DeepSeek-Prover: Advancing Theorem Proving in LLMs through Large-Scale Synthetic Data

Huajian Xin[1,2] **Daya Guo**[1] **Zhihong Shao**[1] **Z.Z. Ren**[1] **Qihao Zhu**[1] **Bo Liu**[1]
Chong Ruan[1] **Wenda Li**[3] **Xiaodan Liang**[2,4*]
[1]DeepSeek [2]Sun Yat-sen University [3]University of Edinburgh [4]MBZUAI
{xinhj, guoday, zhihongshao, rzz, zhuqh, chong.ruan}@deepseek.com,
benjaminliu.eecs@gmail.com, wli8@ed.ac.uk, xdliang328@gmail.com

Abstract

Proof assistants like Lean have revolutionized mathematical proof verification, ensuring high accuracy and reliability. Although large language models (LLMs) show promise in mathematical reasoning, their advancement in formal theorem proving is hindered by a lack of training data. To address this issue, we introduce an approach to generate extensive Lean 4 proof data derived from high-school and undergraduate-level mathematical competition problems. This approach involves translating natural language problems into formal statements, filtering out low-quality statements, and generating proofs to create synthetic data. After fine-tuning the DeepSeekMath 7B model on this synthetic dataset, which comprises 8 million formal statements with proofs, our model achieved whole-proof generation accuracies of 46.3% with 64 samples and 52% cumulatively on the Lean 4 miniF2F test, surpassing the baseline GPT-4 at 23.0% with 64 samples and a tree search reinforcement learning method at 41.0%. Additionally, our model successfully proved 5 out of 148 problems in the Lean 4 Formalized International Mathematical Olympiad (FIMO) benchmark, while GPT-4 failed to prove any. These results demonstrate the potential of leveraging large-scale synthetic data to enhance theorem-proving capabilities in LLMs. Both the synthetic dataset and the model will be made available to facilitate further research in this promising field.

1 Introduction

In modern mathematics, the increasing complexity of proofs presents substantial challenges for peer review. This complexity has led to the acceptance of erroneous proofs, with critical flaws often detected only after considerable time. To address these issues, formal mathematical languages such as Lean [De Moura et al., 2015, Moura and Ullrich, 2021], Isabelle [Paulson, 1994], and Coq [The Coq Development Team] have been developed. These languages enable the creation of computer-verifiable proofs [Avigad, 2023]. However, crafting formal proofs demands significant effort, specialized expertise, and poses challenges even for seasoned mathematicians. Consequently, the significance of automated theorem proving is on the rise [Shulman, 2024].

To reduce the effort involved in writing formal mathematical proofs, several approaches [Polu and Sutskever, 2020, Jiang et al., 2021, Han et al., 2021, Polu et al., 2022, Lample et al., 2022, Jiang et al., 2022a, Yang et al., 2024] have been developed, primarily focusing on search algorithms that explore potential solutions for proposed theorems. However, these methods often struggle with the vast search spaces required for complex theorems, rendering them ineffective for more intricate proofs [Loos et al., 2017]. Recently, advances in large language models (LLMs) have introduced a novel strategy,

*Corresponding author.

Preprint. Under review.

utilizing pre-trained models to guide the search process. Although these new methods [Jiang et al., 2022b, Zhao et al., 2023, Xin et al., 2023] represent significant improvements, they still fall short of practical applicability due to the lack of parallel corpus. Unlike conventional programming languages such as Python or Java, formal proof languages are used by relatively few mathematicians, resulting in limited datasets. Recent advances in autoformalization [Wu et al., 2022] allow more aligned data to be synthesized to train LLM-based automated theorem provers. Nevertheless, the resulting dataset remains too small to fully unleash the capabilities of LLMs.

To address this issue, we propose a method for generating extensive Lean 4 proof data from informal mathematical problems. Our approach translates high-school and undergraduate-level mathematical competition problems into formal statements. We then automate proof generation using a large language model (LLM) and verify the correctness of these proofs within the Lean 4 environment. The primary challenge of this method is to ensure both the scale and quality of the synthetic data.

Quality Assurance: We enhance the quality of generated proofs through a multi-step process. First, we filter out simple statements using a quality scoring model and exclude invalid statements via a hypothesis rejection strategy. Our novel iterative framework then improves proof quality by initially generating synthetic statements from informal math problems using an under-trained LLM fine-tuned on limited data. These statements are used to generate corresponding proofs, which are validated for correctness using a Lean 4 verifier. The correct theorem-proof pairs are subsequently used to further train the initial model. Through several iterations, the model trained on large-scale synthetic data becomes significantly more powerful than the originally under-trained LLMs, resulting in higher-quality theorem-proof pairs.

Scale Assurance: To accelerate the proof generation process, our method addresses the challenge of the large search space for proofs. A significant cause of delays is the generation of unprovable statements that continue to be processed until they reach the time limit. To mitigate this, we propose proving negated statements in parallel. Once either the original statement or its negation is proved, the entire proving process is terminated.

We assess the effectiveness of our method on Lean 4 theorem proving using 488 problems from miniF2F [Zheng et al., 2021] and 148 problems from the FIMO benchmarks [Liu et al., 2023]. We utilize DeepSeekMath 7B [Shao et al., 2024], a state-of-the-art mathematical model, as our base. The results show that our iteratively trained model performs strongly, achieving 46.3% accuracy in whole-proof generation on the miniF2F-test benchmark with 64 samples, surpassing GPT-4 [Achiam et al., 2023] at 23.0% and a reinforcement learning method at 41.0%. Additionally, our approach solved 4 out of 148 problems in the FIMO benchmark with 100 samples, while GPT-4 solved none, and our approach solved 5 with 4096 samples. Ablation experiments indicate that the model progressively solves more problems in miniF2F with each iteration. In summary, our paper makes the following contributions:

- We introduce an iterative method to synthesize 8 million formal statements, each accompanied by a formal proof, from informal math problems. Experimental results demonstrate that this method significantly enhances both the scalability and quality of synthetic data.
- Our model, trained on this synthetic dataset, achieves state-of-the-art performance on benchmarks, with whole-proof generation accuracies of 46.3% using 64 samples and 52% cumulatively on the Lean 4 miniF2F test. This surpasses the baseline GPT-4 at 23.0% with 64 samples and a tree search reinforcement learning method at 41.0%. Additionally, our model successfully proved 5 out of 148 problems in the Lean 4 Formalized International Mathematical Olympiad (FIMO) benchmark, while GPT-4 failed to prove any.
- We contribute to the mathematical and AI communities by creating and open-sourcing a large dataset of high-quality formal mathematical proofs, thereby fostering further research and development in automated theorem proving.

2 Background and Related Works

Automated theorem proving has been a significant area of interest in artificial intelligence research since its inception [Bibel, 2013]. Initial efforts were directed at simpler logical frameworks, which led to the development of highly efficient first-order theorem provers like E [Schulz, 2002] and Vampire [Kovács and Voronkov, 2013]. Nonetheless, these tools often fall short in handling complex

theorems commonly found in modern proof assistants such as Lean [De Moura et al., 2015], Isabelle [Paulson, 1994], and Coq [The Coq Development Team]. The advent of recent deep learning models and model-guided search techniques has reinvigorated the field [Bansal et al., 2019]. This modern approach has not only enhanced the capabilities of ATP systems but also expanded their applicability in solving more intricate mathematical problems.

ATP with Neural Models. With the development of deep learning, several approaches have been proposed to combine neural models with ATP [Loos et al., 2017]. A series of ATP approaches adopts tree search algorithms guided by neural models [Polu and Sutskever, 2020, Han et al., 2021, Polu et al., 2022, Jiang et al., 2022a, Yang et al., 2024]. These approaches primarily utilize reinforcement learning techniques to enhance the accuracy of the model [Kaliszyk et al., 2018, Crouse et al., 2021, Wu et al., 2021, Lample et al., 2022]. Since the search space is significantly large, the searching process consumes considerable time and computing resources.

Another series of ATP approaches harnesses the power of large language models. These approaches typically involve language models that are fine-tuned with open-source proof data and interact with verifiers via a state-action transition program [Polu and Sutskever, 2020, Jiang et al., 2021, Han et al., 2021, Polu et al., 2022, Lample et al., 2022, Jiang et al., 2022a, Yang et al., 2024]. This process iteratively generates proof steps and verifies their correctness with formal verifiers. It then generates the next proof steps based on the proof states returned by the formal verifiers. Although these approaches achieve high performance, they are computationally intensive. To enhance efficiency, recent researches leverage language models to generate complete formal proofs directly [First et al., 2023, Jiang et al., 2022b, Zhao et al., 2023, Xin et al., 2023], thus bypassing the iterative interaction during proof generation.

Autoformalization for Formal Mathematics. Due to the limited availability of formal corpora for training, the performance of current large language models (LLMs) is also constrained. Thus, some approaches propose autoformalization [Wu et al., 2022, Jiang et al., 2022b], which involves converting natural language descriptions into formal statements that can be verified by proof assistants. Several studies have generated synthetic datasets of formal proofs using rule-based transformations of existing theorems [Wu et al., 2020, Wang and Deng, 2020, Xiong et al., 2023]. While effective, these methods are constrained by their reliance on predefined rules and lack flexibility for broader applications. Recent methodologies adopt large language models to translating natural language problems into formal statements [Huang et al., 2024]. However, these datasets remain smaller than needed and are limited to small mathematical benchmarks, leading to only minor improvements in training outcomes for language models. In this paper, we aim to synthesise formal proofs via autoformalization at a much larger scale to boost the performance of a neural prover.

3 Approach

In this section, we introduce our approach, which consists of four key processes as depicted in Figure 1. The initial phase concentrates on generating formal mathematical statements from a broad collection of informal math problems, necessitating further proof. Next, the autoformalized statements are filtered through model scoring and hypothesis rejection methods to select high-quality statements. These statements are then proved by a model called DeepSeek-Prover, with their correctness verified by the formal verifier called Lean 4[2], yielding validated formal statements and proofs. These data serve as synthetic data for fine-tuning the DeepSeek-Prover. After enhancing DeepSeek-Prover, we repeat the entire previously described process. This cycle continues until the improvements in DeepSeek-Prover become marginal. Notably, to enhance proof efficiency, we prove concurrently both the original statements and their negations. This method has the advantage of swiftly discarding the original statement when it is invalid by proving its negation. The details of each phase will be described in the subsequent sections.

3.1 Autoformalization

The generation of formal proof data fundamentally relies on the availability of a substantial corpus of formal statements. In practice, however, amassing a large collection of manually crafted formal statements is challenging. Fortunately, the internet is replete with math-related problems expressed in

[2] `leanprover/lean4 : v4.7.0 − rc2`

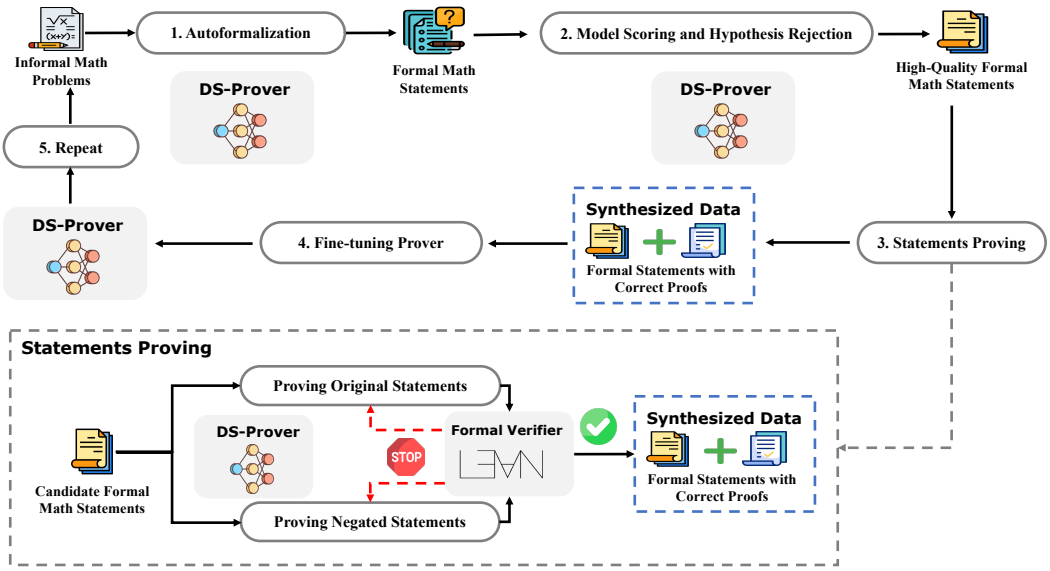

Figure 1: An overview of our approach.

natural language. By autoformalizing these informal mathematical problems, we can generate a vast repository of formal statements.

We have observed that problems with explicit conditions and well-defined goals are typically easier to formalize compared to advanced mathematical topics that necessitate intricate definitions and constructions. Consequently, this paper primarily examines high school and undergraduate-level competition problems, with a particular emphasis on algebra and number theory, and to a lesser extent, combinatorics, geometry, and statistics. Despite their apparent simplicity, these problems often involve complex solution techniques, making them excellent candidates for constructing proof data to improve theorem-proving capabilities in Large Language Models (LLMs). To compile our dataset, we employed web scraping and careful data cleaning techniques to extract problems from online resources featuring high school and undergraduate exercises, exams, and competitions, resulting in a dataset of 869,659 high-quality natural language math problems.

Specifically, we initialized the DeepSeek-Prover using the DeepSeekMath-Base 7B model [Shao et al., 2024]. Initially, the model struggled to convert informal math problems into formal statements. To address this, we fine-tuned the DeepSeek-Prover model using the MMA dataset [Jiang et al., 2023], which comprises formal statements from Lean 4's mathlib[3] that were back-translated into natural language problem descriptions by GPT-4. We then instructed the model to translate these natural language problems into formal statements in Lean 4 using a structured approach.

Prompt:

```
Mathematical Problem in Natural Language:
{$informal_statement_with_answers}
Translate the problem to Lean 4 (only the core declaration):
```lean4
```

Response:

```
{$formal_statement}
```
```

3.2 Quality Filtering

The quality of the autoformalized statements was found to be suboptimal due to two main issues. Firstly, many formal statements were overly simplistic. To address this, we developed scoring criteria and provided examples from miniF2F-valid as few-shot examples to guide the DeepSeek-Prover

[3]The specific mathlib commit used is `64528268b3c2cf578639bc479828882a9ecd3a82`.

model in evaluating the content and quality of these statements using a chain-of-thought approach. Manual review of these scores confirmed that the model's evaluations closely matched human intuition and expectations. Specifically, the model was instructed (see Appendix A.1 for the detailed prompt) to classify the quality of each formal statement into categories: "excellent," "good," "above average," "fair," or "poor." Statements rated as "fair" or "poor" were subsequently excluded.

The second issue pertains to formal statements that, although provable, are based on inconsistent hypotheses leading to vacuous conclusions, rendering the conclusions meaningless in mathematics. For example, consider the following model-generated statement:

```
example (θ : ℝ) (h₀ : ∀ z : ℂ, z ^ 2 = -1 ∧ z ^ 3 = -1 ∧ z ^ 6 = 1) (h₁ :
    Real.tan θ = 2 * Real.sqrt 3) : θ = 5 * Real.pi / 3
```

Here, the hypothesis $z^2 = -1 \wedge z^3 = -1 \wedge z^6 = 1$ for all complex numbers is clearly false, making any derived conclusions meaningless. To eliminate such cases from our dataset, we implemented a hypothesis rejection method. This involves using the DeepSeek-Prover model to attempt proving the formal statement with 'False' as the conclusion. A successful proof indicates an invalid hypothesis, prompting exclusion of the statement. An example is shown below:

```
example (θ : ℝ) (h₀ : ∀ z : ℂ, z ^ 2 = -1 ∧ z ^ 3 = -1 ∧ z ^ 6 = 1) (h₁ :
    Real.tan θ = 2 * Real.sqrt 3) : False := by
  simpa using h₀ 1
```

By applying this dual strategy of model scoring and hypothesis rejection, we curated a refined set of 712,073 high-quality formal statements, providing a robust foundation for further proof synthesis.

3.3 Statement Proving

After creating a substantial corpus of high-quality formal statements, we employed the model to search for proofs of these statements. Traditionally, language models have been used predominantly in a brute-force manner to prove theorems—repeatedly attempting until a valid proof is found or computational resources are exhausted. This approach is inefficient for our purposes. Typically, language models are applied to human-curated formal statements that are carefully crafted and generally true and provable; however, in our task of proving autoformalized statements, many of the statements produced by the model may be incorrect. Indeed, it is unreasonable to expect the model to validate a false proposition within any reliable proof system. This issue becomes more pronounced during large-scale autoformalization, where we observed that at least 20% of the formal statements generated by our model, even after quality filtering, were incorrect, leading to significant computational waste if addressed with brute force.

To minimize resource wastage on unprovable statements and improve the efficiency of the proof search process, we exploited the logical symmetry between a statement and its negation to accelerate proof synthesis. We implemented dual concurrent proof searches for each synthetic statement—one for the statement $\Gamma \vdash P$ and another for its negation $\Gamma \vdash \neg P$. The search terminates as soon as a valid proof is found for either, conclusively demonstrating the unprovability of the other. Each proof search stream attempts up to k proofs unless a valid proof emerges sooner.

All validated proofs, whether they justify the original theorems or their negations, are then aggregated to further train the DeepSeek-Prover. Thus, this dual approach serves as a form of data augmentation, enriching the dataset with both propositions and their negations—even if the original propositions were not correctly formalized by the model.

3.4 Iterative Enhancement

Since the entire pipeline heavily relies on the DeepSeek-Prover, enhancing the model's performance after each iteration is crucial. To achieve this, we consistently fine-tune the model with newly generated data. The updated model is then utilized for subsequent autoformalization iterations. The key insight from this iterative process is that the model incrementally improves in strength and efficacy after each cycle of refinement and application. This iterative process continues until no further gains are observed. Consequently, the theorem-proof pairs generated by the model become increasingly higher in quality with each iteration. This method ensures that the DeepSeek-Prover

consistently enhances its performance, ultimately producing superior theorem-proof pairs through continuous refinement.

4 Experiments

4.1 Experimental Setup

DeepSeek-Prover is built upon DeepSeekMath-Base 7B model [Shao et al., 2024], a decoder-only transformer [Vaswani et al., 2017] pre-trained on a corpus comprising 120 billion math-related tokens. We fine-tuned this model using a global batch size of 512 and a constant learning rate of 1×10^{-4}, incorporating 6,000 warmup steps with synthetic data. DeepSeek-Prover's performance was evaluated against several baselines:

- **GPT-3.5 and GPT-4** [Achiam et al., 2023], developed by OpenAI, are advanced generative AI models known for their effectiveness in diverse tasks, including code generation. Although not explicitly designed for theorem proving, their extensive scale and parameter count confer significant capabilities. In contrast, **DeepSeekMath** is a specialized model, explicitly pre-trained for mathematical content. We utilized both GPT-4 (specifically the GPT-4-turbo 0409 version) and DeepSeekMath to generate complete proofs for given theorems using a methodology similar to ours.

- **GPT-f** [Polu and Sutskever, 2020], utilizing a GPT-2-inspired architecture [Radford et al., 2019], implements an iterative best-first search method to progressively generate and validate proof steps within a formal proof setting until a proof is either completed or resources are depleted. This methodology has been further advanced by **Proof Artifact Co-Training** [Han et al., 2021], **ReProver** [Yang et al., 2024], **Llemma** [Azerbayev et al., 2023], and **COPRA** [Thakur et al., 2023], which employ either specialized fine-tuned models or versatile general-purpose models such as GPT-3.5 and GPT-4 for the generation of proof steps.

4.2 Main Results

This study addresses complex mathematical problems in algebra and number theory. We evaluate the theorem-proving efficacy of our model using the miniF2F [Zheng et al., 2021] and FIMO [Liu et al., 2023] benchmarks. The metric pass@k is employed to denote the scenario where at least one valid proof is discovered among the first k attempts generated by the model.

Results on MiniF2F. The miniF2F benchmark consists of 244 validation and 244 test problems, ranging from basic arithmetic to competition-level problems, e.g., problems from the American Invitational Mathematics Examination (AIME), the American Mathematics Competitions (AMC), and the International Mathematical Olympiad (IMO). We use the version of miniF2F in Lean 4, which was released by the LeanDojo project (`https://github.com/yangky11/miniF2F-lean4`).

Table 1 compares various state-of-the-art methods on the miniF2F dataset. DeepSeek-Prover outperforms all with cumulative scores of 60.2% on miniF2F-valid and 52.0% on miniF2F-test, significantly higher than other methods, including GPT-4 which scores 25.41% and 22.95%, respectively. Even the best tree search method, Hypertree Proof Search with a 600M model, achieves only up to 58.6% on miniF2F-valid and 41.0% on miniF2F-test. DeepSeek-Prover's scalability is evident as its performance improves with increased computational resources, rising from 30.03% using a greedy approach to 50.0% at 65536 generation times, demonstrating its effectiveness in handling complex proof scenarios. Examples of proved theorems of MiniF2F can be found in Appendix A.3.1.

Results on FIMO. The FIMO benchmark comprises 149 formal problems which are sourced from the IMO shortlist translated into Lean 4. Our method successfully proved 4 theorems with 100 attempts per theorem, whereas GPT-4 failed to prove any. By increasing the number of attempts per theorem to 4,096, we successfully proved an additional theorem. Examples of proved theorems of FIMO can be found in Appendix A.3.2.

Table 1: Comparing with state-of-the-arts on the miniF2F dataset.

Method	Model size	Generation times	miniF2F-valid	miniF2F-test
Tree Search Methods				
COPRA (GPT-3.5) [Thakur et al., 2023]	-	1×60	-	9.0%
COPRA (GPT-4) [Thakur et al., 2023]	-	1×60	-	26.6%
Proof Artifact Co-Training [Han et al., 2021]	837M	$1 \times 8 \times 512$	23.9%	24.6%
		$8 \times 8 \times 512$	29.3%	29.2%
ReProver [Yang et al., 2024]	229M	1×3751	-	25.0%
Llemma [Azerbayev et al., 2023]	7B	1×3200	-	26.2%
Llemma [Azerbayev et al., 2023]	34B	1×3200	-	25.8%
Curriculum Learning [Polu et al., 2022]	837M	$1 \times 8 \times 512$	33.6%	29.6%
		$8 \times 8 \times 512$	41.2%	34.5%
		$64 \times 8 \times 512$	47.3%	36.6%
Hypertree Proof Search [Lample et al., 2022]	600M	cumulative	58.6%	-
		64×5000	-	41.0%
Whole-Proof Generation Methods				
GPT-4-turbo 0409	-	64	25.4%	23.0%
DeepSeekMath-Base [Shao et al., 2024]	7B	128	25.4%	27.5%
		cumulative	**60.2%**	**52.0%**
DeepSeek-Prover	7B	1 (greedy)	-	30.0%
		64	-	46.3%
		128	-	46.3%
		8192	-	48.8%
		65536	-	50.0%

4.3 Ablation Studies

4.3.1 The Effectiveness of Large-scale Autoformalization

To demonstrate the effectiveness of large-scale autoformalization, we conducted a comparative analysis as shown in Table 2 between our autoformalized dataset and conventional datasets using expert iteration [Polu and Sutskever, 2020]. This iterative method entails generating formal proofs, fine-tune the model based on successful outcomes, and iterating this process until no additional enhancements are observed. The results indicate that models trained with our autoformalized data significantly outperform those trained solely with mathlib data.

Table 2: Improvement in pass rates for miniF2F at pass@128 in models trained on formal proofs, including those derived from human-authored theorems in Lean 4's mathlib and automatically formalized theorems.

Model	#Tokens	miniF2F-valid	miniF2F-test
-	-	25.4%	27.5%
Mathlib	0.238B	30.3%	31.2%
Autoformalized Statements	3.108B	48.8%	42.6%

4.3.2 The Effectiveness of Formal Statements Scoring

To demonstrate the effectiveness of the model in filtering out low-quality statements, we fine-tuned the DeepSeekMath-Base model using an equal amount of high-score proof data and low-score proof data to verify the quality of the data, as shown in Table 3. The table shows that the model trained on high-score proof data outperformed the model trained on low-score proof data by 4.5%. This enhancement underscores the utility of the model in accurately scoring and effectively filtering out lower-quality statements.

Table 3: Improvement in pass rates for miniF2F at pass@128 in models trained on differently scored proof data.

Scored Class	miniF2F-valid	miniF2F-test
"excellent", "good" and "above average"	48.8%	42.6%
"fair" and "poor"	41.4%	38.1%

4.3.3 The Effectiveness of Iterative Enhancement

Table 4 demonstrates a distinct correlation between the number of iterations in data synthesis and enhanced performance in theorem proving. This evidence underscores the success of our iterative enhancement strategy in augmenting theorem-proving capabilities. Successive iterations not only refine the model's ability to handle complex proofs but also significantly increase the quality and quantity of the synthetic data produced.

Table 4: Improvement in pass rates for miniF2F at pass@128 in models across successive training iterations, facilitated by the incremental integration of synthesized data via autoformalization.

Model	miniF2F-valid	miniF2F-test
iteration 0	38.1%	34.0%
iteration 1	45.1%	39.3%
iteration 2	49.2%	41.4%
iteration 3	54.5%	45.1%
iteration 4	59.4%	46.3%

4.3.4 The Effectiveness of Scaling Synthetic Theorem Proving Data

Our investigation into synthetic theorem proving data reveals a clear correlation between dataset size and model efficacy, as illustrated in Table 5. By examining subsets of the eight million generated proof data points, we observed that performance on the miniF2F benchmark improves proportionally to the exponential increase in dataset size. This pattern highlights the pivotal importance of large-scale datasets for boosting model proficiency in automatically formalizing natural language questions. These findings emphasize the significant potential and necessity of systematic data construction for progressing in the field of automated theorem proving.

Table 5: Improvement in pass rates for miniF2F at pass@128 in models trained with a larger fraction of synthesized data via autoformalization.

Size	miniF2F-valid	miniF2F-test
1,000	22.95%	24.18%
10,000	32.79%	31.97%
100,000	36.07%	37.7%
1,000,000	39.34%	38.11%
8,066,621	42.62%	40.16%

5 Case Studies

This section presents two case studies to demonstrate the application of our methods in autoformalizing theorems. It showcases both successful proofs and the identification of inconsistencies during the Hypothesis Rejection stage.

5.1 Autoformalized Theorem with Complete Proof

Example a. *Problem*: Prove that the determinant of the following matrix is zero.

$$\begin{bmatrix} 1 & \cos(a-b) & \cos(a) \\ \cos(a-b) & 1 & \cos(b) \\ \cos(a) & \cos(b) & 1 \end{bmatrix}$$

Autoformalized Theorem in Lean:

```
example (a b : ℝ) :
  Matrix.det ![![1, Real.cos (a - b), Real.cos a], ![Real.cos (a - b), 1, Real.cos
    b], ![Real.cos a, Real.cos b, 1]] = 0
```

This approach effectively translates the algebraic expression of the matrix and its determinant into a formal language using Lean. The autoformalization captures the essence of the original mathematical statement by defining a specific 3×3 matrix dependent on real numbers a and b, and asserts that its determinant is zero. The formalization employs the `Matrix.det` function to compute the determinant, utilizing the `![...]` notation for lists of lists in Lean to represent the matrix rows.

5.2 Autoformalization of Theorem with Inconsistent Hypotheses

Example b. *Problem*: Given a real number D and the condition that for non-zero real numbers a, b, c, the determinant of the matrix $\begin{bmatrix} a & b & c \\ 1 & 4 & 9 \\ 3 & 1 & 2 \end{bmatrix}$ equals D, prove that $D^2 = 154$.

Autoformalized Theorem in Lean:

```
example (D : ℝ) (h₀ : ∀ a b c : ℝ, a ≠ 0 ∧ b ≠ 0 ∧ c ≠ 0 →
  Matrix.det ![![a, b, c], ![1, 4, 9], ![3, 1, 2]] = D) : D ^ 2 = 154
```

The initial autoformalization incorrectly assumes that the condition $D^2 = 154$ universally applies to all non-zero real numbers a, b, and c. This assumption is not supported by the problem statement, which does not claim universal applicability. Instead, the formalization should aim to either identify specific values of a, b, and c that satisfy $D^2 = 154$ or demonstrate that no such values exist.

The model successfully identifies this inconsistency and provides a counterexample to demonstrate the absurdity of the hypothesis:

```
example (D : ℝ) (h₀ : ∀ a b c : ℝ, a ≠ 0 ∧ b ≠ 0 ∧ c ≠ 0 →
  Matrix.det ![![a, b, c], ![1, 4, 9], ![3, 1, 2]] = D) : False := by
  have h₁ := h₀ 1 2 3
  have h₂ := h₀ 1 4 9
  simp [Matrix.det_fin_three] at h₁ h₂
  linarith
```

A corrected version of the autoformalized theorem can be proposed as follows:

```
example (a b c : ℝ) (h₀ : a ≠ 0 ∧ b ≠ 0 ∧ c ≠ 0) :
  let D := Matrix.det ![![a, b, c], ![1, 4, 9], ![3, 1, 2]];
  D ^ 2 = 154
```

These examples illustrate the model's capability to verify proofs and identify hypothesis inconsistencies effectively. Further details can be found in Appendix A.2.

6 Conclusion

In this paper, we presented a method to generate extensive synthetic proof data from high-school and undergraduate-level mathematical competition problems. By translating natural language problems into formal statements, filtering out low-quality ones, and using iterative proof generation, we created 8 million proof data points and significantly improved the DeepSeekMath 7B model's performance in ATP when trained on this synthetic data. Our model outperforms GPT-4 and other methods on

benchmarks like miniF2F and FIMO. By open-sourcing our dataset and model, we aim to advance research in automated theorem proving and enhance the capabilities of large language models in formal mathematical reasoning. Currently, our work mainly focuses on algebra and number theory at the middle school and undergraduate levels. In future work, we will aim to expand the diversity of mathematical problems addressed, enhancing the general applicability of our methods in ATP.

Broader Impact

The research presented in this paper has the potential to significantly advance automated theorem proving by leveraging large-scale synthetic proof data generated from informal mathematical problems. This remarkable advancement can enhance the capabilities of large language models in formal theorem proving, contributing to more reliable mathematical proof verification and providing valuable educational resources for students and researchers. By directly releasing the code, model, and data, we aim to ensure the responsible use of our work, fostering further innovation and maintaining high standards of data privacy and intellectual property compliance.

References

J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

J. Avigad. Mathematics and the formal turn, 2023.

Z. Azerbayev, H. Schoelkopf, K. Paster, M. D. Santos, S. McAleer, A. Q. Jiang, J. Deng, S. Biderman, and S. Welleck. Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*, 2023.

K. Bansal, S. Loos, M. Rabe, C. Szegedy, and S. Wilcox. Holist: An environment for machine learning of higher order logic theorem proving. In *International Conference on Machine Learning*, pages 454–463. PMLR, 2019.

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

M. Crouse, I. Abdelaziz, B. Makni, S. Whitehead, C. Cornelio, P. Kapanipathi, K. Srinivas, V. Thost, M. Witbrock, and A. Fokoue. A deep reinforcement learning approach to first-order logic theorem proving. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6279–6287, 2021.

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.

E. First, M. N. Rabe, T. Ringer, and Y. Brun. Baldur: Whole-proof generation and repair with large language models, 2023.

J. M. Han, J. Rute, Y. Wu, E. W. Ayers, and S. Polu. Proof artifact co-training for theorem proving with language models. *arXiv preprint arXiv:2102.06203*, 2021.

Y. Huang, X. Lin, Z. Liu, Q. Cao, H. Xin, H. Wang, Z. Li, L. Song, and X. Liang. Mustard: Mastering uniform synthesis of theorem and proof data. *arXiv preprint arXiv:2402.08957*, 2024.

A. Q. Jiang, W. Li, J. M. Han, and Y. Wu. Lisa: Language models of isabelle proofs. In *6th Conference on Artificial Intelligence and Theorem Proving*, pages 378–392, 2021.

A. Q. Jiang, W. Li, S. Tworkowski, K. Czechowski, T. Odrzygóźdź, P. Miłoś, Y. Wu, and M. Jamnik. Thor: Wielding hammers to integrate language models and automated theorem provers. *Advances in Neural Information Processing Systems*, 35:8360–8373, 2022a.

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*, 2022b.