LeanTutor: Towards a Verified AI Mathematical Proof Tutor

Manooshree Patel, Rayna Bhattacharyya *, Thomas Lu *, Arnav Mehta *, Niels Voss *, Narges Norouzi, Gireeja Ranade

University of California, Berkeley

{manooshreepatel@, rayna_b@, thomaslu@, arnavmehta@, niels_voss@, norouzi@, ranade@eecs.}berkeley.edu

Abstract

This paper considers the development of an AI-based provably-correct mathematical proof tutor. While Large Language Models (LLMs) allow seamless communication in natural language, they are error prone. Theorem provers such as Lean allow for provable-correctness, but these are hard for students to learn. We present a proof-of-concept system (LeanTutor) by combining the complementary strengths of LLMs and theorem provers. LeanTutor is composed of three modules: (i) an autoformalizer/proof-checker, (ii) a next-step generator, and (iii) a natural language feedback generator. To evaluate the system, we introduce PeanoBench, a dataset of 371 Peano Arithmetic proofs in human-written natural language and formal language, derived from the Natural Numbers Game.

1 Introduction

College students use LLMs such as ChatGPT and Claude to start projects, create practice questions, and generate solutions to academic assignments (OpenAI 2025; Anthropic 2025). However, indiscriminate LLM usage can be detrimental to student learning (Goetze 2025), because these systems are not designed from a pedagogical perspective. Specifically, (1) most models are designed to be maximally "helpful" to a user (Askell et al. 2021), and often directly give away the answer, instead of helping a student come up with it on their own (Sonkar et al. 2024), (2) even state-of-the-art models are prone to hallucinations and generate convincing wrong answers (Balunović et al. 2025; Maurya et al. 2024; Gupta et al. 2025), (3) models struggle to identify mistakes in reasoning (Tyen et al. 2024; Miller and DiCerbo 2024), and (4) even if models can produce the correct answer, they cannot necessarily produce correct reasoning to guide the student (Gupta et al. 2025). As a result, faculty are increasingly concerned about the negative impacts of LLMs on student learning, particularly in courses where "critical thinking" is one of the key learning goals, for example mathematical proof courses. The recent CRA Practioner-to-Professor survey reinforced the importance of math for developing critical thinking (Simha and Wright 2025).

There is a desire among educators to find a happy-medium solution — where a student can get the benefits of the instantaneous, private feedback that LLMs offer, while mitigating the negative effects of LLMs. Math educators have been developing intelligent tutoring systems (ITS) (e.g. (Bundy, Moore, and Zinn 2000; Lodder et al. 2021; Barnes and Stamper 2008)) and theorem prover-based tutors (e.g. (Sieg 2007; Wemmenhove et al. 2022)) to help students do math proofs for decades.

Unlike LLMs these systems are definitively correct, but they can be tedious to create (Dermeval et al. 2018). Others require an understanding of challenging formal language syntax (Thoma and Iannone 2022), or constrict student writing by only accepting input in application-specific controlled natural language (Wemmenhove et al. 2022). As a result of these challenges, we do not see widespread adoption of such tutoring systems (unlike autograders for programming (DeNero and Martinis 2014; Hecht et al. 2023; Mitra 2023)).

We propose LeanTutor, a math-proof tutoring system that combines the strengths of LLMs and theorem provers. This system interacts with students in natural language (NL), while using the Lean theorem prover (Moura and Ullrich 2021) to evaluate proof correctness and generate correct next steps on the backend. Specifically, the system can:

- Accept complete/partial/correct/incorrect student proofs
- Verify if the student work is correct or incorrect
- Identify the student error, if applicable, and provide guidance towards a correct proof, without giving away the complete answer.

This paper presents a first design of LeanTutor, working off of a small self-contained dataset, as in (Murphy et al. 2024; Cunningham, Bunescu, and Juedes 2023). This system requires a seamless back-and-forth between the LLM and theorem prover by means of faithful auto-formalization and auto-informalization. However, compared to the standard approach in AI for Math work, the tutoring setting offers a new frame for the problems of auto-formalization and next-step generation, as below.

- Autoformalization for tutoring systems must focus on *faithful autoformalization* of natural language statements (i.e. preserving the semantic meaning as in (Murphy et al. 2024)). Fundamental differences in how informal and formal math is written make this difficult. First, formal proofs are much more fine-grained than informal proofs — proof steps that would be considered rigorous by humans

[*]These authors contributed equally.

Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

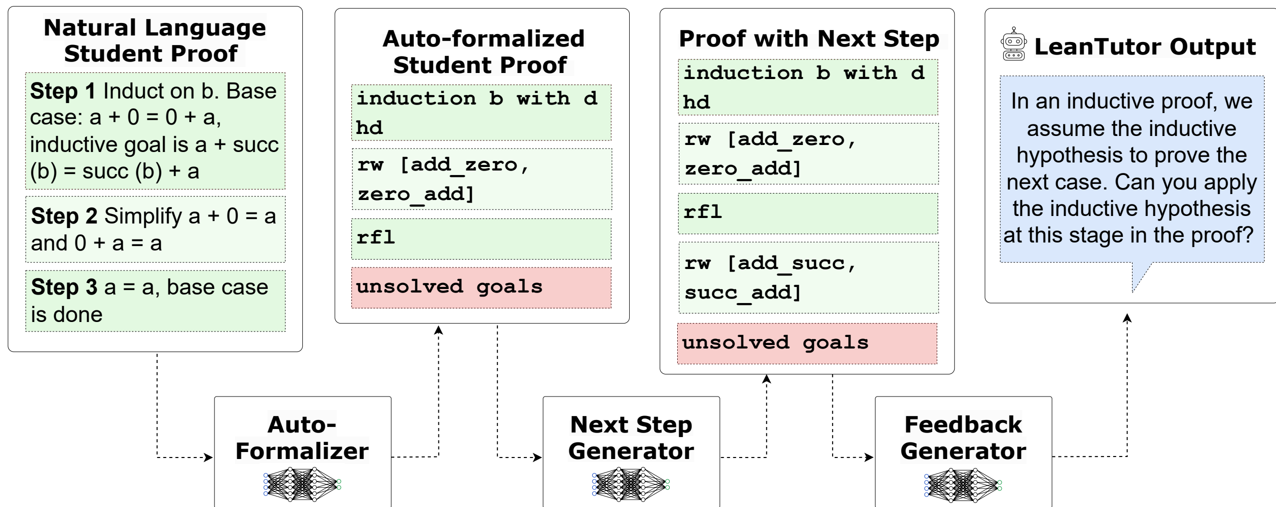

Figure 1: LeanTutor is comprised of three modules: an autoformalizer that automatically formalizes an NL student proof into Lean step-by-step; a next step generator that generates a next feasible tactic for the student proof; and a natural language feedback generator that generates guiding feedback to help the student progress towards a correct proof.

(e.g. arguments that are true by symmetry) can require long-winded proofs in formal languages. Second, informal proofs commonly use forward-reasoning where are formal proofs can be often constructed using backwards-reasoning (Shi et al. 2025). This difference of style can create significant challenges.

- Evaluating faithful autoformalization is particularly challenging, as syntactic correctness (i.e. successful compilation) is insufficient (Liu et al. 2025).
- Autoformalization for a tutoring system must be able to handle not only complete and correct but incomplete and incorrect proofs. Student natural language further have lots of variation and may not have the polish of professional mathematical writing. Previous work on autoformalization has focused on theorem statements and correct whole proofs (Yang et al. 2024).
- The tutoring application may assume knowledge of a complete proof of the theorem under consideration, creating an easier version of the faithful autoformalization problem. Students are working on problems where the instructor knows a solution, unlike mathematicians trying to solve novel problems. We see that this improves autoformalization performance, and leaves open future research problems on how to fully exploit a reference solution.
- Similarly, this reference proof means that the key challenge in next-step-generation is to identify the specific proof approach taken by the student, which may or may not match that of the reference solution. However, the tutoring application can limit the search for next steps to relatively small theorem libraries, compared to novel proof search using huge libraries such as Mathlib (mathlib Community 2020).

LeanTutor simplifies some of these challenges through a carefully aligned informal-formal math dataset, discussed in Section 3, and by proposing a new metric to measure autoformalization accuracy (explained in 5.1), but by no means solves them.

The challenges in developing AI-math-tutors are very similar to those in developing general AI-mathematics-assistants (Riehl 2025) and remain open problems to tackle.

2 Related Work

2.1 Autoformalization via Language Models

A large body of recent work has focused on autoformalizing natural language (NL) theorem statements into formal language (FL), using deep learning methods (Ying et al. 2024; Gao et al. 2024; Wu et al. 2022; Azerbayev et al. 2023a; Lin et al. 2025; Shao et al. 2024). The more difficult task of autoformalizing whole proofs from NL to FL has been explored in fewer works (Jiang et al. 2022; Murphy et al. 2024; Wang et al. 2024a; Huang et al. 2024). State-of-the-art (SOTA) LLMs, without any specific formal language training, have shown strong performance on the task of autoformalization (Wu et al. 2022) and motivate our development of an LLM-agnostic framework for autoformalization.

Both tutoring (as in LeanTutor) and auto-grading require *faithful autoformalization* Murphy et al. (2024). In this paper, we take a similar approach to Kulal et al. (2019) method of translating pseudocode to code, line-by-line, in a C++ program generation task. We make the reasonable assumptions for the classroom setting that all proofs come from a small dataset and at least one valid proof per theorem is known (in both NL and FL). Murphy et al. (2024); Cunningham, Bunescu, and Juedes (2023) successfully formalize proofs in a small dataset where all feasible theorems/tactics are known.

2.2 Neural Theorem Proving

Neural theorem proving reframes theorem proving as a language modeling task (Li et al. 2024a). An abundance of prior

work has made progress towards training language models for theorem proving (Wang et al. 2024a; Welleck et al. 2022; Azerbayev et al. 2023b; Lin et al. 2025; Yang et al. 2023). Additionally, prior work has explored theorem proving frameworks with SOTA LLMs (Jiang et al. 2022; Huang et al. 2024; Thakur et al. 2023). Our next-step generation approach is largely inspired by the COPRA agent (Thakur et al. 2023). The COPRA agent performs a GPT-4 directed depth-first search over sequences of possible tactics, to complete a formal proof. The agent additionally implements a "progress check", which assesses if generated tactics progress the proof.

2.3 Automated Feedback Generation for Programming Assignments

We draw inspiration for LeanTutor's feedback generation module from automated feedback generation in programming classes (D'antoni et al. 2015; Singh, Gulwani, and Solar-Lezama 2013; Suzuki et al. 2017). Since students write their code in a programming environment where compilers enforce formal correctness and autograders ensure that the code passes test cases or return appropriate errors, autonomous tutors can leverage the resulting error messages and metadata to generate high-quality feedback. We build on the five hint types identified by Suzuki et al. (2017) (transformation, location, data, behavior, and example) that can be generated via program synthesis to provide students feedback in an introductory coding class. Similarly, theorem provers provide proof state information and specific error messages, which can be used to pinpoint student errors.

Autoinformalization, translating formal statements into informal ones (Li et al. 2024a), is a parallel task to feedback generation. LLM-based autoinformalization has been explored with success (Wu et al. 2022; Huang et al. 2024).

2.4 Math Proof Tutors

We identify three categories of existing math proof tutors—intelligent tutoring systems, LLM-based tutors, and theorem prover-based tutors. Researchers have made attempts to develop (Autexier, Dietrich, and Schiller 2012; Briggle et al. 2008) or developed intelligent tutoring systems (ITS) for math proofs (Barnes and Stamper 2008; Lodder et al. 2021; Bundy, Moore, and Zinn 2000). ITS require expert authoring of solutions or feedback, making them difficult to develop and scale (Dermeval et al. 2018). LLM-based math tutors have demonstrated benefits such as learning gains (Pardos and Bhandari 2023) and can maintain conversations with no harmful content (Levonian et al. 2025). However, these LLMs fail as tutors, for the reasons outlined in Section 1. Math educators have used theorem provers, such as Lean, to teach proofs (Avigad 2019). These tools have led to unique benefits in students' learning of proofs (Thoma and Iannone 2022), but students struggle to learn the complex syntax required to interact with most (Avigad 2019; Buzzard 2022; Karsten et al. 2023).

3 PeanoBench Dataset

To develop LeanTutor, we construct the PeanoBench dataset, which contains a total of 371 proofs. Each proof has a human-written natural language proof and a semantically equivalent formal language proof in Lean. PeanoBench is derived from the original 80 Peano Arithmetic (PA) proofs in the Natural Number Game 4 (NNG4) (Buzzard et al. 2023) (Apache-2.0 license). We chose these because the formal proofs were already available, making evaluation easier, and the informal versions of PA proofs naturally maintain some similarity in flow and granularity to their formal versions. To create the dataset, we began with a subset of 75 of the original NNG4 proofs[1]) and back-translated, following similar approaches for low-resource languages (Ranathunga et al. 2023; Ying et al. 2024; Wang et al. 2024b; Lu et al. 2024). All NL annotations were written by the first five paper authors.

Proof annotators followed two rules while annotating. (1) Natural language annotations are free of Lean-specific syntax, premises, or tactics. (2) Annotations are written to function as standalone proofs independent of the Lean code. The **one-to-one correspondence between NL proof steps and individual FL tactics** differentiates PeanoBench from prior datasets for Lean autoformalization that pair whole Lean proofs with their whole NL counterpart (Lu et al. 2024; Wang et al. 2024a; Gao et al. 2024).

Correct proofs Correct proofs in PeanoBench are part of one of three groups — (1) the *staff-solutions*, the *equation-based* proofs and the (2) *justification-based* proofs.

The *staff-solution* proofs are derived directly from NNG4 and annotated by two paper authors (before LeanTutor's architecture was designed), and the annotations are very descriptive. The two-other groups of proofs are generated according to "personas" (Cooper 1999), a technique used in user-interface design as well as for synthetic data generation (Ge et al. 2024). The NL for the *equation-based* persona consists primarily of algebraic manipulations and few words. The NL proof in the *justification-based* persona explicitly contains the theorems and mathematical definitions used in the proof. Each proof was annotated by one of five (paper author) annotators and proofread by a different annotator. When possible, we varied the proof's Lean code (whether this be a major logical difference or rearranging commutative tactics) in addition to changing the NL comments according to the persona. Thus, we have a total of 225 correct proofs.

Incorrect Proofs We additionally generate incorrect proofs by mimicking logical errors. Logical errors are a common mistake made by students, where they believe they have a valid proof, but it is in fact invalid (Weber 2001). A common example for such a logical error is to assume a statement $\mathcal{P} \implies \mathcal{Q}$, but forgetting that there are steps required to justify this. We imitate this mistake and create incorrect proofs by randomly skipping a step from the last three lines of the proof.

Proofs that are only one line long are removed from the incorrect set. Two limitations of this approach are that 1) this does not capture all possible error-types made by students, and 2) this programmatic deletion approach can sometimes lead to unrealistic errors. In total, we end with 73×2 incorrect

[1]We removed the attempted proof of Fermat's Last Theorem and proofs which contain the `simp` tactic, as the applicability of the `simp` tactic is broad and can sometimes easily close complex goal

proofs in the *equation-based* and *justification-based* personas. We do not generate incorrect proofs using the *staff-solution*.

Staff solutions proofs are only offered as context to the model. System performance is evaluated on the correct and incorrect *equation-based* and *justification-based* proofs.

4 System Design

LeanTutor has three modules: an autoformalizer, a next-step generator, and an automatic feedback generator (Figure 1).

4.1 Autoformalizer and Proof Checker

We autoformalize student NL proofs one step at a time, and check for compliation at each step. This approach is similar in spirit to previous works breaking autoformalization into subtasks (Jiang et al. 2022), but closest to the work of Kulal et al. (2019). Figure 2 illustrates this process of translating a single student proof step into Lean, and repeating the process until the student is finished with their proof or the student makes an error in their proof. We prompt a general-purpose LLM to autoformalize the student proofs. (We do not pursue the post-training of or inference on open-source models at this time due to a severe limitation in both open-source models intended for whole proof autoformalization and the size of PeanoBench.) To support the autoformalization task, we add several key pieces of information in-context of our model:

- *Staff Solution*: We provide one reference proof in both NL and FL. This reference proof may or may not align with the student proof being autoformalized. We do not leverage this staff solution beyond providing it in the context, but aim to do so in future work.
- *Theorem and Tactic Dictionary*: We create a dictionary of all theorems and tactics in the dataset, where the keys are the formal Lean names of the theorems and tactics, and the values are natural language descriptions of each.
- *5-shot examples*: We include five examples of translations of a natural language proof step and corresponding Lean formalization following Murphy et al. (2024).

Proof Checker The input to autoformalizer module will include both correct and incorrect proofs. As shown in Figure 2, each autoformalizing student proof step is appended to the Lean theorem statement and previously formalized steps. The proof is compiled, via LeanInteract (Poiroux, Kuncak, and Bosselut 2025). If the compiler output indicates only `unsolved goals`, we assume the student step is correct and proceed with autoformalizing remaining steps. For any other error message (`unknown tactic`, `error:unexpected identifier`, etc.), we assume the student step is incorrect and mark this proof step as erroneous. (Note: We end the autoformalization process once the first error is located.) Furthermore, a compiler error can indicate either an incorrect student proof step, or an autoformalization error. This is a limitation of the system that we intend to address in future work.

4.2 Next Step Generator

The Next Step Generator (NSG) is launched when the student proof is not identified as complete and correct by the autoformalizer/proof checker. The NSG takes as input the formalized partial student proof (with the incorrect step removed). It aims to output a Lean tactic that can lead to a complete proof. Similar to (Thakur et al. 2023), the module performs an LLM-directed depth-first proof search. An LLM is instructed to generate 12 candidate tactics with a rank-ordering of their likelihood of being a correct next step. The prompt includes a list of all tactics/premises used in the NNG4 world of that theorem.

The 12 generated tactic candidates are appended to the existing proof and run through the Lean compiler (via LeanInteract (Poiroux, Kuncak, and Bosselut 2025)). Compiling tactics are then filtered through a *progress check*, which follows Thakur et al. (2023) and Sanchez-Stern et al. (2020). In the progress check, we (1) ensure we are not using any theorems on a list of forbidden theorems (we define this list to include the theorem we are currently trying to prove and theorems that are introduced after the theorem being proven in the order defined by NNG4) and (2) avoid cyclic tactics that would cause the proof-tree to revisit a goal state (Thakur et al. 2023). We build a proof-search tree using all tactics that fulfill the compilation and progress check and do a depth-first search until a complete proof is found. We bound the tree depth to eight, which is sufficient for most of the proofs in our case. If a proof cannot be found, we report this to the following feedback generator module.

4.3 Natural Language Feedback Generator

The feedback-generation module combines information from previous modules to provide natural language feedback to the student. This module takes as input the student's autoformalized proof, the Lean compiler error message (if present), and the next Lean tactic generated from the NSG module. To aid in error identification, we include six common errors students have made in inductive proofs (Baker 1996) in our prompt.

We use this information to automatically generate three types of feedback common in ITS. Similar to the automatic feedback generated by D'antoni et al. (2015), we (1) identify the student error and (2) generate a hint or question that guides the student to the next step. We also generate (3) an explicit next step the student could take, similar to a *bottom-out hints* (Suzuki et al. 2017). This third part of our feedback is very similar to the auto-informalization task in automated theorem proving (Li et al. 2024a).

5 Experiments

We evaluate performance of the entire LeanTutor system on incorrect proofs. In this experiment, a baseline model and LeanTutor are both given incorrect proofs as input and generate NL feedback as output. Human evaluators then assess the generated feedback across four axes: Accuracy, Relevance, Readability, and Answer Leakage, on a 5-point scale. These experiments are detailed in section 5.4. To understand the impact of key innovations in our autoformalizer, namely the presence of staff solutions and the step-by-step autoformalization approach, we perform experiments and ablations on just our Autoformalizer. To assess our model's performance at the faithful autoformalization task, we present

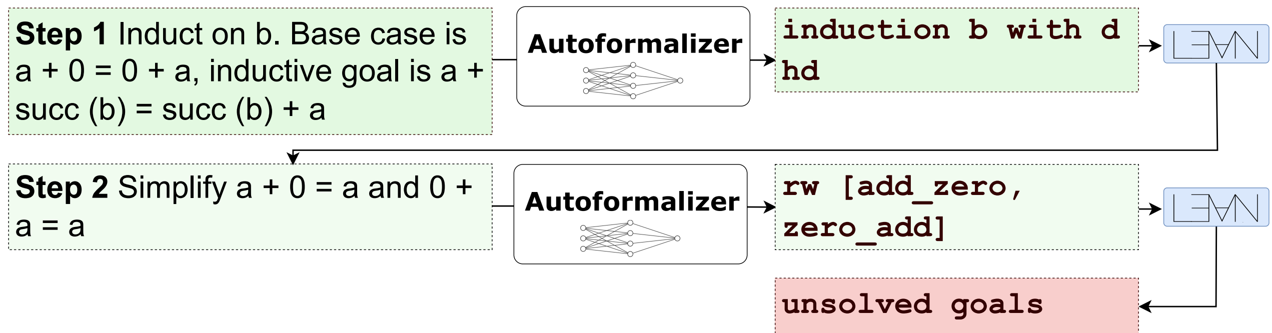

Figure 2: Autoformalizer architecture. The natural langauge student step is provided to the autoformalizer, and the output is checked by the Lean compiler. The formalization of each step is appended to formalizations of previous steps to check for correctness.

Experiment	Correct Tactics	Correct Proofs	Incorrect Proofs
Baseline	32.9% ± 3.1%	6.7% ± 4.0%	14.4% ± 5.7%
Baseline + Staff Solution	**56.8% ± 3.2%**	18.0% ± 6.1%	**30.1% ± 7.4%**
Baseline (whole proof)	28.2% ± 2.9%	10.7% ± 4.9%	13.0% ± 5.4%
Baseline + Staff Solution (whole proof)	51.8% ± 3.3%	**26.7% ± 7.0%**	21.9% ± 6.7%

Table 1: Autoformalization performance across correct and incorrect proofs. Autoformalization is done step-by-step in the Baseline and Baseline + Staff Solution experiments. In the experiments labeled with (whole proof), the whole NL proof is autoformalized into Lean code at once. Binomial error bars were computed using Jeffreys prior with a 95% confidence interval.

a novel metric. These experiments are explained in section 5.2. All experiments cost less than $4.00 to run on `gpt-4o-mini-2024-07-18`. We expect that the autoformalization performance can be boosted by using other more powerful LLMs. However, since the focus of our paper is presenting a model-agnostic framework for the LeanTutor model, we do not optimize over different LLMs.

5.1 Metric for Faithful Autoformalization

A few metrics have been developed to assess faithful autoformalization (Murphy et al. 2024; Liu et al. 2025; Lin et al. 2025; Li et al. 2024b). Both the measures proposed by Murphy et al. (2024); Liu et al. (2025) are too coarse for our use case. Li et al. (2024b) verify Isabelle formalizations and rely on Sledgehammer, Lin et al. (2025) use an LLM-as-a-judge, Murphy et al. (2024) use an SMT solver to prove equivalence between two statements, and Liu et al. (2025) define a new equivalence relation. While we also use this paradigm, we focus primarily on human evaluations, since LLM evaluations can include hallucinations and be unreliable.

We develop a metric that performs *relaxed exact matching*. Our metric has two phases. Firstly, exact *tactic-matching* is attempted in which the generated tactic string is matched with the ground truth tactic string. If string matching fails, we move to the second phase: *state-matching*. In *state-matching* we compare the two tactics by checking if the proof states (the proof state rendered once the predicted and ground truth tactics have been appended to the existing predicted and ground truth proofs respectively) are syntactically identical up to variable naming. We call our metric *relaxed*, because we accommodate differing variable names between the input and ground truth proofs. To do this, proof states are segmented by goal and/or casework and we locate all variables through a custom Python implementation of Lean Identifiers (Community 2024). Variables in all goal state segments are standardized and string matching can ensue. If this check fails as well, we deem the predicted tactic as not a faithful autoformalization of the input NL proof stem.

5.2 Autoformalizer Evaluation

Our *baseline model* adapts the autoformalization prompt proposed by Murphy et al. (2024) to our dataset. Their autoformalization prompt was designed for a small dataset use case in which all tactics/premises can be provided in-context; this is appropriate for PeanoBench. Our baseline prompt contains the theorem statement in both NL and FL, the tactic and theorem dictionaries, five examples of the formalization task, and the student input that needs to be formalized.

All model outputs are evaluated at *pass@1*, in contrast with some prior work in autoformalization in which performance has been evaluated at pass@8 (Liu et al. 2025). For correct proof formalizations, accuracies at both the tactic and proof levels were measured. Tactic-level accuracies were determined using the metric described above. Proof-level accuracy was measured by verifying that all NL statements in a given proof were autoformalized into the correct tactics, in the correct order. Thus, proof-level accuracy measures a much higher performance bar than does tactic-level accuracy. For incorrect proof formalizations, we report only proof-level accuracy. Thus, for incorrect proofs a formalization is considered successful if (1) all correct proof steps until the first incorrect step were formalized correctly and (2) formalization

of the incorrect proof step leads to a Lean compiler error.

Findings We report results in Table **??**. Tactic-level results are out of 900 total tactics, correct proofs results are out of 150 total proofs, and incorrect proof results are out of 146 proofs. A proof is considered correct of *all* tactics in that proof were correctly formalized. The Baseline + Staff Solution model displays superior performance in all categories compared to the baseline.

We find that the model relies heavily on the staff solution in its context in generating the formalization. In 89% of correct tactic formalizations, the tactic predicted was in the staff solution (meaning that the tactic was present somewhere in the staff solution provided in context, but possibly in a different position than the predicted tactic). When the autoformalization was wrong, the model copied a tactic in the staff-solution, instead of predicting the real tactic corresponding to the NL input, in 51% of cases. When the expected formalization was not in the staff solution, LeanTutor was able to correctly formalize the NL proof step in 32% of cases. These results demonstrate a limitation of the current approach's ability to autoformalize NL proofs whose formalization does not correspond to the formalized staff solution provided in-context.

Ablations We compare our autoformalizer model to one ablation: generating whole proofs all at once instead of step-by-step generations (experiments labeled with (whole proof) in Table **??**). With this approach, the autoformalized whole proof does not necessarily contain the same number of tactics as our ground truth whole proof. We truncate proof lengths to `min(len(generated proof), len(ground truth proof))` (the length of a proof referring to the number of tactics in the proof) and align both proofs to each other tactic-by-tactic. We compute tactic-level and proof-level accuracy in the same manner described above[2]. Considering the models with staff solutions, the step-by-step autoformalization approach has comparable performance to the whole proof autoformalization on correct proofs. However, the step-by-step approach outperforms the whole proof approach on incorrect proofs, by 8%. As many incoming proofs to a tutoring system will be incorrect, better performance on incorrect proofs is advantageous.

We performed additional experiments, evaluating the impact of adding the student's natural language proof and Lean goal state information in-context of the autoformalizer. However, these experiments performed lower than the Baseline + Staff Solution experiment.

5.3 Metric for LeanTutor Feedback

In the system-level evaluation of LeanTutor, a student NL proof is input and NL feedback is generated as output. We evaluate the generated outputs on four axes: *Accuracy*, *Relevance*, *Readability*, and *Answer Leakage*, motivated by the metrics used in Mitra et al. (2024); Mozafari et al. (2025). We evaluate each of our three categories of feedback (error identification, hint/question generation and explicit next step) along each axis using a 5-point scale.

We define what it means to receive the highest rating of 5 for each axis. A score of 1 indicates complete disagreement with the following definitions. **Accuracy**: The generated error/hint/next-step is correctly and accurately identified (Mitra et al. 2024). **Relevance**: The generated error/hint/next step is relevant to the error/proof following Mitra et al. (2024); Mozafari et al. (2025). **Readability**: The generated feedback is coherent (Mitra et al. 2024). **Answer Leakage**: The generated feedback does not disclose the answer in any way (Mozafari et al. 2025).

5.4 LeanTutor Evaluation

We evaluate our full system on incorrect proofs. Across our experiments, we use `gpt-4o-mini-2024-07-18` (temperature = 0.0). Two external evaluators (graduate students in the Computer Science department with prior teaching experience in math courses) scored all generated feedback from the baseline and LeanTutor models. Both evaluators were provided the same evaluation protocol document and had a brief discussion with a paper author, to explain the scoring task. The evaluations were model-blind—the evaluators did not which model generated which feedback. Each evaluator scored half the proofs, so each proof's feedback was evaluated by a single evaluator.

We evaluate our end-to-end system on a subset of incorrect proofs from PeanoBench. We only consider incorrect proofs that were "successfully autoformalized" by the LeanTutor autoformalizer. Of the 44 "successfully autoformalized" proofs, we randomly selected one to three proofs per world, totaling 21 proofs[3]. These proofs are passed through our Next Step Generator and Feedback Generator modules. We compare to a simple baseline, providing the LLM with the erroneous student proof and prompting the model to generate the three feedback types.

Our system-level evaluation (Table 2) indicates LeanTutor outperforms the baseline model on the *Accuracy* and *Relevance* metrics. Performance on the *Readability* and *Answer Leakage* metrics are comparable for both models. (Note: We expect high answer leakage in the scores for "next step" feedback; a score of 1 is expected).

For completeness, we performed the same evaluation, with `gpt-4` as the judge, following the "LLM-as-a-judge" paradigm. The "judge" was calibrated to three proofs' feedback scores given by the paper authors. The results show that the baseline model edges out LeanTutor in almost every scoring axis on every feedback type. However, these results do not align with our human evaluators' scores, which highlights issues with the paradigm.

[2] Our metric is imperfect for evaluating generated whole proofs. Thus, we also evaluate how many generated whole proofs (in the correct proof experiments) also completed successfully, with the Lean compiler displaying `no goals`. The Baseline (whole proof) model produced 28 compiling proofs and the Baseline + Staff Solution model (whole proof) produced 50 compiling proofs. Note, that a complete Lean proof doesn't serve as an appropriate measure for faithful autoformalization.

[3] We excluded proofs where the skipped step did not lead to a Lean compiler error.

Feedback Type	Accuracy	Relevance	Readability	Answer Leakage
Baseline Error Identification	**3.4**	3.5	**4.5**	**4.6**
LeanTutor Error Identification	**3.4**	**3.8**	4.3	4.5
Baseline Hint/Question	3.1	2.9	**4.7**	4.3
LeanTutor Hint/Question	**3.7**	**3.7**	**4.7**	**4.4**
Baseline Next Step	2.8	2.9	**4.6**	**2.2**
LeanTutor Next Step	**3.7**	**3.9**	4.4	1.1

Table 2: Average (across all proofs) scores of generated feedback from baseline and LeanTutor experiments on 21 incorrect proofs. The generated feedback was scored on a scale of 1-5 in which a score closer to 5 indicates desired performance.

6 Limitations

LeanTutor presents a proof-of-concept design for a formally-verified mathematical proof tutoring system, and leaves many open questions for future research. Some limitations come from assumptions in our system design, that may make it difficult to generalize our system. (1) Assuming a one-to-one correspondence between NL proof steps and FL tactics, which generally will not scale to more complicated proofs. (2) We assume the presence of an already formalized staff solution, which could be a significant burden on an instructor in the absence of a good autoformalizer. (3) Our metric for faithful autoformalization applies only when ground truth formalizations exist. We aim to explore approaches that only use an informal staff solution.

A second set of limitations comes from our dataset construction. (1) While students commonly miss steps in writing proofs, there are other types of errors that are not captured in the incorrect proofs dataset. (2) All of the natural language in PeanoBench has been written by paper authors, as opposed to non-author students (the varied personas are an earnest effort to incorporate realistic natural language variations).

Finally, a critical limitation in our system design is that if autoformalization fails, we may provide incorrect feedback or not be able to respond to the student. As a result, in our evaluation, we did not evaluate end-to-end system performance on proofs with incorrect autoformalization. Relatedly, we assume a student proof is incorrect if the Lean compiler errors. However, errors may also result from incorrect autoformalization, which could lead to false positives (though spot checking revealed this was not a big issue).

7 Conclusions and Future Work

Our future goal is to deploy LeanTutor in large undergraduate mathematics classes such as discrete math and linear algebra. Our next-steps include:

- more effective use of formal/informal reference proofs in the Autoformalizer and NSG.
- implement backtracking in the next-step generation module to discard unviable student steps and generate a feasible next step, starting from the most recent point in the student's proof that can lead to a complete proof.

Additionally, we have the open challenges on faithful autoformalization due to granularity and reasoning paths as discussed in the introduction.

Finally, the tutoring application really forces a focus on the *human* factors that must go into joint human-AI systems. While the current paper does not focus on these factors, it opens the door to working on them. Our hope is that LeanTutor's approach of combining state-of-the-art LLMs with the Lean theorem prover inspires future systems that combine LLMs with external verifiers in educational applications.

Acknowledgments

The authors would like to acknowledge support from: NSF CAREER grant ECCS-2240031, the 2023 CITRIS Institute Seed Grant, the UC Berkeley Instructional Technology and Innovation MicroGrant Program, Google and the Renaissance Philanthropy AI for Math 2025 fund.

References

Anthropic. 2025. Anthropic education report: How university students use Claude. https://www.anthropic.com/news/anthropic-education-report-how-university-students-use-claude. Accessed: 2025-11-28.

Askell, A.; Bai, Y.; Chen, A.; Drain, D.; Ganguli, D.; Henighan, T.; Jones, A.; Joseph, N.; Mann, B.; DasSarma, N.; et al. 2021. A general language assistant as a laboratory for alignment. *arXiv preprint arXiv:2112.00861*.

Autexier, S.; Dietrich, D.; and Schiller, M. 2012. Towards an intelligent tutor for mathematical proofs. *arXiv preprint arXiv:1202.4828*.

Avigad, J. 2019. Learning logic and proof with an interactive theorem prover. *Proof technology in mathematics research and teaching*, 277–290.

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

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

Baker, J. D. 1996. Students' Difficulties with Proof by Mathematical Induction. In *Annual Meeting of the American Educational Research Association*.

Balunović, M.; Dekoninck, J.; Jovanović, N.; Petrov, I.; and Vechev, M. 2025. MathConstruct: Challenging LLM

Reasoning with Constructive Proofs. *arXiv preprint arXiv:2502.10197*.

Barnes, T.; and Stamper, J. 2008. Toward automatic hint generation for logic proof tutoring using historical student data. In *International conference on intelligent tutoring systems*, 373–382. Springer.

Briggle, A.; et al. 2008. Towards an intelligent tutoring system for propositional proof construction. *Current Issues in Computing and Philosophy*, 175: 145.

Bundy, A.; Moore, J.; and Zinn, C. 2000. An intelligent tutoring system for induction proofs. In *CADE-17 Workshop on Automated Deduction in Education*, 4–13.

Buzzard, K. 2022. Teaching formalisation to mathematics undergraduates.

Buzzard, K.; Eugster, J.; Pedramfar, M.; Bentkamp, A.; Massot, P.; Carey, S.; Farabella, I.; and Browne, A. 2023. NNG4: natural number game in Lean 4. https://github.com/leanprover-community/NNG4. Accessed: 2025-11-28.

Community, L. 2024. Defining new syntax — identifiers. https://lean-lang.org/doc/reference/latest/Notations-and-Macros/Defining-New-Syntax/. Accessed: 2025-11-28.

Cooper, A. 1999. *The inmates are running the asylum*. Springer.

Cunningham, G.; Bunescu, R. C.; and Juedes, D. 2023. Towards autoformalization of mathematics and code correctness: Experiments with elementary proofs. *arXiv preprint arXiv:2301.02195*.

D'antoni, L.; Kini, D.; Alur, R.; Gulwani, S.; Viswanathan, M.; and Hartmann, B. 2015. How can automatic feedback help students construct automata? *ACM Transactions on Computer-Human Interaction (TOCHI)*, 22(2): 1–24.

DeNero, J.; and Martinis, S. 2014. Teaching composition quality at scale: human judgment in the age of autograders. In *Proceedings of the 45th ACM technical symposium on Computer science education*, 421–426.

Dermeval, D.; Paiva, R.; Bittencourt, I. I.; Vassileva, J.; and Borges, D. 2018. Authoring tools for designing intelligent tutoring systems: a systematic review of the literature. *International Journal of Artificial Intelligence in Education*, 28: 336–384.

Gao, G.; Wang, Y.; Jiang, J.; Gao, Q.; Qin, Z.; Xu, T.; and Dong, B. 2024. Herald: A natural language annotated Lean 4 dataset. *arXiv preprint arXiv:2410.10878*.

Ge, T.; Chan, X.; Wang, X.; Yu, D.; Mi, H.; and Yu, D. 2024. Scaling synthetic data creation with 1,000,000,000 personas. *arXiv preprint arXiv:2406.20094*.

Goetze, C. 2025. The Real Reason Why Students Are Using AI to Avoid Learning. *Time*.

Gupta, A.; Reddig, J.; Calo, T.; Weitekamp, D.; and MacLellan, C. J. 2025. Beyond Final Answers: Evaluating Large Language Models for Math Tutoring. *arXiv preprint arXiv:2503.16460*.

Hecht, R.; Liu, R.; Zenke, C.; and Malan, D. J. 2023. Distributing, Collecting, and Autograding Assignments with GitHub Classroom. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2*, 1179–1179.

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

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

Karsten, N.; Jacobsen, F. K.; Eiken, K. J.; Nestmann, U.; and Villadsen, J. 2023. ProofBuddy: A Proof Assistant for Learning and Monitoring. *arXiv preprint arXiv:2308.06970*.

Kulal, S.; Pasupat, P.; Chandra, K.; Lee, M.; Padon, O.; Aiken, A.; and Liang, P. S. 2019. Spoc: Search-based pseudocode to code. *Advances in Neural Information Processing Systems*, 32.

Levonian, Z.; Henkel, O.; Li, C.; Postle, M.-E.; et al. 2025. Designing Safe and Relevant Generative Chats for Math Learning in Intelligent Tutoring Systems. *Journal of Educational Data Mining*, 17(1).

Li, Z.; Sun, J.; Murphy, L.; Su, Q.; Li, Z.; Zhang, X.; Yang, K.; and Si, X. 2024a. A Survey on Deep Learning for Theorem Proving. In *First Conference on Language Modeling*.

Li, Z.; Wu, Y.; Li, Z.; Wei, X.; Zhang, X.; Yang, F.; and Ma, X. 2024b. Autoformalize mathematical statements by symbolic equivalence and semantic consistency. *arXiv preprint arXiv:2410.20936*.

Lin, Y.; Tang, S.; Lyu, B.; Wu, J.; Lin, H.; Yang, K.; Li, J.; Xia, M.; Chen, D.; Arora, S.; et al. 2025. Goedel-Prover: A Frontier Model for Open-Source Automated Theorem Proving. *arXiv preprint arXiv:2502.07640*.

Liu, Q.; Zheng, X.; Lu, X.; Cao, Q.; and Yan, J. 2025. Rethinking and Improving Autoformalization: Towards a Faithful Metric and a Dependency Retrieval-based Approach. In *The Thirteenth International Conference on Learning Representations*.

Lodder, J.; Heeren, B.; Jeuring, J.; and Neijenhuis, W. 2021. Generation and use of hints and feedback in a Hilbert-style axiomatic proof tutor. *International Journal of Artificial Intelligence in Education*, 31: 99–133.

Lu, J.; Wan, Y.; Liu, Z.; Huang, Y.; Xiong, J.; Liu, C.; Shen, J.; Jin, H.; Zhang, J.; Wang, H.; et al. 2024. Process-driven autoformalization in lean 4. *arXiv preprint arXiv:2406.01940*.

mathlib Community, T. 2020. The Lean Mathematical Library. In *Proceedings of the ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP)*. ACM.

Maurya, K. K.; Srivatsa, K.; Petukhova, K.; and Kochmar, E. 2024. Unifying AI Tutor Evaluation: An Evaluation Taxonomy for Pedagogical Ability Assessment of LLM-Powered AI Tutors. *arXiv preprint arXiv:2412.09416*.

Miller, P.; and DiCerbo, K. 2024. LLM Based Math Tutoring: Challenges and Dataset.

Mitra, C.; Miroyan, M.; Jain, R.; Kumud, V.; Ranade, G.; and Norouzi, N. 2024. RetLLM-E: retrieval-prompt strategy

for question-answering on student discussion forums. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 23215–23223.

Mitra, J. 2023. Studying the impact of auto-graders giving immediate feedback in programming assignments. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, 388–394.

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

Mozafari, J.; Piryani, B.; Abdallah, A.; and Jatowt, A. 2025. HintEval: A Comprehensive Framework for Hint Generation and Evaluation for Questions. *arXiv preprint arXiv:2502.00857*.

Murphy, L.; Yang, K.; Sun, J.; Li, Z.; Anandkumar, A.; and Si, X. 2024. Autoformalizing Euclidean geometry. *arXiv preprint arXiv:2405.17216*.

OpenAI. 2025. College students and ChatGPT adoption in the US. https://openai.com/global-affairs/college-students-and-chatgpt/. Accessed: 2025-11-28.

Pardos, Z. A.; and Bhandari, S. 2023. Learning gain differences between ChatGPT and human tutor generated algebra hints. *arXiv preprint arXiv:2302.06871*.

Poiroux, A.; Kuncak, V.; and Bosselut, A. 2025. LeanInteract: A Python Interface for Lean 4.

Ranathunga, S.; Lee, E.-S. A.; Prifti Skenduli, M.; Shekhar, R.; Alam, M.; and Kaur, R. 2023. Neural machine translation for low-resource languages: A survey. *ACM Computing Surveys*, 55(11): 1–37.

Riehl, E. 2025. Testing artificial mathematical intelligence. https://emilyriehl.github.io/files/testing.pdf. Accessed: 2025-11-28.

Sanchez-Stern, A.; Alhessi, Y.; Saul, L.; and Lerner, S. 2020. Generating correctness proofs with neural networks. In *Proceedings of the 4th ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, 1–10.

Shao, Z.; Wang, P.; Zhu, Q.; Xu, R.; Song, J.; Bi, X.; Zhang, H.; Zhang, M.; Li, Y.; Wu, Y.; et al. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Shi, J.; Torczon, C.; Goldstein, H.; Pierce, B. C.; and Head, A. 2025. QED in Context: An Observation Study of Proof Assistant Users. *Proceedings of the ACM on Programming Languages*, 9(OOPSLA1): 337–363.

Sieg, W. 2007. The AProS project: Strategic thinking & computational logic. *Logic Journal of the IGPL*, 15(4): 359–368.

Simha, R.; and Wright, H. 2025. The CRA Practitioner-to-Professor (P2P) Survey. Technical report.

Singh, R.; Gulwani, S.; and Solar-Lezama, A. 2013. Automated feedback generation for introductory programming assignments. In *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, 15–26.

Sonkar, S.; Ni, K.; Chaudhary, S.; and Baraniuk, R. G. 2024. Pedagogical alignment of large language models. *arXiv preprint arXiv:2402.05000*.

Suzuki, R.; Soares, G.; Glassman, E.; Head, A.; D'Antoni, L.; and Hartmann, B. 2017. Exploring the design space of automatically synthesized hints for introductory programming assignments. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, 2951–2958.

Thakur, A.; Tsoukalas, G.; Wen, Y.; Xin, J.; and Chaudhuri, S. 2023. An in-context learning agent for formal theorem-proving. *arXiv preprint arXiv:2310.04353*.

Thoma, A.; and Iannone, P. 2022. Learning about proof with the theorem prover lean: the abundant numbers task. *International Journal of Research in Undergraduate Mathematics Education*, 1–30.

Tyen, G.; Mansoor, H.; Cărbune, V.; Chen, Y. P.; and Mak, T. 2024. LLMs cannot find reasoning errors, but can correct them given the error location. In *Findings of the Association for Computational Linguistics ACL 2024*, 13894–13908.

Wang, R.; Zhang, J.; Jia, Y.; Pan, R.; Diao, S.; Pi, R.; and Zhang, T. 2024a. Theoremllama: Transforming general-purpose llms into lean4 experts. *arXiv preprint arXiv:2407.03203*.

Wang, R.; Zhang, J.; Jia, Y.; Pan, R.; Diao, S.; Pi, R.; and Zhang, T. 2024b. TheoremLlama: Transforming General-Purpose LLMs into Lean4 Experts. arXiv:2407.03203.

Weber, K. 2001. Student difficulty in constructing proofs: The need for strategic knowledge. *Educational studies in mathematics*, 48(1): 101–119.

Welleck, S.; Liu, J.; Lu, X.; Hajishirzi, H.; and Choi, Y. 2022. Naturalprover: Grounded mathematical proof generation with language models. *Advances in Neural Information Processing Systems*, 35: 4913–4927.

Wemmenhove, J.; Arends, D.; Beurskens, T.; Bhaid, M.; McCarren, S.; Moraal, J.; Garrido, D. R.; Tuin, D.; Vassallo, M.; Wils, P.; et al. 2022. Waterproof: educational software for learning how to write mathematical proofs. *arXiv preprint arXiv:2211.13513*.

Wu, Y.; Jiang, A. Q.; Li, W.; Rabe, M.; Staats, C.; Jamnik, M.; and Szegedy, C. 2022. Autoformalization with large language models. *Advances in Neural Information Processing Systems*, 35: 32353–32368.

Yang, K.; Poesia, G.; He, J.; Li, W.; Lauter, K.; Chaudhuri, S.; and Song, D. 2024. Formal mathematical reasoning: A new frontier in ai. *arXiv preprint arXiv:2412.16075*.

Yang, K.; Swope, A.; Gu, A.; Chalamala, R.; Song, P.; Yu, S.; Godil, S.; Prenger, R. J.; and Anandkumar, A. 2023. LeanDojo: Theorem proving with retrieval-augmented language models. *Advances in Neural Information Processing Systems*, 36: 21573–21612.

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