various mathematical domains including number theory, algebra, and geometry, domains such as *combinatorics* (e.g., subset counting and graph theory) remain challenging for AI-assisted theorem proving. For example, the two IMO problems that AlphaProof and AlphaGeometry failed to solve were combinatorics problems. Combinatorics is of course fundamental in math and computer science and has applications in cryptography, coding theory, and optimization [18]–[20]. However, while properties and theorems in combinatorics are easy to state informally (e.g., in English), they are *notoriously difficult to formalize*, even when done manually by human experts. For example, while the `mathlib` library [21] in LEAN has successfully formalized many major concepts in algebra, calculus, and topology, its collection of theorems and lemmas in combinatorics is underdeveloped and misses major combinatorics concepts including generating functions and recurrence relations. This limitation not only hinders using combinatorics in LEAN, but it also makes it difficult to develop sufficient training data for using AI to assist in combinatorics theorem proving.

In this NIER paper, we propose a new system, AutoForm4Lean, that leverages LLMs to assist in formalizing combinatorics problems in the LEAN proof assistant. AutoForm4Lean combines the power of LLMs with SE/FM techniques (e.g., synthesis, bug fixing, and counterexample generation) to synthesize, validate, and refine formalizations of combinatorics problems. We have developed a AutoForm4Lean prototype, and preliminary results show that AutoForm4Lean can generate correct formalizations of combinatorics problems with a much higher success rate than existing LLM-based systems.

In addition to automatically formalizing combinatorics problems, the AutoForm4Lean project aims to provide a new dataset of combinatorics problems in LEAN to help train

LLMs to better understand this mathematical domain. We will contribute our results to the LEAN community, e.g., to enrich its `mathlib` library with formalized combinatorics concepts and theorems.

II. Background and Example

Consider the example in Fig. 1, in which we want to translate the combinatorial statement on the number of subsets fitting a criterion (left-hand side) to a proper LEAN statement. To achieve this, our proposed system, AutoForm4Lean, uses a combination of LLM and LEAN to iteratively generate and refine candidate formalizations, and then check for correctness.

A. Autoformalizing

Autoformalization, the task of automatically translating statements into proper syntax for proof assistant, is a well-known problem in the literature [16], [22]–[24]. Unsurprisingly, recent and state-of-the-art autoformalization techniques leverage LLM to synthesize such statements (e.g., [24], [25] for LEAN). However, despite the many successes, such approaches do not work well with combinatorics problems. For instance, we apply the latest LLM GPT-4o [26] from OpenAI (the maker of ChatGPT) to our counting example and ask it to candidate statements in LEAN. Out of 64 candidates generated, 61 do not compile correctly (due to various forms of syntax or notation errors), and only *three* (4.7%) are *syntactically* correct.

Writing syntactically correct formulation is similar to a programmer writing code that does not compile correctly, in which case they will look at the compiler error messages (e.g., `incorrect type on line` X), make the appropriate fixes, and repeat until the program is compiled cleanly. Using this idea, AutoForm4Lean consists of an iterative "bug-fixing" approach that integrates LLM to synthesize candidate LEAN code and LEAN to check for errors. If errors exist, AutoForm4Lean uses LEAN feedback to help LLM improve its synthesis process.

When applied to the counting example, our AutoForm4Lean prototype was able to generate 2 correct LEAN statements from just 10 candidates in two iterations. In the first iteration, AutoForm4Lean uses LLM to generate 5 candidates, which are all rejected by LEAN. The tool then gathers error messages from LEAN and in the second iteration instructs LLM to generate 5 new candidates from the previous ones, taking into account the error feedback. This time AutoForm4Lean got 2 candidates that LEAN successfully compiles. Fig. 1 summarizes the process: in the 1st (initial) iteration, AutoForm4Lean obtains error messages and uses them as feedback to generate the correct candidates in the 2nd iteration. Notice how the errors (shown in red in the candidates in the first iteration) were modified to be the correct ones (shown in green in the candidates in the second iteration).

B. Correctness of Formalization

Generating syntactically correct statements is one thing, ensuring that they are correctly translated from the original statement is another. This difference is analogous to a program being syntactically correct (passes the compilation state) and semantically correct (behaves as expected), with the latter is often much more challenging.

For the two candidates generated in Fig. 1, only the first one, `count_valid_subsets_general`, correctly formulates and checks for valid subsets via the function `valid_subset_general` (not shown). The second one, while being a valid LEAN statement, is incorrect and does not even check for valid subsets(does not even use the set S of the input problem).

AutoForm4Lean includes an LLM-based checker to determine and filter incorrect candidates. Specifically, the LLM checks that the candidate satisfies several requirements such as if it defines and uses mathematical objects described in the original problem (e.g., set, type, variables) and contains specific combinatoric details from the original problem.

In our example, the first candidate satisfies all requirements. It uses elements and variables from in the original problem (e.g., set S and variable k) and defines a helper function `valid_subset_general` to capture the validity of a subset. However, the second candidate fails because it does not use element sets or check subsets.

III. Dataset Construction

AutoForm4Lean relies on LLMs for autoformalization and correctness checking of combinatorics problems. The power of LLMs comes from the data they are trained on. However, LLM data for combinatorics is underdeveloped—in fact, even in non-LLM scenarios, topics related to combinatorics are relatively scarce in proof assistants like LEAN [21]. Among two large-scale datasets MMA [27] and Lean Workbook [28] on LEAN formalizations, MMA is just an extraction of the `mathlib` library with few combinatoric instances, and only 893/57,000+=1.5% examples in Lean Workbook are combinatorics problems, a very small number for any meaningful LLM training.

AutoForm4Lean tackles this problem by using an LLM to generate synthetic data for LLM. Fig. 2 shows the process. First, we give the LLM existing combinatoric problems (the informal "English" problems) as seeds and ask it to generate new problems. We then attempt to falsify (i.e., disprove) the generated problems by using LLM to find counterexamples. If the falsifier fails to find a counterexample, we add the problem to our collection. Next, we integrate the collection into the autoformalization (i.e., iterative bug-fixing, §II-A) and correctness checking process (§II-B) in AutoForm4Lean to obtain LEAN formalization of combinatoric problems. These data are combined to form our dataset and then are used to train LLMs specialized in combinatorics problems.

A. Problem Generator

The problem generation component focuses on generating combinatorics problems which are used to train LLMs. While we can use existing combinatoric problems from textbooks and existing math problems databases, they are limited in terms of

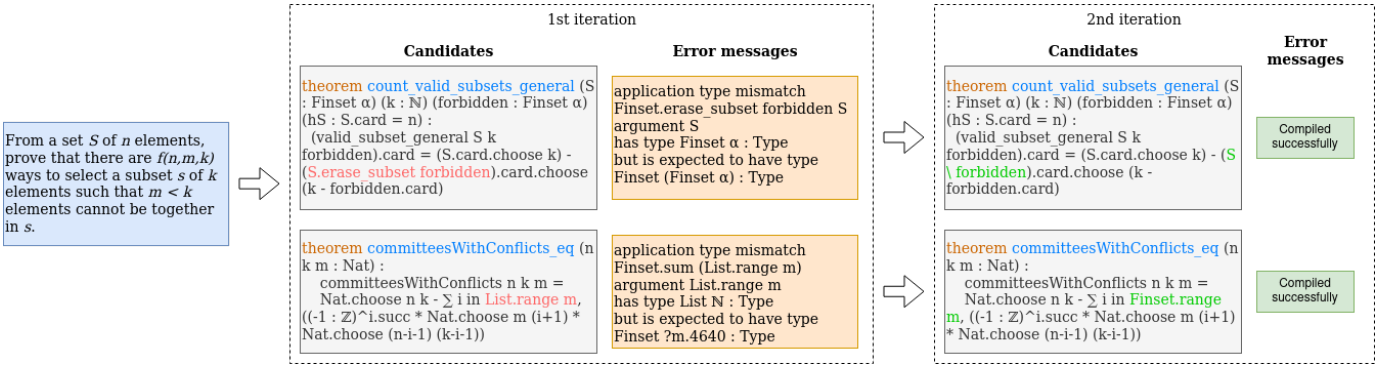

Fig. 1. Autoformalization in AutoForm4Lean. The errors are highlighted in red, while the fixes are highlighted in green.

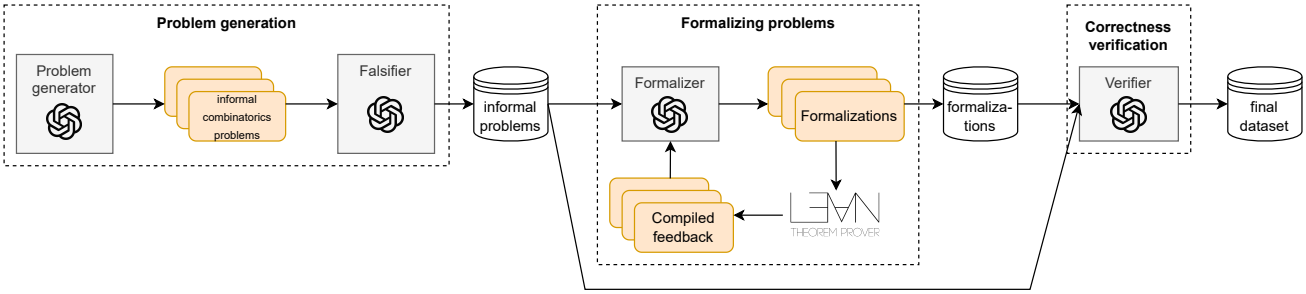

Fig. 2. Overview of AutoForm4Lean.

quantity and accessibility. Moreover, they are often concrete problems that are not suitable for formalization. For example, "A 6-person committee is formed from 10 people, with two people refusing to work together. How many ways can the committee be formed?" has concrete numbers that are not related to any combinatoric statements or theorems.

To address this issue, the problem generator component of AutoForm4Lean generates *abstract combinatorics problems* from these concrete problems by generalizing concrete values to variables. For example, AutoForm4Lean turns the concrete problem above to: "From a set S of n elements, select a subset s of k elements. If m elements cannot be together in s, how many ways can we select s?". Next, AutoForm4Lean turns this into a proof problem: "Prove that from a set S of n elements, there are $f(n, m, k)$ ways to select a subset s of k elements such that $m < k$ elements cannot be together in s" that can be formalized in LEAN.

B. Problem Falsifier

LLMs often have the hallucination issue [29], which in this context means generating combinatoric problems that are meaningless and incorrect. We thus need to sanitize the generated problems.

To do this, AutoForm4Lean employs a *falsifier* component to disprove generated problems by finding counterexamples, e.g., concrete inputs that falsify the proof. The falsifier aims to synthesize a *program implementation* of the problem in Python and several test cases (including edge/corner cases). If the program fails any of the tests, the problem is disproved and AutoForm4Lean discards it.

For example, after generating the proof question in §III-A, AutoForm4Lean synthesizes a program to represent it with three functions: one function to count the number of valid subsets, one that implements the formula of $f(n, m, k)$, and a (driver) function that compares the outputs of the two functions. AutoForm4Lean next generates concrete test cases for the program, e.g., $(n = 10, m = 2, k = 6)$, and removes the proof question as it makes the program fail.

After removing invalid problems, we now have a dataset of combinatoric problems suitable for LEAN formalization (Fig. 2). We then use this dataset to train LLMs specialized in combinatorics problems.

C. Autoformalization and Checking

Once a dataset of combinatorics problems is obtained, we use it to train LLMs for autoformalization and correctness checking as shown in Fig. 2 and §II. Here we provide additional details on these two processes.

a) Autoformalization: For the iterative bug-fixing process for autoformalization (§II-A), AutoForm4Lean first generates a list of n candidates formalizations and puts them in a queue. For each loop iteration, AutoForm4Lean selects a candidate p from the queue and invokes LEAN to compile p. If the compilation succeeds, AutoForm4Lean adds p to its combinatorics database. Otherwise, AutoForm4Lean uses LLM to analyze the error message from LEAN and synthesize a repaired candidate p' and puts it back into the queue. The loop

terminates when either the queue is empty or AutoForm4Lean reaches a maximum number of iterations.

We note that error messages from LEAN can be ambiguous and difficult to debug. For example, in the first iteration of Fig. 1, the error messages of the second candidate contain a meta-variable "?m.4640", which is cryptic and difficult to understand. Fortunately, AutoForm4Lean was able to analyze the whole error message, determine that there is a type mismatch, and provide a repair by changing from "List.range m" to "Finset.range m".

b) Correctness Checking: §II-B discusses the importance of correctness in formalization. To ensure the formalization matches the original informal problem, AutoForm4Lean uses an LLM-based checker to evaluate the correctness of the formalization based on several criteria:

- Type correctness: Are all mathematical objects (e.g., sets, functions, variables) properly typed and consistent with the original problem?
- Completeness: Are helper definitions (e.g., functions, abbreviations, constructions) properly formalized (with proofs where applicable)?
- Faithfulness: Does the formalization accurately reflect the meaning of the original problem?
- Logical correctness: Are there any logical errors in the formalization? (e.g., a helper function that is well-defined but logically incorrect).

IV. Preliminary Results

We developed an AutoForm4Lean prototype and used it to generate a small dataset of combinatorics problems and their formalizations. This dataset is then used to train the underlying LLMs for autoformalization and correctness checking. Our preliminary results show the effectiveness of AutoForm4Lean, which generates 10/10 syntactically correct formalizations, 8 of which are semantically correct.

Our dataset is created using the problem generation and falsification components described in Fig. 2. Specifically, we collect 10 combinatorics problems from math textbooks and contest forum[1]. Next, we apply AutoForm4Lean to generate formalizations (problem generation) and attempt to invalidate them by creating counterexamples (problem falsification). About 40% of the generated problems were falsified, and in the end, we obtain a dataset of 100 combinatorics problems.

Next, we explore several ways to generate LEAN formalizations. In this preliminary study, we compare three approaches: (i) oversampling, where we generate 64 formalization candidates without iterative bug-fixing, (ii) iterative, where we generate 5 initial candidates with iterative bug-fixing, and (iii) iterative + in-context learning (ICL) [30], where we add examples from our dataset to manually guide the LLM's behaviors (i.e., add these examples to the LLM prompt as a form of demonstration).

Table I shows the results, with # succ. comp meaning successfully compiled by LEAN (i.e., syntactically correct) and correct (LLM) and correct (human) indicating the number of correct formalizations as judged by the LLM-based verifier and a human evaluator (us). Here, we see that even with a "tiny" dataset of 100 problems, AutoForm4Lean was able to achieve promising results. Even with oversampling where no iterative bug-fixing is applied, 7/10 formalizations were successfully compiled, and 6/10 were considered correct by the LLM-based verifier and human evaluator. The iterative approach outperforms the oversampling approach as expected, with 9/10 successfully compiled and 6/10 considered correct by the human evaluator. Finally, the iterative + ICL approach outperforms the other two, with 10/10 formalizations successfully compiled and 8/10 formalizations considered correct.

Tab. I
Preliminary results.

	# suc. comp.	# correct (LLM)	# correct (human)
Oversampling	7/10	6/10	6/10
Iterative	9/10	6/10	6/10
Iterative + ICL	**10/10**	**7/10**	**8/10**

These initial results demonstrate the effectiveness of AutoForm4Lean, even with a limited dataset. We note that as our dataset grows, we anticipate that the benefits of ICL may diminish. With a larger corpus of formalized combinatorics problems, training a specialized LLM is likely to become a more effective strategy.

V. Conclusion and Future plans

We have presented AutoForm4Lean, a system that uses LLMs and SE/FM techniques to autoformalize and check the correctness of combinatorics problems for the LEAN proof assistant. Our preliminary results show that AutoForm4Lean is effective in generating correct formalizations of combinatorics problems compared to the state-of-the-art.

We are working on generating a larger and more diverse dataset of combinatorics problems to train LLM to reason about combinatorics problems and improve our AutoForm4Lean system. Autoformalization is just the first step, and we next plan to use the generated dataset and ideas here to synthesize the full proof steps in LEAN for combinatorics problems, with the goal of helping mathematicians to fully formalize and prove their combinatorics work.

Acknowledgment

We thank the anonymous reviewers for their helpful comments. This material is based in part upon work supported by the National Science Foundation under grant numbers 2422036, 2319131, 2238133, and 2200621, and by an Amazon Research Award.

References

[1] The Coq Development Team, "The Coq reference manual – release 8.19.0," https://coq.inria.fr/doc/V8.19.0/refman, 2024.

[1]https://artofproblemsolving.com/community

[2] L. De Moura, S. Kong, J. Avigad, F. Van Doorn, and J. von Raumer, "The lean theorem prover (system description)," in *Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25*. Springer, 2015, pp. 378–388.

[3] G. Gonthier, "Formal proof—the four-color theorem," *Notices Amer. Math. Soc.*, vol. 55, no. 11, pp. 1382–1393, 2008.

[4] Tereance Tao, "A digitisation of the proof of the Polynomial Freiman-Ruzsa Conjecture in Lean 4," 2024. [Online]. Available: https://teorth.github.io/pfr/

[5] D. Castelvecchi, "Mathematicians welcome computer-assisted proof in 'grand unification'theory," *Nature*, vol. 595, no. 7865, pp. 18–19, 2021.

[6] Lean Community, "100 theorems," 2024. [Online]. Available: https://leanprover-community.github.io/100.html

[7] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago *et al.*, "Competition-level code generation with alphacode," *Science*, vol. 378, no. 6624, pp. 1092–1097, 2022.

[8] D. Castelvecchi, "Are chatgpt and alphacode going to replace programmers?" *Nature*, 2022.

[9] S. Chakraborty, S. K. Lahiri, S. Fakhoury, M. Musuvathi, A. Lal, A. Rastogi, A. Senthilnathan, R. Sharma, and N. Swamy, "Ranking llm-generated loop invariants for program verification," *arXiv preprint arXiv:2310.09342*, 2023.

[10] K. Pei, D. Bieber, K. Shi, C. Sutton, and P. Yin, "Can large language models reason about program invariants?" in *International Conference on Machine Learning*. PMLR, 2023, pp. 27 496–27 520.

[11] E. First, M. Rabe, T. Ringer, and Y. Brun, "Baldur: Whole-proof generation and repair with large language models," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 1229–1241.

[12] P. Lammich, "Generating verified llvm from isabelle/hol," in *10th International Conference on Interactive Theorem Proving (ITP 2019)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2019.

[13] K. Yang, A. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. J. Prenger, and A. Anandkumar, "Leandojo: Theorem proving with retrieval-augmented language models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[14] S. Welleck and R. Saha, "Llmstep: Llm proofstep suggestions in lean," *arXiv preprint arXiv:2310.18457*, 2023.

[15] J. M. Han, J. Rute, Y. Wu, E. Ayers, and S. Polu, "Proof artifact co-training for theorem proving with language models," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=rpxJc9j04U

[16] Z. Li, J. Sun, L. Murphy, Q. Su, Z. Li, X. Zhang, K. Yang, and X. Si, "A survey on deep learning for theorem proving," *arXiv preprint arXiv:2404.09939*, 2024.

[17] AlphaProof and A. teams, "Ai achieves silver-medal standard solving international mathematical olympiad problems," https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/, 2024.

[18] F. Ruskey, "Combinatorial generation," *Preliminary working draft. University of Victoria, Victoria, BC, Canada*, vol. 11, p. 20, 2003.

[19] M. Saračević, S. Adamović, and E. Bišević, "Application of catalan numbers and the lattice path combinatorial problem in cryptography," *Acta Polytechnica Hungarica*, vol. 15, no. 7, pp. 91–110, 2018.

[20] D. T. Dao, "Applications of combinatorics in coding theory," 2023.

[21] "Mathlib library," https://leanprover-community.github.io/mathlib4_docs/, last accessed: October 8th, 2024.

[22] C. Kaliszyk, J. Urban, and J. Vyskočil, "Automating formalization by statistical and semantic parsing of mathematics," in *Interactive Theorem Proving: 8th International Conference, ITP 2017, Brasília, Brazil, September 26–29, 2017, Proceedings 8*. Springer, 2017, pp. 12–27.

[23] Q. Wang, C. Kaliszyk, and J. 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*. Springer, 2018, pp. 255–270.

[24] Z. Azerbayev, B. Piotrowski, H. Schoelkopf, E. W. Ayers, D. Radev, and J. Avigad, "Proofnet: Autoformalizing and formally proving undergraduate-level mathematics," *arXiv preprint arXiv:2302.12433*, 2023.

[25] A. Agrawal, S. Gadgil, N. Goyal, A. Narayanan, and A. Tadipatri, "Towards a mathematics formalisation assistant using large language models," *arXiv preprint arXiv:2211.07524*, 2022.

[26] "Hello GPT-4o," https://openai.com/index/hello-gpt-4o/, last accessed: October 7th, 2024.

[27] A. Q. Jiang, W. Li, and M. Jamnik, "Multilingual mathematical autoformalization," *arXiv preprint arXiv:2311.03755*, 2023.

[28] H. Ying, Z. Wu, Y. Geng, J. Wang, D. Lin, and K. Chen, "Lean workbook: A large-scale lean problem set formalized from natural language math problems," *arXiv preprint arXiv:2406.03847*, 2024.

[29] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin *et al.*, "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions," *arXiv preprint arXiv:2311.05232*, 2023.

[30] T. B. Brown, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.