CombiBench: Benchmarking LLM Capability for Combinatorial Mathematics

Junqi Liu [*1,2] **Xiaohan Lin** [*2,3] **Bolton Bailey** [4] **Jonas Bayer** [4] **Yaël Dillies** [4,5] **Weijie Jiang** [6] **Xiaodan Liang** [3] **Roman Soletskyi** [4] **Haiming Wang** [2] **Yunzhou Xie** [4,7] **Beibei Xiong** [6] **Zhengfeng Yang** [6] **Jujian Zhang** [4,7] **Lihong Zhi** [1] **Zekai Zhu** [4] **Jia Li** [4] **Zhengying Liu** [2]

Abstract

Neurosymbolic approaches integrating large language models with formal reasoning have recently achieved human-level performance on mathematics competition problems in algebra, geometry, and number theory. In comparison, combinatorics remains a challenging domain, characterized by a lack of appropriate benchmarks and theorem libraries. To address this gap, we introduce CombiBench, a comprehensive benchmark comprising 100 combinatorial problems, each formalized in Lean 4 and paired with its corresponding informal statement. The problem set covers a wide spectrum of difficulty levels, ranging from middle school to IMO and university level, and spans more than ten combinatorial topics. CombiBench is suitable for testing IMO-solving capabilities since it includes all IMO combinatorics problems since 2000 (except IMO 2004 Problem 3, whose statement contains an image). Furthermore, we provide a comprehensive and standardized evaluation framework, dubbed *Fine-Eval* (for **F**ill-in-the-blank **in** **L**ean Evaluation), for formal mathematics. It accommodates not only proof-based problems but also, for the first time, the evaluation of fill-in-the-blank questions. Using Fine-Eval as the evaluation method and Kimina Lean Server as the backend, we benchmark several LLMs on CombiBench. State-of-the-art LLMs solve only 48% of CombiBench problems, highlighting its complexity and importance for LLM evaluation.

[1]Academy of Mathematics and Systems Science, University of Chinese Academy of Sciences [2]Moonshot AI [3]Sun Yat-sen University [4]Project Numina [5]Stockholm University [6]East China Normal University [7]Imperial College London . Correspondence to: Jia Li <jia@projectnumina.ai>, Zhengying Liu <liuzhengying@moonshot.cn>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

1. Introduction

Automated theorem proving (ATP) has long been a prominent research area at the intersection of artificial intelligence and mathematics (Bibel, 2013), aiming to develop computer programs capable of automatically verifying or discovering mathematical proofs. In recent years, the integration of large language models (LLMs) with formal theorem provers such as Lean (2021), Coq (2023), and Isabelle (2008) has received increased attention. These formal systems can rigorously verify the proof generated by an LLM step by step and provide feedback, thereby enhancing the models' performance in automated theorem proving.

In this research area, the design of high-quality benchmark tests is essential, as they offer valuable guidance for improving model capabilities. Three well-known competition-based benchmarks in automated theorem proving include miniF2F (Zheng et al., 2021), FIMO (Liu et al., 2023) and PutnamBench (Tsoukalas et al., 2024). miniF2F primarily features high school-level mathematics problems and competition questions, FIMO consists of algebra and number theory problems from the International Mathematical Olympiad (IMO), and PutnamBench originates from the prestigious North American university-level Putnam competition. Next to these, ProofNet (2023) is a benchmark for autoformalization and formal proving of undergraduate-level mathematics.

However, each of these benchmarks has notable limitations. For instance, miniF2F is limited to high school mathematics, FIMO and ProofNet are written in the (now outdated) formal proof language Lean 3 (de Moura et al., 2015), and PutnamBench includes only a small number of combinatorial problems. It is also worth noting that a Lean 4 version of ProofNet is provided in DeepSeek-Prover-V1.5 (Xin et al., 2024b). We analyze the number and proportion of combinatorics problems in existing benchmarks in Table 1, as well as their applicability to Lean 4.

The above formal mathematics benchmarks in Lean 4 scarcely cover combinatorial mathematics. Moreover, combinatorics remains one of the weakest areas for current auto-

Table 1. Benchmark comparison of combinatorics problems

BENCHMARK	COUNT	RATIO	LEAN 4 FRIENDLY
MINIF2F	0	0	😄
FIMO	0	0	😭
PUTNAMBENCH	29	4.4%	😄
PROOFNET	0	0	😄
COMBIBENCH	**100**	**100%**	😄

mated theorem-proving systems.

In 2024, Google DeepMind's AlphaProof reportedly solved 4 out of 6 problems at the International Mathematical Olympiad (IMO) using Lean 4, achieving a level comparable to human silver medalists; notably, the two unsolved problems were combinatorial.

In 2025, ByteDance's Seed-Prover reportedly achieved an IMO silver-medal score (30 points), fully solving 4 out of 6 problems during the contest and obtaining a partial score on one additional problem; again, the remaining unsolved problems were combinatorial. Notably, Seed-Prover reportedly completed a full solution to one of these combinatorial problems after the contest, suggesting progress in this domain.

Taken together, these two IMO case studies highlight a consistent bottleneck: across consecutive years, the problems that remained unsolved by leading formal-systems-based approaches were combinatorial.

This outcome can be attributed to three primary factors:

- The absence of robust benchmarks to assess models' proficiency in combinatorial mathematics;
- The limited coverage of combinatorial content within Lean's math library—Mathlib (mathlib Community, 2020).
- In combinatorial mathematics, the gap between informal and formal is larger than in other areas. For a specific problem, it is often necessary to add some definitions that are unique to that problem.

CombiBench aims to address these gaps by providing testing standards for evaluating the capabilities of large language models in this area.

Besides, traditional formal math benchmarks (such as miniF2F, FIMO) primarily consist of theorem-proving tasks, where evaluation methods focus on assessing whether a model can complete a proof. However, PutnamBench introduces a new standardized formalization for fill-in-the-blank problems, which existing evaluation approaches cannot handle effectively. Unlike standard proof-based problems, fill-in-the-blank questions require the model to construct a solution and verify its correctness. Current evaluation methods are insufficient for this problem, focusing solely on proving given propositions without addressing solution generation and validation. To bridge this gap, we developed a novel evaluation framework, named FINE-EVAL, to assess model performance on fill-in-the-blank problems. This approach adapts evaluation techniques to new problem formats and provides a more comprehensive measure of a model's mathematical reasoning abilities, particularly in solution construction and verification.

The contributions of this work are outlined as follows:

- We introduce CombiBench, a benchmark of 100 combinatorics problems formalized in Lean 4, paired with their corresponding informal statements. The benchmark spans a wide range of difficulty levels (from middle school to IMO and university level) and covers more than ten combinatorial topics.
- We propose a standardized evaluation framework for formal mathematics that can assess both proof-based problems and fill-in-the-blank questions.

2. Related work

2.1. IMO

The International Mathematical Olympiad (IMO) is one of the most prestigious and competitive global mathematics competitions for high school students. Established in 1959, the IMO aims to challenge and inspire young mathematicians worldwide. Each year, teams of up to six students from over 100 countries participate in the event, which consists of solving six highly complex mathematical problems over two days. The problems span various areas of mathematics, including algebra, geometry, number theory, and combinatorics, requiring not only deep mathematical knowledge but also creativity and problem-solving skills. Many individuals who have won the Fields Medal—the highest honor in mathematics—have participated in the IMO. More recently, the annual IMO competition has also become widely recognized as a grand challenge for Artificial Intelligence (AI) (Challenge, 2019). Artificial Intelligence Mathematical Olympiad (AIMO) has set up a $10 million prize to reward the first AI system that reaches the level of an IMO gold medalist (Prize, 2023).

2.2. Automated Theorem Proving

Early research in automated theorem proving focused on first-order theorem provers designed for simple logical frameworks (2002) and theorem provers based on symbolic

engines (Chou et al., 2000). With the advent of deep learning technologies, there has been a significant shift toward using large language models (LLMs) to automate the theorem-proving process.

GPT-f (2020) is an automated theorem prover and proof assistant developed for the Metamath formalization language. Its introduction marks one of the first instances in which the formal mathematics community accepted and incorporated proofs generated by deep learning. The emergence of GPT-f has significantly advanced the field of automated theorem proving. Subsequently, Draft, Sketch, and Prove (DSP) (2022) was introduced, using informal proofs to aid the generation of formal ones. LEGO-Prover (2023a) builds a library through modular formal proofs, allowing LLMs to retrieve existing skills and generate new ones during the proof process. DT-Solver (2023b) introduces a dynamic-tree Monte Carlo search algorithm. BFS-Prover (Xin et al., 2025) uses a best-first search approach and attained state-of-the-art performance among theorem provers based on search algorithms.

The aforementioned work employs a search-based approach to predict the next step in the proof. Recently, progress in this area has led to an alternative approach, in which the language model generates the entire proof directly. DeepSeek-Prover (2024a) and Goedel-Prover (2025) both use this method; in particular, DeepSeek-Prover incorporates both search-based and whole-proof approaches. Despite notable progress, these methods face significant challenges.

While LLMs excel at pattern matching and sequence generation, effectively capturing the deep, structured, and often non-linear reasoning required for complex formal proofs remains difficult. Kimina-Prover Preview (2025) proposed a novel reasoning-driven exploration paradigm for formal theorem proving, using reinforcement learning to enhance whole-proof generation. After Kimina-Prover Preview, the research paradigm in this area increasingly shifted toward combining informal and formal reasoning, leading to systems such as Goedel-Prover-V2 and Seed-Prover.

More recently, Seed-Prover 1.5 introduced an agentic prover architecture to better bridge the gap with natural-language reasoning by treating Lean as a tool and allowing the prover to autonomously invoke multiple tools during the proof process.

Besides trained theorem-proving models, agentic approaches have also become increasingly popular. For example, Hilbert reports strong performance on PutnamBench by combining informal reasoning with formal verification in an agentic framework.

Numina-Lean-Agent further demonstrates the potential of this direction: it reports solving all Putnam 2025 problems and matching the reported full-score performance of a closed-source system (AxiomProver). It also reports outperforming other systems, including the closed-source Seed-Prover 1.5 and Aristotle.

2.3. Existing Benchmarks

With the rapid advancement of automated theorem proving, several formal mathematical benchmarks in the Lean language have been introduced in recent years. miniF2F is a benchmark designed to evaluate automated theorem-proving systems in various formal systems. It includes various mathematical problems, such as exercises from prestigious math olympiads (AMC, AIME, IMO) and high school and undergraduate coursework. The main objective of miniF2F is to provide a standardized benchmark to directly evaluate and compare theorem-proving systems. Initially implemented in Lean and Metamath, it has since been extended to HOL Light and Isabelle, broadening its applicability across different formal proof environments. The current state-of-the-art result in the miniF2F test set is 99.6%, achieved by Seed Prover.

For combinatorial problems, miniF2F and FIMO focus exclusively on algebra and number theory problems from the IMO. PutnamBench includes 26 combinatorial problems, but its coverage of combinatorial topics (e.g. countability, power sets, discrete structures, games) is limited. LeanComb (2025) introduces a data enhancement approach and offers a benchmark specifically focused on combinatorial identities. The proofs of these identities often involve analytical techniques rather than purely combinatorial reasoning.

3. CombiBench

CombiBench is the first benchmark focused on combinatorial competition problems written entirely in Lean 4. It is a manually produced benchmark that includes 100 combinatorial mathematics problems of varying difficulty and knowledge levels.

Range of Topics. For the selection of topics in combinatorics, we follow Brualdi's classical combinatorics textbook "Introductory Combinatorics" (Brualdi, 2004). The book consists of fourteen chapters and is widely used in undergraduate and graduate courses in combinatorics.

- What Is Combinatorics?
- Permutations and Combinations
- The Pigeonhole Principle
- Generating Permutations and Combinations
- The Binomial Coefficients
- The Inclusion-Exclusion Principle and Applications

- Recurrence Relations and Generating Functions
- Special Counting Sequences
- Systems of Distinct Representatives
- Combinatorial Designs
- Introduction to Graph Theory
- More on Graph Theory
- Digraphs and Networks
- Pólya Counting

The book systematically introduces the fundamental concepts, methods, and applications of combinatorics, covering many important topics such as permutations and combinations, the pigeonhole principle, generating functions, graph theory, and combinatorial designs.

Composition and Diversity. CombiBench consists of 10 easy problems from https://www.hackmath.net/, 42 exercises from Brualdi's book, 36 IMO problems, and 12 problems from other math competitions, as summarized in Table 2. This composition spans a broad range of difficulty, from introductory to IMO-level and beyond, reflecting good diversity in difficulty.

Table 2. Problem source distribution

SOURCE	COUNT
HACKMATH	10
BRUALDI'S BOOK	42
IMO	36
APMO	2
BALTICWAY	1
EGMO	1
IMO-SHORTLIST	4
IZHO	2
BXMO	1
USAMO	1

For the IMO portion, we started from all IMO combinatorics problems since 2000 (37 problems in total). Two of these problems include images (Problem 3 from 2004 and Problem 5 from 2023), which complicates direct formalization. After inspection, we found that the image in IMO 2023 Problem 5 is purely illustrative and does not add information beyond the text, so we kept it; we excluded IMO 2004 Problem 3, resulting in 36 IMO problems.

For the exercises from Brualdi's book, we randomly sampled problems from 14 chapters, choosing three problems from each chapter. This ensured that the 42 problems were evenly distributed across all 14 chapters, helping to guarantee the diversity of the topics covered in our selection.

Some problems (e.g., IMO 2024 Problems 3 and 5) have already been formalized in Mathlib. For these, we directly reuse the Mathlib statements in CombiBench, along with any auxiliary definitions required by the statements.

Naming Convention. Depending on the source of the problems, we adopt different naming conventions. For the 10 easy problems, we name them `hackmath_1`, `hackmath_2`, etc. Exercises from Brualdi's book are named using the format `brualdi_#chapter_#number`. Each competition problem is named as "`#competition_#year_p#number`". For instance, "`imo_2019_p5`" refers to the IMO Problems 2009 Problem 5, and "`imo_2019_p5_1`" refers to the first sub-question of the IMO Problems 2009 Problem 5. And "`brualdi_ch8_6`" refers to Problem 6 from the exercises section of Chapter 8 in Brualdi's book.

Classification of Problems. In CombiBench, 45% of the problems require first providing a solution and then proving its correctness, such as the following problem:

- Determine the number of permutations of $\{1, 2, \ldots, 8\}$ in which no even integer is in its natural position.

Therefore, these problems do not directly state a proposition and are not naturally formalized without first introducing a candidate answer. Previous benchmarks, such as miniF2F and FIMO, sidestep this issue by rewriting the statement to prove that a given solution satisfies the constraints. However, this rewriting reduces the overall difficulty, since a key challenge is often deriving the solution itself. PutnamBench introduces a standardized formalization method for such problems that better preserves the difficulty of the original informal statement. The following is an example of formalizing the above problem in PutnamBench style:

```
abbrev brualdi_ch6_11_solution : ℕ := sorry

/--
Determine the number of permutations of
    {1,2,...,8} in which no even integer is
    in its natural position.
-/
theorem brualdi_ch6_11
    (sols : Finset (Equiv.Perm (Finset.Icc
    1 8)))
    (h_sols : ∀ σ, σ ∈ sols ↔ (∀ i, Even
    i.1 → σ i ≠ i)) :
    sols.card = brualdi_ch6_11_solution :=
    by sorry
```

Challenges. Our formalization team consists of five doctoral students and one master's student, each with more than one year of Lean experience; the team includes a major Mathlib contributor and a Mathlib reviewer. During formalization, we observed that (beyond the simplest problems)

most problems require more than 30 minutes to formalize. For International Mathematical Olympiad (IMO)-level problems, nearly all require more than 3 hours, and some take more than 8 hours. Overall, these numbers suggest that IMO-level combinatorics remains challenging and time-consuming to formalize in Lean.

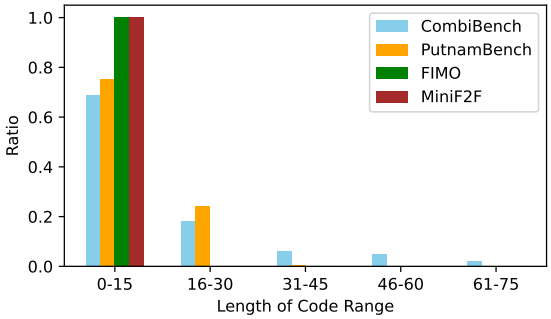

Figure 1. Code length distribution

One reason for this difficulty is the limited coverage of combinatorics in Mathlib, which often forces us to introduce problem-specific definitions and lemmas. We analyze the lengths of the formal statements in CombiBench, miniF2F, PutnamBench, and FIMO. Specifically, we exclude blank lines, comments, and lines beginning with `import` or `open`, and count only the remaining lines that constitute the problem statement. Figure 1 summarizes the results. All formalizations in FIMO and miniF2F are within 15 lines. In contrast, PutnamBench and CombiBench contain many statements with 16–30 lines, and statements exceeding 30 lines are almost exclusively from CombiBench. These results suggest that formalizing combinatorics competition problems typically requires substantially more problem-specific infrastructure.

We also count statement lengths across all CombiBench problems (again excluding blank lines). More than half of the problems exceed 10 lines, more than a quarter exceed 20 lines, and the longest statement contains 67 lines. We include this longest example in Appendix B.

Data Quality.

We ensure correctness at both syntactic and semantic levels. Lean automatically checks that each formal statement is well-formed and type-correct. Semantic correctness (faithfulness to the informal problem) cannot be fully automated, so each completed formalization is reviewed by at least two team members; in addition, we use a language model to assess semantic consistency between the informal statement and the formalization. As an additional sanity check, we use Kimina-Prover-72B to run Pass@128 sampling on the negation of each problem statement; none of these negations were proven.

Human expert proofs provide an important reference point but are expensive: for a Lean user with over one year of experience, producing a complete formal proof of an IMO combinatorics problem typically requires roughly 20 hours. Fully proving the entire benchmark would therefore be prohibitively costly. Nevertheless, as a small-scale validation, we completed formal proofs for more than half of the problems.

To prevent data leakage, we do not release ground-truth proofs in our public GitHub and Hugging Face repositories. We ask submitters to provide full proofs when reporting results on CombiBench; we verify them in Lean and then add them to the leaderboard after manual review. If submitters wish to share proofs publicly, we recommend packaging/encrypting them and releasing sufficient information for reproducibility (e.g., model weights, API endpoints, technical reports, and code). We will continue incorporating community feedback and maintaining CombiBench over time.

We also run experiments to model the relationship between problem difficulty and model discrimination. We measure the success rate on each problem in the theorem-proving task and analyze the results to demonstrate that CombiBench possesses strong discriminatory power in C.

4. Evaluation

We construct experiments to grasp CombiBench's difficulties with state-of-the-art theorem-proving approaches and general-purpose LLMs. Previous evaluation methods include whole-proof generation and tree search. Whole-proof generation directly queries a language model to generate a complete proof, while tree-search methods gradually build the evidence by searching for possible proof strategies. Both methods work well for verifying proof problems, but cannot verify fill-in-the-blank problems. To address the issue, we propose a new evaluation pipeline to verify both types of problems in Lean 4 named FINE-EVAL(**F**ill-in-the-blank **in** **L**ean Evaluation). We formalize the matching of the LLM prediction answer and the ground truth into a theorem-proving problem to rigorously verify the fill-in-the-blank question and avoid the complicated rules and format requirements as approaches in the math word problem.

4.1. Evaluation method

FINE-EVAL interacts with two servers, LLM and Lean. We deployed Kimina Lean server(2025) on a 64-core 512G machine and called it in parallel with 10 workers. LLM takes as input an entire formal statement with the proof and the blanks to be filled in replaced by '`sorry`'s, and writes a snippet of complete Lean 4 code. Since LLMs sometimes cheat by commenting out all the code to pretend it

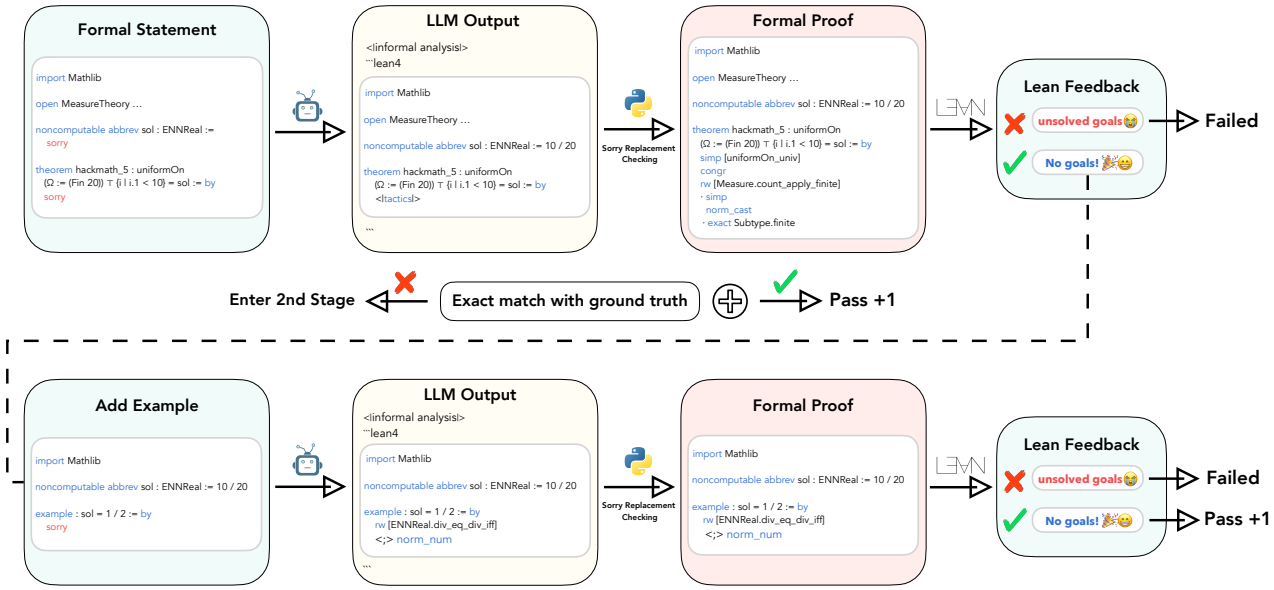

Figure 2. Pipeline of the two-stage FINE-EVAL.

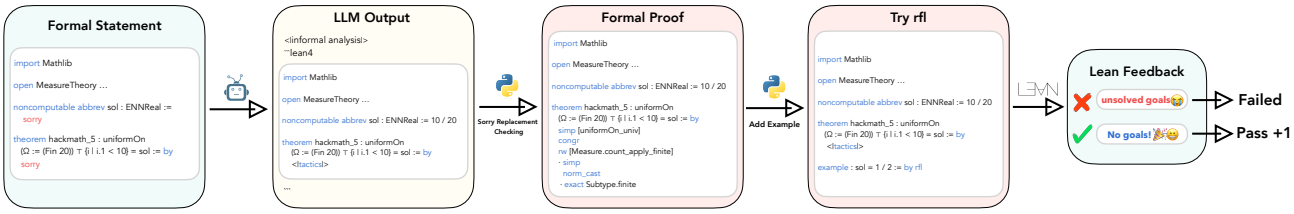

Figure 3. Pipeline of the one-stage FINE-EVAL.

passed compilation, we show an example in the appendix E. Therefore, we need to check the code after comments are removed.

To ensure that verification in Lean is sound and not achieved by introducing additional assumptions, we restrict the allowed axiom set. Let $\mathcal{T}$ be the set of all type-checked theorems in Lean, and let $\mathcal{A}$ denote the universe of all valid axioms. We define the standard axiomatic basis as

$$\mathcal{B} = \{\texttt{propext}, \texttt{Classical.choice}, \texttt{Quot.sound}\}.$$

Let $\mathrm{Ax} : \mathcal{T} \to \mathcal{P}(\mathcal{A})$ be the dependency function that maps a theorem $t \in \mathcal{T}$ to the set of axioms required for its proof. We accept a proof only if it is verified relative to the standard library, i.e.,

$$\mathrm{Ax}(t) \subseteq \mathcal{B}.$$

In practice, we inspect $\mathrm{Ax}(t)$ using the Lean command #print axioms t.

Any output that fails to compile in Lean, contains unresolved `sorry`, or depends on axioms outside $\mathcal{B}$ is considered a failure. Otherwise, we treat the proof as successful and then check whether the solution provided by the model matches the ground truth. If the solution and ground truth are exactly matched, we believe that the model has successfully solved the problem. If not, we then try to verify that the answer that LLM predicts is equivalent to the ground truth. To avoid unnecessary LLM calls, we construct a formal statement as follows, asserting that "`xx_solution` = `ground_truth`" and try to prove it using two common tactics: '`rfl`' and '`norm_num`'.

```
example : imo_2006_p2_solution =
    ground_truth := by
    try rfl
    try norm_num
```

If the lean verifier returns that it cannot be proved, we consider that the matching answer is non-trivial and add the statement to the input of the first round of LLM to prove it again. We use the same criterion as the first phase to determine whether the LLM's proof is successful. Figure 2 demonstrates the process of FINE-EVAL. In the first stage,

Table 3. Evaluation results on CombiBench across different computational budgets and methods reveal that all tested approaches perform poorly, solving only a few problems at most.

Model	*with solution*			*without solution*		
	Pass@1	Pass@8	Pass@16	Pass@1	Pass@8	Pass@16
Reasoning Model						
Claude-3.7-Sonnet-thinking(Anthropic, 2025)	0	2	2	0	0	0
DeepSeek-V3.1(Guo et al., 2025)	0	2	2	**1**	2	2
Gemini-2.5-pro-preview(Google, 2025)	0	2	4	0	2	3
Theorem Prover						
Kimina-Prover-72B(Wang et al., 2025)	**2**	**6**	8	**1**	**4**	**7**
Goedel-Prover-V2-32B(Ren et al., 2025)	2	6	8	-	-	-
DeepSeek-Prover-V2-671B(Ren et al., 2025)	-	-	10	-	-	-
Seed-Prover(Chen et al., 2025b)		**30** (Accumulative)		-	-	-
Seed-Prover 1.5(Chen et al., 2025a)		**48** (Accumulative)		-	-	-

the model answers "10 / 20" and the corresponding proof at once, which can be verified by the lean 4 server. Next, we try to use "norm_num" and "rfl" to automatically prove that 10 / 20 is equivalent to 1 / 2. When these attempts fail, we enter the second stage and let the model try to prove this problem, and the model fully demonstrates that the answer it predicts is equivalent to the ground truth. To prevent models from first submitting a trivial solution, then deceptively eliciting the ground truth to subsequently construct the actual proof, we impose a length limit on the second-stage proof. The number of characters in the output proof, after removing spaces and newline characters, must not exceed 42.

In practical applications, the two-stage verification method can often be complex. Consequently, we also provide a simplified evaluation approach, as illustrated in Figure 3. This simplified method initially follows the same procedure as the first stage of the Fine. However, if an exact match fails, we directly employ the `rfl` tactic to verify if the solution generated by the model is definitionally equivalent to the ground truth. This enables the evaluation of fill-in-the-blank problems using only a single Large Language Model (LLM) call. While this approach imposes stricter requirements on the model's predicted answer, it is more easily embedded into other workflows.

Metric For fill-in-the-blank questions, we treat a complete two-stage FINE-EVAL as a single verification. Following PutnamBench, we also conducted a comparative experiment, replacing the corresponding sorry with ground truths and letting the model complete the proof. We report pass@N at different sample budgets as the performance metric.

4.2. Baseline results

To demonstrate the significant challenge that CombiBench poses to LLMs, we evaluate it on different LLMs using various computational budgets. We consider two types of models, one is the model fine-tuned on the automated theorem proving task (indicated by "theorem prover"), the size of such models is usually 7B or less (except Kimina-Prover Preview) and the other is the general reasoning model (indicated by "reasoning model"). We evaluate pass@16 of these models on CombiBench using whole-proof generation. Considering the economic cost and reproducibility, we do not evaluate with a larger sample budget and tree-search method. We set temperature 1.0 during evaluation, and use a maximum tokens length equals 16,000. The system prompt and user prompt are shown in A.

The experimental results are presented in Table 3. We found that all models solved only a few problems except Seed Prover, likely due to the absence of formal combinatorial question libraries and the inherent difficulty of such questions. None of the theorem-proving models were able to solve all problems, which we attribute to the limited generalization capabilities of models when facing out-of-distribution data. The general reasoning model answered a few questions correctly, and its performance showed minimal difference between the with_solution and without_solution settings, likely due to its strong informal mathematical reasoning ability. In the without_solution case, both evaluation methods mentioned earlier produced identical results. This observation implies that, due to the model's current capability limitations, the solutions predicted are typically either completely correct or entirely wrong. As a result, sophisticated or flexible answer-checking mechanisms are less critical at this stage.

After evaluating three state-of-the-art open-source theorem provers—Kimina-Prover, DeepSeek-Prover v2, and Goedel-Prover v2—we observe that all of them are able to solve only a very limited number of problems on CombiBench. One possible explanation is that these open-source provers were developed approximately half a year ago, and thus

their coverage and reasoning capabilities in combinatorial mathematics remain insufficient.

In contrast, Seed Prover exhibits a markedly different level of performance. Benefiting from nearly a year of continuous development, Seed Prover 1.5 successfully solves 48 out of the 100 problems in the CombiBench test set (48%), substantially outperforming all other evaluated provers by a wide margin.

This performance gap can be largely attributed to architectural differences. Unlike the open-source provers, the closed-source Seed-Prover employs a significantly more sophisticated and fine-grained workflow, supporting advanced mechanisms such as lemma decomposition. In particular, Seed-Prover v1.5 has evolved into a fully agentic prover, in which autonomous planning and coordination play a central role. This increased level of autonomy appears to be a key factor behind its substantial performance improvement.

5. Conclusion

This paper introduces CombiBench, a formal-language benchmark for evaluating the capability of AI models on combinatorics competition problems, developed in Lean 4. CombiBench spans a variety of topics in combinatorics, featuring problems that range in difficulty from high school level to IMO and university-level. It is the first comprehensive benchmark specifically designed to assess the capability of language models in solving combinatorial mathematics problems.

Additionally, we introduce a new evaluation method for fill-in-the-blank problems, employing a two-stage approach. In the first stage, the model is required to generate a solution and prove its correctness. If the filled-in solution is equal to the ground truth and the proof compiles successfully, the problem is deemed solved. Even if the solution differs from the ground truth, it might still be a valid mathematical solution. In this case, the evaluation moves to a second stage, where the model must demonstrate that its proposed solution is equivalent to the ground-truth answer.

Our experimental results indicate that CombiBench remains challenging, while also reflecting rapid progress over time. For example, DeepSeek-Prover-V2 and Seed-Prover 1.5 were released roughly eight months apart: DeepSeek-Prover-V2 solved 10 out of 100 problems, while Seed-Prover 1.5 solved 48 out of 100 problems (48%). Despite this improvement, most problems remain unsolved by existing models. We identify two primary factors contributing to these failures:

- Absence of combinatorial mathematics content in existing theorem libraries: models struggle with formalization because they lack pre-built definitions, lemmas, and theorems relevant to combinatorial topics, forcing them to construct everything from scratch.
- The significant gap between natural language problem statements and formalized proofs: In combinatorial mathematics, the transition from informal to formal reasoning is particularly challenging, and current models struggle to bridge this gap effectively.

Looking ahead, we will contribute the newly formalized definitions from CombiBench to Mathlib and concurrently develop a dedicated theorem library for combinatorics. Beyond providing a benchmark, we view CombiBench and FINE-EVALas concrete infrastructure that already supports measurable progress in the community (e.g., the improvement from 10/100 to 48/100 over eight months), and we expect it to continue accelerating research by enabling reproducible comparisons and rapid iteration.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Anthropic. Claude 3.7 sonnet, 2025. URL https://www.anthropic.com/news/claude-3-7-sonnet.

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.

W. Bibel. *Automated theorem proving*. Springer Science & Business Media, 2013.

R. A. Brualdi. *Introductory combinatorics*. Pearson Education India, 2004.

I. G. Challenge. IMO Grand Challenge — imo-grand-challenge.github.io. https://imo-grand-challenge.github.io/, 2019. [Accessed 01-06-2024].

J. Chen, W. Chen, J. Du, J. Hu, Z. Jiang, A. Jie, X. Jin, X. Jin, C. Li, W. Shi, et al. Seed-prover 1.5: Mastering undergraduate-level theorem proving via learning from experience. *arXiv preprint arXiv:2512.17260*, 2025a.

L. Chen, J. Gu, L. Huang, W. Huang, Z. Jiang, A. Jie, X. Jin, X. Jin, C. Li, K. Ma, et al. Seed-prover: Deep and broad reasoning for automated theorem proving. *arXiv preprint arXiv:2507.23726*, 2025b.

S.-C. Chou, X.-S. Gao, and J.-Z. Zhang. A deductive database approach to automated geometry theorem proving and discovering. *Journal of Automated Reasoning*, 25(3):219–246, 2000.

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*, pages 378–388. Springer, 2015.

Google. Gemini 2.5: Our most intelligent ai model, 2025. URL https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/#gemini-2-5-thinking.

D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

A. Q. Jiang, S. Welleck, J. P. Zhou, W. Li, J. Liu, M. Jamnik, T. Lacroix, Y. Wu, and G. Lample. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. *arXiv preprint arXiv:2210.12283*, 2022.

Y. Lin, S. Tang, B. Lyu, J. Wu, H. Lin, K. Yang, J. Li, M. Xia, D. Chen, S. Arora, et al. Goedel-prover: A frontier model for open-source automated theorem proving. *arXiv preprint arXiv:2502.07640*, 2025.

C. Liu, J. Shen, H. Xin, Z. Liu, Y. Yuan, H. Wang, W. Ju, C. Zheng, Y. Yin, L. Li, M. Zhang, and Q. Liu. Fimo: A challenge formal dataset for automated theorem proving, 2023.

T. mathlib Community. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, POPL '20. ACM, Jan. 2020. doi: 10.1145/3372885.3373824. URL http://dx.doi.org/10.1145/3372885.3373824.

L. d. Moura and S. Ullrich. The lean 4 theorem prover and programming language. In *Automated Deduction–CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings 28*, pages 625–635. Springer, 2021.

S. Polu and I. Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.

Prize. AIMO Prize — aimoprize.com. https://aimoprize.com/, 2023. [Accessed 01-06-2024].

Z. Ren, Z. Shao, J. Song, H. Xin, H. Wang, W. Zhao, L. Zhang, Z. Fu, Q. Zhu, D. Yang, et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

M. D. Santos, H. Wang, H. de Saxcé, R. Wang, M. Baksys, M. Unsal, J. Liu, Z. Liu, and J. Li. Kimina lean server: Technical report. *arXiv preprint arXiv:2504.21230*, 2025.

S. Schulz. E–a brainiac theorem prover. *Ai Communications*, 15(2-3):111–126, 2002.

The Coq Development Team. The Coq Proof Assistant, Sept. 2023.

G. Tsoukalas, J. Lee, J. Jennings, J. Xin, M. Ding, M. Jennings, A. Thakur, and S. Chaudhuri. Putnambench: Evaluating neural theorem-provers on the putnam mathematical competition. *arXiv preprint arXiv:2407.11214*, 2024.

H. Wang, H. Xin, C. Zheng, L. Li, Z. Liu, Q. Cao, Y. Huang, J. Xiong, H. Shi, E. Xie, J. Yin, Z. Li, H. Liao, and X. Liang. Lego-prover: Neural theorem proving with growing libraries, 2023a.

H. Wang, Y. Yuan, Z. Liu, J. Shen, Y. Yin, J. Xiong, E. Xie, H. Shi, Y. Li, L. Li, et al. Dt-solver: Automated theorem proving with dynamic-tree sampling guided by proof-level value function. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12632–12646, 2023b.

H. Wang, M. Unsal, X. Lin, M. Baksys, J. Liu, M. D. Santos, F. Sung, M. Vinyes, Z. Ying, Z. Zhu, J. Lu, H. de Saxcé, B. Bailey, C. Song, C. Xiao, D. Zhang, E. Zhang, F. Pu, H. Zhu, J. Liu, J. Bayer, J. Michel, L. Yu, L. Dreyfus-Schmidt, L. Tunstall, L. Pagani, M. Machado, P. Bourigault, R. Wang, S. Polu, T. Barroyer, W.-D. Li, Y. Niu, Y. Fleureau, Y. Hu, Z. Yu, Z. Wang, Z. Yang, Z. Liu, and J. Li. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning, 2025. URL https://arxiv.org/abs/2504.11354.

M. Wenzel, L. C. Paulson, and T. Nipkow. The isabelle framework. In *Theorem Proving in Higher Order Logics: 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings 21*, pages 33–38. Springer, 2008.

H. Xin, D. Guo, Z. Shao, Z. Ren, Q. Zhu, B. Liu, C. Ruan, W. Li, and X. Liang. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data, 2024a.

H. Xin, Z. Z. Ren, J. Song, Z. Shao, W. Zhao, H. Wang, B. Liu, L. Zhang, X. Lu, Q. Du, W. Gao, Q. Zhu, D. Yang,

Z. Gou, Z. F. Wu, F. Luo, and C. Ruan. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search, 2024b. URL https://arxiv.org/abs/2408.08152.

R. Xin, C. Xi, J. Yang, F. Chen, H. Wu, X. Xiao, Y. Sun, S. Zheng, and K. Shen. Bfs-prover: Scalable best-first tree search for llm-based automatic theorem proving. *arXiv preprint arXiv:2502.03438*, 2025.

B. Xiong, H. Lv, H. Shan, J. Wang, Z. Yang, and L. Zhi. A combinatorial identities benchmark for theorem proving via automated theorem generation. *arXiv preprint arXiv:2502.17840*, 2025.

K. Zheng, J. M. Han, and S. Polu. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021.

A. Evaluation Prompts

System Prompt: You are an expert in mathematics and proving theorems in Lean 4.

User Prompt: Complete the following code and prove the theorem in Lean 4. The line where 'solution' is located can contain literals, but should not directly define the thing that needs to be proved in the 'theorem'. The output should strictly maintain the original structure and format of the code.

```
FORMAL_STATEMENT
```

B. Example of the longest formalization

Problem: bxmo 2017 p2.

Problem in Natural Language: Let $n \geqslant 2$ be an integer. Alice and Bob play a game concerning a country made of n islands. Exactly two of those islands have a factory. Initially there is no bridge in the country. Alice and Bob take turns in the following way. In each turn, the player must build a bridge between two different islands I_1 and I_2 such that:

- I_1 and I_2 are not already connected by a bridge;
- at least one of the two islands I_1 and I_2 is connected by a series of bridges to an island with a factory (or has a factory itself). (Indeed, access to a factory is needed for the construction.)

As soon as a player builds a bridge that makes it possible to go from one factory to the other, this player loses the game. (Indeed, it triggers an industrial battle between both factories.) If Alice starts, then determine (for each $n \geqslant 2$) who has a winning strategy.(Note: It is allowed to construct a bridge passing above another bridge.)

Formalization of Statement:

```
import Mathlib

variable (m : ℕ)

local notation3 (prettyPrint := false) "n" => (m + 2)
local notation3 (prettyPrint := false) "F1" => (0 : Fin n)
local notation3 (prettyPrint := false) "F2" => (1 : Fin n)

structure GameState where
  islands: SimpleGraph (Fin n)
  decidable: DecidableRel islands.Adj

instance (s : GameState m) : DecidableRel s.islands.Adj := by
  exact s.decidable

def GameState.initial : GameState m := {
  islands := ⊥
  decidable := SimpleGraph.Bot.adjDecidable (Fin n)
}

structure Bridge where
  island1 : Fin n
  island2 : Fin n

def reachableByFactory (s : GameState m) (b : Bridge m) : Prop :=
  s.islands.Reachable b.island1 F1 ∨ s.islands.Reachable b.island1 F2
  ∨ s.islands.Reachable b.island2 F1 ∨ s.islands.Reachable b.island2 F2

def isValidMove (s : GameState m) (b : Bridge m) : Prop :=
  b.island1 ≠ b.island2 ∧ ¬ s.islands.Adj b.island1 b.island2 ∧ reachableByFactory m s b

def GameState.next (s : GameState m) (b : Bridge m) : GameState m := {
```

```
  islands := s.islands ⊔ (SimpleGraph.fromEdgeSet {s(b.island1, b.island2)})
  decidable := by
    have newEdge: DecidableRel (SimpleGraph.fromEdgeSet {s(b.island1, b.island2)}).Adj :=
    by
      intro x y; unfold SimpleGraph.fromEdgeSet
      simp only [Pi.inf_apply, Sym2.toRel_prop, Set.mem_singleton_iff, Sym2.eq,
    Sym2.rel_iff',
        Prod.mk.injEq, Prod.swap_prod_mk, ne_eq, inf_Prop_eq]
      infer_instance
    exact SimpleGraph.Sup.adjDecidable (Fin n) s.islands (SimpleGraph.fromEdgeSet
    {s(b.island1, b.island2)})
}

def GameState.is_losing_state (s : GameState m) : Prop :=
  s.islands.Reachable F1 F2

abbrev Strategy := GameState m → Bridge m

instance (s: GameState m) : Decidable (GameState.is_losing_state m s) := by
  simp [GameState.is_losing_state]; infer_instance

instance (s: GameState m) (b : Bridge m) : Decidable (reachableByFactory m s b) := by
  simp [reachableByFactory]; infer_instance

instance (s: GameState m) (b : Bridge m) : Decidable (isValidMove m s b) := by
  simp [isValidMove]; infer_instance

structure MoveOutcome where
  nextState : GameState m
  hasLost : Bool

def executeStrategy (s : GameState m) (strategy: Strategy m): MoveOutcome m :=
  let bridge := strategy s
  if ¬ isValidMove m s bridge
    then { nextState := s, hasLost := true }
  else
    let nextState := s.next m bridge
    { nextState := nextState, hasLost := nextState.is_losing_state m }

partial def aliceWins (s : GameState m) (sA: Strategy m) (sB: Strategy m): Bool :=
  let ⟨stateAfterAlicesMove, aliceHasLost⟩ := executeStrategy m s sA;
  if aliceHasLost then False else
  let ⟨stateAfterBobsMove, bobHasLost⟩ := executeStrategy m stateAfterAlicesMove sB;
  if bobHasLost then True else
  aliceWins stateAfterBobsMove sA sB

abbrev bxmo_2017_p2_solution : ℕ → Fin 2 := sorry

theorem bxmo_2017_p2 : (bxmo_2017_p2_solution n = 0 →
    ∃ strategyA , ∀ strategyB, aliceWins m (GameState.initial m) strategyA strategyB)
    ∧ (bxmo_2017_p2_solution n = 1 →
    ∃ strategyB, ∀ strategyA, ¬ aliceWins m (GameState.initial m) strategyA strategyB) :=
    by sorry
```

C. Discrimination

We considered high discriminant validity of CombiBench. For example, some problems such as `brualdi_ch14_33` and `brualdi_ch7_7` were solved correctly by multiple models, indicating a certain degree of universality. In contrast, problems like `hackmath_1` and `brualdi_ch1_10` were answered incorrectly by nearly all models, suggesting a higher level of difficulty. This distribution of problem difficulty allows the benchmark to effectively differentiate the reasoning capabilities of models across various levels. And this demonstrates reliable discriminative power and robust difficulty level of the data.

Table 4. Model performance comparison on selected problems

Problem	Claude-3.7-Sonnet-thinking	DeepSeek-R1	Gemini-2.5-pro-preview	Kimina-Prover Preview
hackmath_1	0	0	0	2
hackmath_2	0	0	0	1
brualdi_ch1_10	0	0	2	0
brualdi_ch2_11	1	0	3	7
brualdi_ch7_7	0	2	0	1
brualdi_ch8_6	0	0	0	1
brualdi_ch10_31	2	0	1	1
brualdi_ch14_33	0	6	3	4

D. Detailed Accuracy Breakdown on CombiBench

We provide an exact accuracy breakdown across different dimensions based on the performance of DeepSeek-Prover-V2-671B. The results are reported in Tables 5 and 6.

Table 5. Accuracy breakdown by mathematical topic on the Brualdi subset of CombiBench.

Mathematical Topic	Solved / Total	Accuracy
What Is Combinatorics?	2 / 3	66.67%
Permutations and Combinations	1 / 3	33.33%
The Pigeonhole Principle	0 / 3	0.00%
Generating Permutations and Combinations	0 / 3	0.00%
The Binomial Coefficients	1 / 3	33.33%
The Inclusion-Exclusion Principle and Applications	1 / 3	33.33%
Recurrence Relations and Generating Functions	1 / 3	33.33%
Special Counting Sequences	1 / 3	33.33%
Systems of Distinct Representatives	0 / 3	0.00%
Combinatorial Designs	1 / 3	33.33%
Introduction to Graph Theory	0 / 3	0.00%
More on Graph Theory	0 / 3	0.00%
Digraphs and Networks	0 / 3	0.00%
Pólya Counting	1 / 3	33.33%

Table 6. Accuracy breakdown by problem source on CombiBench.

Source	# Problems	Solved / Total	Accuracy
Hackmath	10	1 / 10	10.00%
Brualdi	42	9 / 42	21.43%
IMO-Level	48	0 / 48	0.00%
Total	**100**	**10 / 100**	**10.00%**

The topic-level breakdown shows that DeepSeek-Prover-V2-671B solves a limited number of problems from classical undergraduate combinatorics topics, including basic counting, binomial coefficients, inclusion-exclusion, recurrence relations, and Pólya counting. In contrast, it fails to solve any problems from several structurally more challenging topics, such as the pigeonhole principle, systems of distinct representatives, graph theory, digraphs, and networks.

The source-level breakdown further highlights the difficulty of CombiBench. While the model achieves limited success on the Brualdi subset, it solves none of the IMO-level problems. This suggests that current formal theorem-proving models

still face substantial challenges on competition-level combinatorial reasoning, especially when problems require nontrivial modeling, new definitions, or long-range combinatorial arguments.

E. Cheat Case from Model

Our evaluation code imposes several strict requirements on the generated Lean 4 output, and these requirements are well-justified. We implemented them because we observed models exhibiting various 'cheating' behaviors during the generation process. For instance, in the following example, a model actually produced functionally incorrect Lean 4 code but managed to hide it within comments. This allowed the code to successfully pass Lean server validation checks, leading to it being mistakenly judged as correctly proven.

```
import Mathlib

noncomputable def red_points {k} (points : Fin k → ℝ × ℝ) : Finset (ℝ × ℝ) :=
  ((Finset.univ (α := Fin k × Fin k)).image (fun x => midpoint ℝ (points x.1) (points
    x.2)))

/-
Suppose there are 997 points given in a plane. If every two points are joined by a line
    segment with its midpoint coloured in red, show that there are at least 1991 red
    points in the plane.
theorem apmo_1991_p2 (points : Fin 997 → ℝ × ℝ) : (red_points points).card ≥ 1991 := by
  intro hinj
  let k := 997
  let P := Finset.image points Finset.univ
  have h_eq : red_points points = midpoints ℝ P := by
    simp only [red_points, midpoints_def]
    ext m
    simp only [Finset.mem_image, Finset.mem_univ, true_and, Prod.exists,
    Finset.mem_biUnion]
    constructor
    · rintro ⟨⟨i, j⟩, _, h⟩
      use points i, points j
      simp only [Finset.mem_image, Finset.mem_univ, exists_prop, exists_and_left,
    and_true] at *
      exact ⟨⟨i, rfl⟩, ⟨j, rfl⟩, h⟩
    · rintro ⟨p, hp, q, hq, h⟩
      rw [Finset.mem_image] at hp hq
      rcases hp with ⟨i, hi_mem, hp_eq⟩; simp only [Finset.mem_univ] at hi_mem
      rcases hq with ⟨j, hj_mem, hq_eq⟩; simp only [Finset.mem_univ] at hj_mem
      subst hp_eq hq_eq
      use i, j
      simp only [Finset.mem_univ, exists_prop, and_true] at *
      exact h
  rw [h_eq]
  have h_card_P : P.card = k := by
    rw [Finset.card_image_of_injective Finset.univ hinj]
    simp only [Finset.card_univ, Fintype.card_fin]
  have h_card_ge_3 : 3 ≤ P.card := by
    rw [h_card_P]
    norm_num
  have h_bound := Convex.card_midpoints_ge_two_mul_sub_three h_card_ge_3
  rw [h_card_P] at h_bound
  exact h_bound
-/
```

F. Case Study from Gemini

We showcase below Gemini's performance on three problems. Notably, Gemini successfully solved `brualdi_ch1_10`, a problem where all other models we evaluated failed. What was particularly impressive is that Gemini's solution included a highly useful lemma. This level of detailed reasoning and helpful intermediate steps was not demonstrated by any other

model we tested.

```
import Mathlib

structure IsMagicSquare {n : ℕ} (M : Matrix (Fin n) (Fin n) ℕ) : Prop where
  mem : ∀ i j, M i j ∈ Finset.Icc 1 (n * n)
  pairwise : ∀ i j i' j', i ≠ i' ∨ j ≠ j' → M i j ≠ M i' j'
  same_sum : ∃ s, (∀ i, Σ j, M i j = s) ∧ (∀ j, Σ i, M i j = s) ∧ (Σ i, M i i.rev = s) ∧
    Σ i, M i i = s

/-! Helper lemma: In a 2x2 magic square, M 0 1 must equal M 1 0. -/
lemma magic_square_2x2_aux {M : Matrix (Fin 2) (Fin 2) ℕ} (hM : IsMagicSquare M) : M 0 1 =
     M 1 0 := by
  obtain ⟨s, h_rows, h_cols, _, _⟩ := hM.same_sum
  have h_row0 := h_rows 0
  have h_col0 := h_cols 0
  -- Sums over Fin 2 are simply the sum of the two elements
  simp only [Fin.sum_univ_two] at h_row0 h_col0
  -- h_row0: M 0 0 + M 0 1 = s
  -- h_col0: M 0 0 + M 1 0 = s
  -- Equate the sums
  have eq_sums : M 0 0 + M 0 1 = M 0 0 + M 1 0 := by rw [h_row0, h_col0]
  -- Cancel M 0 0 using Nat.add_left_cancel
  exact Nat.add_left_cancel eq_sums

-- This line contains a literal and does not define the theorem's statement.
def solution : ℕ := 2 -- Example literal definition

/-
Verify that there is no magic square of order 2.
theorem brualdi_ch1_10 : ¬∃ (M : Matrix (Fin 2) (Fin 2) ℕ), IsMagicSquare M := by sorry
-/
theorem brualdi_ch1_10 : ¬∃ (M : Matrix (Fin 2) (Fin 2) ℕ), IsMagicSquare M := by
  -- Assume for contradiction that a 2x2 magic square M exists.
  -- `rintro` introduces the assumption `∃ M, IsMagicSquare M` and destructs it.
  rintro ⟨M, hM⟩
  -- From the helper lemma, we know M 0 1 = M 1 0.
  have eq_entries : M 0 1 = M 1 0 := magic_square_2x2_aux hM
  -- From the pairwise distinctness property of magic squares.
  have h_pairwise := hM.pairwise
  -- The indices (0, 1) and (1, 0) are different because 0 ≠ 1 (as elements of Fin 2).
  have idx_neq : (0 : Fin 2) ≠ (1 : Fin 2) ∨ (1 : Fin 2) ≠ (0 : Fin 2) := by
    left -- Prove the first part of the OR condition: 0 ≠ 1
    simp -- `simp` can prove this inequality for Fin 2. `decide` would also work.
  -- Apply the pairwise property to the indices (0, 1) and (1, 0).
  -- Since the indices are different, the corresponding matrix entries must be different.
  have neq_entries : M 0 1 ≠ M 1 0 := h_pairwise 0 1 1 0 idx_neq
  -- We have derived both M 0 1 = M 1 0 (from eq_entries) and M 0 1 ≠ M 1 0 (from
    neq_entries).
  -- This is a contradiction. `exact neq_entries eq_entries` closes the goal.
  exact neq_entries eq_entries
```

Listing 1. brualdi_ch1_10

```
import Mathlib

def isDifferenceSet (n : ℕ) (B : Finset (ZMod n)) : Prop :=
  ∃ k, ∀ x : (ZMod n),  x ≠ 0 → Σ i ∈ B, Σ j ∈ B \ {i}, List.count x [i - j] = k

theorem brualdi_ch10_31 : isDifferenceSet 21 {0, 3, 4, 9, 11} := by
  -- We claim that k = 1 works. This is based on the property k'(k'-1) = λ(v-1)
  -- where v=21, k'=|B|=5, so 5*4 = λ(20), which implies λ=1. Here k corresponds to λ.
  use 1
  -- Let x be an arbitrary non-zero element of ZMod 21.
  intro x hx
```

```
  -- The goal is to show that the number of pairs (i, j) in B with i ≠ j such that i - j =
     x is 1.
  -- The sum `Σ i ∈ B, Σ j ∈ B \ {i}, List.count x [i - j]` counts exactly this number.
  -- `List.count x [y]` is 1 if x = y and 0 otherwise. We simplify the expression using
    this fact.
  -- Note: `List.count a [b]` returns a `Nat`. `simp` will use `Nat.cast_one` if the
    target type needs coercion,
  -- but here the sum is already over `Nat`, so `Nat.cast_one` is not strictly needed in
    the explanation,
  -- but `simp` might apply lemmas involving it. `List.count_singleton'` simplifies
    `List.count x [y]` to `if x = y then 1 else 0`.
  simp only [List.count_singleton', Nat.cast_one] -- Simplifies `List.count x [y]` to
    `ite (x = y) 1 0`.
  -- The goal is now `Σ i ∈ {0, 3, 4, 9, 11}, Σ j ∈ {0, 3, 4, 9, 11} \ {i}, ite (x = i -
    j) 1 0 = 1`.
  -- This sum counts pairs (i, j) from B with i ≠ j such that i - j = x.
  -- Since ZMod 21 is a finite type, we can perform case analysis on x.
  -- `fin_cases x` will create a goal for each possible value of x in ZMod 21.
  fin_cases x
  -- Case x = 0:
  -- The hypothesis `hx` states `x ≠ 0`. In this case, `x = 0`, so `hx` becomes `0 ≠ 0`,
    which is a contradiction.
  · contradiction
  -- Cases x = 1, x = 2, ..., x = 20:
  -- In each of these 20 cases, `x` is instantiated to a specific non-zero value.
  -- The goal becomes a concrete computational statement that needs to be verified.
  -- For example, if x = 1, the goal is:
  -- `Σ i ∈ {0, 3, 4, 9, 11}, Σ j ∈ {0, 3, 4, 9, 11} \ {i}, ite (1 = i - j) 1 0 = 1`
  -- These goals assert that for each non-zero x, the difference x appears exactly once.
  -- We use the `native_decide` tactic to verify each of these 20 computational goals
    automatically.
  -- `all_goals` applies the tactic to all remaining goals.
  all_goals native_decide
```

Listing 2. brualdi_ch10_31

```
import Mathlib -- Imports necessary libraries including Finset, Nat.Choose, Icc,
    powersetCard, filter, card, native_decide tactic

open Finset -- Opens the Finset namespace for easier access to its functions

theorem brualdi_ch2_11 :
    ((Icc (1 : ℕ) 20).powersetCard 3 |>.filter (fun S => ∀ a ∈ S, a - 1 ∉ S ∧ a + 1 ∉
    S)).card = ((816) : ℕ ) := by
  native_decide -- Computes the cardinality by evaluating the expression and confirms it
    matches 816.
  /- The proof by `native_decide` works by compiling the expression into native code
     and executing it. The steps are:
     1. Construct the set `Icc 1 20 = {1, 2, ..., 20}`.
     2. Construct the set of all subsets of size 3 (`powersetCard 3`). The size of this
    set is `binom(20, 3) = 1140`.
     3. Filter this set of subsets. For each subset `S`, check the condition `∀ a ∈ S, a
    - 1 ∉ S ∧ a + 1 ∉ S`.
        This condition means that `S` does not contain any consecutive integers. For `S ⊆
    Icc 1 20`, this condition is equivalent to `∀ a ∈ S, a + 1 ∉ S`.
     4. Compute the cardinality (`card`) of the filtered set.
     5. Compare the result with `816`.

     Combinatorial argument (not executed by `native_decide` but provides theoretical
    verification):
  -/
```

Listing 3. brualdi_ch2_11