Formally Verified AI Co-Reasoning: An Education Case Study

Arnav Mehta [*1] **Thomas Lu** [*1] **Chenjun Guo** [*1] **Niels Voss** [†1] **Manooshree Patel** [†1] **Rayna Bhattacharyya** [1]
Peter Donovan [1] **Gireeja Ranade** [1]

Abstract

Proof assistants such as Lean4 offer machine-checkable verification of mathematical reasoning, but their adoption is limited because formal languages have a steep learning curve and can be challenging to use. Large language models offer a way to lower this barrier to entry through *natural-language co-reasoning systems*: proof assistants that users interact with entirely in natural language, and which use formal verification to provide feedback to users who submit incomplete and incorrect proofs. This paper explores such a co-reasoning system in an educational setting.

We make three contributions. First, we identify **desiderata** for natural language co-reasoning systems—*faithfulness*, *accuracy*, *appropriate granularity*, *transparency*, and *incremental editability*. Second, we introduce an **evaluation framework** for the first three of these properties in the context of educational proofs. Third, we present a **case study** on an educational natural-language co-reasoning system for math proofs, whose auto-formalization harness we evaluate on student-written proofs from an undergraduate linear algebra class against the three desiderata.

1. Introduction

Proof assistants like Lean4 (de Moura & Ullrich, 2021) and Isabelle (Paulson, 1994) have the ability to assist math research and education because they provide a mechanism to scalably and reliably check mathematical proofs. They have been adopted by some mathematicians in large-scale formalization projects such as Mathlib (mathlib Community, 2020), but adoption of proof assistants in mainstream mathematics remains limited. Formal languages pose challenges to users due to syntax and readability (Shi et al., 2025) and proofs in formal languages require a much greater level of detail and justification than those in natural language (Patel et al., 2026).

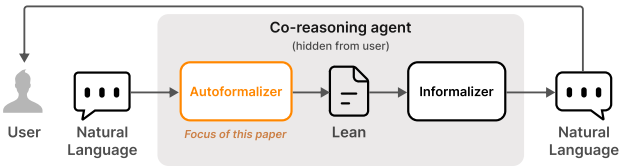

Figure 1. NL co-reasoning system architecture

Recent advances in LLMs have shown promise in bridging the gap between natural and formal language. LLMs now assist users of proof assistants with tactic suggestion (Song et al., 2024; Welleck & Saha, 2023), premise selection (Yang et al., 2023), proof repair (Thakur et al., 2025; Lu et al., 2024c), conjecturing (Onda et al., 2025), informalization (Gao et al., 2024), proof generation (Ren et al., 2025; Chen et al., 2025a; Lin et al., 2025), and auto-formalization (Wu et al., 2022; Lu et al., 2024b). These tools make it easier for users to interact with a proof assistant, but they still require knowledge of the formal language.

We introduce the notion of a **natural language co-reasoning** system (*NL co-reasoning*): a proof assistant that a user interacts with entirely in natural language. Such a system allows users to iteratively develop their informal proofs: they may submit incomplete or incorrect proofs, receive formally verified natural-language feedback, and revise their reasoning accordingly. Figure 2 shows our co-reasoning interface, the left panel shows the user inputted proof, the central panel shows the informalization of each step, and the right panel shows the Lean4 code. Although NL co-reasoning systems are designed to be used entirely through natural-language interaction, we make the underlying Lean4 code available for users who wish to inspect the formal proof; this view is hidden by default.

A user's workflow is as follows (see Figure 1): (1) the user inputs their reasoning in free-form natural language; (2) the input is auto-formalized into a formal language such as Lean4; (3) the resulting formal program is compiled; (4) the generated code and compiler output are informalized

[*]Equal contribution. [†]Equal contribution. [1]University of California, Berkeley, USA. Correspondence to: Arnav Mehta <arnavmehta@berkeley.edu>, Gireeja Ranade <ranade@berkeley.edu>.

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

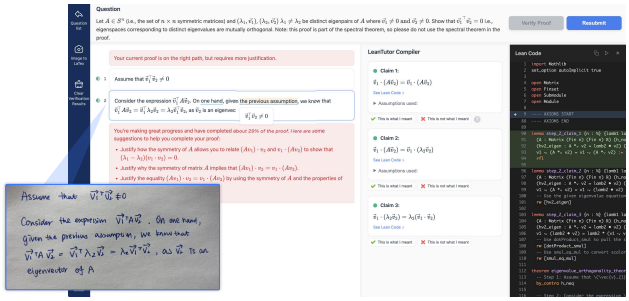

Figure 2. This figure shows the co-reasoning interface. The left panel shows the student's input proof (transcribed from handwriting via Image-to-LaTeX) with feedback on its correctness, completeness, and suggested next steps. The central panel shows the LeanTutor Compiler's informalization of each formalized step. The right panel shows the corresponding Lean code.

back into natural language as verification feedback; and (5) the user uses this feedback to revise and resubmit their reasoning. Steps (2)–(4) are hidden from the user, who is never required to read or write formal code.

Existing auto-formalization approaches use formal proof primarily as a post-hoc check—to verify correctness of a natural language proof after it is considered complete (Wu et al., 2022; Lu et al., 2024b). NL co-reasoning instead integrates formal verification into the proof discovery process itself, using it to surface errors to the user at intermediate steps.

Co-reasoning systems that users interact with in natural language are valuable because they can deliver formally verified feedback without disrupting a user's existing workflow. In mathematics for instance, both expert mathematicians proving new conjectures and students learning to write proofs can use NL co-reasoning systems to receive per-step verification and feedback that is trustworthy (Ospanov et al., 2025; Avigad, 2019; Lodder et al., 2021; Keenan & Omar, 2024; Patel et al., 2026). Beyond math, legal and tax professionals must reason within the constraints of statutes and tax codes (Jurayj et al., 2026; PramaanaLabs), and clinicians must produce evidence-based justifications for their diagnostic conclusions (Hekmatnejad et al., 2019; Singh et al., 2026). In each case, a co-reasoning system can productively surface inconsistencies, with no additional burden to the user.

Our main contributions are: (1) desiderata for natural language co-reasoning systems; (2) an evaluation framework for faithfulness, accuracy and appropriate granularity in the context of education; and (3) a case study applying this framework to an educational NL co-reasoning system, whose auto-formalization harness is evaluated on student-written proofs from an undergraduate linear algebra course.

2. Natural language co-reasoning desiderata

Building on prior work (Patel et al., 2026; Cabral et al., 2025; Murphy et al., 2024), we identify five desiderata for realizing such a co-reasoner. The first three characterize properties of the system itself and can be evaluated using quantitative measures introduced in Section 3. The remaining two concern user interaction and require user feedback for assessment. We therefore address the first three in this work and leave the latter two to future work.

1. **Faithfulness:** the formal proof accurately reflects both the structure (Cabral et al., 2025) and semantics (Murphy et al., 2024) of the natural-language proof provided as input.

2. **Accuracy:** the system provides mathematically accurate feedback for each reasoning step, i.e. it identifies correct steps as correct, and incorrect steps as incorrect.

3. **Appropriate granularity:** the system must require users to justify their reasoning at a level of detail appropriate for the target application. Formal languages such as Lean require fully rigorous formal proofs. This can impose unnecessary burdens on users, who may naturally think at a higher level of granularity (Guo et al., 2025). The level of granularity required depends on the context; an advanced mathematical course may allow students to assume lemmas that compress substantial logical jumps, while an introductory course will require more justification. Thus, it must be possible to calibrate an NL co-reasoning system to the needs of the context it is deployed in.

4. **Transparency:** A user must be able to view and interpret the system state without having to read and interpret the associated formal language. System state includes the inputs and outputs of the formal system, error messages, and other user-facing data surfaced by the proof assistant. For example, for Lean4 co-reasoning systems, goal states must be informalized before being displayed to the user. In Figure 2, the user does not need to open the Lean code panel to verify proofs in the system.

5. **Incremental Editability:** Users of NL co-reasoning systems revise their proofs iteratively in response to feedback. When a user makes an edit, the system should update only the formal code affected by that change (Hu et al., 2025). If every revision produces an entirely new formalization, the user would not be able to edit an isolated part of their proof.

The educational setting is a particularly interesting case study for NL co-reasoning and provides a natural environment to measure the above desiderata. Students learning to

write proofs naturally generate diverse informal arguments, including incomplete and incorrect proofs. Instructors implicitly or explicitly communicate a notion of sufficiently rigorous or justified proofs (that is specific to their classrooms) (Yackel & Cobb, 1996).

Evaluating these properties presents a significant challenge. Faithfulness is not typically considered when judging auto-formalization success, as a compiling proof is often treated as a successful output. Thus, auto-formalization systems are typically evaluated by Pass@k on benchmarks (Azerbayev et al., 2023; Wang et al., 2024; Ying et al., 2024; Liu et al., 2025b; Huang et al., 2025). Much of the prior work also assumes well-formed informal statements (Lu et al., 2024b; Liu et al., 2025a), while a co-reasoning system must handle proofs that are incomplete, ambiguous, or incorrect.

3. Metrics

3.1. Faithfulness

Measuring the semantic equivalence between two statements is not a well-defined problem. Thus, we attempt to measure faithfulness by considering two proxy metrics—*information preservation* and *non-dilution*. Essentially, these measure "did the auto-formalizer retain everything the user included?" and "did the auto-formalizer refrain from adding anything the user did not include?"

3.1.1. Information Preservation and Non-Dilution

Let a *natural-language proof* Π_{NL} and a *formal-language proof* Π_{FL} each be decomposed into a set of content-bearing pieces, which we call *units*. The choice of what counts as a unit is a parameter of the definition; we call it a *resolution choice* and fix one before measuring. For example, we may examine a Lean4 proof at the level of tactics, which is a high resolution perspective, or at the level of `have` statements, which is lower resolution. Given a resolution choice, write $U_{NL}(\Pi_{NL})$ and $U_{FL}(\Pi_{FL})$ for the resulting unit sets. We further fix, as part of the resolution choice, a relation $\sim$ between NL units and FL units, where $u_{NL} \sim u_{FL}$ indicates that the two units express the same mathematical information.

Definition (Information preservation) Π_{FL} *preserves the information* of Π_{NL} iff every unit in the natural-language proof has a corresponding unit (under $\sim$) in the formal proof—nothing got dropped. We measure this as:

$$\text{IP} = \frac{\text{\# NL units matched in FL under} \sim}{\text{\# NL units}}$$

Definition (Non-dilution) Π_{FL} is *non-diluting* with respect to Π_{NL} iff every unit in the formal proof has a corresponding unit in the natural-language proof—nothing was added. We measure this as:

$$\text{ND} = \frac{\text{\# FL units matched in NL under} \sim}{\text{\# FL units}}$$

Together, the two properties say the formal proof is exactly what the user said, no more and no less. Note both scores are in the range $[0, 1]$.

The definition of a "unit" in both natural and formal-language proofs is fundamental to this definition. We provide working definitions for both.

3.1.2. Units in natural language

A natural language proof comprises of two kinds of reasoning steps (Guo et al., 2025; Sieg, 1990): *declarations*, which introduce new variables, objects, or hypotheses (e.g. "let x be the zero vector"), and *assertions*, which make a mathematical claim (e.g. "the matrix M is symmetric").

We adopt the unitization procedure of Guo et al. (2025), which decomposes a natural-language proof into a sequence of *declaration* and *assertion* steps exactly as written by the user. Each resulting step is one natural-language (NL) unit.

3.1.3. Units in formal language:

The choice of proof-unit resolution is application-dependent. In educational settings, the coarsest unit may be a single theorem, since students are typically asked to prove one theorem at a time. In other settings, such as formalizing a legal document, the coarsest unit may instead be an entire document comprising multiple theorems. To evaluate faithfulness using the metrics above, one must first specify the proof units that the system is designed to support. In our case, we evaluate co-reasoning systems at three levels of resolution, ordered from coarse to fine. These reflect the common levels at with Lean4 proofs are structured, though this list is not intended to be exhaustive. The proposed metrics are agnostic to the choice of resolution and apply to any decomposition that defines meaningful proof units.

Low-resolution: Top-level statements. A Lean4 file typically contains named `theorems` and `lemmas`, each of which asserts a proposition. The unit is the *initial goal state* at the start of each such declaration. This corresponds to the resolution used in statement auto-formalization benchmarks (Azerbayev et al., 2023; Ying et al., 2024; Gulati et al., 2024; Poiroux et al., 2024).

Medium resolution: Intermediate sub-claims. Within a proof, tactics such as `have`, `suffices`, and `show` introduce intermediate propositions; the unit is the goal state following each such statement. This granularity corresponds to proof-sketch and sub-goal decomposition approaches in the

literature (Jiang et al., 2022; Ren et al., 2025; Cabral et al., 2025).

High resolution: Tactic states. In a tactic-mode proof, Lean4 exposes a goal state after every tactic. The unit is each such goal state. A block of n tactics typically yields $n + 1$ goal states. This is the granularity used by tactic-prediction models, which learn to predict the next tactic given the current proof state (Yang et al., 2023; Welleck & Saha, 2023).

3.2. Measuring Appropriate Granularity

We wish to identify when a co-reasoning system operates at an 'appropriate granularity'. Here it is easier to define what failure looks like than to characterize success. We do not want users to provide extensive justification for mathematical facts that may seem trivial to humans but require long proofs in Lean4. On the other hand, particularly in educational contexts, we do not want to allow students to make large logical leaps and have gaps in their reasoning be filled in by a powerful auto-formalizer. Evaluating whether the first of the undesirable behaviors exists in the system requires extensive user studies that are out of scope for the current paper. We focus on detecting the second kind of undesirable behavior in this work. To do this we consider the task of auto-grading formal proofs.

Proof auto-graders (Brown & Michel, 2010; Moore, 2016; Miller et al., 2018) automatically assign scores to student proofs. A central part of that task is identifying *logical leaps*: steps that are stated by a student without sufficient justification (Fallis, 2003; Weber, 2008; Inamdar et al., 2025). A co-reasoner that can detect insufficiently justified steps should, in principle, provide information useful for grading. We use this idea to define our proxy metric.

Concretely, we compare a baseline grading procedure that does not perform granularity detection (*lenient*) with one that does (*strict*). We then measure whether granularity detection improves agreement with human graders.

To grade a formal proof against a rubric, we first associate each rubric item with the formal proof steps that are relevant to evaluating that item. A grading procedure then maps those steps to a binary credit decision for the item. In *lenient* mode, all steps are provided to the grading procedure, whereas in *strict* mode, steps flagged as insufficiently justified by the granularity detector are discarded before the credit decision is made.

If the granularity detector correctly identifies insufficiently justified steps, then removing those steps before grading should bring the resulting credit decisions closer to those made by human graders. We therefore measure the value of the detector by the improvement in grading agreement obtained when switching from lenient to strict mode.

Let C denote the set of student submissions, and A^i_m denote the agreement rate on rubric item i under grading mode m:

$$A^i_m = \frac{1}{|C|} \sum_{c \in C} \mathbf{1}[\, g_m(c)_i = h(c)_i \,],$$

where $h(c)_i$ and $g_m(c)_i$ are the human and system credit decisions for rubric item i on submission c, and $m \in \{\text{strict}, \text{lenient}\}$.

We define the granularity score for rubric item i as the gain in agreement obtained by using strict mode:

$$\Delta^i = A^i_{\text{strict}} - A^i_{\text{lenient}}.$$

The overall granularity score is the average gain across rubric items:

$$\Delta = \frac{1}{|R|} \sum_{i \in R} \Delta^i.$$

We weight all rubric items equally, regardless of the number of points assigned to each.

A large positive Δ means that withholding credit for insufficiently justified steps brings the system's grading closer to the human's. This is preliminary evidence that the detector identifies the gaps a human grader would penalize.

This measure is indirect and has three limitations. First, human graders are imperfect (Thomas et al., 2025), yet we treat them as ground truth. Second, even a correct detector may produce a small Δ if insufficiently justified steps are rare in the evaluation data. Third, the metric assumes that the grading procedure on the formal proof is accurate.

3.3. Measuring Accuracy

We define accuracy as whether the system correctly identifies the mathematical validity of each natural-language proof unit, as defined in Section 3.1.2, under the problem context. We report true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) at the unit level.

4. A case study: A educational co-reasoning system for math proofs

We developed a proof-of-concept NL co-reasoning system for proof tutoring. The system is comprised of an auto-formalizer and an auto-informalizer. It accepts free-form student proofs and allows users to formally verify their proofs,

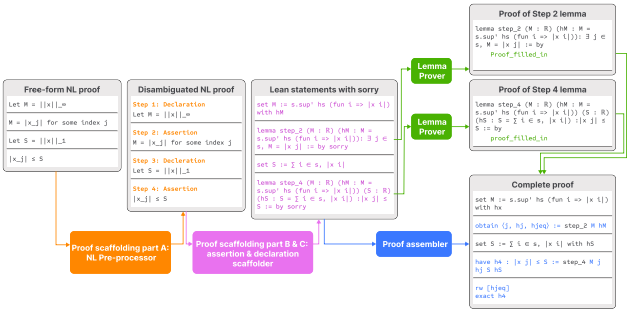

Figure 3. Auto-formalization architecture of our system

solely in natural language. This tool was deployed in an undergraduate linear algebra course. In our case study, we assess the auto-formalizer's **faithfulness**, **appropriate granularity**, and **accuracy**. Evaluating the remaining desiderata require user studies that out of the scope of this work.

4.1. Auto-formalization harness

Our system preserves the sub-claim structure of a student's proof by converting each sub-claim into an independent Lean4 lemma, proving those lemmas, and assembling them into a proof of the original theorem. To faithfully capture the student's reasoning, we maintain the local context established by the proof. Whenever a student introduces new assumptions or variables, these are added to the context and inherited by all subsequent sub-claims until they go out of scope. For example, if a student writes "let u be a unit vector," then all later sub-claims within that scope are checked under the corresponding hypotheses that u is defined and is a unit vector. The pipeline has three stages (Figure 3):

Proof scaffolder: Extracts an ordered list of sub-claims from the student's proof and converts each into a self-contained Lean4 lemma statement. We call these scaffold lemmas.
Lemma prover: Proves each scaffold lemma via a compiler-error repair loop augmented with a curated library of *supporting mini lemmas*—routine facts that can be invoked during proof construction (Jiang et al., 2022)
Proof assembler: Stitches the scaffold lemmas into a proof of the original theorem.

4.2. Stage 1: Proof scaffolding

The scaffolder turns each student-written natural langauge sub-claim into a Lean4 lemma statement (with proof left as `sorry`) in three steps:

NL pre-processing: We build on the pre-processing system of Guo et al. (2025), which decomposes natural language proofs into minimal independent units at the user-specified granularity, in three steps: (1) resolve referential expressions such as "they" or "by similar reasoning" to their referents; (2) classify each step as a declaration or assertion (Sieg, 1990); (3) for assertions, extract antecedent and consequent. The output is a list of disambiguated steps, with implication assertions labeled with its antecedent and consequent.

Assertion scaffolding: An LLM then converts each assertion step into a Lean4 proposition — the type after the colon in

```
lemma name (hypotheses): prop := by ...
```

For implication assertions, the antecedent is lifted into a hypothesis (`h_antecedent : ...`) and the consequent becomes the proposition. The proof is left as `sorry`; the hypothesis signatures for the lemmas are completed in the declaration scaffold step.

Declaration scaffolding: To recover the hypotheses in scope at each assertion, we prompt an LLM to translate each student declaration into a Lean4 tactic. We assemble a partial proof by placing these tactics in order and inserting `extract_goal` (mathlib Community, 2020) at every assertion position. `extract_goal` is a Mathlib tactic that serializes the current goal state as a standalone theorem whose hypotheses are exactly those in scope at the call site, giving us the hypotheses to attach to the subsequent assertion's lemma (Figure 4).

Type checking and error correction: Lean4 enforces strict type correctness, so each lemma statement is type-checked before proof synthesis. If a generated lemma statement fails to compile, it is passed into a loop in which an LLM attempts to revise the statement. Importantly, the LLM is provided with explicit instructions to correct only formalization errors and never alter the mathematical content of the assertion. However, this is not always enforceable in practice. A compilation error could be caused by a mathematical error made by the student: for instance, a matrix multiplication with incompatible dimensions will trigger a type error in Lean4 even when the assertion faithfully reflects the student's claim. In such cases, the correction loop cannot fix the compilation error without editing the mathematical content, which we identify as a limitation of this approach.

4.3. Stage 2: Lemma proving

The prover attempts to prove the generated Lean4 lemma statements. Real-time feedback demands speed and low cost (Patel et al., 2026; Barnes & Stamper, 2008; Lodder et al., 2021), yet formal proof search is expensive and slow (Yang et al., 2023; Xin et al., 2024b; Liu et al., 2026).

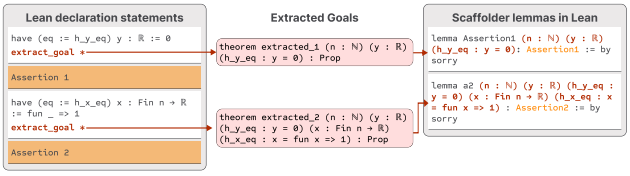

Figure 4. using `extract_goals` to get the local context for each assertion

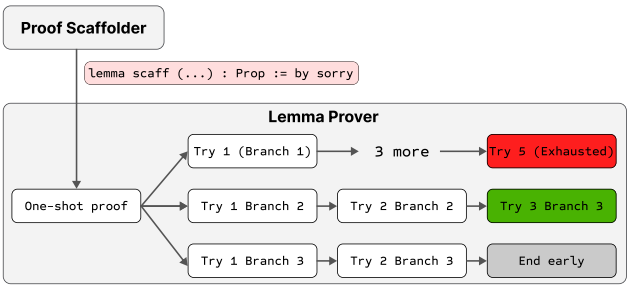

Figure 5. Lemma proving stage: scaffolded lemma is proved with a one-shot attempt followed by branch for proof diversity. prover concludes if one of the branches succeed or every branch exhausts 5 attempts

We address this with two design choices. First, we augment the prover with *supporting mini lemmas* that supply problem-specific facts, saving the prover the burden of deriving low-level results from scratch (Jiang et al., 2022). Scaffold lemmas corresponding to earlier assertions in the proof are also available to the prover. Second, a staged pipeline (Figure 5) tries a one-shot proof (Xin et al., 2024a), then falls back to parallel sampling for diversity (Xin et al., 2024b; Lin et al., 2025), with a compiler-driven repair loop of up to 5 attempts per branch (Thakur et al., 2025; Zhang et al., 2025). Together these shrink the search space enough that a lightweight model such as Gemini Flash (Comanici et al., 2025) suffices as the prover.

4.4. Supporting mini lemma generation

A *supporting mini lemma* is a small auxiliary result that the prover may invoke when proving sub-claims in a students' proof. Supporting mini lemmas serve two purposes. First, they provide reusable facts that allow the prover to focus on higher-level reasoning. Second, they provide a mechanism to calibrate and detect the "size" of sub-claims.

A stronger mini-lemma set permits larger inferential steps, while a weaker set requires more intermediate reasoning to be made explicit.

Rather than retrieving from a large library such as Mathlib (mathlib Community, 2020; Yang et al., 2023), we construct problem-specific mini lemmas tailored to the target theorem. In this paper, the mini lemmas are written by humans, though automated generation is a promising direction for future work. This design allows us to maintain a consistent proof granularity across problems (Gusakov et al., 2025).

Main and side lemmas. Mini lemmas differ in mathematical significance, a distinction used by the granularity detector in the next section. Each is annotated by the authors as either *main* or *side*. A *main* lemma is a key mathematical fact for the problem relative to the course rubric, and its appearance indicates substantive reasoning; a *side* lemma is auxiliary Lean4 machinery — type coercions, definitional unfolding, and other boilerplate without significant mathematical content. The annotation is made once per problem and reused across submissions; see Appendix E for an example.

4.5. Granularity Detection

Each *main* mini lemma corresponds to a rubric-scorable step (Section 4.4). The number of main mini lemmas invoked in the proof of a scaffold lemma serves as a proxy for the amount of mathematical reasoning compressed into that student claim. If the prover invokes more than one main mini lemma to discharge a single scaffold lemma, we flag the claim as a **large jump**. This is the systems' *appropriate granularity detector* as introduced in Section 3.2.

4.6. Stage 3: Proof assembler

The proof assembler's purpose is to verify that the student's top-level argument is logically valid: given the sub-claims as stated, does the main theorem follow? Given (i) the theorem statement, (ii) the ordered scaffold lemmas, (iii) the Lean4 statements for the declarations, and (iv) the student's NL proof, an LLM is prompted to produce a Lean4 proof invoking the lemmas and declaration tactics in the order mentioned in the student's proof. Similar to Thakur et al. (2025), compilation failures are returned to the LLM for revision, up to 5 attempts.

The assembler runs whether or not every scaffold lemma was closed. Unresolved lemmas are left as `sorry` and treated as assumed facts during assembly, analogous to sub-goal decomposition in recent auto-provers (Ren et al., 2025; Zhou et al., 2025).

The assembler is prompted to close the main theorem using the proven scaffold lemmas, the declaration statements, and routine tactics: `rw`, arithmetic normalization (`ring`, `linarith`, `nlinarith`) and `positivity`. It is instructed not to introduce substantial mathematical facts.

Calibrating how much mathematical content the assembler is allowed to add is domain-dependent and remains open: if the assembler is too conservative, it may fail even when the student's argument is correct; if it is too permissive, it

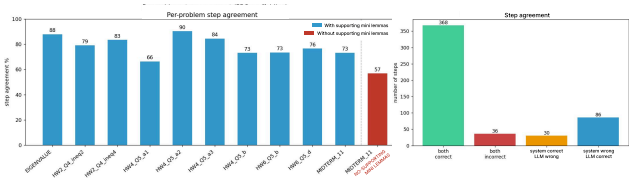

Figure 6. System accuracy evaluation. Left: per-problem step-level agreement between the system and the LLM evaluator; the red bar shows the one problem without supporting mini lemmas. Right: aggregate agreement and disagreement counts.

may fill gaps the student did not provide. In future work, we hope to replace the LLM assembler with a custom Lean4 tactic that makes this boundary explicit.

5. Evaluations

We evaluate the system on 100 natural-language proofs from real student homework submissions: 10 proof problems with 10 student solutions each. For each proof, we run the student's natural-language steps through the system pipeline and record step-level verification outcomes.

5.1. Accuracy

Accuracy is defined as whether the system's verification outcome matches the mathematical validity of each proof step under the problem context. As a baseline, an LLM evaluator judges each natural language proof step. We manually audited a random 10% of the LLM judge's decisions and found that 34 of 35 steps (97%)were judged correctly. Appendix F reports the evaluator prompt. The system's outputs are then compared with the baseline judgments (Figure 6), and all disagreements are manually reviewed. The main source of false positive and false negative come from scaffolder generating unfaithful lemma statements, we manually review all cases and report the analysis in Appendix C. We perform an ablation study on one problem where the lemma prover (Section 4.3) does not have access to supporting mini lemmas. Figure 6 shows that accuracy agreement reduces from 70% to 58%.

5.2. Faithfulness

We measure faithfulness as defined in Section 3.1, defining the correspondence relation $\sim$ to be LLM-as-a-judge that decides whether a natural language unit and a informalized formal-language unit convey the same mathematical content. We randomly picked and manually audited 10% of these decisions to validate them. Table 2 shows the human and LLM disagreement report. Appendix A details our methodology.

We report information preservation and non-dilution across the 100 natural-language proofs at varying granularity (Table 1). For 26 of the 100 solutions, the system does not

	Information Preservation	Non-Dilution
All Proofs (100)		
Top-Level	0.6305	0.7616
Sub-Claims	0.6704	0.3968
Tactic States	0.3645	0.1000
Complete Proofs Only (74)		
Top-Level	0.7194	0.7926
Sub-Claims	0.7625	0.4051
Tactic States	0.4096	0.1040

Table 1. Information preservation and non-dilution scores for formalized proofs, by granularity level.

Human vs. LLM Disagreement		
Top-Level	Sub-Claims	Tactic States
6.19%	3.09%	19.59%

Table 2. Human–LLM judge disagreement rate on the faithfulness mapping, by granularity level, over a manually audited 10% sample (10 proofs, 97 NL steps).

generate a complete formal-language proof, due to lemma statements that failed to compile. For example, in Figure 7, a student step initiate a matrix with mismatched dimensions fails the Lean4 typechecker, as do all steps that follow. Information preservation naturally suffers here, since fewer formal statements are produced; we therefore report separate results for the 74 problems that yield a complete formal-language proof.

Our results show a trade-off between non-dilution and information preservation as unit resolution changes: finer units expose more FL content, raising information preservation ($0.72 \rightarrow 0.76$) but lowering non-dilution ($0.79 \rightarrow 0.40$) through additional low-level scaffolding. No resolution level tested approaches perfect information preservation or non-dilution. We also approached faithfulness measurement using cosine similarity in a vector embedding space (Appendix B), but found it unreliable: semantic similarity does not track mathematical equivalence. A verbose student step and its clean informalized equation can be equivalent yet score low, while a minor auto-correction (e.g., an added absolute value) can alter the mathematical meaning yet score high.

5.3. Appropriate Granularity

We measure granularity using the metric defined in Section 3.2. To do this, we run the pipeline on one problem (P10 in Appendix D), using 50 student submissions. Each submission is compared to the ground truth grades assigned by a TA using a 7-item rubric worth 8 points in total (Appendix E.1).

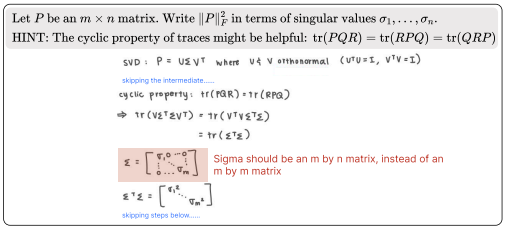

Figure 7. An example student solution where the formal lemma statement does not compile. The solution contains a dimension error (red): the student writes Σ as an $m \times m$ matrix when it should be $m \times n$.

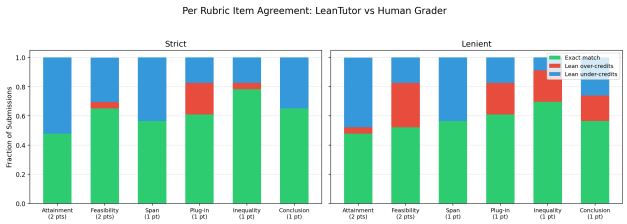

Figure 8. Per-item agreement between our system and the human grader under strict mode (left) and lenient mode (right) for problem P10 ($n = 50$ submissions). Each bar shows the fraction of submissions where our system exactly matched the human (green), over-credited (red), or under-credited (blue). Strict mode achieves higher exact-match rates on most rubric items.

For a given student proof, we collect the list of *main mini lemmas* invoked across all scaffold lemmas (Section 4.4); scaffold lemmas that the prover fails to close contribute nothing. Across the 50 proofs, submissions invoke 4.1 main mini lemmas on average. The detector then flags a mini lemma as insufficiently justified when it appears *only* inside large-jump scaffold steps (Section 4.5) — 71 of the 540 scaffold lemmas in the corpus (13%) were flagged as large-jumps. *Strict mode* grades with large-jump flagged mini lemmas removed and *lenient mode* keeps them. Each rubric item is then resolved as a logical condition over the surviving main mini lemmas: `any_of` L (one lemma in L suffices, for multiple valid approaches), `all_any` $\{L_1, \ldots, L_k\}$ (one from each L_i, for multiple required sub-claims), or `all_of` L (every lemma in L).

5.4. Granularity Results

Strict mode agrees with the human grader on 62.1% of item-level decisions versus 57.1% for lenient. $\Delta = +5.0\%$, and strict matches or exceeds lenient on every rubric item and is strictly better on four of seven (Figure 8). These results provide preliminary evidence that the granularity detector is identifying insufficiently justified steps correctly.

A limitation of this evaluation is that rubric items are scored solely based on supporting mini lemmas. However, human graders also award credit for intermediate results, correct use of hypotheses, and other forms of proof progress that may not be reflected in supporting mini lemmas. Extending the rubric representation to capture these forms of evidence is an important direction for future work.

6. Related Work

Faithful Autoformalization While many prior works on autoformalization focus on translating informal mathematical statements into Lean propositions (Ying et al., 2024; Gao et al., 2024; Wu et al., 2022; Azerbayev et al., 2023), whole-proof autoformalization is comparatively more difficult (Jiang et al., 2022; Huang et al., 2024; Patel et al., 2026; Tarrach et al., 2024). A natural desiderata for whole-proof autoformalization is faithfulness – the requirement that a formal proof accurately reflects both the structure and semantics of its natural-language counterpart. Metrics to evaluate faithfulness assess syntactic correctness, semantic similarity, and structural equivalence (Jiang et al., 2023; Moore & Shah, 2025; Lu et al., 2024a; Liu et al., 2025a; Li et al., 2024) between the informal and autoformalized proofs. We extend these criteria by defining information preservation and non-dilution, which extend structural equivalence by requiring a one-to-one pairing between formal-language and natural-language units.

A number of approaches have been proposed for faithful proof-level autoformalization. LeanTutor (Patel et al., 2026) formalizes informal proofs step-by-step to preserve semantic equivalence, and ProofFlow (Cabral et al., 2025) constructs an explicit dependency graph of proof steps before formalizing each node as a lemma, preserving structural fidelity. Quan et al. (2025) and Guo et al. (2025) parse the structure of informal proofs to reduce downstream ambiguity. Despite this progress, faithfulness remains an open challenge: LLMs have been observed to mistranslate informal proofs, fabricate or omit axioms, and produce logically inconsistent formal statements (Kim et al., 2026; Moore & Shah, 2025).

Appropriate Granularity The role of granularity—the level of detail at which a system operates, relative to the user's reasoning—has been acknowledged, but not fully studied, in the autoformalization literature. LeanTutor (Patel et al., 2026) assumes a one-to-one mapping between Lean4 tactics and informal steps, but observes that more complex operations in Lean 4 lack direct natural-language corespondents. Nevertheless, following the user's level of granularity is valuable; Tarrach et al. (2024) iteratively elaborates informal proof steps when formalization fails, and Wang et al. (2025) generates formal proof states aligned with the informal proof's logical structure. When evaluating formalization faithfulness, ProofFlow (Cabral et al., 2025) acknowledges that multiple levels of formal-language granularity may exist and be valid, but does not attempt to

further categorize them. Our work can be viewed as a development in the spirit of Guo et al. (2025), which identifies the "unspoken logic" implicit in informal proofs and develop a preprocessing pipeline that surfaces this latent structure before formalization; we explicitly define and study the different levels of formal language granularity, and argue that granularity should be be *calibratable to a specific use case*; a system grading proofs in an introductory course must demand more justification at each step than one used in a graduate course.

Accuracy Common automated theorem proving and autoformalization evaluations often report compile success, proof success, or pass@k (Azerbayev et al., 2023; Ying et al., 2024; Wu et al., 2022; Yang et al., 2023; Xin et al., 2024a; Lin et al., 2025; Ren et al., 2025; Chen et al., 2025b). Our step-level framing is closer to work on natural-language proof verification and proof grading (Hu et al., 2025; Patel et al., 2026; Ma et al., 2025; Mahdavi et al., 2025; Gonzalez et al., 2026; Mahinpei et al., 2026).

7. Limitations

This work evaluates the auto-formalization harness of an educational NL co-reasoning system for math proofs, not a full end-to-end tutoring system. We do not evaluate the natural-language feedback shown to students or whether using the system improves proof-writing ability; we leave this to future work. Our results should therefore be read as evidence about the formalization and verification pipeline rather than about educational effectiveness.

Our experiments are limited to undergraduate linear algebra proofs from one course, so performance may differ in other mathematical domains or courses with different justification standards.

The supporting mini-lemmas are also problem-specific and human-authored, so the reported results do not reflect the cost or reliability of constructing them automatically.

The accuracy evaluation uses an LLM-based reference evaluator with manual review of disagreements. This provides a useful proxy, though some student steps are ambiguous or valid only under course-specific conventions, making step-level correctness inherently difficult to label in edge cases.

Our faithfulness metrics are proxy measures that depend on proof unitization, LLM informalization, embedding similarity, and one-to-one matching. These choices may not capture every valid correspondence between informal and formal steps, and do not fully encode dependency structure or proof order.

The granularity evaluation is indirect: we measure whether strict grading improves agreement with a human rubric rather than directly measuring large-jump detection. As it is also run on a single problem, these results should be treated as preliminary.

8. Acknowledgment

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.

The authors would also like to thank Aditya Chandrasekhar, Dakota Feldkamp, and Krishiv Kothari for their work on the human auditing of the evaluation results.

References

Avigad, J. Learning logic and proof with an interactive theorem prover. In Hanna, G., Reid, D. A., and de Villiers, M. (eds.), *Proof Technology in Mathematics Research and Teaching*, volume 14 of *Mathematics Education in the Digital Era*, pp. 277–290. Springer, Cham, 2019. doi: 10.1007/978-3-030-28483-1_13.

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

Barnes, T. and Stamper, J. Toward automatic hint generation for logic proof tutoring using historical student data. In Woolf, B. P., Aïmeur, E., Nkambou, R., and Lajoie, S. (eds.), *Intelligent Tutoring Systems (ITS 2008)*, volume 5091 of *Lecture Notes in Computer Science*, pp. 373–382, Berlin, Heidelberg, 2008. Springer. doi: 10.1007/978-3-540-69132-7_41.

Brown, D. E. and Michel, S. Assessing proofs with rubrics: The RVF method. In *Proceedings of the 13th Conference on Research in Undergraduate Mathematics Education*, 2010.

Cabral, R., Do, T. M., Yu, X., Tai, W. M., Feng, Z., and Shen, X. Proofflow: A dependency graph approach to faithful proof autoformalization. *arXiv preprint arXiv:2510.15981*, 2025.

Chen, J., Chen, W., Du, J., Hu, J., Jiang, Z., Jie, A., Jin, X., Jin, X., Li, C., Shi, W., Wang, Z., Wang, M., Wei, C., Wei, S., Xin, H., Yang, F., Gao, W., Yuan, Z., Zhan, T., Zheng, Z., Zhou, T., and Zhu, T. H. Seed-prover 1.5: Mastering undergraduate-level theorem proving via learning from experience, 2025a. URL https://arxiv.org/abs/2512.17260.

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

Comanici, G., Bieber, E., Schaekermann, M., et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

de Moura, L. and Ullrich, S. The Lean 4 theorem prover and programming language. In *International Conference on Automated Deduction*, pp. 625–635. Springer, 2021.

Fallis, D. Intentional gaps in mathematical proofs. *Synthese*, 134(1–2):45–69, 2003. doi: 10.1023/A:1022131513275.

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

Gonzalez, S., Amiri Bavandpour, A., Ye, P., Zhang, E., et al. QEDBench: Quantifying the alignment gap in automated evaluation of university-level mathematical proofs. *arXiv preprint arXiv:2602.20629*, 2026.

Gulati, A., Ladsaria, D., Mishra, S., Sidhu, J., and Miranda, B. An evaluation benchmark for autoformalization in Lean4. In *ICLR 2024 Tiny Papers*, 2024.

Guo, C., Patel, M., Hartmann, B., Zamfirescu-Pereira, J., Chasins, S., and Ranade, G. Unspoken logic: Understanding and bridging the gap between free-form and llm-interpretable natural language mathematical proofs. In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS 2025*, 2025.

Gusakov, A., Nelson, P., and Watt, S. M. Structuring definitions in mathematical libraries. *arXiv preprint arXiv:2509.10828*, 2025. URL https://arxiv.org/abs/2509.10828.

Hekmatnejad, M., Simms, A. M., and Fainekos, G. Model checking clinical decision support systems using SMT, 2019. URL https://arxiv.org/abs/1901.04545. arXiv preprint arXiv:1901.04545 (v2, revised 4 March 2019).

Hu, X., Zhou, Q., Grechuk, B., and Tyukin, I. Y. Stepproof: Step-by-step verification of natural language mathematical proofs, 2025. URL https://arxiv.org/abs/2506.10558.

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

Huang, Y., Jin, X., Liang, S., Li, P., and Liu, Y. Formarl: Enhancing autoformalization with no labeled data, 2025. URL https://arxiv.org/abs/2508.18914.

Inamdar, A., Macar, U., Vazirani, M., Tarnow, M., Mustapha, Z., Dittren, N., Sadeh, S., Verma, N., and Salleb-Aouissi, A. LogicLearner: A tool for the guided practice of propositional logic proofs, 2025.

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

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

Jurayj, W., Holzenberger, N., and Durme, B. V. Language models and logic programs for trustworthy tax reasoning, 2026. URL https://arxiv.org/abs/2508.21051.

Keenan, M. and Omar, C. Learner-centered design criteria for classroom proof assistants. In *Workshop on Human Aspects of Types and Reasoning Assistants (HATRA)*, 2024. URL https://hazel.org/papers/learner_centered_proofs_hatra_2024.pdf.

Kim, K., Poiroux, A., and Bosselut, A. Do LLMs game formalization? Evaluating faithfulness in logical reasoning. In *ICLR 2026 Workshop on Verified AI (VerifAI-2)*, March 2026.

Li, Z., Wu, Y., Li, Z., Wei, X., Yang, F., Zhang, X., and Ma, X. Autoformalize mathematical statements by symbolic equivalence and semantic consistency. *Advances in Neural Information Processing Systems*, 37:53598–53625, 2024.

Lin, Y., Tang, S., Lyu, B., Yang, Z., Chung, J.-H., Zhao, H., Jiang, L., Geng, Y., Ge, J., Sun, J., et al. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. *arXiv preprint arXiv:2508.03613*, 2025.

Liu, J., Zhou, Z., Zhu, Z., Santos, M. D., He, W., Liu, J., Wang, R., Xie, Y., Zhao, J., Wang, Q., et al. Numina-Lean-Agent: An open and general agentic reasoning system for formal mathematics. *arXiv preprint arXiv:2601.14027*, 2026.

Liu, Q., Zheng, X., Lu, X., Cao, Q., and Yan, J. Rethinking and improving autoformalization: Towards a faithful metric and a dependency retrieval-based approach. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=hUb2At2DsQ.

Liu, X., Bao, K., Zhang, J., Liu, Y., Chen, Y., Liu, Y., Jiao, Y., and Luo, T. Atlas: Autoformalizing theorems through lifting, augmentation, and synthesis of data. *arXiv preprint arXiv:2502.05567*, 2025b.

Lodder, J., Heeren, B., Jeuring, J., and Neijenhuis, W. Generation and use of hints and feedback in a hilbert-style axiomatic proof tutor. *International Journal of Artificial Intelligence in Education*, 31(1):99–133, 2021. doi: 10.1007/s40593-020-00222-2.

Lu, J., Wan, Y., Huang, Y., Xiong, J., Liu, Z., and Guo, Z. Formalalign: Automated alignment evaluation for autoformalization. *arXiv preprint arXiv:2410.10135*, 2024a.

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

Lu, M., Delaware, B., and Zhang, T. Proof automation with large language models. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2024c.

Ma, W. et al. Reliable fine-grained evaluation of natural language math proofs. *arXiv preprint arXiv:2510.13888*, 2025.

Mahdavi, H., Mahdavinia, P., Malek, S., Mohammadipour, P., Hashemi, A., Daliri, M., Farhadi, A., Khasahmadi, A., Mireshghallah, N., and Honavar, V. RefGrader: Automated grading of mathematical competition proofs using agentic workflows. *arXiv preprint arXiv:2510.09021*, 2025.

Mahinpei, R., Druchyna, S., and Horta Ribeiro, M. When LLMs help – and hurt – teaching assistants in proof-based courses. *arXiv preprint arXiv:2602.23635*, 2026.

mathlib Community, T. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, POPL '20, pp. 367–381. ACM, January 2020. doi: 10.1145/3372885.3373824. URL `http://dx.doi.org/10.1145/3372885.3373824`.

Miller, D., Infante, N., and Weber, K. How mathematicians assign points to student proofs. *Journal of Mathematical Behavior*, 49:24–34, 2018.

Moore, H. and Shah, A. Evaluating autoformalization robustness via semantically similar paraphrasing. *arXiv preprint arXiv:2511.12784*, 2025.

Moore, R. C. Mathematics professors' evaluation of students' proofs: A complex teaching practice. *International Journal of Research in Undergraduate Mathematics Education*, 2(2):246–278, 2016. doi: 10.1007/s40753-016-0029-y.

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

Onda, N., Kasaura, K., Oriike, Y., Taniguchi, M., Sannai, A., and Sonoda, S. LeanConjecturer: Automatic generation of mathematical conjectures for theorem proving. *arXiv preprint arXiv:2506.22005*, 2025.

Ospanov, A., Feng, Z., Sun, J., Bai, H., Shen, X., and Farnia, F. HERMES: Towards efficient and verifiable mathematical reasoning in LLMs. *arXiv preprint arXiv:2511.18760*, 2025. URL `https://arxiv.org/abs/2511.18760`.

Patel, M., Bhattacharyya, R., Lu, T., Mehta, A., Voss, N., Norouzi, N., and Ranade, G. Leantutor: Towards a verified AI mathematical proof tutor. In *Proceedings of the Fortieth AAAI Conference on Artificial Intelligence*, AAAI '26, pp. 40670–40678. AAAI Press, January 2026. URL `https://ojs.aaai.org/index.php/AAAI/article/view/41514`.

Paulson, L. C. Isabelle: A generic theorem prover. In Bundy, A. (ed.), *Proceedings of the 12th International Conference on Automated Deduction (CADE-12)*, volume 814 of *Lecture Notes in Computer Science*, pp. 37–41. Springer, 1994.

Poiroux, A. et al. Reliable evaluation and benchmarks for statement autoformalization, 2024.

PramaanaLabs. Pramaana labs. `https://www.pramaanalabs.ai/`, 2026. Accessed: 2026-05-08.

Quan, X., Valentino, M., Dennis, L. A., and Freitas, A. Faithful and robust llm-driven theorem proving for nli explanations. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17734–17755, 2025.

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

Shi, J., Torczon, C., Goldstein, H., Pierce, B. C., and Head, A. Qed in context: An observation study of proof assistant users. *Proceedings of the ACM on Programming Languages*, 9(OOPSLA1):1–27, April 2025. doi: 10.1145/3720426.

Sieg, W. *Proof theory*. Carnegie Mellon University, 1990.

Singh, V., Ganguly, D., Yu, H., Zhou, C., Singh, P., Lee, B., Chaudhary, V., and Datta, G. Toward guarantees for clinical reasoning in vision language models via formal verification, 2026. URL https://arxiv.org/abs/2602.24111.

Song, P., Yang, K., and Anandkumar, A. Towards large language models as copilots for theorem proving in lean. *arXiv preprint arXiv:2404.12534*, 2024.

Tarrach, G., Jiang, A. Q., Raggi, D., Li, W., and Jamnik, M. More details, please: Improving autoformalization with more detailed proofs. In *AI for Math Workshop@ ICML 2024*, 2024.

Thakur, A. et al. Apollo: Automated llm and lean collaboration for advanced formal reasoning. *arXiv preprint arXiv:2505.05758*, 2025.

Thomas, D. R., Borchers, C., and Koedinger, K. R. Beyond agreement: Rethinking ground truth in educational AI annotation. *arXiv preprint arXiv:2508.00143*, 2025.

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

Wang, Z., Yang, B., Li, C., Zhang, Y., Zhou, S., Dong, B., and Wen, Z. Translating informal proofs into formal proofs using a chain of states. *arXiv preprint arXiv:2512.10317*, 2025.

Weber, K. How mathematicians determine if an argument is a valid proof. *Journal for Research in Mathematics Education*, 39(4):431–459, 2008.

Welleck, S. and Saha, R. LLMSTEP: LLM proofstep suggestions in Lean. *arXiv preprint arXiv:2310.18457*, 2023.

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

Xin, H., Guo, D., Shao, Z., Ren, Z., Zhu, Q., Liu, B., Ruan, C., Li, W., and Liang, X. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data. *arXiv preprint arXiv:2405.14333*, 2024a.

Xin, H., Ren, Z., Song, J., Shao, Z., Zhao, W., Wang, H., Liu, B., Zhang, L., Lu, X., Du, Q., et al. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. *arXiv preprint arXiv:2408.08152*, 2024b.

Yackel, E. and Cobb, P. Sociomathematical norms, argumentation, and autonomy in mathematics. *Journal for research in mathematics education*, 27(4):458–477, 1996.

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

Ying, H., Wu, Z., Geng, Y., Wang, J., Lin, D., and Chen, K. Lean workbook: A large-scale lean problem set formalized from natural language math problems. *Advances in Neural Information Processing Systems*, 37:105848–105863, 2024.

Zhang, X. et al. Leanabell-prover-v2: Verifier-integrated reasoning for formal theorem proving via reinforcement learning. *arXiv preprint arXiv:2507.08649*, 2025.

Zhou, Y., Zhao, J., Zhang, Y., Wang, B., Wang, S., Chen, L., Wang, J., Chen, H., Jie, A., Zhang, X., et al. Solving formal math problems by decomposition and iterative reflection. *arXiv preprint arXiv:2507.15225*, 2025.

A. Description of Faithfulness Evaluation

Given a formal proof in Lean4, we extract content units at the three levels of granularity given in Section 3.1.3. At all three levels, our units consist of Lean4 goal states, in order to fully disambiguate variable names and definitions. For example, a top-level unit could consist of the goal state immediately following the statement of a `theorem`. All formal units are then informalized into natural language using Claude Sonnet 4.6. Below, we give the prompt used at each level, together with an example of an original formal statement and its informalized version. We observed that the LLM judge is less accurate at the tactic level than at the other levels. We hypothesize the formatting of "Before... After..." could confuse the LLM judge.

Top level. Prompt:

```
f"""--- ENTRY {idx} ---
Kind: {entry.kind}
Name: {entry.name}
Lean source:
{entry.lean_code}

Source-level proposition, if explicit:
{entry.proposition}

Lean-derived checked proposition:
{checked_prop}

Lean goal state to informalize:
{goal_state}
"""
       )

   return f"""PROBLEM CONTEXT
---------------
This is {problem_label}. The overall theorem/problem
     context is:

{lean_theorem}

NOTATION CONVENTIONS USED IN THIS PROBLEM
-----------------------------------------
{custom_instructions}

LEAN GOAL STATES TO INFORMALIZE ({len(batch)})
----------------------------------------------
{chr(10).join(items)}

TASK
----
For each entry, informalize the mathematical claim
    represented by the Lean goal
state to informalize.

Usually this means informalizing the ` ...` line.
    For inferred `have`
statements such as `have := term`, the displayed
    goal state has been synthesized
so that the newly-created hypothesis type appears as
     the goal; informalize that
created fact.

Do not informalize the unchanged outer goal when the
     entry is an inferred `have`
that merely adds a hypothesis.

Return JSON with a single field `informalizations`,
    an array of length
{len(batch)} in the same order as the entries above.
     Each entry has one field:
`informalization`.
"""
```

Input example:

```
lemma step_3_transpose_mul (A : Matrix (Fin n) (Fin
    n) ) ... (h_A_symm : A.IsSymm) :
   A * v1 = lamb1  v1
goal fed to the model:  A * v1 = lamb1  v1
```

Output informalization:

$$A^T v_1 = \lambda_1 v_1$$

Sub-claims level. Prompt:

```
f"""--- ENTRY {idx} ---
Kind: {entry.kind}
Name: {entry.name}
Lean source:
{entry.lean_code}

Source-level proposition, if explicit:
{entry.proposition}

Lean-derived checked proposition:
{checked_prop}

Lean goal state to informalize:
{goal_state}
"""
       )

   return f"""PROBLEM CONTEXT
---------------
This is {problem_label}. The overall theorem/problem
     context is:

{lean_theorem}

NOTATION CONVENTIONS USED IN THIS PROBLEM
-----------------------------------------
{custom_instructions}

LEAN GOAL STATES TO INFORMALIZE ({len(batch)})
----------------------------------------------
{chr(10).join(items)}

TASK
----
For each entry, informalize the mathematical claim
    represented by the Lean goal
state to informalize.
```

```
Usually this means informalizing the ` ...` line.
    For inferred `have`
statements such as `have := term`, the displayed
    goal state has been synthesized
so that the newly-created hypothesis type appears as
     the goal; informalize that
created fact.

Do not informalize the unchanged outer goal when the
     entry is an inferred `have`
that merely adds a hypothesis.

Return JSON with a single field `informalizations`,
    an array of length
{len(batch)} in the same order as the entries above.
     Each entry has one field:
`informalization`.
"""
```

Input example:

```
have h_left : (A * v1)  v2 = lamb1 * (v1  v2) :=
    step_5_symm_apply
goal fed to the model:  A * v1  v2 = lamb1 * v1  v2
```

Output informalization:

$$(Av_1) \cdot v_2 = \lambda_1(v_1 \cdot v_2)$$

Tactic level. Prompt:

```
PROBLEM CONTEXT
---------------
This is {problem_label}. The overall theorem we are
    proving is:

{lean_theorem}

NOTATION CONVENTIONS USED IN THIS PROBLEM
-----------------------------------------
{custom_instructions}

PROOF STEPS TO INFORMALIZE ({len(batch)} steps)
-----------------------------------------------
{steps_block}

TASK
----
For EACH step above, informalize ONLY the goal (the
    `` line(s)) of the BEFORE
and AFTER states into one short, human-style English
     statement of what is being
claimed. Use LaTeX where helpful. Do NOT list all
    the hypotheses; assume the
standard problem context. If a post-state is "no
    goals", set its
informalized_end to "Proof complete." (or similar).

Return JSON with a single field `informalizations`,
    an array of length
{len(batch)} in the same order as the steps above.
    Each entry has two fields:
informalized_start, informalized_end.
```

Input example:

```
exact hv1_eigen
goal fed to the model:
BEFORE:  A * v1 = lamb1  v1
AFTER:  A * v1  v2 = lamb1 * v1  v2
```

Output informalization:

> Before this step, the goal is: $Av_1 = \lambda_1 v_1$
> After this step, the goal is: $(A^T v_1) \cdot v_2 = \lambda_1(v_1 \cdot v_2)$

B. Description of Faithfulness Evaluation (embedding model)

The embeddings of the informalizations are computed using BGE-M3. The natural-language units, produced using the unitization procedure of (Guo et al., 2025), are embedded using the same model, and we compute the formal-language natural-language pairing that maximizes total cosine similarity using the Hungarian method. Our pairing is one-to-one; each unit can be paired to at most one other unit, and only pairings with a cosine similarity of 0.6 or greater are permitted. Based on the number of paired and unpaired units, we compute information preservation and non-dilution scores following Section 3.1.

C. Accuracy examples

There are 86 cases where the LLM evaluator marks a step as correct but the system marks it as incorrect. Manual review shows that 6 are LLM evaluator errors, and 4 are lemma-prover failures, where the prover fails to prove a faithful scaffolded lemma (Sections 4.3 and 4.2). The remaining cases occur when **the scaffolder generates an unfaithful or invalid lemma because the student writes ambiguous or under-specified steps**. Appendix **??** gives three examples in which students omit declarations, types, or assumptions that humans can infer, but the system produces false negatives due to unfaithful lemma statements.

Example 1: Ambiguous assumption markers. In Figure 9 (bottom), the proof requires both directions of an iff statement. For the forward direction, the system must add $\|P\|_F = 0$ as a hypothesis before proving $P = 0$. However, the student begins with "Given $\|P\|_F = 0$," which the preprocessor classifies as a restatement of known context rather than as an assumption for the current proof goal. As a result, the generated Lean4 statement omits the needed hypothesis `(h_norm_zero : P.frobeniusNorm = 0)`, making the claim false.

Example 2: Undeclared variables and implicit conventions. In Figure 9 (top), the student writes $aP = a\sum_i \sigma_i u_i v_i^\top = \sum_i a\sigma_i u_i v_i^\top$. The variables i, σ_i, u_i, and

v_i are not declared, but a human reader infers them from notation used in lectures and textbooks, where U and V commonly refer to the left and right singular-vector matrices in an SVD. A faithful formalization would need either an explicit SVD-expansion hypothesis for P, or explicit declarations of σ_i, u_i, and v_i with their types and index ranges.

Example 3: Intuitively correct but formally invalid steps. In Figure 9 (center), the student writes $\langle P, P\rangle + 2\langle P, B\rangle + \langle B, B\rangle = \|P\|_F^2 + 2\|PB\|_F^2 + \|B\|_F^2$. The term $\|PB\|_F$ is undefined because P and B are both $m \times n$ matrices, so the system cannot prove the step. However, because the intended expansion is clear, a human grader or LLM evaluator may not focus on notation error and mark the step as correct.

Other false negatives arise from two cases: (1) the student asserts a correct but irrelevant step, or a step that uses a non-trivial fact, absent from our mini helper lemma library; and (2) The proof of a scaffold lemma depends on an earlier scaffold lemma whose proof has failed.

For the 24 false positive cases, we identify three causes. First, despite the scaffold lemma correction loop being instructed never to alter mathematical content (Section **??**), we observe it doing so for some type errors — a design flaw to be addressed in future work. Second, some student steps are mathematically true only in context, e.g., `expressionA * a = expressionB * a` holds trivially when `a = 0`, but an LLM judging the claim in isolation marks it false. Third, the pre-processor (Section 4.2) occasionally misclassifies an assertion as a declaration; since declarations are always treated as valid, this produces a false positive.

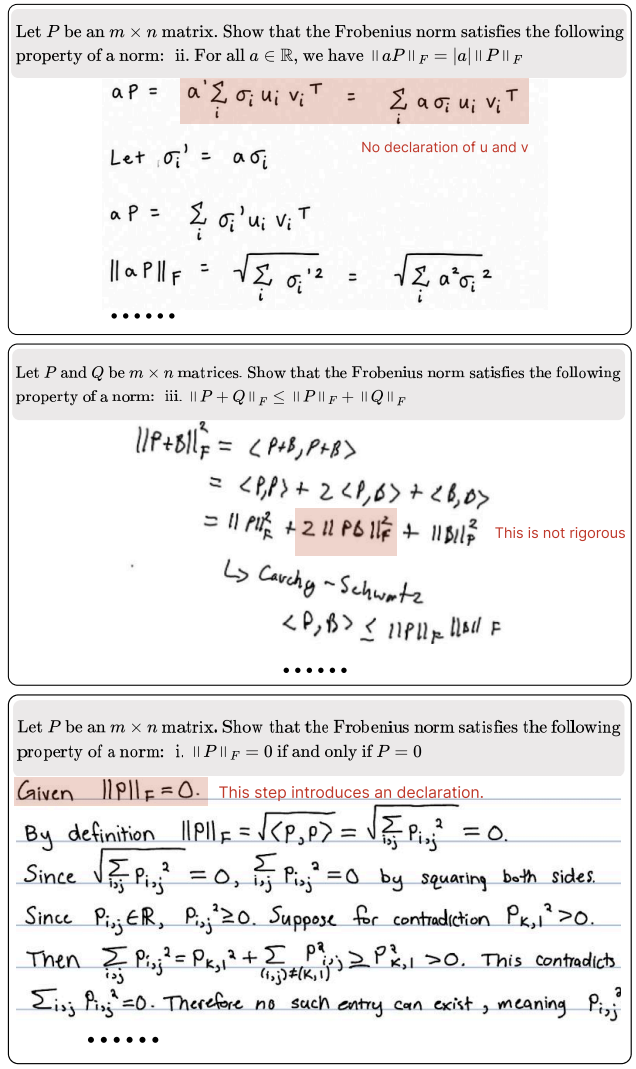

Figure 9. examples for student steps which lead to unfaithful autoformalization

D. Problem Statements

The following ten problems from EECS 127 were used in our evaluation. Each problem was attempted by $n = 10$ students.

1. **Eigenvector Orthogonality** (HW3 Q3)
   Let $A \in S^n$ (i.e., the set of $n \times n$ symmetric matrices) and $(\lambda_1, \mathbf{u}_1), (\lambda_2, \mathbf{u}_2)$, with $\lambda_1 \neq \lambda_2$, be distinct eigenpairs of A. Show that $\mathbf{u}_1^\top \mathbf{u}_2 = 0$, i.e., eigenspaces corresponding to distinct eigenvalues are mutually orthogonal.

2. **Norm Inequality:** $\|\cdot\|_\infty \le \|\cdot\|_2$ (HW2 Q4)
   Show that the following inequality holds for any $\vec{x} \in \mathbb{R}^n$:
   $$\|x\|_\infty \le \|x\|_2.$$

3. **Norm Inequality:** $\|\cdot\|_1 \le \sqrt{n}\,\|\cdot\|_2$ (HW2 Q4)
   Show that the following inequality holds for any $\vec{x} \in \mathbb{R}^n$:
   $$\|x\|_1 \le \sqrt{n} \cdot \|x\|_2.$$

4. **Frobenius Norm: Zero Iff**
   Let P be an $m \times n$ matrix. Show that the Frobenius norm satisfies $\|P\|_F = 0$ if and only if $P = 0$.

5. **Frobenius Norm: Homogeneity**
   Let P be an $m \times n$ matrix. Show that for all $a \in \mathbb{R}$, $\|aP\|_F = |a|\,\|P\|_F$.

6. **Frobenius Norm: Triangle Inequality**
   Let P and Q be $m \times n$ matrices. Show that $\|P + Q\|_F \le \|P\|_F + \|Q\|_F$.

7. **Frobenius Norm via Singular Values**
   Let P be an $m \times n$ matrix with singular values $\sigma_1, \ldots, \sigma_n$. Show that $\|P\|_F^2 = \sum_{i=1}^n \sigma_i^2$.

8. **PSD Implies Non-negative Diagonal**
A symmetric matrix $M \in \mathbb{R}^{n\times n}$ is positive semidefinite if $\vec{x}^\top M\vec{x} \geq 0$ for all $\vec{x} \in \mathbb{R}^n$. Assume $A \in \mathbb{R}^{n\times n}$ is symmetric. Show that if $A \succeq 0$ then $A_{ii} \geq 0$ for all i.

9. **$Q^\top Q$ is PSD**
Show that for any matrix $Q \in \mathbb{R}^{m\times n}$, $A = Q^\top Q$ is positive semidefinite.

10. **Constrained Eigenvalue Optimization**
Let $A \in \mathbb{R}^{n\times n}$ be symmetric with eigenvalue-eigenvector pairs
$(\lambda_1, \vec{v}_1), \ldots, (\lambda_n, \vec{v}_n)$ where $\lambda_1 > \cdots > \lambda_n$. Consider
$$p^* = \max_{\vec{x}\in\mathbb{R}^n} \vec{x}^\top A\vec{x}$$
s.t. $\quad \|\vec{x}\|_2^2 = 1 \quad \vec{x}^\top\vec{v}_1 = 0.$
Show that $p^* = \lambda_2$.

E. Supporting mini lemma example

Problem statement:

```
theorem Problem4_ineq1
 {n : ℕ}
 (hn : 0 < n)
 (x : Fin n → ℝ) :
 1 / Real.sqrt ↑n * ‖WithLp.toLp 2 x‖ ≤ ‖WithLp.toLp
 ↪  ⊤ x‖ :=


---- AXIOMS START
-- @side
-- max xᵢ ≥ 0 when all xᵢ ≥ 0
theorem max_is_positive {n : ℕ} (x : Fin n → ℝ)
(h : ∀ i : Fin n, 0 ≤ x i) : 0 ≤ ⨆ i : Fin n, x i
↪  := by
 sorry

-- @side
-- ‖x‖₂ = √(∑ |xᵢ|²)
theorem def_l2_norm {n : ℕ} (x : Fin n → ℝ)
: ‖WithLp.toLp 2 x‖ = √(∑ i : Fin n, |x i| ^ 2)
↪  := by
 sorry

-- @side
-- ‖x‖∞ = max |xᵢ|
theorem def_linfty_norm {n : ℕ} (x : Fin n → ℝ)
: ‖WithLp.toLp ⊤ x‖ = ⨆ i : Fin n, |x i| := by
 sorry

-- @side
-- x ≤ y iff x² ≤ y² for non-negative x, y
theorem square_bs_ineq (x y : ℝ) (hx : 0 ≤ x)
(hy : 0 ≤ y) : x ≤ y ↔ x ^ 2 ≤ y ^ 2 := by
 sorry

-- @side
-- |x|² = x²
lemma abs_sq_eq_sq (x : ℝ) : |x| ^ 2 = x ^ 2
:= by
 sorry

-- @side
-- (max |xᵢ|)² = max |xᵢ|²
theorem move_squares_into_max {n: ℕ} (x : Fin n →
↪  ℝ)
: (⨆ i : Fin n, |x i|) ^ 2
= ⨆ i : Fin n, |x i| ^ 2  := by
 sorry

-- y/z ≤ x iff y ≤ x*z for positive z
theorem division_ineq (x y : ℝ)
{z : ℝ} (hz : 0 < z) : y / z ≤ x  ↔  y ≤ x * z := by
 sorry

-- x ≤ y implies √x ≤ √y for non-negative x, y
theorem sqrt_ineq (x y : ℝ)
(hx : 0 ≤ x) (hy : 0 ≤ y) : x ≤ y → √x ≤ √y :=
↪  by
 sorry

-- @side
-- sum of a constant is the constant times the
↪  number of elements in the sum
recall Finset.sum_const : ∀ {ι : Type u_1} {M : Type
↪  u_4} {s : Finset ι}
[AddCommMonoid M] (b : M), ∑ _x ∈ s, b = s.card • b

-- @side
-- ∑ c = n * c
theorem me_Finset_sum_const {n : ℕ}
(c : ℝ) : ∑ _ : Fin n, c = ↑n * c := by
 simp

-- fᵢ ≤ gᵢ for all i implies ∑ fᵢ ≤ ∑ gᵢ
theorem me_Finset_sum_le_sum {n : ℕ}
(f g : Fin n → ℝ) (h : ∀ i : Fin n, f i ≤ g i) :
↪  ∑ i : Fin n, f i ≤ ∑ i : Fin n, g i := by
 sorry

-- @side
recall le_ciSup : ∀ {α : Type u_1} {ι : Sort u_4}
↪  [ConditionallyCompleteLattice α]
{f : ι → α},
 BddAbove (Set.range f) → ∀ (c : ι), f c ≤ iSup f

-- fᵢ ≤ max fⱼ
theorem me_le_ciSup {n : ℕ}
(f : Fin n → ℝ) (i : Fin n) : f i ≤ ⨆ i : Fin n, f
↪  i := by
 sorry

-- @side
-- (1/√n)*√(n*x) = √x for positive n and
↪  non-negative x
theorem cancel_sqrt_n {n : ℕ}
(hn : 0 < n) (x : ℝ) (hx : 0 ≤ x) :
  1 / Real.sqrt n * Real.sqrt (n * x) = Real.sqrt x
  ↪  := by
 sorry

-- @side
-- 1 ≤ a*(b/c) iff c ≤ a*b for positive c
theorem mul_div_le_iff (a b c : ℝ)
```

```
(hc : 0 < c) : 1 ≤ a * (b / c) ↔ c ≤ a * b := by
  rw [div_eq_mul_inv, ← mul_assoc, le_mul_inv_iff₀
  ↪  hc, one_mul]

-- @side
-- ∑ 1 = n
theorem sum_one_over_n {n : ℕ} :
(∑ _ : Fin n, 1 = ↑n) := by
  simp

-- @side
recall norm_smul {α : Type u_1} {β : Type u_2}
[Norm α] [Norm β] [SMul α β] [NormSMulClass α β] (r
↪  : α) (x : β) :
  ‖r • x‖ = ‖r‖ * ‖x‖

-- @side
recall Real.norm_of_nonneg {r : ℝ}
(hr : 0 ≤ r) : ‖r‖ = r

-- @side
theorem one_div_sqrt_eq_sqrt_one_div (x : ℝ) :
  (1 / Real.sqrt x) = Real.sqrt (1 / x)
  := by
  rw [Real.sqrt_div]
  simp
  positivity

-- @side
theorem mul_one_div_sqrt (a n : ℝ) :
  a * (1 / Real.sqrt n) = a / Real.sqrt n := by
  field_simp

---- AXIOMS END
```

E.1. Rubric

These rubric items were crafted by the course instructor.

Second Principal Component (8 pts) *Shows that $\bar{x} = \bar{v}_2$ is achievable.*

1. +2 pts: Concludes that the feasible point $\bar{x} = \bar{v}_2$ attains $\bar{v}_2^\top A\bar{v}_2 = \bar{v}_2^\top \lambda_2 \bar{v}_2 = \lambda_2$.
2. +2 pts: Shows that $\bar{x} = \bar{v}_2$ is feasible: $\bar{v}_1^\top \bar{v}_2 = 0$ and $\|\bar{v}_2\|_2 = 1$.

Proves that $p^* \leq \lambda_2$.

1. +1 pt: Claims $\bar{x} \in \operatorname{span}(\bar{v}_2, \ldots, \bar{v}_n)$, e.g., $\bar{x} = \sum_{k=2}^n y_k \bar{v}_k$.
2. +1 pt: Substitutes into the objective and derives
$$\bar{x}^\top A\bar{x} = \sum_{i=2}^n \lambda_i y_i^2.$$
3. +1 pt: Uses
$$\lambda_i y_i^2 \leq \lambda_2 y_i^2 \qquad (i \geq 2).$$
4. +1 pt: Concludes
$$\bar{x}^\top A\bar{x} \leq \lambda_2 \sum_{i=2}^n y_i^2 = \lambda_2.$$
5. +0 pts: Incorrect but non-blank.
6. +0 pts: Blank.

F. Prompt for LLM evaluator

System prompt:

`You are an expert mathematics professor evaluating whether individual proof steps are correct given the proof question. Respond with valid JSON only, no extra text.`

User prompt:

QUESTION: question

STUDENT PROOF STEPS (OCR transcription, 1-indexed): steps

Evaluate the student's proof: - List the 1-based step indices of any mathematically INCORRECT steps in "incorrect_step" (use [] if every step is correct). - If "incorrect_step" is non-empty, set "incomplete" to "NA". - If "incorrect_step" is empty, judge whether the proof is INCOMPLETE — i.e. missing steps or cases needed to fully establish the result — and set "incomplete" to "Yes" or "No". - "reason": one concise explanation of your judgment.

Respond with JSON only: <"incorrect_step": [<ints>], "incomplete": "Yes"|"No"|"NA", "reason": "<text>">