

参赛学生姓名： ALLEN HE

中学： 华润小径湾贝塞斯国际学校 (BIPH)

省份： 广东

国家/地区： 中国

指导老师姓名： Cody Kennedy, 严骏驰

指导老师单位： BIPH, 上海交通大学

论文题目： LLM Mathematical Reasoning
Grounded with Formal Verification

LLM Mathematical Reasoning Grounded with Formal Verification

Allen He

Abstract

As Artificial Intelligence (AI) systems improve in processing power and reasoning ability, it can become helpful to users in fields such as mathematics. For example, in a recent event, Fields Medalist *Terence Tao* explained that proof checkers and AI programs are dramatically changing mathematics. He said, “in three years AI will become useful for mathematicians. It will be a great co-pilot.” An AI system, such as a large language model (LLM), can be a great tool for assisting the user in solving mathematical problems, as it can create preliminary ideas and even (partial or intermediate) solutions for complex problems and explain them in understandable natural language (referred to as *informal solution* hereafter). However, when utilizing LLM techniques to solve such mathematical problems, it is common for an LLM generated solution to have illogical proof steps, such as using incorrect premises, skipping steps in the reasoning, or containing calculation errors, making answers both prone to mistakes and unjustified, namely *hallucination*. It echoes the opinion held by Terence Tao, that an LLM is more like a guessing machine in mathematics which inspires the mathematician in solving problems, rather than providing the solution directly [18]. On the other hand, Formal Theorem Proving (FTP), which is commonly used among mathematicians and computer scientists, serves as a tool for strictly checking the correctness of proofs (referred to as *formal solution* hereafter). Therefore, it is desirable to make the LLM generate solutions grounded on proofs verified by FTP, so as to mitigate LLM hallucination problems. In this work, we propose the method of augmenting an LLM with FTP to solve mathematical quantitative reasoning problems, namely, Reasoning Grounded with Formal Verification (RGFV). Compared to previous LLM-based approaches, RGFV takes a detour by formal proof, utilizing auto-formalization and formal theorem proving. It augments the LLM with a constructed formal statement and verified formal proof to generate informal solutions. Our experiments and ablation studies on a collection of mathematical competition problems show that, compared to previous work, RGFV improves the accuracy of the answers significantly from 46.7% to 65.8% overall, and even more significantly on the subcategory of algebra, from 66.7% to 96.7%. Moreover, it significantly improves the quality of the generated solutions. The method also helps AI be more reliable and useful for students studying mathematics.

Keywords: Large language model; mathematical proving; artificial intelligence;

Contents

| | |
|---|-----------|
| Abstract | 2 |
| 1. Introduction | 4 |
| 2. Related Work | 5 |
| 2.1 Large Language Model Based Mathematical Reasoning | 5 |
| 2.1.1 Large Language Model | 5 |
| 2.1.2 LLM Based Mathematical Reasoning | 5 |
| 2.1.3 In-Context Learning, Prompt and Demonstration | 6 |
| 2.2 Formal Theorem Proving | 6 |
| 2.2.1 Automated Theorem Proving | 6 |
| 2.2.2 ATP with LLMs | 7 |
| 3. Method | 8 |
| 3.1 Statement Auto-formalization | 9 |
| 3.2 Generating Natural Language Solution Candidates | 9 |
| 3.3 Formal Proof Candidate Generation & Verification | 9 |
| 3.4 Solution Auto-Informalization | 10 |
| 4. Experiment and Results | 10 |
| 4.1 Experiment Setup | 10 |
| 4.2 Baseline and Evaluation Metrics | 13 |
| 4.3 Main Experiment Results | 14 |
| 4.4 Ablation Study | 15 |
| 4.5 Case Study | 17 |
| 5. Conclusion and Future Work | 21 |
| 6. Acknowledgements | 21 |
| References: | 23 |
| Appendix | 25 |
| Few-shot demonstrations | 25 |
| Prompt for RGFV pipeline | 28 |

1. Introduction

LLMs have recently been used for generating informal solutions for mathematical problems, and some have achieved a level of proficiency already. For example, OpenAI’s GPT-4 achieved the 89th percentile in the SAT math portion [19]. However, there is one major flaw in the informal solutions generated by LLMs: *hallucination*. LLMs will often generate outputs that deviate from factual error or coherence, which is detrimental to the solution, leading to incorrect answers or inconsistencies between answer and proof.

In mathematical proving, there are generally two types of solutions: natural language solutions and formal language solutions. The former is often referred to as an *informal solution*, and the latter is accordingly referred to as a *formal solution*. Informal solutions involve establishing the truth of mathematical statements building on existing knowledge via intuitive reasoning and natural language explanations, and formal solutions require presenting theorems and proofs in a machine-verifiable format, ensuring their correctness using rigorous logical rules. While informal solutions are easy for humans to understand, they are hard to verify formally (i.e., mathematically). Formal solutions, contrarily, are simple and easy for computers to verify, but hard for humans to understand. Therefore, by integrating these two types of methods, we aim to get the best of both worlds, generating verified solutions in natural language.

In this work, we propose using LLMs with formal theorem proving to alleviate the problem of LLM hallucination in mathematical reasoning. Briefly, we *translate* the original natural language problem into structured formal language [1] to create a *formal statement*. This step is also referred to as *formalization*. Then an auto-theorem prover (ATP) can be used to generate and verify a *formal proof*. In order to further guide the ATP and boost its success rate, we utilize the LLM to propose informal solution candidates given the informal problem, i.e., *guess* a set of likely solutions. Then we translate these informal solution candidates into formal language, called *formal proof templates*, which can be used as guidance to help the ATP generate the formal proof. In this process, an informal solution candidate might skip some logical steps which are necessary for formal reasoning. Therefore, we generate the formal proof template that has the major steps of reasoning, but leave blank low-level proof details (e.g., premise selection, calculation, etc.), and let the ATP fill in the gaps to generate a full formal proof. The translations between the informal language and the formal language are performed by a pre-trained LLM with In-Context Learning (ICL), which will be elaborated on in greater detail in future sections.

Once a formal proof is generated and verified (i.e., passes the Isabelle Checker), it is *translated* back via *informalization* into an informal solution in natural language. Since the solution is grounded on a verified proof, the effects of hallucination are greatly mitigated. The proposed approach is evaluated on the MATH [2] dataset, encompassing questions from multiple fields in mathematics.

This work has made the following highlights:

- The LLM based **Reasoning Grounded with Formal Verification** (RGFV) framework for solving mathematical problems, which is an early attempt to leverage verified formal proofs with LLMs to generate reliable natural language solutions.
- Using LLMs with the In-Context Learning method for translations between formal and informal language in both directions.
- Comprehensive evaluations on the effectiveness of the RGFV method on public benchmarks. Our evaluation shows that RGFV improves the accuracy of the answers significantly from 46.7% to 65.8%. Moreover, it also significantly improves the quality and understandability of the generated solutions.

2. Related Work

2.1 Large Language Model Based Mathematical Reasoning

2.1.1 Large Language Model

Large Language Models (LLMs) have seen significant progress in natural language processing tasks, driven by the development of transformer-based architectures [20] such as GPT [19,25,26] and BERT [27]. LLMs have demonstrated impressive performance across a wide range of tasks, including text generation, question answering, summarization, and translation [21,22,23,24]. These models leverage vast amounts of data and computational power to learn context, semantics, logics and linguistic patterns, enabling them to generate human-like text and perform few-shot or zero-shot learning. Decoder-only Large Language Models (LLMs), also known as autoregressive models, are a subclass of transformer-based architectures that focus exclusively on the decoder component of the transformer framework. Unlike models that employ both encoder and decoder layers (e.g., BERT), decoder-only LLMs are designed to predict the next token in a sequence based on previously generated tokens, making them particularly effective for text generation tasks. The most well-known examples of decoder-only LLMs include the GPT series [25,26], with GPT-3 [25] and GPT-4 [19] representing significant milestones in the development of autoregressive language models.

2.1.2 LLM Based Mathematical Reasoning

Informal mathematical reasoning involves solving a mathematical problem based on existing knowledge, using intuitive reasoning and natural language in explanations. This mirrors how people tackle and solve problems in everyday mathematics. For instance, to solve an arithmetic problem, one needs to comprehend basic concepts such as primes and might apply a series of reasoning and calculation steps to find an answer. Despite the ubiquity of informal mathematical reasoning, as the difficulty of mathematics increases, the solutions and proofs tend to be more intricate and complex, making verifying their correctness increasingly difficult.

Recently, large language models have showcased their proficiency in tackling a diverse spectrum of mathematical problems [4,5], spanning from middle school math word problems (e.g., GSM8K [6]) to high school mathematical competition challenges (e.g., MATH[2]). Utilizing prompting techniques such as Chain-of-Thoughts [7] and Tree-of-Thoughts [8], these models can mimic human-like planning and reasoning, producing step-by-step solutions and answers. However, these solutions and answers suffer from hallucination problems, when the LLM does not certainly guarantee the accuracy and logical soundness of solutions. They are prone to generate solutions that seem correct, but upon closer inspection contain logical or computational mistakes.

More recent work such as the self-consistency method [9] addresses this problem by using a majority vote on multiple solutions generated by large language models, providing a more trustworthy answer. Although it gives better performance compared to previous work, it is still often observed that the original informal solution contains unjustifiable reasoning steps accompanying a correct answer, or that the majority of the answers are not correct, instead another less-popular solution might give the correct answer.

Different from these approaches, RGFV uses a formal theorem proving system to find the correct reasoning path, i.e., it augments the LLM with a verified formal proof to mitigate any hallucination problems, and generate reliable natural language solutions.

2.1.3 In-Context Learning, Prompt and Demonstration

In-context learning [11] refers to the ability of a model to perform a task based on a few examples or instructions provided within the input itself, without the need for additional fine-tuning or parameter updates. This approach is particularly notable in the context of large pre-trained language models, such as GPT-3 [25], where the model's vast knowledge base and contextual understanding allows it to adapt to new tasks dynamically without extra training.

A *prompt* is textual input that guides the model's behavior by specifying the task, providing examples, and/or setting the context. Prompts can range from simple instructions to more complex structured inputs that include multiple examples or constraints.

Demonstrations are a specific type of prompt that involve providing the model with examples of the task at hand, such as examples of proofs for math problems. These examples serve as a form of implicit supervision, allowing the model to grasp and carry out the task by observing how similar inputs have been processed in the prompt.

2.2 Formal Theorem Proving

2.2.1 Automated Theorem Proving

Formal theorem proving (FTP) represents theorems and proofs in a machine-verifiable format, ensuring their correctness using rigorous logical rules and procedures, while automated theorem proving (ATP) aims to verify formal statements automatically without human intervention.

Theorem provers, including E [29] and Vampire [30], mainly operate on first-order logic (FOL) in order to autonomously generate logical consequences from a set of axioms until a proof or refutation is derived, or computational limits are reached. Similarly, geometric ATP systems such as GEX [31] prove geometry problems by iteratively applying deduction rules. Despite the sophisticated designs of these ATP systems, the inherently required vast search space often limits their practicality in more complex problems.

Automated Theorem Proving systems, like Isabelle [1], leverage sophisticated algorithms such as dependent-type theory and heuristics to handle complex mathematical theories and large-scale proofs. The integration of ATP with interactive theorem provers has further expanded its capabilities, allowing for collaborative human-machine theorem proving. Recent developments in machine learning, particularly the application of neural networks, enhance ATP systems with data-driven approaches. AI models, like neural networks, can interact with ATP systems to prove and discover new theories. These innovations aim to improve the efficiency and scalability of theorem proving, making it more accessible and applicable to a broader range of mathematical domains.

2.2.2 ATP with LLMs

There are two types of tasks in the field of LLMs for ATP: (1) Auto-formalization and (2) Formal Theorem proving.

Auto-formalization refers to the task where an informal statement is autonomously translated into a formal statement written in formal language like Isabelle [1], Lean [14] and Coq [3]. There are several methods available that can achieve this goal. Firstly, LLMs pre-trained on formal language corpus can be utilized to complete this task via In-Context Learning [11]. The effectiveness of this method has previously been studied [12], which demonstrates that LLM can be employed to auto-formalizing informal proofs into formal ones like Isabelle. Secondly, a special-purpose translation system can be trained on datasets of formal proofs including AFP (Archive of Formal Proofs) and the Mathlib library built for formal theorem provers [17], so as to perform the auto-formalization task. However, this approach heavily relies on the availability of training data consisting of informal-formal statement pairs. Our approach adopts the first method of using a pre-trained LLM, and therefore does not require training.

Formal Theorem Proving aims to prove a formal statement, written in formal language, that can be checked by the theorem prover. There are two ways to complete this goal [10, 13, 32]. (1) Proof Search method [32,33] is a tree search method that generates proofs that are step-by-step conditioned on the proof state (a set of conditions and goals to refine) given by a theorem prover, such as Isabelle. The proof search method proves the theorem progressively and refines the proof state until the “no goals” state is reached, which occurs when the proof search method successfully finds a formal proof. (2) In [10, 13], LLMs are utilized to perform formal theorem proving via one step generation. By In-Context Learning, LLM first generates an informal proof. Then, the informal proof is auto-formalized into a formal proof sketch. Finally, the Isabelle prover verifies the sketch and finishes proving if it passes the check.

Rather than performing theorem proving in the formal language domain, RGFV aims to generate informal solutions that are grounded by LLM reasoning with formal theorem verification, i.e., given the complete and verified formal proof, the LLM can generate reliable natural language solutions. Since most of the available data in mathematic reasoning is written in natural language, RGFV attempts to tackle a more realistic but challenging problem than ATP.

3. Method

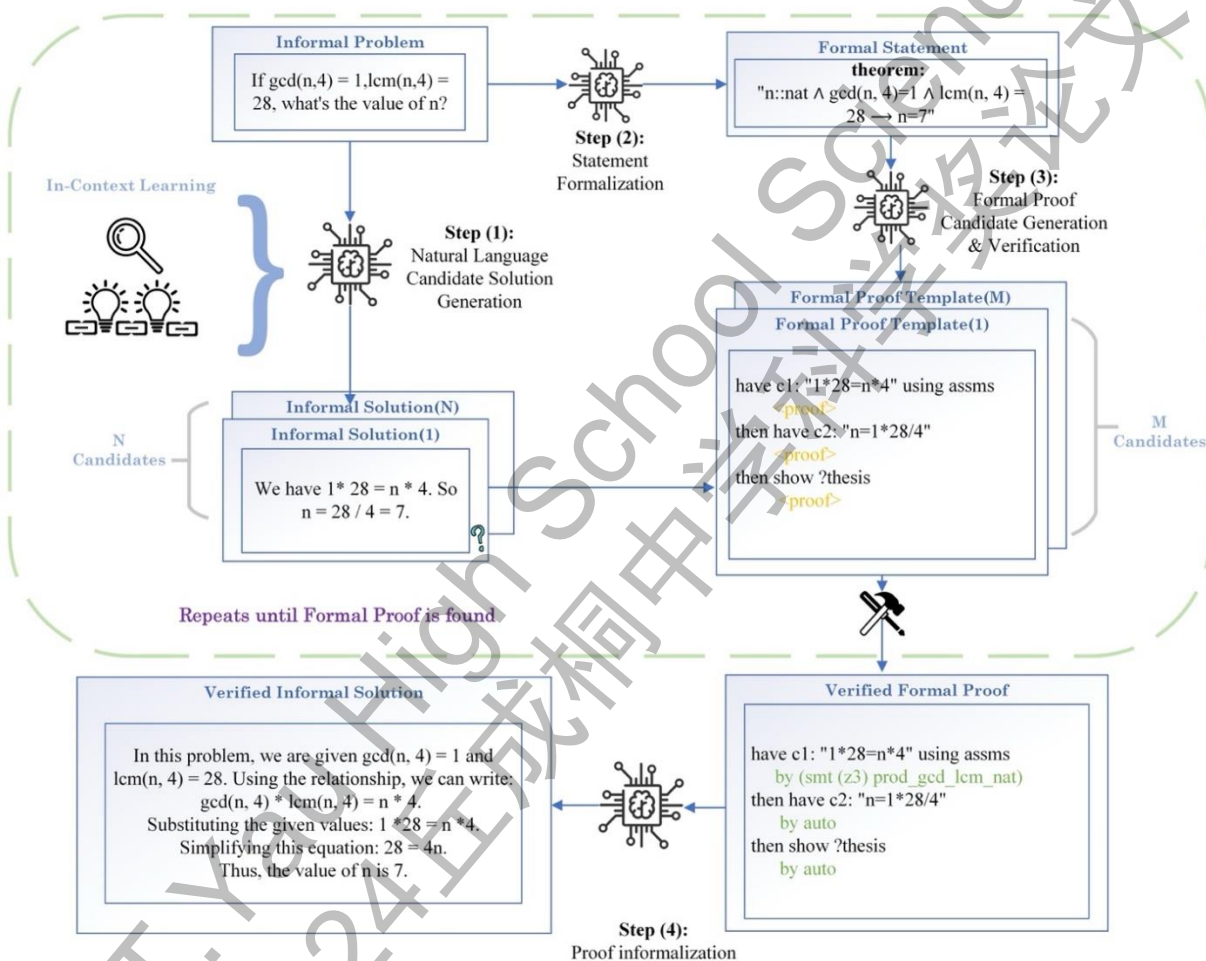


Figure 1: The overview of the RGFV framework. Starting from the informal problem, we first use the LLM with few-shot prompts to generate N informal solution candidates with potential answers. Then, we translate the informal problem into a formal statement via formalization, also by LLM. Thirdly, we prompt the LLM to generate a set of M formal proof templates given an informal solution candidate and the formal statement. Following, we run the Isabelle prover to fill in the missing low-level proof step gaps in the template and verify if the formal proof is correct or not. This process repeats until timeout or a formal proof is verified by the Isabelle prover. Once verified, we find the correct reasoning path in the formal language domain. Finally, we perform an auto-informalization process to obtain the verified, hallucination-free informal solution.

3.1 Statement Auto-formalization

Given an informal mathematical problem and a potential answer, we denote the problem as a propositional function $P_i(\cdot)$, the solution as p_i , the answer as a value \hat{x} . Then, an informal statement can be instantiated as $P_i(\hat{x})$. For example, given the problem “if $\gcd(n, 4) = 1, \text{lcm}(n, 4) = 28$, Find the natural number n ” and an answer candidate, e.g., 7, we can construct a proving statement “if $\gcd(n, 4) = 1, \text{lcm}(n, 4) = 28$, then $n = 7$ ”.

In this step, the informal statement is fed into an LLM to generate a formal statement in Isabelle. Isabelle requires the formal statement to be assertive, such as “prove that $x = 2$,” yet there are many math questions that are not proofs, such as the algebraic equation “ $x+2=4$ ” which is not a provable question. Therefore, to avoid this problem, the informal statement is transformed by a LLM into a proper, assertive formal statement, such as: “let $x+2=4$, prove $x=2$.” We give the LLM few-shot example in-context learning, then prompt it to create formal statements for the whole dataset.

Formally, we sample the formal statement as follows

$$s_f \sim p_{\text{formalizer}}(\cdot \mid P_i(\hat{x}), \{(P_{i,j}(\hat{x}_j), s_{f,j})\}_{j=1}^D)$$

where s_f is the formal statement, $p_{\text{Autoformalizer}}$ is the distribution of the auto-formalizer model in which we use a LLM (controlled by sampling method and prompt), each of $(P_{i,j}(\hat{x}_j), s_{f,j})$ is a demonstration. A demonstration contains a pair consisting of an informal statement and a formal statement example. D is the number of demonstrations, which is usually a number in the range from 20 to 50. In few-shot in-context learning, a prompt is fed as the input to the LLM, instructing the LLM to receive these D demonstrations as few-shot examples, and generate s_f for $P_i(\hat{x})$ through sampling.

3.2 Generating Natural Language Solution Candidates

In this process, LLM generates a set of informal proof candidates for the original problem posed in natural language. We use the DeepSeek-coder-v2 [5] LLM for this task, for its low expense and high capability as a domestic, readily available LLM in China. Our framework is agnostic to the specific LLM choice. It receives the few-shot examples, then is prompted to generate a set of N solutions for a subsequent problem via in-context learning. These solution candidates are not guaranteed to be accurate and may contain logical fallacies or discontinuities, i.e., *hallucinations*. However, they can be viewed as good guesses of likely proofs, and can be used to guide the ATP.

3.3 Formal Proof Candidate Generation & Verification

We leverage Isabelle to verify proofs for the formal statement. Isabelle [1] is one of the mainstream formal theorem provers, along with Lean [14] and Coq [3], which have the ability to verify a proof in formal language. We selected Isabelle for its powerful tools, such as Sledgehammer, which can

help to fill in low-level proving gaps in proof generation and verification. The DeepSeek-coder-v2 LLM creates templates of Isabelle proofs according to the statement and prior informal solution candidates, prompted with few-shot examples. These templates do not contain the theorems utilized in the proofs; those are instead filled in by the Sledgehammer tool in Isabelle. These complete proofs are then checked by Isabelle to find a proof that has correct logical reasoning and a correct answer.

Given informal statement $P_i(\hat{x})$, the corresponding informal solution p_i and formal statement s_f , we sample the formal proof p_f as follows

$$p_f \sim p_{\text{FormalProver}}(\cdot \mid P_i(\hat{x}), p_i, s_f, \{(P_{i,j}(\hat{x}_j), p_{i,j}, s_{f,j})\}_{j=1}^D)$$

where $p_{\text{FormalProver}}$ is the distribution of the Formal Proof generation model, which we use the DeepSeek-coder-v2 [5] LLM with in-context learning. These D triples, $(P_{i,j}(\hat{x}_j), p_{i,j}, s_{f,j})$, are the few-shot examples for in-context learning.

3.4 Solution Auto-Informalization

If a complete formal proof is verified to be logically correct by Isabelle, it is translated back into natural language by the LLM to give the more reliable informal solution, i.e., informalization. The LLM is prompting with few-shot examples to perform informalization. This auto-informalized solution is more reliable than the informal solution that is generated only conditioning on the original informal problem.

Given informal statement $P_i(\hat{x})$, the corresponding informal solution p_i and formal statement s_f and verified complete formal proof p_f , we reconstruct a formal-proof-grounded informal solution as follows

$$\tilde{p}_i \sim p_{\text{Informalizer}}(\cdot \mid P_i(\hat{x}), p_i, s_f, p_f, \{(P_{i,j}(\hat{x}_j), p_{i,j}, s_{f,j}, p_{f,j})\}_{j=1}^D)$$

where $p_{\text{informalizer}}$ is the distribution of the informal solution generation model, which we use the DeepSeek-coder-v2 [5] LLM with in-context-learning, too.

4. Experiment and Results

4.1 Experiment Setup

Evaluation Dataset. We evaluate our method on a constructed subset of the MATH dataset [2], which is a prevalent high-school level mathematical problem-solving benchmark. It comprises 12,500 challenging problems (5,000 for test) sourced from high school mathematics competitions, including the AMC 10, AMC 12, and AIME. These problems are categorized into different subjects

with difficulty levels ranging from 1 to 5. In order to study the effect of using formal proving in our method, we further select problems which have corresponding formal statements in the miniF2F^[15] dataset. The MiniF2F dataset is a benchmark in formal theorem proving, where some of the problems originate from the MATH dataset, which has been formalized by experts in the formal theorem proving field.

```
Informal Statement:
The greatest common divisor of positive integers m
and n is 6. The least common multiple of m and n is
126. What is the least possible value of m+n? Show
that it is 60.
Formal Statement:
"theorem mathd_numbertheory_277:
  fixes m n ::nat
  assumes "gcd m n = 6"
  and "lcm m n = 126"
  shows "60 ≤ m + n"
```

(a) a Number Theory problem

```
Informal Statement:
Real numbers x and y have an arithmetic mean of 7
and a geometric mean of  $\sqrt{19}$ . Find  $x^2+y^2$ . Show that
it is 158.
Formal Statement:
theorem mathd_algebra_332:
  fixes x y ::real
  assumes "(x + y) / 2 = 7"
  and "sqrt (x * y) = sqrt 19"
  shows "x^2 * y^2 = 158"
```

(b) an Algebra problem

Figure 2. Examples of informal problem and the formal statement of problems in the Number Theory and Algebra subjects, respectively.

To avoid any too-heavy computation requirements of the LLM, we perform evaluation on a subset of the original datasets. Specifically, we randomly sampled 60 Algebra problems and 60 Number Theory problems from the intersection of MATH and miniF2F, leading to a total of 120 problems in our evaluation set. The distribution of difficulty levels of the constructed test set is akin to the original MATH dataset, while the number of all levels of problems are almost at the same size. Note that since our method only relies on a pre-trained LLM with in-context learning, no training data is needed except some few-shot examples as demonstrations. We construct the demonstration examples from data sampled outside our evaluation dataset. Figure 2 above presents several examples of informal problem and the formal statement in our evaluation dataset. **Model.** We use the deepseek-coder LLM [5] in the experiments. The model has 236 billion parameters. Although deepseek-coder is assumed to be a specialized AI model designed to assist developers and

programmers in writing, optimizing, and debugging code, its training data contains many examples of mathematics, too, and shows proficiency in solving math problems according to its technical report [5].

Few-shot Prompting Examples. We construct 36 demonstrations from MATH train split. Each demonstration consists of an informal problem, informal solution, formal statement and formal proof. We use all the demonstrations in the pipeline of RGFV including (1) Informal Solution Generation, (2) Statement Formalization, (3) Formal Theorem Proving, and the necessary fields are used in each step. More details of demonstrations can be found in the appendix.

Informal Solution Generation. We few-shot prompt the deepseek-coder LLM to generate $N = 10$ informal solutions per problem conditioned on the informal problem statement and demonstrations of (informal problem, informal solution) pairs to leverage the in-context learning ability of LLM to generate the informal solution. We use the default sampling parameters in OpenAI API [28] where $temperature = 0.7$ and nucleus sampling with $top_p = 1.0$. The prompt can be found in the appendix.

Statement Formalization. We use the above 36 examples with (informal problem, informal solution, formal statement) tuples for few-shot demonstrations. Because Isabelle theorem prover is only capable of proving, not solving, the answer in the informal solution must combine with the informal problem to construct an informal statement for statement formalization. We perform single statement formalization concerning each informal solution generated in the informal solution generation stage and in total $N = 10$ formal statements are generated per problem. The default sampling parameters in OpenAI API [28] where $temperature = 0.7$ and nucleus sampling with $top_p = 1.0$ is adopted and the prompt can be found in the appendix.

Formal Proof Generation & Verification. We use the above 36 examples with (formal statement, formal proof) pairs for few-shot demonstrations to perform formal proof template generation. To save computation resources, we query the language model once ($M = 1$) for formal proof generation for each formal statement sample so that each informal problem is in total formalized $N \times M = 10 \times 1$ times, generating 10 formal proof templates. We then use Isabelle to verify the potential formal theorem (the formal statement together with the formal proof template). Since the template might contain low-level proving gaps, we prompt the deepseek-coder LLM to use sledgehammer [34] along with 11 common tactics (*auto*, *simp*, *blast*, *fastforce*, *force*, *eval*, *presburger*, *sos*, *arith*, *linarith*, and *auto simp: field_simps*) to close low-level open conjectures to generate a complete formal proof. Then, the Isabelle theorem prover is called to verify the correctness of the proof.

Proof Auto-Informalization. Once the formal proof passes the verification of the Isabelle theorem prover, the original informal problem is viewed as being formally solved. The verified formal proof will be utilized to perform informalization, also via the deepseek-coder LLM with in-context learning, to generate a final informal solution with correct reasoning steps, and therefore resolving

the issue of hallucination in the solution. The default sampling parameters are used as mentioned above, and the prompt can be found in the appendix.

4.2 Baseline and Evaluation Metrics

Our task includes several steps: (1) generating informal solution candidates and potential answers for a given problem, (2) constructing a formal statement based on the informal problem and a potential answer, (3) generating formal proof templates based on the informal solution candidates, (4) generating and verifying formal proofs using Isabelle and sledgehammer tools, (5) composing the verified formal proof and generating a grounded informal solution. Therefore, for a comprehensive evaluation, we evaluate from three perspectives.

Accuracy. To evaluate the effectiveness of RGFV, we measure its accuracy. In this evaluation, we define "accuracy" as the ratio of correctly-answered problems among all the problems. Note that, if ATP cannot find a correct proof, we will fall back to the most likely solution and answer generated by the LLM without using formal proof. As shown in Figure 1, the formalization and proving process will repeat k times, referred to as k trials. We use $Acc@k$ to denote the accumulated accuracy in k trials. The reason to try multiple trials is that the sampling process of the LLM gives diverse outputs that are different each time, like a guessing machine. Therefore, iterating k trials increases the coverage, increasing the chance of finding a correct proof.

Solution Quality. The goal of this project is to improve the quality of informal solutions for mathematical problems. For a fair and objective evaluation, we leverage a supermodel GPT-4 [19] to score the quality of solutions. Specifically, for each pair of solutions (e.g., one from baseline method, and one from RGFV), we use 2-shot prompting to score the two solutions by $\{-1, 0, 1\}$. We ask GPT-4 to compare the quality of the two solutions from four perspectives: Correctness, Clarity, Readability and Rigor, and give a relative score comparing RGFV with the baseline. To avoid the LLM's bias on the order of solutions presented, we randomly permute the two solutions and repeat the scoring eight times. The final score is an average of these eight trials.

Verification Pass Rate. To analysis the robustness of formal verification, we also report pass rate, which is the ratio of formally-verified problems among all the problems. We use $Pass@k$ to denote the accumulated pass rate in k trials.

We use the (1) single-best solution method and (2) self-consistency method proposed in [9] recently as our baseline. Both baselines exploit a LLM with in-context learning to generate informal solutions and answers given the original problem without using formal proofs. The former uses the LLM to output the most likely solution, while the latter will perform a majority-vote to determine the final answer so as to enhance the self-consistency. We compare accuracy on answers with the above methods. As the ability of LLM's auto-formalization has its limitations, and not all the formal proof will pass the checking in theorem prover, we will also report the pass rate concerning the auto-formalization process.

4.3 Main Experiment Results

The main results are shown in table 1 below. We compare our method with the (1) single-decoding solution method and (2) self-consistency method [9]. Concerning the effect of verifier-grounded proofs on the final accuracy on the problem, we report the **Accuracy** on answer of our proposed method. The higher accuracy on answer shows the effect of verifier-grounded proofs helping the LLM generate a more reliable answer.

| Method | Accuracy (Overall) | Accuracy (Algebra) | Accuracy (Number Theory) |
|-----------------------------|--------------------|--------------------|--------------------------|
| single-decoding method [9] | 38.3% | 58.3% | 18.3% |
| self-consistency method [9] | 46.7% | 66.7% | 26.7% |
| RGFV | 65.8% | 96.7% | 35.0% |

Table 1. Accuracy of RGFV compared with single-decoding and self-consistency method, with number of trails k set to 10.

Results in Table 1 show that, compared to the stronger baseline of self-consistency method [9], the RGFV boost the overall accuracy from 46.7% to 65.8%, a 19.1% absolute improvement, demonstrating the effectiveness of grounding the reasoning on formal verification. We also observed that the accuracy on the subject of Number Theory is relatively lower than Algebra, likely due to the fact that Number Theory problems often require more steps of reasoning, which highlights both the importance and challenge for AI to perform reasoning tasks. However, the improvement is still very significant for Number Theory problems, improving from 26.7% to 35.0%, a 31.1% improvement relatively.

Additionally, we evaluated the quality of the informal solution generated by the auto-informalization process. We adopt GPT-4 [19] as the super model to conduct the evaluation. Comparing RGFV with the self-consistency method [9], we measure the relative quality of the solutions generated by these two approaches from 4 perspectives: Correctness, Clarity, Readability, Rigor. The relative score is ranged from -1 to 1.

| perspective | Correctness | Clarity | Readability | Rigor |
|---------------------------|-------------|---------|-------------|---------|
| RGFV vs. self-consistency | +0.2769 | +0.6789 | +0.4132 | +0.7801 |

Table 2. GPT4 super model evaluation scores on Correctness, Clarity, Readability and Rigor.

Results in Table 2 show that RGFV outperforms the baseline in all four aspects, especially in the category of Rigor, showing that our method gives significant advantages. Overall, RGFV

outperforms the baseline by a large margin of 0.5373, which is very significant. We present typical examples in section 4.5.

4.4 Ablation Study

In this section, we first evaluate the ability of the LLM to do statement formalization, i.e., its ability of *translation* between informal and formal language. We provide the ground truth (GT) formal statement in the pipeline to replace the LLM-generated formal statement. Since this concerns the formal proving step, we measure the formal proof pass rate.

| Method | Pass Rate (Oveall) | Pass Rate (Algebra) | Pass Rate (Number theory) |
|--|--------------------|---------------------|---------------------------|
| RGFV (w/ GT formal statement) | 65.0% | 93.3% | 36.7% |
| RGFV (w/ LLM-generated formal statement) | 63.3% | 91.7% | 35.0% |

Table 3. The pass rate of RGFV (with/without ground truth formal statement) on sub-category on Algebra and Number Theory.

The result shows that (1) even with GT formal statement, the pass rate is still far from perfect, especially on the Number Theory subject. This indicates that the bottleneck relates more to the formal theorem proving process. This observation is consistent with the recent progress in Formal Theorem Proving [13], that the $PassRate@100$ on `minif2f` is around 50%. (2) the gap between using GT formal statement vs. LLM-generated formal statement is quite small, in the range of only 1-2%. This shows that LLM performs reasonably well on the task of *translation* between informal and formal languages. (3) the pass rate on Algebra is much higher than the pass rate on Number Theory, showing RGFV is more capable of solving Algebra problem than Number Theory, partially because Algebra problem focuses more on computation, which is relatively easier, while Number Theory problems need more steps and reasoning.

Compared to the results in table 1, we also observed that the accuracy of RGFV is slightly higher than its pass rate, indicating that there are a few cases where no correct formal proof can be found, but the answer is still correct. Note that if ATP cannot find a correct proof, we will back off to the answer generated by the LLM only. This indicates there are cases where although RGFV cannot find any verified proof, the LLM still makes a good guess on the answer.

Insight into the iteration k with pass rate:

Figure 3 below shows that pass rate monotonically increases with respect to the iteration k , and starts becoming saturated after $k=8$. We use the default sampling parameters with $temperature = 0.7$ and nucleus sampling with $top_p = 1.0$. This validates the saying held by Terence Tao that LLM is a good guessing machine [18], and we let the LLM to guess multiple times so to increase the chance for the ATP to find a valid proof. More iterations could improve the pass rate but will cause a large cost to invocations of deepseek-coder API. Therefore, we will report $pass@10$.

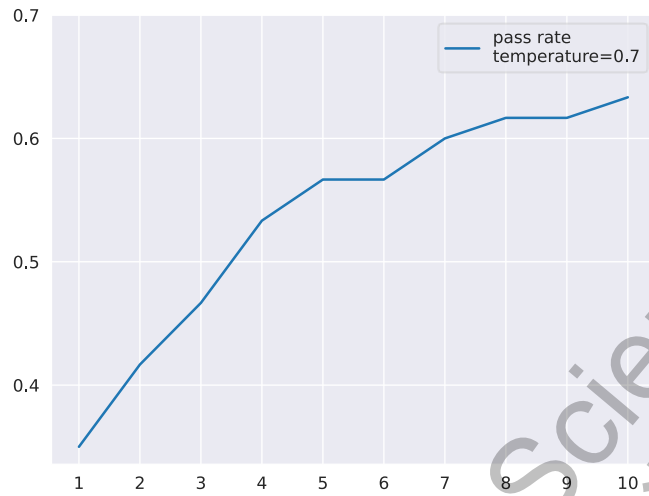


Figure 3: The accumulative pass rate up to 10 iterations, using the default value of temperature=0.7.

The effect of temperature for the LLM to generate formal proof:

Temperature is indeed an important hyperparameter in the context of LLMs when sampling the output. It controls the randomness and diversity of the model's output. At low temperature, the model tends to produce more deterministic and consistent outputs, while at high temperature, the model becomes more exploratory and produces more diverse and random outputs. The model outputs logits z_i for each token in the vocabulary, and the Softmax function converts these logits into probabilities p_i . The softmax function is given by:

$$p_i = \frac{\exp(z_i/t)}{\sum_j \exp(z_j/t)}$$

When the temperature t approaches 0, The system becomes fully deterministic, choosing the token with the highest probability. As t approaches infinity, the probability distribution becomes uniform, meaning all tokens have an equal chance of being sampled.

We study the effect of temperature on the pass rate with 10 iterations, Figure 4 shows how different temperature affect the final pass rate@10.

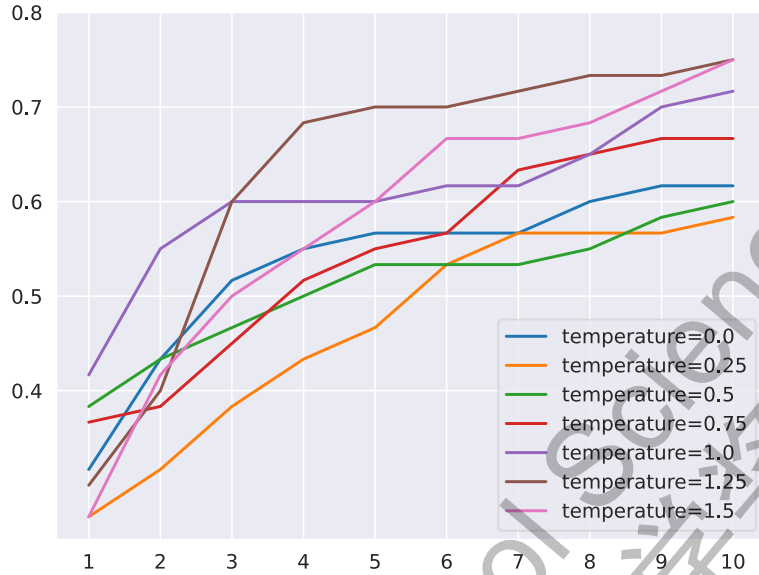


Figure 4: Accumulative pass rate curves with different temperature parameters.

Compared to the default sampling parameters in OpenAI API [28] where $temperature = 0.7$, a higher temperature (temperature=1.25 or 1.5) yields even better performances, boosting the pass rate up to 75%. High temperatures enable the large language model to generate more diverse solutions, so as to increase the likelihood of having ones that lead to correct formal statements and proofs that meet the verifier's check. This underscores the importance of exploiting LLM as a *guessing machine* to solve challenging mathematical problems.

4.5 Case Study

RGFV can alleviate hallucination through formal verification. We show three cases below to illustrate that, while LLM-generated informal solutions have some hallucination in their proving steps such as computational errors, illogical reasoning steps, etc., our RGFV method can generate a good informal solution grounded with formal verification to eliminate these hallucination errors.

Informal Problem from Algebra: For what real value of k is $\frac{13-\sqrt{131}}{4}$ a root of $2x^2 - 13x + k = 0$? (The Answer is $\frac{19}{4}$.)

Generated Informal Solution with Hallucination:
 To determine the value of k for which $\frac{13-\sqrt{131}}{4}$ is a root of the quadratic equation $2x^2 - 13x + k = 0$, we need to use the properties of quadratic equations and their roots.
 Given that $\frac{13-\sqrt{131}}{4}$ is a root, let's denote it as x_1 . The quadratic equation can be written as: $2x^2 - 13x + k = 0$
 Since $x_1 = \frac{13-\sqrt{131}}{4}$ is a root, it must satisfy the equation: $2\left(\frac{13-\sqrt{131}}{4}\right)^2 - 13\left(\frac{13-\sqrt{131}}{4}\right) + k = 0$
 First, simplify $\left(\frac{13-\sqrt{131}}{4}\right)^2$:

$$\left(\frac{13-\sqrt{131}}{4}\right)^2 = \frac{(13-\sqrt{131})^2}{16}, (13-\sqrt{131})^2 = 169 - 26\sqrt{131} + 131 = 300 - 26\sqrt{131}$$

$$\left(\frac{13-\sqrt{131}}{4}\right)^2 = \frac{300 - 26\sqrt{131}}{16} = \frac{75 - 13\sqrt{131}}{4}$$

Next, substitute this back into the equation:

$$2 \cdot \frac{75 - 13\sqrt{131}}{4} - 13 \cdot \frac{13 - \sqrt{131}}{4} + k = 0, \frac{-19 - 13\sqrt{131}}{4} + k = 0$$

Thus, the value of k is:

$$\frac{19 + 13\sqrt{131}}{4}$$

Generated Formal Statement and Formal Solution:

```

theorem
  fixes k :: real
  assumes "2 * (13 - sqrt 131) / 4 + (13 - sqrt 131) / 4 - 13 * (13 - sqrt 131) / 4 + k = 0"
  shows "k = 19 / 4"
proof -
  (* We use the quadratic formula to find the roots of the equation *)
  have "2 * (13 - sqrt 131) / 4 + (13 - sqrt 131) / 4 - 13 * (13 - sqrt 131) / 4 + k = 0"
  using assms by auto
  (* The quadratic formula is given by: *)
  (* x = (-b ± sqrt(b^2 - 4ac)) / 2a *)
  (* For the equation 2x^2 - 13x + k = 0, we have a = 2, b = -13, and c = k. *)
  (* Plugging these values into the quadratic formula, we get: *)
  (* x = (13 ± sqrt(169 - 8k)) / 4 *)
  (* We are given that one of the roots is (13 - sqrt 131) / 4. *)
  (* Therefore, we set: *)
  (* sqrt(169 - 8k) = sqrt 131 *)
  (* Squaring both sides, we obtain: *)
  (* 169 - 8k = 131 *)
  (* Thus, the value of k is: *)
  (* k = 19 / 4 *)
  then show "k = 19 / 4" by (auto simp: field_simps)
qed
  
```

Generated Informal Solution by Auto-Informalization:
 To determine the value of k for which $\frac{13-\sqrt{131}}{4}$ is a root of the quadratic equation $2x^2 - 13x + k = 0$, we can follow these steps:

- Identify the quadratic formula:** The quadratic formula for finding the roots of a quadratic equation $ax^2 + bx + c = 0$ is given by: $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
- Substitute the coefficients:** For the given quadratic equation $2x^2 - 13x + k = 0$, we have $a = 2$, $b = -13$, and $c = k$. Plugging these values into the quadratic formula, we get: $x = \frac{13 \pm \sqrt{169 - 8k}}{4}$
- Set one root equal to the given value:** We are given that one of the roots is $\frac{13-\sqrt{131}}{4}$. Therefore, we set: $\frac{13 - \sqrt{169 - 8k}}{4} = \frac{13 - \sqrt{131}}{4}$
- Equate the numerators:** Since the denominators are the same, we can equate the numerators directly: $13 - \sqrt{169 - 8k} = 13 - \sqrt{131}$
- Simplify the equation:** Simplifying the equation, we get: $\sqrt{169 - 8k} = \sqrt{131}$
- Square both sides:** To eliminate the square roots, we square both sides of the equation: $169 - 8k = 131$
- Solve for k :** Solving for k , we first isolate k by subtracting 131 from both sides: $169 - 131 = 8k$

$$38 = 8k$$

Then, divide both sides by 8:

$$k = \frac{38}{8} = \frac{19}{4}$$

8. Conclusion: Thus, the value of k is:

$$\frac{19}{4}$$

Final Answer: The final answer is $\frac{19}{4}$. This confirms that the value of k for which $\frac{13-\sqrt{131}}{4}$ is a root of the quadratic equation $2x^2 - 13x + k = 0$ is indeed $\frac{19}{4}$.

(a) LLM hallucination leading to incorrect computational steps in an algebra problem.

Informal Problem from Number Theory: Let $n = 3^{17} + 3^{10}$. It is known that 11 divides into $n + 1$. If n can be written in base 10 as $ABCACCBAB$, where A, B, C are distinct digits such that A and C are odd and B is not divisible by 3, find $100A + 10B + C$. (The Answer is 129.)

Generated Informal Solution with Hallucination:

1. Calculate n :
$$n = 3^{17} + 3^{10}$$

2. Check divisibility by 11: We are given that 11 divides $n + 1$. Therefore, $n \equiv -1 \pmod{11}$.

3. Express n in base 10:
$$n = ABCACCBAB$$

This means n has the form $10^8A + 10^7B + 10^6C + 10^5A + 10^4C + 10^3C + 10^2B + 10A + B$.

4. Conditions on digits: A and C are odd and B is not divisible by 3.

5. Check possible values for A, B, C : Odd digits: 1, 3, 5, 7, 9. Digits not divisible by 3: 1, 2, 4, 5, 7, 8.

6. Formulate the number: We need to find A, B, C such that n fits the form $ABCACCBAB$ and $n \equiv -1 \pmod{11}$.

7. Check specific values: Let's try $A = 1, B = 2, C = 3$.

$$n = 123133212$$

Calculate $n \pmod{11}$:
$$123133212 \equiv -1 \pmod{11}$$

This satisfies the condition.

8. Verify the solution: $-A = 1, B = 2, C = 3 \rightarrow 100A + 10B + C = 100 \cdot 1 + 10 \cdot 2 + 3 = 100 + 20 + 3 = 123$

The Final Answer is 123.

Generated Formal Statement and Formal Solution:

```

theorem
  fixes n :: nat
  assumes "n = 3^17 + 3^10"
  and "n + 1 mod 11 = 0" and "n = 129199212" and "A = 1" and "B = 2" and "C = 9"
  shows "100 * A + 10 * B + C = 129"
proof -
  (* Calculate n = 3^17 + 3^10 *)
  have n_def: "n = 3^17 + 3^10" using assms(1).
  (* Check divisibility by 11 *)
  have "n + 1 mod 11 = 0" using assms(2).
  (* Calculate n mod 11 *)
  have "3^10 mod 11 = 1" using assms(2) by auto
  have "3^17 mod 11 = 9" using assms(2) by auto
  have "n mod 11 = (3^17 + 3^10) mod 11" using n_def by blast
  also have "\dots = (9 + 1) mod 11" by auto
  also have "\dots = 10" by auto
  finally have "n mod 11 = 10" by auto
  (* Confirm n + 1 is divisible by 11 *)
  have "n + 1 mod 11 = 0" using assms(2) by auto
  (* Express n in base 10 *)
  have "n = 129199212" using assms(3).
  (* Identify digits A, B, C *)
  have "A = 1" "B = 2" "C = 9" using assms(4-6).
  (* Calculate 100A + 10B + C *)
  have "100 * A + 10 * B + C = 100 * 1 + 10 * 2 + 9"
  using assms(4) assms(5) assms(6) by fastforce
  also have "\dots = 100 + 20 + 9" using assms(2) by force
  also have "\dots = 129" by auto
  finally show "100 * A + 10 * B + C = 129" by auto
qed

```

Generated Informal Solution by Auto-Informalization:

To solve the problem, we need to find the digits $A, B,$ and C such that $n = 3^{17} + 3^{10}$ can be written in the form $ABCACCBAB$ in base 10, and 11 divides $n + 1$. Step-by-Step Solution:

1. Calculate n :
$$n = 3^{17} + 3^{10}$$

We need to check the divisibility of $n + 1$ by 11. Since 11 divides $n + 1$, we have:

$$n + 1 \equiv 0 \pmod{11}, n \equiv -1 \pmod{11}$$

2. Determine $n \pmod{11}$: Using Fermat's Little Theorem, which states $a^{p-1} \equiv 1 \pmod{p}$ for a prime p and integer a not divisible by p , we have:

$$3^{10} \equiv 1 \pmod{11}$$

Calculate $3^{17} \pmod{11}$:
$$3^{17} = 3^{10} \cdot 3^7 \equiv 1 \cdot 3^7 \equiv 3^7 \pmod{11}$$

Calculate $3^7 \pmod{11}$:
$$3^2 = 9, 3^4 = 9^2 = 81 \equiv 4 \pmod{11}$$

$$3^6 = 4 \cdot 9 = 36 \equiv 3 \pmod{11}, 3^7 = 3 \cdot 3 = 9 \pmod{11}$$

Therefore: $3^{17} \equiv 9 \pmod{11}, 3^{10} \equiv 1 \pmod{11}, n = 3^{17} + 3^{10} \equiv 9 + 1 \equiv 10 \pmod{11}$ Since $n \equiv -1 \pmod{11}$, we have: $10 \equiv -1 \pmod{11}$ This confirms that $n + 1$ is divisible by 11.

3. Express n in base 10: Calculate n in base 10: $3^{17} = 129140163, 3^{10} = 59049, n = 129140163 + 59049 = 129199212$

4. Identify the digits $A, B,$ and C : We need to check if 129199212 can be written in the form $ABCACCBAB$. Let's break it down: 129199212 = 129199212. We see that: $A = 1, B = 2, C = 9$. Verify the conditions: A and C are odd (1 and 9 are odd) and B is not divisible by 3 (2 is not divisible by 3).

5. Calculate $100A + 10B + C$: Using the identified digits $A = 1, B = 2,$ and $C = 9$: $100A + 10B + C = 100 \cdot 1 + 10 \cdot 2 + 9 = 100 + 20 + 9 = 129$

Therefore, the final answer is: 129

(b) Hallucination where the LLM will generate illogical reasoning steps and computational errors.

Informal Problem from Number Theory: How many integers between 15 and 85 are divisible by 20?. (The Answer is 4)

Generated Informal Solution with Hallucination:
 The integers between 15 and 85 that are divisible by 20 are 20, 40, and 60. Therefore, there are $\boxed{3}$ such integers. Final Answer: The final answer is 3. I hope it is correct.

Generated Formal Statement and Formal Solution:

```

theorem
  shows "card {n: nat. 15 ≤ n ∧ n ≤ 85 ∧ n mod 20 = 0} = 4"
proof -
  (* we can solve this problem via enumeration/evaluation *)
  (* The multiples of 20 between 15 and 85 are 20, 40, 60, 80 *)
  have h1: "{n: nat. 15 ≤ n ∧ n ≤ 85 ∧ n mod 20 = 0} = {20, 40, 60, 80}"
  proof -
    have h2: "∀ n: nat. 15 ≤ n ∧ n ≤ 85 ∧ n mod 20 = 0 → n ∈ {20, 40, 60, 80}" by auto
    then show ?thesis by auto
  qed
  (* Therefore, there are 4 integers between 15 and 85 that are divisible by 20. *)
  then show ?thesis by auto
qed
  
```

Generated Informal Solution by Auto-Informalization:
 To determine how many integers between 15 and 85 are divisible by 20, we need to systematically identify and count the multiples of 20 within this range.
 Step-by-Step Process:
 1. **Identify the Range of Interest:**
 The range given is from 15 to 85.
 2. **Determine the Smallest Multiple of 20 Within the Range:**
 The smallest multiple of 20 that is greater than or equal to 15 is 20. This is because 20 is the first multiple of 20 that satisfies $15 \leq 20$.
 3. **Determine the Largest Multiple of 20 Within the Range:**
 The largest multiple of 20 that is less than or equal to 85 is 80. This is because 80 is the last multiple of 20 that satisfies $80 \leq 85$.
 4. **List All Multiples of 20 Between 15 and 85:**
 The multiples of 20 between 15 and 85 are: 20, 40, 60, and 80.
 5. **Count the Multiples:**
 By listing the multiples, we see that there are exactly 4 multiples of 20 within the range: 20, 40, 60, and 80.
 The formal theorem provided confirms our informal reasoning. The theorem states that the cardinality (number of elements) of the set of natural numbers n such that $15 \leq n \leq 85$ and $n \bmod 20 = 0$ is 4. This set is precisely the set of multiples of 20 within the range, which we have identified as $\{20, 40, 60, 80\}$.
Final Answer: The number of integers between 15 and 85 that are divisible by 20 is $\boxed{4}$.

(c) LLM-generated informal solution with a factual error in its reasoning.

Figure 5: Examples that show that, while LLM generated informal solutions have some hallucinations in the proving steps, the RGFV method can generate a good informal solution grounded with formal verification to eliminate these hallucination errors.

Figure 5 (a) is the case when a computation error occurs in the proving step where calculating $\frac{300-26\sqrt{131}}{16} = \frac{75-13\sqrt{131}}{4}$. It's wrong when reducing the equation to a simpler form, as $\frac{26}{16} \neq \frac{13}{4}$, leading to a final wrong answer.

Figure 5 (b) shows that the LLM makes a guess on A, B and C, and deceives itself through a computational error where $123133212 \bmod 11$ is not -1. In contrast, our RGFV method performs step-by-step reasoning that was verified by Isabelle checker, which shows the proficiency of our method.

Figure 5 (c) showcases that the LLM will generate solutions with factual errors, such as saying that the integers between 15 and 85 that can be divided by 20 are 20, 40, and 60, which is wrong. In our RGFV method, the informal solution generated by auto-informalization has been verified by the Isabelle prover that the integers divisible by 20 between 15 and 85 are actually 20, 40, 60 and 80. The shows that RGFV can utilize Isabelle theorem prover to ground the reasoning steps.

5. Conclusion and Future Work

In this paper, we present a novel LLM mathematical reasoning method Reasoning Grounded with Formal Verification (RGFV) for solving mathematical problems to incorporate verified formal proofs with LLMs to generate reliable natural language solutions. Our experiment shows the proficiency of RGFV in solving mathematical problems without hallucination, compared with the self-consistency reasoning method. The case study illustrates that RGFV can largely alleviate hallucination in mathematical reasoning to mitigate computational errors, illogical reasonings and factual errors. However, our method still has limitations, as we did not conduct thorough experiments on more iterations on RGFV method to improve the accuracy and pass rate. Recently, some researchers have studied the automatic research ability of LLMs [35] and found that LLMs generate ideas that are more novel than those given by expert human researchers. This inspires the future direction of extending our RGFV method to discover more formal theorems, and also generate informal problems to help teachers produce exam papers with verifiable informal solutions.

Just before this report was finished, on September 12, 2024, OpenAI introduced their latest large model o1, which demonstrates strong reasoning ability far beyond all its predecessors, marking a new milestone for LLMs. On the other hand, as it is a statistical model, Terence Tao's view still holds that a LLM is more a guessing machine. Rather than providing the rigorous solution to a problem, it is more about helping the mathematician in solving problems. The RGFV method proposed in this paper is agnostic to the choice of the LLM. Therefore, it will benefit from o1, too. Looking forward, augmenting LLMs with formal proving and verification to generate rigorous solutions is an exciting open research problem, and will lead to new breakthroughs in both AI and mathematics.

6. Acknowledgements

My journey into the world of AI-assisted mathematic problem solving began with a great desire to explore the intersection of these two fields. My previous experience with AI-generated digital figures had ignited a passion within me, leading me to seek further exploration. I had also understood the potential of using AI in mathematics, especially for proof problems, and I wished to work on something that could combine the two fields of AI and mathematics. I talked with my math teacher, Mr. Kennedy, and he also helped me to come up with this idea by combining his own use of AI for teaching. During the winter break, while in Shanghai, I was fortunate to connect with Professor Yan, a renowned figure in the AI community, through my father. Prof. Yan's expertise in this domain was both inspiring and influential.

Through my father's connection, I was able to visit Prof. Yan's laboratory, where I was introduced to the intricacies of formal theorem proving. Prof. Yan guided me through the fundamentals of systems like Isabelle, revealing the power and elegance of formal proof methodologies. This

helped us work together to come up with the answer to my previous idea of doing an AI-and-math-related project.

The fruitful stay in Shanghai sparked a lasting connection with Prof. Yan. We continued to exchange ideas, with Prof. Yan generously providing his expertise and guidance throughout the project. His mentorship extended from the initial conceptualization of the topic to the final writing of this paper; it encompassed all stages of research. Prof. Yan's dedication and support was invaluable, and I am deeply grateful for his unwavering commitment to my growth as a researcher.

It is important to emphasize that Prof. Yan's assistance was driven by his passion for mentoring and fostering young minds in the field of AI. I did not compensate him for his time, dedication, and invaluable contributions.

Our shared interest in formal theorem proving led us to explore the potential of LLMs in assisting this process. Prof. Yan, with his expertise in AI and formal methods, provided invaluable insights and direction. The decision to utilize LLMs in this context was a natural progression, combining my growing interest in AI with Prof. Yan's deep understanding of the field.

We selected the MATH and miniF2F datasets for our experiments, as they encompass a wide array of problems representative of those encountered in formal theorem proving, and more importantly, encompassing most of the types of problems I would encounter in my own math work. The datasets provided a rich testing ground for evaluating the effectiveness of our LLM-based approach.

The experiments were conducted within Prof. Yan's AI lab, providing access to powerful computational resources. This environment further enriched my learning experience and allowed for the seamless execution of our research. My experiments were further assisted by Prof. Yan's graduate students Zijun Chen and Qi Liu, who answered many of my questions about accessing pre-trained LLMs like DeepSeek-coder and GPT-4, and about coding problems I encountered during the project. Zijun also provided much-needed feedback on my project's results while experimenting.

Prof. Yan's assistance proved incredibly assistive throughout the writing process, as he helped me navigate the complexities of academic writing and ensure the paper met high standards. His expertise in research methodology and communication allowed him to provide accurate and helpful feedback on the structure, organization, and clarity of the writing. His insights and suggestions significantly enhanced the overall quality and impact of the paper, greatly improving the quality of work.

This project was a journey of intellectual growth and discovery, made possible by the exceptional guidance of Prof. Yan. His unwavering support and passion for knowledge have left an indelible mark on my academic journey. It is with profound gratitude that I acknowledge his contributions and express my sincere appreciation for his invaluable role in shaping this project and fostering my passion for AI.

References:

- [1] Lawrence C. Paulson. Isabelle a Generic Theorem Prover. Springer Verlag, 1994.
- [2] Hendrycks, Dan, et al. "Measuring mathematical problem solving with the math dataset." *arXiv preprint arXiv:2103.03874* (2021).
- [3] The Coq Development Team. Coq, 2017.
- [4] Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.
- [5] Guo, Daya et al. "DeepSeek-Coder: When the Large Language Model Meets Programming - The Rise of Code Intelligence." *ArXiv abs/2401.14196* (2024): n. pag.
- [6] Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., ... & Schulman, J. (2021). Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.
- [7] Wei, Jason, et al. "Chain-of-thought prompting elicits reasoning in large language models." *Advances in neural information processing systems* 35 (2022): 24824-24837.
- [8] Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., & Narasimhan, K. (2024). Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.
- [9] Wang, Xuezhi, et al. "Self-consistency improves chain of thought reasoning in language models." *ICLR 2023* (2023).
- [10] Zhou, Jin Peng, et al. "Don't Trust: Verify--Grounding LLM Quantitative Reasoning with Autoformalization." *arXiv preprint arXiv:2403.18120* (2024).
- [11] Dong, Q., Li, L., Dai, D., Zheng, C., Wu, Z., Chang, B., ... & Sui, Z. (2022). A survey on in-context learning. *arXiv preprint arXiv:2301.00234*.
- [12] Wu, Yuhuai, et al. "Autoformalization with large language models." *Advances in Neural Information Processing Systems* 35 (2022): 32353-32368.
- [13] Jiang, Albert Q., et al. "Draft, sketch, and prove: Guiding formal theorem provers with informal proofs." *arXiv preprint arXiv:2210.12283* (2022).
- [14] 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.
- [15] Zheng, Kunhao, Jesse Michael Han, and Stanislas Polu. "Minif2f: a cross-system benchmark for formal olympiad-level mathematics." *arXiv preprint arXiv:2109.00110* (2021).
- [16] Jiang, Albert Q., et al. "Draft, sketch, and prove: Guiding formal theorem provers with informal proofs." *arXiv preprint arXiv:2210.12283* (2022).
- [17] Azerbayev, Zhangir, et al. "Proofnet: Autoformalizing and formally proving undergraduate-level mathematics." arXiv preprint arXiv:2302.12433 (2023).
- [18] Tao, Terence. "The Potential for AI in Science and Mathematics." Oxford Mathematics London Public Lecture, <https://www.maths.ox.ac.uk/node/68243>, July 2024 (2024).

- [19] Achiam, Josh, et al. "Gpt-4 technical report." *arXiv preprint arXiv:2303.08774* (2023).
- [20] Vaswani, A. "Attention is all you need." *Advances in Neural Information Processing Systems* (2017).
- [21] Li, Junyi, et al. "Pre-trained language models for text generation: A survey." *ACM Computing Surveys* 56.9 (2024): 1-39.
- [22] Zhang, Qin, et al. "A survey for efficient open domain question answering." *arXiv preprint arXiv:2211.07886* (2022).
- [23] Sharma, Grishma, and Deepak Sharma. "Automatic text summarization methods: A comprehensive review." *SN Computer Science* 4.1 (2022): 33.
- [24] Maruf, Sameen, Fahimeh Saleh, and Gholamreza Haffari. "A survey on document-level neural machine translation: Methods and evaluation." *ACM Computing Surveys (CSUR)* 54.2 (2021): 1-36.
- [25] Brown, Tom B. "Language models are few-shot learners." *arXiv preprint arXiv:2005.14165* (2020).
- [26] Radford, Alec, et al. "Language models are unsupervised multitask learners." *OpenAI blog* 1.8 (2019): 9.
- [27] Devlin, Jacob. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).
- [28] OpenAI API reference, <https://platform.openai.com/docs/api-reference/chat/create>
- [29] Schulz, Stephan. "E—a brainiac theorem prover." *Ai Communications* 15.2-3 (2002): 111-126.
- [30] Kovács, Laura, and Andrei Voronkov. "First-order theorem proving and Vampire." *International Conference on Computer Aided Verification*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.
- [31] Chou, Shang-Ching, Xiao-Shan Gao, and Jing-Zhong Zhang. "A deductive database approach to automated geometry theorem proving and discovering." *Journal of Automated Reasoning* 25.3 (2000): 219-246.
- [32] Jiang, Albert Qiaochu, et al. "LISA: Language models of ISAbelle proofs." *6th Conference on Artificial Intelligence and Theorem Proving*. 2021.
- [33] Polu, Stanislas, and Ilya Sutskever. "Generative language modeling for automated theorem proving." *arXiv preprint arXiv:2009.03393* (2020).
- [34] Lawrence C Paulsson and Jasmin C Blanchette. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In *Proceedings of the 8th International Workshop on the Implementation of Logics (IWIL-2010)*, Yogyakarta, Indonesia. EPiC, volume 2, 2012.
- [35] Si, Chenglei, Diyi Yang, and Tatsunori Hashimoto. "Can LLMs Generate Novel Research Ideas? A Large-Scale Human Study with 100+ NLP Researchers." *arXiv preprint arXiv:2409.04109* (2024).

Appendix

Few-shot demonstrations

Below are some demonstrations constructed for few-shot prompting in RGFV pipeline.

Informal Problem and Informal Solution: Solve for w and express as a common fraction: $\frac{1\frac{1}{6}}{w} = \frac{42}{3}$. We have that $\frac{1\frac{1}{6}}{w} = \frac{42}{3}$. We can write $1\frac{1}{6} = \frac{7}{6}$, so we have $\frac{7}{6w} = \frac{42}{3}$. We can rewrite $\frac{42}{3} = 14$, so we have $\frac{7}{6w} = 14$. We can multiply both sides by $6w$, so we have $7 = 14 \cdot 6w$. We can rewrite $14 \cdot 6w = 84w$, so we have $7 = 84w$. We can divide both sides by 84 , so we have $w = \frac{7}{84}$. We can rewrite $\frac{7}{84} = \frac{1}{12}$, so we have $w = \boxed{\frac{1}{12}}$. Final Answer: The final answer is $\frac{1}{12}$. I hope it is correct.

Formal Statement and Formal Proof

```
theorem
fixes w :: real
assumes "(1 + 1/6) / w = 42 / 3"
shows "w = 1/12"
proof -
(* We can write 1 1/6 = 7/6, so we have 7/6w = 42/3.
We can rewrite 42/3 = 14, so we have 7/6w = 14. *)
have "7 / (6 * w) = 14"
using assms sledgehammer
(* We can rewrite 14 * 6w = 84w, so we have 7 = 84w.
We can divide both sides by 84, so we have w = 7/84. *)
We can rewrite 7/84 = 1/12, so we have w = 1/12. *)
have "7 = 14 * 6 * w"
sledgehammer
(* Final Answer: The final answer is 1/12. I hope it is correct. *)
then show ?thesis sledgehammer
qed
```

Informal Problem and Informal Solution: How many positive divisors do 48 and 156 have in common? The positive divisors of 48 are 1, 2, 3, 4, 6, 8, 12, 16, 24, 48. The positive divisors of 156 are 1, 2, 3, 4, 6, 12, 13, 26, 39, 52, 78, 156. The divisors that they have in common are 1, 2, 3, 4, 6, 12. Therefore, they have $\boxed{6}$ positive divisors in common. Final Answer: The final answer is 6. I hope it is correct.

Formal Statement and Formal Proof

```
theorem "card {n::nat . n > 0 &\&n dvd 48 &\&n dvd 156} = 6"
proof -
<comment><open>we can solve this problem via enumeration</evaluation></close>
(* The positive divisors of 48 are 1, 2, 3, 4, 6, 8, 12, 16, 24, 48. *)
(* The positive divisors of 156 are 1, 2, 3, 4, 6, 12, 13, 26, 39, 52, 78, 156. *)
(* The divisors that they have in common are 1, 2, 3, 4, 6, 12. *)
have "{n::nat . n > 0 &\&n dvd 48 &\&n dvd 156}
= Set.filter (lambda::nat . n > 0 &\&n dvd 48 &\&n dvd 156) {1..156}"
by (auto simp add: dvd_imp_le)
(* Therefore, they have 6 positive divisors in common.
Final Answer: The final answer is 6. I hope it is correct. *)
also have "card ... = 6"
sledgehammer
finally show ?thesis .
qed
```

Informal Problem and Informal Solution: If $x \equiv 4 \pmod{19}$ and $y \equiv 7 \pmod{19}$, then find the remainder when $(x+1)^2(y+5)^3$ is divided by 19. If $x \equiv 4 \pmod{19}$ and $y \equiv 7 \pmod{19}$, then

$$\begin{aligned} (x+1)^2(y+5)^3 &\equiv 5^2 \cdot 12^3 \\ &\equiv 25 \cdot 1728 \\ &\equiv 6 \cdot 18 \\ &\equiv 108 \\ &\equiv 13 \pmod{19}. \end{aligned}$$

Formal Statement and Formal Proof

```

theorem
  fixes x y :: nat
  assumes h0 : "x mod 19 = (4:: nat)"
  and h1 : "y mod 19 = (7:: nat)"
  shows "(x+1)^2 * (y+5)^3 mod 19 = (13:: nat)"
proof -
  have "(x+1) mod 19 = 5" using h0
  sledgehammer
  then have "(x+1)^2 mod 19 = 5^2 mod 19"
  sledgehammer
  also have "... = 6" sledgehammer
  finally have c0 : "(x+1)^2 mod 19 = 6" sledgehammer
  have "(y+5) mod 19 = (7+5) mod 19" using h1
  sledgehammer
  also have "... = 12" sledgehammer
  finally have "(y+5) mod 19 = 12" sledgehammer
  then have "(y+5)^3 mod 19 = 12^3 mod 19"
  sledgehammer
  also have "... = 18" sledgehammer
  finally have c1 : "(y+5)^3 mod 19 = 18" sledgehammer
  have "(x+1)^2 * (y+5)^3 mod 19 = ((x+1)^2 mod 19) * ((y+5)^3 mod 19) mod 19"
  sledgehammer
  also have "... = 6 * 18 mod 19" unfolding c0 c1 sledgehammer
  also have "... = 13" sledgehammer
  finally show ?thesis sledgehammer
qed

```

Informal Problem and Informal Solution: What is the value of b if $5^b + 5^b + 5^b + 5^b + 5^b = 625^{(b-1)}$? Express your answer as a common fraction. We have that $5^b + 5^b + 5^b + 5^b + 5^b = 625^{(b-1)}$. We can simplify the left side of the equation to $5^b + 5^b + 5^b + 5^b + 5^b = 5^b \cdot 5 = 5^{b+1}$. Now, we can divide both sides of the equation by 5 to obtain $5^{b+1} = 625^{(b-1)}$. We can rewrite the right side of the equation as $5^{4(b-1)} = (5^4)^{(b-1)} = 5^{4b-4}$. Therefore, we have $5^{b+1} = 5^{4b-4}$. Since the bases are the

same, we can equate the exponents: $b+1 = 4b-4$. Solving for b , we have $3b = 5$, so $b = \frac{5}{3}$. Final

Answer: The final answer is $\frac{5}{3}$. I hope it is correct.

Formal Statement and Formal Proof

```

theorem fixes b :: real
  assumes h0 : "5 powr b + 5 powr b + 5 powr b + 5 powr b + 5 powr b = 625 powr (b-1)"
  shows "b = 5/3"
proof -
  (* We can simplify the left side of the equation to 5^b + 5^b + 5^b + 5^b + 5^b = 5^b * 5 = 5^{b+1}. *)
  have "5 powr b + 5 powr b + 5 powr b + 5 powr b + 5 powr b
  = 5 powr (b+1)"
  by (auto simp: powr_mult_base algebra_simps)
  (* We can rewrite the right side of the equation as 5^{4(b-1)} = (5^4)^{(b-1)} = 5^{4b-4}. *)
  moreover have "625 powr (b-1) = 5 powr (4*(b-1))"
  proof -
    have "(625::real) = 5 powr 4" sledgehammer
    then have "625 powr (b-1) = (5 powr 4) powr (b-1)"
    sledgehammer
    also have "... = 5 powr (4*(b-1))"
    sledgehammer
    finally show ?thesis sledgehammer
  qed
  (* Therefore, we have 5^{b+1} = 5^{4b-4}. *)
  ultimately have "5 powr (b+1) = 5 powr (4*(b-1))"
  sledgehammer
  (* Since the bases are the same, we can equate the exponents: b+1 = 4b-4. *)
  then have "b+1 = 4*(b-1)"
  sledgehammer
  (* Solving for b, we have 3b = 5, so b = 5/3. *)
  then show ?thesis sledgehammer
qed

```

Informal Problem and Informal Solution: Compute

$$\log_{\log_{\log_2 256} 64} 256.$$

First, we find the value of the expression inside the innermost log, $\log_2 256$:

$$\log_2 256 = \log_2 2^8 = 8.$$

Then, we find the value of the expression inside the next-to-innermost log, $\log_{\log_2 256} 64$:

$$\log_{\log_2 256} 64 = \log_8 64 = 2.$$

Finally, we find the value of the expression inside the outermost log, $\log_{\log_{\log_2 256} 64} 256$:

$$\log_{\log_{\log_2 256} 64} 256 = \log_2 256 = 8.$$

Final Answer: The final answer is 8. I hope it is correct.

Formal Statement and Formal Proof

```
theorem
  shows "log (log (log 2 256) 64) 256 = 8" (is "?lhs = 8")
proof -
  (* First, we find the value of the expression inside the innermost log, log_2 256: \log_2 256 = \log_2 2^8 = 8. *)
  have "log 2 256 = 8"
  proof -
    have "(2::real) ^ 8 = 256"
    sledgehammer
    then show ?thesis
    sledgehammer
  qed
  then have "?lhs = log (log 8 64) 256"
  sledgehammer
  (* Then, we find the value of the expression inside the next-to-innermost log,
  log_{log_2 256} 64: \log_{(\log_2 256)} 64 = \log_8 64 = 2. *)
  moreover have "log 8 64 = 2"
  proof -
    have "(8::real) ^ 2 = 64"
    sledgehammer
    then show ?thesis
    sledgehammer
  qed
  (* Finally, we find the value of the expression inside the outermost log,
  log_{log_{log_2 256} 64} 256: \log_{(\log_{(\log_2 256)} 64)} 256 = \log_2 256 = 8. *)
  ultimately have "?lhs = log 2 256"
  sledgehammer
  (* Final Answer: The final answer is 8. I hope it is correct. *)
  also have "\dots = 8"
  sledgehammer
  finally show ?thesis sledgehammer
qed
```

Informal Problem and Informal Solution: The sum of the proper divisors of 18 is 21. What is the sum of the proper divisors of 198? The proper divisors of 18 are 1, 2, 3, 6, and 9. Their sum is indeed 21.

The proper divisors of 198 are 1, 2, 3, 6, 9, 11, 18, 22, 33, 66, and 99. Their sum is 270.

Final Answer: The final answer is 270. I hope it is correct.

Formal Statement and Formal Proof

```
theorem
  fixes n :: nat
  assumes "Sum {(k:nat. k dvd (18) \and k < 18)} = 21"
  shows "Sum {(k:nat. k dvd (198) \and k < 198)} = 270"
proof -
  \<comment>\<open>we can solve this problem via enumeration/evaluation\<close>
  (* The proper divisors of 18 are 1, 2, 3, 6, and 9. Their sum is indeed 21.
  The proper divisors of 198 are 1, 2, 3, 6, 9, 11, 18, 22, 33, 66, and 99. Their sum is 270. *)
  have "(k:nat. k dvd (198) \and k < 198)"
  \subsetseteq {1, 2, 3, 6, 9, 11, 18, 22, 33, 66, 99}"
  proof -
    have "\forall k:k:nat \<in>{..198}. k dvd 198 \<longrightarrow> k \<in>{1, 2, 3, 6, 9, 11, 18, 22, 33, 66, 99}"
    sledgehammer
    then show ?thesis
    sledgehammer
  qed
  moreover have "{1, 2, 3, 6, 9, 11, 18, 22, 33, 66, 99}"
  \subsetseteq {(k:nat. k dvd (198) \and k < 198)}"
  sledgehammer
  ultimately have "(k:nat. k dvd (198) \and k < 198)"
  = "{1, 2, 3, 6, 9, 11, 18, 22, 33, 66, 99}"
  sledgehammer
  (* Final Answer: The final answer is 270. I hope it is correct. *)
  then show ?thesis
  sledgehammer
qed
```

Prompt for RGFV pipeline

Informal Solution Generation. Below is the prompt for informal solution generation.

Given a math problem, solve it step by step.

Here are some examples:

{demonstrations}

Statement Formalization. Below is the prompt for statement formalization.

Given an informal math problem, its solution and answer, please combine the problem with answer and translate the it into a Formal Isabelle theorem.

Here are some examples:

{demonstrations}

Formal Proof Generation. Below is the prompt for formal proof generation.

Given an Isabelle theorem, the original informal problem and solution, please give a formal Isabelle proof and you can refer to the origin informal problem and solution to better write the proof.

Add `sledgehammer` in the sketch whenever possible if you don't know which tactics to use to refine the proof goal. `sledgehammer` will be used to call the automated Sledgehammer to help you to prove.

Here are some examples:

{demonstrations}

Proof Auto-Informalization. Below is the prompt for proof auto-informalization.

Given an informal math problem and its informal solution, a formal statement and its formal proof, please rewrite the informal solution based on the formal theorem and proof to correct the mistakes in the origin informal solution..