Evaluating the Robustness of Proof Autoformalization in Lean 4

Zhengtao Gui [1] **Sheng Yang** [2] **Zhouxing Shi** [2]

Abstract

Proof autoformalization aims to translate a mathematical informal proof written in natural language into a formal proof in a formal language such as Lean 4. Several works have developed LLM-based models for proof autoformalization. However, existing evaluations have typically focused on translating well-formed informal proofs from curated datasets. We argue that a robust proof autoformalizer must remain faithful even for informal proofs that diverge from these idealized ones, and we present the first study on the robustness of proof autoformalization models. We formulate two categories of perturbations and evaluate robustness under each: a global perturbation paraphrases the informal proof in a different style, under which the formalization should remain consistent; a local perturbation alters a value, symbol, or proof step, possibly in a counterfactual way, and a robust formalization should faithfully reflect the perturbation rather than reverting to the original one or inferring a different one on its own. We build a benchmark with both perturbations on miniF2F and MATH-500, and automatically measure how stable a proof autoformalization's correctness is under global perturbations and how faithfully its output reflects local perturbations. We evaluate seven recent models, all of which are sensitive to global perturbations and mostly fail to remain faithful under local perturbations. Code and data are available via `https://github.com/ucr-rai/robust-proof-autoformalization`.

1. Introduction

Large language models (LLMs) have recently demonstrated strong capabilities in mathematical reasoning (Lightman et al., 2024; Guo et al., 2025), which is typically conducted in natural language; however, such natural-language reasoning is inherently challenging to verify reliably, and it is hard to detect subtle logical errors reliably. In contrast, formal theorem provers, such as Coq (Barras et al., 1999), Isabelle (Nipkow et al., 2002), and Lean 4 (Moura & Ullrich, 2021), represent mathematical statements and proofs in a rigorous formal language, enabling formal verification of whether a proof is correct for a given statement.

Given the strong capabilities of LLM reasoning developed in the natural-language (NL) space and the formal reliability provided by the formal-language (FL) space, bridging NL and FL reasoning has emerged as an active research direction. The bridge can run in both directions: NL reasoning can guide the generation of FL proofs (Jiang et al., 2023b; Ren et al., 2025; Wang et al., 2025), while FL verification can in turn verify NL reasoning once it is faithfully translated into FL (Liu et al., 2025a; Yao et al., 2025b; Jana et al., 2025; Cabral et al., 2025).

The translation from NL mathematics into FL is known as *autoformalization*, which can be categorized into statement autoformalization and proof autoformalization, depending on whether only the statement or also the proof is translated. Most of the previous work has focused on statement autoformalization (Gao et al., 2024; Jiang et al., 2024; Poiroux et al., 2025; Wang et al., 2025; Wu et al., 2026). Proof autoformalization, which is substantially more challenging, has gained attention only recently (Jana et al., 2025; Cabral et al., 2025) and is the focus of this work. However, these methods are typically only trained and tested on well-formed proofs from standard datasets, and it remains unclear how robust they are when the input deviates from idealized proofs.

In this work, we present the first study on the robustness of proof autoformalization. Specifically, we evaluate robustness under two categories of perturbations applied to the NL input. First, when the NL proof is correct but exhibits stylistic variations, the resulting FL proof should remain correct. This robustness is crucial for a proof autoformalization model to remain effective in practice across NL proofs of varying styles and quality. We thus design *global perturbations*, which paraphrase the entire NL proof in different styles while preserving the mathematical semantics. Second, when a subtle change is introduced into the NL proof — even a counterfactual one that represents an error — the

[1]University of California, Irvine [2]University of California, Riverside. Correspondence to: Zhouxing Shi <zhouxing.shi@ucr.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).

autoformalization should faithfully reflect this change in the FL proof rather than inferring a different one on its own, for instance by silently correcting the error. This robustness lays the groundwork for future work that may combine autoformalization with Lean-based formal verification to verify NL proofs and detect errors in them, so that NL reasoning can in turn be strengthened by feedback from the FL side. We therefore design *local perturbations*, which modify a single value, symbol, or proof step in the NL proof, possibly in a counterfactual way.

We summarize our contributions as follows:

- We present the first study on the robustness of proof autoformalization, designing global and local perturbations with accompanying robustness metrics to assess properties that a robust autoformalization model should satisfy.
- We build a benchmark that instantiates these perturbations on miniF2F (Zheng et al., 2021) and MATH-500 (Lightman et al., 2024). We evaluate the robustness of a variety of proof autoformalization methods including latest ones such as ProofBridge (Jana et al., 2025) and ProofFlow (Cabral et al., 2025).
- Our experiments reveal that all existing proof autoformalization methods exhibit limited robustness, typically suffer from unstable correctness under global perturbations and limited faithfulness under local perturbations, indicating substantial room for improvement.

2. Related Work

Formal Theorem Proving. Formal theorem proving aims to generate an FL proof for a given FL statement, such that the generated proof can be rigorously verified by a proof assistant such as Lean 4, which guarantees that the proof is correct for the given statement. Existing methods generally fall into two paradigms. The first is next-tactic prediction, which generates one proof step (tactic) at a time conditioned on the current proof state, with representative systems including GPT-f (Polu & Sutskever, 2020), LISA (Jiang et al., 2021), and PACT (Han et al., 2021). The second is whole-proof generation, which produces a complete FL proof in a single pass. Powered by recent advances in LLMs, a number of whole-proof generation models have been developed and have achieved strong performance, including DeepSeek-Prover-V1 (Xin et al., 2024), which uses expert iteration; TheoremLlama (Wang et al., 2024), which adopts curriculum training; DeepSeek-Prover-V2 (Ren et al., 2025), which integrates NL-guided reasoning with formal proof generation; Kimina-Prover (Wang et al., 2025), which uses compilation-based RL rewards (Jana et al., 2023); and Goedel-Prover-V2 (Lin et al., 2025), which combines scaffolded data synthesis, verifier-guided self-correction, and model averaging. The primary focus of all these methods is on generating correct FL proofs, while we focus on bridging NL and FL proofs so that the generated FL proof faithfully reflects the given NL proof.

Autoformalization. Autoformalization translates informal language into formal language and is broadly divided into statement and proof autoformalization. Statement autoformalization is relatively more established. Traditional rule-based autoformalization is hard to implement and generalize (Jiang et al., 2023a). Thus, recent work mainly focuses on LLM-based methods, including in-context learning (Wu et al., 2022), back-translation-based data synthesis (Jiang et al., 2023a; Azerbayev et al., 2023; Wu et al., 2025), retrieval-augmented generation (Liu et al., 2025b), natural language inference (Ying et al., 2024), expert iteration with LLM judges (Wang et al., 2025), and dataset construction with translator fine-tuning, as in Herald-translator (Gao et al., 2024).

Proof autoformalization has received relatively limited attention and is a more recent research direction. FormL4 (Lu et al., 2024) trains on GPT-4-informalized Mathlib proofs with step-wise Lean feedback, while ProofBridge (Jana et al., 2025) uses NL-FL representation learning and cross-modal retrieval to guide proof formalization. Additionally, some recent methods perform proof autoformalization step by step, such as StepProof (Hu et al., 2025) using Isabelle, and ProofFlow (Cabral et al., 2025), which preserves proof structure via dependency DAGs and intermediate lemmas. In this work, we focus on evaluating the robustness of existing proof autoformalization methods in Lean 4.

Metrics for autoformalization. For proof autoformalization, recent evaluations introduce proof-level metrics. ProofBridge (Jana et al., 2025) evaluates the correctness of generated formalizations using Type Correctness (TC) and Semantic Correctness (SC). TC is determined by whether the generated FL passes Lean's type checker, whereas SC is computed only for type-correct FL and measures whether the generated FL is semantically equivalent to the gold-standard FL. ProofFlow (Cabral et al., 2025) proposes *proof-score*, a unified metric that evaluates proof formalization through syntactic correctness, semantic faithfulness, and structural fidelity. It checks whether formalized steps compile, whether they match the corresponding natural-language steps, and whether their proof-step dependencies are preserved by a DAG structure. These metrics, evaluated on well-formed NL proofs alone, do not assess the robustness of proof autoformalization, which we investigate in this work.

3. Method

We study the robustness of proof autoformalization models in Lean 4. For a target model f, the task is to translate a natural-language (NL) theorem-proof pair $M_{\mathrm{NL}} = \langle T_{\mathrm{NL}}, P_{\mathrm{NL}} \rangle$ into a formal-language (FL) pair $\widehat{M}_{\mathrm{FL}} = \langle \widehat{T}_{\mathrm{FL}}, \widehat{P}_{\mathrm{FL}} \rangle$:

$$\widehat{M}_{\mathrm{FL}} = f(M_{\mathrm{NL}}),$$

where T_{NL} and P_{NL} denote the NL theorem statement and proof, and $\widehat{T}_{\mathrm{FL}}$ and $\widehat{P}_{\mathrm{FL}}$ denote their FL counterparts. We investigate the robustness under two categories of perturbations applied to the NL input. A *global* perturbation paraphrases the whole NL proof P_{NL}, and we evaluate robustness to it by measuring the correctness of the resulting formalization (§3.1); a *local* perturbation changes a single value, symbol, or proof step within the NL input, and we evaluate faithfulness by measuring whether the formalization faithfully reflects the change (§3.2).

3.1. Global Perturbations

For global perturbations, we use LLMs to paraphrase the entire NL proof. Specifically, we use two LLMs: Gemini-2.5-Flash (Google, 2025) (**G**), representing a closed-source model, and Qwen3.5-397B-A17B (Qwen Team, 2026) (**Q**), representing an open-source model. Given the original NL proof, denoted as **Orig**, we consider two paraphrasing modes: *free-form paraphrasing* (**-FF**) uses an LLM to restate the NL proof in its own wording and style while preserving the mathematical meaning and overall logic; and *step-by-step paraphrasing* (**-Step**) uses an LLM to reorganize the NL proof into one with a clearly numbered sequence of steps. NL proofs paraphrased with these two modes can represent stylistic variations in NL proofs encountered in real-world scenarios. For example, an original proof may write:

> **Orig:** Since $x^2 - 5x + 6 = (x-2)(x-3)$, the roots are 2 and 3.

A free-form paraphrase may put the same idea in different wording:

> **FF:** Factoring the quadratic shows that it vanishes exactly at $x = 2$ and $x = 3$.

A step-by-step paraphrase instead makes the derivation explicit:

> **Step:**
>
> 1. Factor $x^2 - 5x + 6 = (x-2)(x-3)$.
> 2. Set each factor to zero.
> 3. Thus $x = 2$ or $x = 3$.

All three versions prove the same claim, but they expose the autoformalizer to different wording, levels of detail, and proof organization. Using Gemini-2.5-Flash and Qwen3.5-397B-A17B, respectively, we generate four types of globally perturbed NL proofs: **G-FF**, **Q-FF**, **G-Step**, and **Q-Step**. Together with **Orig**, this yields five NL proof styles per problem. Details are in Appendix A.2.

Correctness Metrics. Since these global perturbations are expected to preserve the mathematical meanings, a robust proof autoformalization should achieve comparable correctness across all five styles. We therefore measure robustness by the stability of correctness across styles: large variation across styles indicates sensitivity to surface wording rather than the underlying mathematics. As shown in Figure 1, each generated FL output is evaluated by three complementary correctness metrics: **Type Correctness (TC)** checks whether the generated FL output compiles, which formally guarantees the correctness of the FL proof for the given FL statement. **Statement Semantic Correctness (StmtSC)** and **Proof Semantic Correctness (ProofSC)** assess whether the generated formal statement and formal proof, respectively, semantically matches their NL counterparts. **FullyCorrect** is satisfied only when all three metrics are satisfied simultaneously: Type Correctness, Statement Semantic Correctness, and Proof Semantic Correctness. For scoring, TC is checked by the Lean compiler, while StmtSC and ProofSC are assigned by an LLM judge; we detail the judging procedure in Appendix A.3.3 and audit its reliability in Appendix A.5.

3.2. Local Perturbations

A local perturbation changes a single value, symbol, or proof step in the NL input. We deliberately allow such perturbations to be possibly counterfactual, in order to reveal the different behaviors of two types of autoformalizers. A **Faithful Autoformalizer** produces a Lean formalization that faithfully reflects the NL input it is given, even in the presence of counterfactual edits, which is the desired behavior for robust autoformalization. In contrast, a **Biased Autoformalizer** produces an autoformalization that silently overrides the edit with its own inference. This occurs because a spurious bias acquired during training may lead the model to fill in seemingly plausible content based on the FL statement, the context of the FL proof, and patterns seen in well-formed proofs, rather than faithfully following the actual NL input. For example, given a proof "from $2x + 3 = 17$ we get $2x = 14$, so $x = 7$", we may change the intermediate step to $2x = 16$, which no longer holds arithmetically. A Faithful Translator should carry this step into the FL proof and produce `2 * x = 16`, whereas a Biased Autoformalizer may revert to 14 on its own and produce `2 * x = 14`.

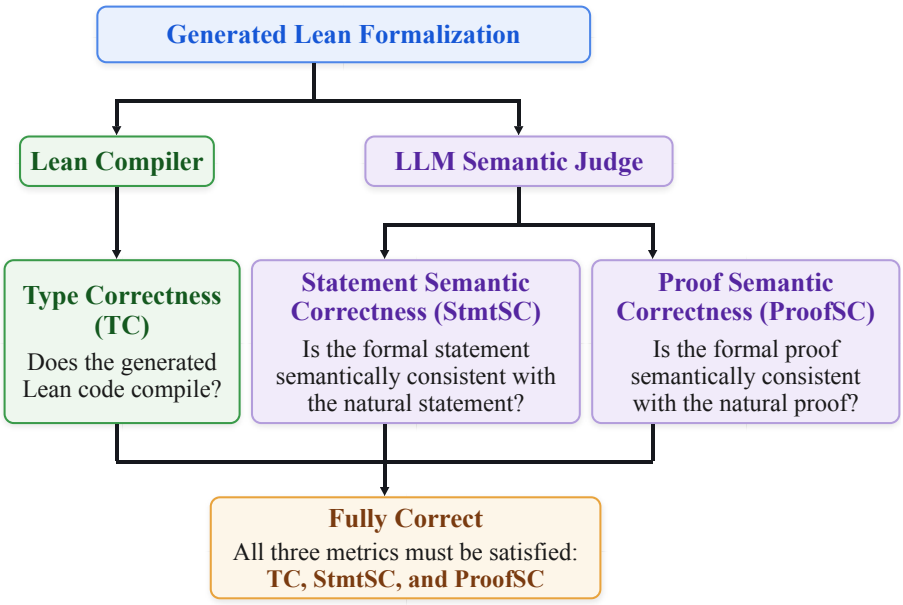

Figure 1. Correctness metrics pipeline for global perturbations. Type Correctness (TC) is checked by the Lean compiler, while Statement Semantic Correctness (StmtSC) and Proof Semantic Correctness (ProofSC) are evaluated by an LLM judge. A generated formalization is counted as FullyCorrect only when all three conditions are satisfied.

We design three specific types of edits for the local perturbations. A **number edit** changes a numeric literal to a different value of similar magnitude and the same sign; a **symbol edit** replaces a relation or operator with its opposite (e.g. $+$ with $-$, or $\leq$ with $\geq$); and a **step deletion** removes the justification of one substantive proof step while keeping its stated outcome, so the proof asserts the step's result without proving it. For example, consider a proof step that derives a lower bound:

> Since $(x-3)^2 \geq 0$, expanding gives $x^2-6x+9 \geq 0$. Hence $x^2 - 6x + 10 \geq 1$.

A number edit may change the final bound from 1 to 2. A symbol edit may change the relation $x^2 - 6x + 10 \geq 1$ to $x^2 - 6x + 10 \leq 1$, or change the operator in $x^2 - 6x + 10$ from $+$ to $-$. A step deletion may remove the justification "Since $(x - 3)^2 \geq 0$, expanding gives $x^2 - 6x + 9 \geq 0$" while keeping the stated outcome "Hence $x^2 - 6x + 10 \geq 1$." In all three cases, the edited proof may no longer be mathematically valid, but a faithful formalization should still reflect the edited text rather than silently repairing it. This is particularly important when proof autoformalization is applied to verify an NL proof—possibly incorrect—by checking its autoformalized counterpart in a formal proof assistant such as Lean.

Number and symbol edits are applied in both the proof and the statement; step deletion applies only to the proof. The statement edits act as a control against which the proof-level results can be read. To build an edit, an LLM (Gemini-2.5-Flash) first marks the eligible locations in the input: numeric literals, relations and operators, and substantive proof steps. We keep only locations that pass rule-based validity checks, discarding cases that would render the problem malformed, such as numbers inside subscripts or superscripts and relations inside chained expressions like $0 < p < 15$. From the valid locations, we choose one for each of the statement and the proof. For a number or symbol edit, this is a single-span substitution at a marked position; for a step deletion, we instead rank the eligible steps by a quality score and delete the justification of the top-ranked one. Appendix A.2 gives the full marking, filtering, and ranking rules, and Table 1 reports the per-type edit counts.

Faithfulness Metrics. For local perturbations, we classify each output as faithful, reverted, or unclear: a *faithful* output reflects the edited input, a *reverted* output matches the original unedited input, and an *unclear* output matches neither (for example, it is empty, fails to compile, or never mentions the edited entity). This label is an objective property of the output rather than a subjective judgment: it is determined entirely by which value the output carries (the edited one, the original, or neither). We assign these labels with an LLM judge; such labels can be easily inspected by humans, and we audit the labeling reliability in Appendix A.5. From these labels we compute the **Faithful Rate (FR)** and **Reverted Rate (RR)** as the fractions of the identifiable (faithful or reverted) outputs, so that $\mathrm{FR} + \mathrm{RR} = 1$; a

Table 1. Dataset statistics for local perturbations, reporting the number of statement-level / proof-level edit instances on miniF2F and MATH-500.

	miniF2F	MATH-500
Number-edit	227 / 241	490 / 489
Symbol-edit	130 / 195	195 / 349
Step-deletion	– / 129	– / 192

higher FR marks a Faithful Autoformalizer and a higher RR indicates a Biased Autoformalizer. The unclear outputs are excluded from FR/RR, and we report their share as the **Other/Unclear Rate (OUR)**, analyzing its composition with examples in Appendix A.4.

4. Experiments

We first describe the common experimental setup (§4.1), then report two experiments: one measuring robustness to global perturbations through the stability of correctness (§4.2), and one measuring faithfulness to local perturbations (§4.3). Finally, we validate that the perturbations themselves are correctly constructed (§4.4).

4.1. Setup

Datasets. We build our evaluation based on two mathematical reasoning datasets: miniF2F (Zheng et al., 2021) and MATH-500 (Lightman et al., 2024). miniF2F contains 244 olympiad-style problems drawn from AIME, AMC, and IMO; MATH-500 contains 500 problems from the MATH benchmark (Hendrycks et al., 2021). MATH-500 problems are originally posed as computational problems that require producing concrete answers; since a Lean statement must be a theorem that requires a proof rather than a computational answer, we append each gold answer to its statement as a "Show that it is X" clause, turning the problem into a closed proposition requiring a proof, as similarly done in prior work using Lean for answer selection (Yao et al., 2025a; Liu et al., 2025a).

For global perturbations, each of the problems appears in five styles, which gives 3,720 problem-style instances in total. For local perturbations, we generate one edit per problem for each applicable combination of edit type and location: number and symbol edits in both the statement and the proof, and step deletion in the proof only. An edit is created only where an eligible location exists, and thus the counts vary by type and dataset. Table 1 shows the full breakdown, totaling 2,637 local perturbations.

Models. We evaluate seven models in total, which we organize into two groups. **(i) Repurposed models.** Kimina-Prover-RL-1.7B and Kimina-Prover-Distill-8B (Wang et al., 2025), DeepSeek-Prover-V2-7B (Ren et al., 2025), and Goedel-Prover-V2-8B (Lin et al., 2025) are specialized theorem provers in Lean 4, and they are trained to generate a formal proof for a given formal statement. Although these are specialized models for proof generation rather than autoformalization, given the scarcity of proof autoformalization models, we follow Jana et al. (2025) and repurpose these models for proof autoformalization with new prompts. Additionally, we adopt Gemini-3.1-Pro which is a general-purpose LLM and can thus be prompted to perform proof autoformalization. **(ii) Specialized proof autoformalization models.** ProofBridge and ProofFlow are the two models in our evaluation built specifically for proof autoformalization, in contrast to the repurposed provers and the general-purpose LLM above. **ProofBridge** (Jana et al., 2025) is trained by Supervised Fine-tuning (SFT) from Kimina-Prover-RL-1.7B on proof autoformalization data. Although ProofBridge is enhanced with a retrieval-augmented approach which guides the autoformalization with similar cases from the database, this component has not been included in their released code; we thus focus on the SFT model in this work. Nevertheless, we do not expect the retrieval-augmented approach to improve the robustness we study, as it uses extra references from the database rather than the current NL input. **ProofFlow** (Cabral et al., 2025) is a formalization pipeline designed to preserve proof structure. It constructs a directed acyclic graph (DAG) of proof-step dependencies, formalizes intermediate lemmas, and then attempts to complete the proof through these lemmas. We use each model's own standard decoding settings (Appendix A.1) without performing any per-model hyperparameter search of our own. All models generate a single output per input; we do not consider multiple samples per input (a.k.a. Pass@k), given the computational cost of evaluation across various perturbations.

All semantic and faithfulness metrics (StmtSC, ProofSC, FR/RR/OUR) are assigned by Gemini-2.5-Flash, so that cross-model comparisons are not confounded by judge differences. Type checking uses Lean 4 (Moura & Ullrich, 2021) with a 900-second per-problem timeout. Details are documented in Appendix A.1.

4.2. Robustness to Global Perturbations

Table 2 reports FullyCorrect on the original input and four semantic-preserving rewrites. A robust autoformalizer should achieve comparable FullyCorrect across columns, but all the models evaluated show sensitivity to NL proof presentation. On miniF2F, the variation can be large: ProofBridge drops from 21.3% on Orig to 17.6% under both Q-FF and Q-Step, while DeepSeek-Prover-V2-7B rises from 1.2% on Orig to 12.7% on Q-FF. Gemini-3.1-Pro is the strongest model in aggregate on both datasets, yet its miniF2F score still ranges from 17.2% to 22.5% across styles.

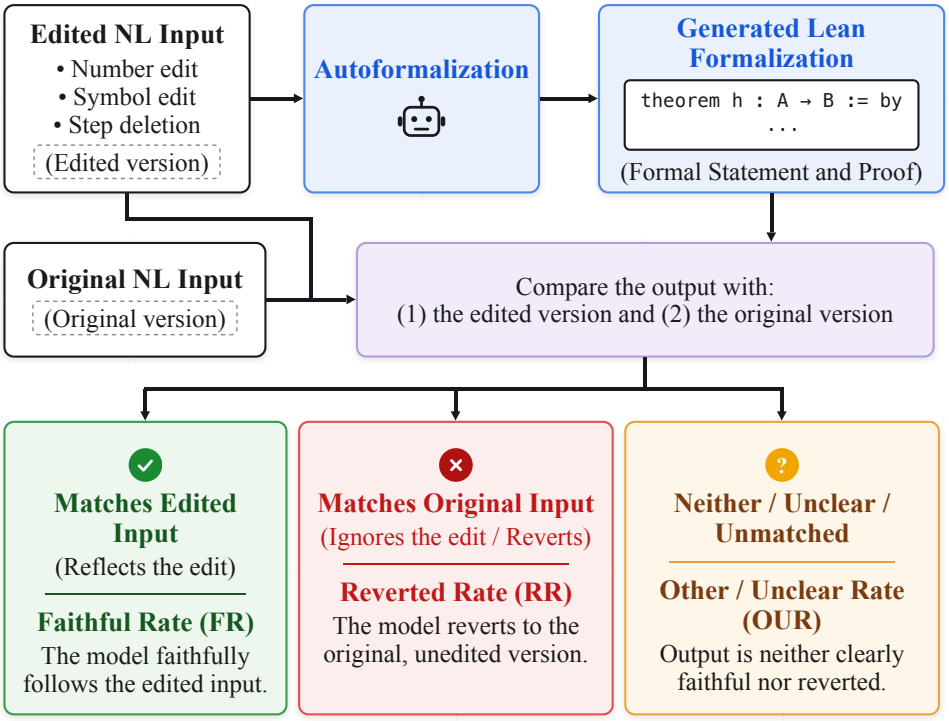

Figure 2. Overview of the faithfulness metrics for local perturbations. We compare each Lean output against both the edited and original input and label it *faithful* (matches the edit), *reverted* (matches the original), or *unclear* (neither). The Faithful (FR) and Reverted (RR) Rates are the shares of *identifiable* (faithful or reverted) outputs that are faithful and reverted; the Other/Unclear Rate (OUR) is the unclear share of all outputs.

Rewriting has mixed effects across models and datasets. On miniF2F, rewrites can substantially improve some models: DeepSeek-Prover-V2-7B increases from 1.2% on Orig to 12.7% on Q-FF, Goedel-Prover-V2-8B from 1.2% to 4.1%, and Gemini-3.1-Pro from 17.2% to 22.5%. On MATH-500, however, the same kind of rewriting often lowers performance: for example, Kimina-Prover-Distill-8B drops from 19.8% on Orig to 16.4–19.0% on the rewrites, and Gemini-3.1-Pro drops from 26.4% to 23.6–24.6%. Thus, rewriting on its own is neither uniformly helpful nor uniformly harmful, and its effect depends on both the dataset and the model.

Overall, no input style dominates uniformly, and semantic-preserving rewrites can change both absolute scores and model rankings. For example, on miniF2F, ProofBridge is strongest on Orig, whereas Gemini-3.1-Pro is strongest on all four rewritten styles. This indicates that current autoformalizers are not stable under global perturbations with paraphrased NL inputs, and that evaluations solely on a fixed original style can substantially over- or under-state a model's true autoformalization ability.

4.3. Faithfulness under Local Perturbations

We use the three local perturbations of §3.2, each testing a specific faithful behavior. Tables 3 and 4 summarize FR (the higher the better) and RR (the lower the better) across proof-level and statement-level local perturbations, respectively.

Models are least faithful on proof-level number and symbol edits. For the five models other than ProofBridge and ProofFlow, the average FR on these edits is only 6.3%. RR is above 90% in almost all corresponding cells. For example, on miniF2F number-proof edits, Kimina-Prover-Distill-8B has 94.7% RR and Gemini-3.1-Pro has 95.9% RR. ProofBridge and ProofFlow, which are specialized proof autoformalization models, are relatively more faithful on proof-level number and symbol edits. ProofBridge has 41.8% average FR on these edits, and ProofFlow has 26.1%, both higher than the 6.3% average of the other five models. However, this advantage is still limited: even for these proof-level edits, both systems still revert in more than half of identifiable cases. Such reversion is consistent with the bias described in §3.2. A proof-level edit is only a local change, with the statement and most of the proof context unchanged. The input overall resembles a well-formed proof, and thus a Biased Autoformalizer tends to produce a plausible FL proof that is consistent with the statement and the proof

Table 2. Robustness of proof autoformalization models under global perturbations, measured by FullyCorrect (%) on miniF2F and MATH-500 across five styles of natural-language proof inputs.

Model	**miniF2F**					**MATH-500**				
	Orig	G-FF	G-Step	Q-FF	Q-Step	Orig	G-FF	G-Step	Q-FF	Q-Step
Kimina-Prover-RL-1.7B	9.4	10.2	11.5	11.5	8.6	6.6	8.2	6.8	8.2	9.6
Kimina-Prover-Distill-8B	19.3	16.0	16.4	19.7	16.0	19.8	16.4	16.8	16.8	19.0
DeepSeek-Prover-V2-7B	1.2	8.6	7.4	12.7	8.6	8.2	7.4	5.8	6.4	5.0
Goedel-Prover-V2-8B	1.2	2.5	4.1	3.7	4.1	4.4	2.0	2.0	3.8	3.0
Gemini-3.1-Pro	17.2	21.7	18.9	21.7	22.5	26.4	23.6	24.6	23.8	23.8
ProofBridge	21.3	19.7	18.4	17.6	17.6	13.4	14.4	12.6	14.8	15.2
ProofFlow	9.0	6.1	8.2	7.0	9.8	10.8	8.2	8.0	8.4	13.0

Table 3. FR/RR results (%) for proof-level local perturbations, computed over identifiable outputs (faithful or reverted). Unclear outputs that neither faithful nor reverting are excluded; their fraction is reported in the appendix.

Model	**Number edit**				**Symbol edit**				**Step deletion**			
	miniF2F		MATH-500		miniF2F		MATH-500		miniF2F		MATH-500	
	FR (↑)	RR (↓)	FR (↑)	RR (↓)	FR (↑)	RR (↓)	FR (↑)	RR (↓)	FR (↑)	RR (↓)	FR (↑)	RR (↓)
Kimina-Prover-RL-1.7B	7.3	92.7	5.2	94.8	3.6	96.4	8.1	91.9	51.8	48.2	70.7	29.3
Kimina-Prover-Distill-8B	5.3	94.7	2.9	97.1	8.3	91.7	3.1	96.9	25.9	74.1	51.3	48.7
DeepSeek-Prover-V2-7B	8.3	91.7	6.4	93.6	7.9	92.1	8.5	91.5	48.4	51.6	53.6	46.4
Goedel-Prover-V2-8B	10.1	89.9	6.6	93.4	2.6	97.4	9.2	90.8	43.7	56.3	60.9	39.1
Gemini-3.1-Pro	4.1	95.9	6.3	93.7	8.3	91.7	4.1	95.9	47.4	52.6	56.9	43.1
ProofBridge	39.4	60.6	41.4	58.6	44.9	55.1	41.4	58.6	33.7	66.3	53.8	46.2
ProofFlow	32.6	67.4	24.4	75.6	27.0	73.0	20.2	79.8	24.8	75.2	33.8	66.2

context, while silently overriding the local edit.

Step deletion shows a different pattern. Its FR is much higher than for proof-level number and symbol edits, but reversion remains frequent. For example, on miniF2F, step-deletion RR ranges from 48.2% to 75.2%, indicating that models still often fill in the removed reasoning step on their own. The model with the highest FR on step deletion is Kimina-Prover-RL-1.7B, with 51.8% FR on miniF2F and 70.7% on MATH-500, though this may partly reflect its limited ability to infer missing reasoning owing to its smaller size rather than genuinely better faithfulness.

Statement edits are much easier for models to follow than proof edits. For every model, FR is higher when the same type of edit is made in the statement rather than in the proof. For example, ProofBridge goes from 39.4%/44.9% FR on miniF2F proof-level number/symbol edits to 78.6%/80.0% on statement-level edits. Gemini-3.1-Pro also rises from 4.1%/8.3% to 22.8%/23.3%. This is consistent with the bias described above: since a Biased Autoformalizer tends to produce an FL proof consistent with the statement, an edit in the statement is naturally more likely to be reflected in the output.

4.4. Perturbation Reliability

To ensure the validity of the results under constructed perturbations, we carefully checked the quality of the perturbations. A local edit should change only the targeted number, symbol, or step, leaving the rest of the input untouched. We verify this with an automated script that compares every edited input against its original character by character: in all 2,637 cases the two differ in exactly one contiguous span (the intended edit) and are otherwise identical (100%). For global perturbations, an automated script check finds that 96.4% of all 2,976 rewrites preserve every numeric and symbolic token of the original, with the remaining 3.6% having formatting differences (digit grouping, numbers spelled out, and similar) but not content changes. A manual audit of a stratified sample of 100 (original, rewrite) pairs, covering the four rewrite styles across both datasets, confirms that each of the edits changes only the wording while preserving all values, claims, reasoning steps, and semantic meanings. The detailed procedure is given in Appendix A.5.

5. Conclusion

This work studies the robustness of proof autoformalization under global paraphrases and local edits. Our results on

Table 4. FR/RR results (%) for statement-level local perturbations, computed over identifiable outputs (faithful or reverted). Unclear outputs are excluded; their fraction is reported in the appendix.

Model	**Number edit**				**Symbol edit**			
	miniF2F		MATH-500		miniF2F		MATH-500	
	FR	RR	FR	RR	FR	RR	FR	RR
Kimina-Prover-RL-1.7B	82.9	17.1	62.4	37.6	90.4	9.6	82.8	17.2
Kimina-Prover-Distill-8B	75.0	25.0	49.1	50.9	83.1	16.9	67.2	32.8
DeepSeek-Prover-V2-7B	66.3	33.7	48.1	51.9	73.8	26.2	69.4	30.6
Goedel-Prover-V2-8B	58.9	41.1	44.6	55.4	73.1	26.9	59.3	40.7
Gemini-3.1-Pro	22.8	77.2	14.2	85.8	23.3	76.7	26.9	73.1
ProofBridge	78.6	21.4	57.4	42.6	80.0	20.0	70.8	29.2
ProofFlow	52.8	47.2	49.3	50.7	64.8	35.2	57.7	42.3

seven recent models reveal two types of robustness failures: under global perturbations, the models are unstable when exposed to semantic-preserving changes in NL proof inputs; under local perturbations, they often fail to faithfully reflect local edits. Our findings suggest that proof autoformalization should be evaluated beyond correctness on well-formed proofs from standard datasets; robustness to NL proofs of varying styles and quality should be considered in future development to build reliable proof autoformalizers that can enable broader applications.

Acknowledgement

This work is supported in part by an NVIDIA Academic Grant award.

This work used the Delta system at the National Center for Supercomputing Applications [award OAC 2005572] through allocation CIS250704 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296.

This work used resources available through the National Research Platform (NRP) at the University of California, San Diego. NRP has been developed, and is supported in part, by funding from National Science Foundation, from awards 1730158, 1540112, 1541349, 1826967, 2112167, 2100237, and 2120019, as well as additional funding from community partners.

Limitations

Our study has several limitations that suggest directions for future work. First, the evaluated datasets consist of mathematical competition problems and undergraduate-level mathematics, which provide a controlled setting; we have not yet extended the study to more advanced domains. Second, we target only the Lean 4 proof assistant; extending the benchmark to other systems such as Isabelle and Coq is also valuable. Third, we study three local perturbation types (number, symbol, and proof-step edits) and two global rewriting styles; a wider range of edits and styles could be explored. Lastly, our perturbations are generated independently of the models being evaluated, but stronger ones may be generated adversarially with respect to the models. We leave these extensions to future work.

References

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.

Barras, B., Boutin, S., Cornes, C., Courant, J., Coscoy, Y., Delahaye, D., de Rauglaudre, D., Filliâtre, J.-C., Giménez, E., Herbelin, H., et al. The coq proof assistant reference manual. *INRIA, version*, 6(11):17–21, 1999.

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.

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.

Goedel-LM. Goedel-formalizer-v2-8b. https://huggingface.co/Goedel-LM/Goedel-Formalizer-V2-8B, 2025a. Hugging Face model card.

Goedel-LM. Goedel-prover-v2-8b. https://huggingface.co/Goedel-LM/Goedel-Prover-V2-8B, 2025b. Hugging Face model card.

Google. Gemini 2.5 flash. https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-5-flash, 2025. Google Cloud model documentation.

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

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

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS Datasets and Benchmarks)*, 2021.

Hu, X., Zhou, Q., Grechuk, B., and Tyukin, I. Y. Stepproof: Step-by-step verification of natural language mathematical proofs. *arXiv preprint arXiv:2506.10558*, 2025.

Jana, P., Jha, P., Ju, H., Kishore, G., Mahajan, A., and Ganesh, V. Cotran: An llm-based code translator using reinforcement learning with feedback from compiler and symbolic execution. *arXiv preprint arXiv:2306.06755*, 2023.

Jana, P., Kale, K., Tanriverdi, A. E., Song, C., Vishwanath, S., and Ganesh, V. Proofbridge: Auto-formalization of natural language proofs in lean via joint embeddings. *arXiv preprint arXiv:2510.15681*, 2025.

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

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

Jiang, A. Q., Welleck, S., Zhou, J. P., Lacroix, T., Liu, J., Li, W., Jamnik, M., Lample, G., and Wu, Y. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *International Conference on Learning Representations*, 2023b.

Jiang, A. Q., Li, W., and Jamnik, M. Multi-language diversity benefits autoformalization. *Advances in Neural Information Processing Systems*, 37:83600–83626, 2024.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 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, C., Yuan, Y., Yin, Y., Xu, Y., Xu, X., Chen, Z., Wang, Y., Shang, L., Liu, Q., and Zhang, M. Safe: Enhancing mathematical reasoning in large language models via retrospective step-aware formal verification. *arXiv preprint arXiv:2506.04592*, 2025a.

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*, 2025b.

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*, 2024.

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

Nipkow, T., Wenzel, M., and Paulson, L. C. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer, 2002.

Poiroux, A., Weiss, G., Kunčak, V., and Bosselut, A. Reliable evaluation and benchmarks for statement autoformalization. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 17958–17980, 2025.

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

Qwen Team. Qwen3.5-397b-a17b. https://huggingface.co/Qwen/Qwen3.5-397B-A17B, 2026. Hugging Face model card.

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.

Wang, H., Unsal, M., Lin, X., Baksys, M., Liu, J., Santos, M. D., Sung, F., Vinyes, M., Ying, Z., Zhu, Z., et al. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*, 2025.

Wang, R., Zhang, J., Jia, Y., Pan, R., Diao, S., Pi, R., and Zhang, T. Theoremllama: Transforming general-purpose

llms into lean4 experts. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 11953–11974, 2024.

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.

Wu, Y., Huang, D., Shi, W., Wang, W., Pu, Y., Gao, L., Liu, S., Nan, Z., Yuan, K., Zhang, R., et al. Inverse-coder: Self-improving instruction-tuned code llms with inverse-instruct. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 25525–25533, 2025.

Wu, Y., Huang, D., Wan, R., Peng, Y., Shang, S., Cao, C., Qi, L., Zhang, R., Zhang, X., Du, Z., et al. Stepfun-formalizer: Unlocking the autoformalization potential of llms through knowledge-reasoning fusion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33980–33988, 2026.

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*, 2024.

Yao, J., Wang, R., and Zhang, T. FANS: Formal answer selection for LLM natural language math reasoning using Lean4. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 3181–3200. Association for Computational Linguistics, 2025a. doi: 10.18653/v1/2025.emnlp-main.158. URL https://aclanthology.org/2025.emnlp-main.158/.

Yao, J., Wang, R., and Zhang, T. Fans: Formal answer selection for llm natural language math reasoning using lean4. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 3181–3200, 2025b.

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.

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

A. Appendix

A.1. Implementation Details

Inference. All models run zero-shot from one fixed autoformalization prompt, with no hyperparameter search. We serve the open-weight single-stage models (Kimina-Prover-RL-1.7B, Kimina-Prover-Distill-8B, ProofBridge, DeepSeek-Prover-V2-7B, and Goedel-Prover-V2-8B) with vLLM, generating one sample per input under our Pass@1 evaluation setting. Gemini-3.1-Pro is queried through its API.

For ProofFlow, we use the released dependency-DAG pipeline but adapt its component models and output interface to our evaluation. The graph builder is Gemini-2.5-Flash, and the lemma formalizer and tactic completer are `Goedel-LM/Goedel-Formalizer-V2-8B` (Lin et al., 2025; Goedel-LM, 2025a) and `Goedel-LM/Goedel-Prover-V2-8B` (Lin et al., 2025; Goedel-LM, 2025b). We keep DAG following, previous formal context, and supplying the original informal proof enabled. We use three retries for graph construction, lemma formalization, and proving, and set the internal ProofFlow pass budget to three; this is smaller than the retry budget used in some released ProofFlow benchmark scripts, which use more attempts for formalization and proving. To avoid giving ProofFlow credit for partial intermediate progress, our adapter exports only verified final Lean theorems to the downstream TC/SC pipeline. If ProofFlow formalizes intermediate lemmas but does not verify a final theorem, we treat the output as an error. We do not report ProofFlow's original ProofScore; instead, all models are evaluated with the same TC, StmtSC, ProofSC, FullyCorrect, FR, RR, and OUR metrics.

Compute budget. All the main experiments runs on a cluster with NVIDIA A100 (40/80 GB), H200, and A40 GPUs. We estimate the project's total cost at about 1,900 GPU-hours. Roughly half is LLM inference: the five open-weight models served with vLLM, plus the multi-stage ProofFlow pipeline (a graph model, a formalizer, and a solver with intermediate Lean checking), which is by far the most expensive per problem. The rest is Lean type-checking on GPU nodes, together with development runs. Gemini-3.1-Pro and the Gemini-2.5-Flash judge run through a hosted API and are not in this GPU total.

LLM judge. We use Gemini-2.5-Flash (Google, 2025) (model identifier `gemini-2.5-flash`) to assign StmtSC, ProofSC, and FR/RR/OUR. Using the same judge for every metric keeps cross-model rankings from being affected by differences between judges.

Lean toolchain. We check type correctness with Lean 4.15.0 and Mathlib v4.15.0, through a persistent Lean REPL that keeps Mathlib loaded across calls. Each type-check has a per-problem timeout of 900 seconds.

Reproducibility. The (anonymized) project repository contains all perturbation datasets (number, symbol, and step-deletion edits across miniF2F and MATH-500), the scored output files for every model and condition, the human-audit task files, and the inference and judge prompts (also in Appendix A.3.4).

Licenses and terms. We use each artifact under its public license, and each proprietary API under its service terms. The Lean 4 toolchain, Mathlib, and the Lean version of miniF2F are released under Apache-2.0. The MATH dataset, from which MATH-500 is derived, is released under the MIT license. The ProofBridge and ProofFlow codebases are released under the MIT license. The open-weight checkpoints `AI-MO/Kimina-Prover-RL-1.7B`, `AI-MO/Kimina-Prover-Distill-8B`, `Goedel-LM/Goedel-Prover-V2-8B`, `Goedel-LM/Goedel-Formalizer-V2-8B`, and `Qwen/Qwen3.5-397B-A17B` are released under Apache-2.0. For `deepseek-ai/DeepSeek-Prover-V2-7B`, the repository code is MIT-licensed, while the model weights are governed by the DeepSeek Model License. Gemini-3.1-Pro and Gemini-2.5-Flash are proprietary models that we access through Google's Gemini API and use subject to the Gemini API terms. We use all artifacts and services for non-commercial research evaluation, and we release our perturbation datasets, evaluation prompts, and evaluation code under the MIT license.

A.2. Perturbation Construction Details

This section explains how we build the two kinds of perturbations from §3.1 and §3.2.

Global perturbations. For each problem we produce four semantic-preserving rewrites of the natural-language statement and proof: two free-form rephrasings (G-FF, Q-FF) and two numbered step-by-step rewrites (G-Step, Q-Step), generated by Gemini-2.5-Flash and Qwen3.5-397B-A17B. Both prompts (Appendix A.3.1) start with a shared set of rules: do not change any number, symbol, hypothesis, or logical step, and keep the rewrite within $\pm 30\%$ of the original length. Together with the original input, this gives five styles per problem.

Local perturbations. All three local edit types share the same three-stage pipeline. First, an LLM (Gemini-2.5-Flash) labels candidate edit locations. Second, we filter the candidates and pick one per region, where a region is the statement or the proof (so a problem can receive both

a statement edit and a proof edit). Third, a deterministic rule writes the edited text as a single-span substitution at the recorded character offset.

Number edits. The labeler marks every editable number in the statement and proof, skipping structural numbers (subscripts, superscripts, indices) and numbers inside LaTeX formatting arguments. We pick one per region. A SHA256-seeded deterministic rule then sets the new value: it shifts the number by an amount that scales with its size (for example, an integer up to 10 changes by 1 to 3, and one up to 100 by 1 to 5) and keeps the sign, so the edited problem stays well-formed.

Symbol edits. The labeler marks relation and operator symbols (other than $=$ and $\neq$) and tags each as a relation or an operator. We swap each one by a fixed table: relations flip direction ($<\leftrightarrow>$, $\leq\leftrightarrow\geq$) and operators go to their inverse ($+ \leftrightarrow -$, $\times \leftrightarrow \div$). We skip symbols inside relation chains (e.g., $0 < p < 15$) and inside subscripts or superscripts, and we oversample relations to keep the relation/operator mix balanced.

Step deletions. The labeler (the LLM) does the language-level work: it splits each proof into steps, splits each step into a reasoning part and an outcome, and flags whether the step is *deletable*. A step is flagged deletable only if the reasoning is substantive (not a filler connective such as "thus" or "we have") and the outcome stands on its own: it must be a contiguous substring of the step, must not start with a dangling or anaphoric fragment (e.g. "which…", "this gives…"), and must not be a bare number or symbol. Everything after this is a deterministic script with no further LLM calls. The script re-checks these hard rules, then sorts each deletable step into a quality tier (gold, silver, or bronze) by how readable its outcome is (capitalization, English words, and a proper sentence ending). It selects the top step by a fixed, deterministic ranking: tier first, then a reasoning-substance score (whether the reasoning contains mathematics and a substantive verb), then outcome readability, then reasoning length. Finally, the script removes the chosen step's reasoning while keeping its stated outcome, and requires the deletion to remove between 0.5% and 80% of the proof. Step deletions apply to the proof only.

A.3. Prompt Templates

This appendix collects all prompts used in our pipeline: the perturbation rewriting prompts, the autoformalization prompt, and the semantic-correctness and faithfulness judge prompts.

A.3.1. Perturbation Rewriting Prompts

We first give the shared rules that every semantic-preserving rewrite must follow, then the two prompts that generate the global perturbations.

Shared rewriting rules. These rules are prepended to both rewriting prompts below. They require the rewrite to keep the original meaning, length, and logical steps, and to avoid fixing or extending the original.

Shared Rewriting Rules

```
This is a MEANING-PRESERVING transformation. Your
rewrite must be a faithful translation of the
original — a reader must be able to recover the
original's exact mathematical content from your
version. Specifically:

- Do NOT change any numerical constant, variable
name, function name, set, inequality direction,
or quantifier.
- Do NOT change, weaken, strengthen, or generalize
the hypotheses or the conclusion.
- Do NOT add new mathematical facts, lemmas,
intermediate results, or justifications that were
not in the original.
- Do NOT remove, merge, or reorder logical steps
of the original argument.
- Do NOT fix or "clean up" the original. If the
original is awkward, incomplete, or even wrong,
preserve it faithfully — your job is translation,
not editing.
- The rewritten version should be roughly the same
length as the original, within ±30%.

Use LaTeX math notation exactly as in the original,
e.g., x^2, \frac{a}{b}, \neq, \notin, \tan. Write
every backslash literally as a single backslash —
do NOT escape them, do NOT wrap anything in JSON,
do NOT add code fences.

Output format — return EXACTLY these two
tagged blocks and NOTHING else before or
after. Your entire response must start
with <informal_statement> and end with
</informal_proof>.

<informal_statement>
... rewritten problem statement ...
</informal_statement>

<informal_proof>
... rewritten proof ...
</informal_proof>
```

We use the following prompt to generate global perturbations by paraphrasing, while preserving the original mathematical meaning.

paraphrasing Prompt

```
You are a mathematics professor. Paraphrase the following problem statement and its proof into different wording while preserving the exact same mathematical meaning. Change only the phrasing, sentence structure, and word choice; keep every mathematical object, value, and logical step identical.

{faithfulness_rules}

Original problem statement:
%STATEMENT%

Original proof:
%PROOF%
```

We use the following prompt to convert the original natural-language statement and proof into a numbered step-by-step format while preserving the original mathematical content.

Step-by-Step Rewriting Prompt

```
You are a mathematics professor. Faithfully translate the following problem statement and its proof into a numbered step-by-step format. This is a pure formatting change: break the existing content into numbered steps with short bold step titles. Do NOT add explanations, details, motivation, or any content that is not already in the original — only reorganize what is there.

The proof should follow this structure:
1. **Step Title:** <content taken directly from the original>
2. **Step Title:** <content taken directly from the original>

The statement may use the same numbered-step format if it naturally decomposes, or remain as a single paragraph if it does not — but its mathematical content must stay identical.

{faithfulness_rules}

Original problem statement:
%STATEMENT%

Original proof:
%PROOF%
```

A.3.2. Autoformalization Prompts

We use the following system and user prompts to autoformalize each natural-language statement and proof into a Lean 4 theorem together with its proof.

Autoformalization System Prompt

```
You are an expert in mathematics. Your task is to convert informal, natural-language proofs into correct Lean 4 formalizations.
```

Autoformalization User Prompt

```
Your task is to take as input an informal proof in natural language and autoformalize it in Lean 4 with a header. Think step-by-step and ensure that the output formal theorem is compilable with Lean 4 (version 4.15.0).

{example_block}

Here is the **actual** informal proof in natural language:

<informal_statement>
{informal_statement}
</informal_statement>

<informal_proof>
{informal_proof}
</informal_proof>

Now first think step-by-step for the actual output and autoformalize it in Lean 4 with a header. Importantly, enclose the final formal proof in Lean 4 inside the following tags:

<formal_proof>
```lean4
(Provide your entire Lean 4 proof with header here)
```
</formal_proof>
```

A.3.3. Semantic Correctness Judge Prompts

We use two judges to score semantic correctness, one for the proof and one for the statement. The first checks whether the generated Lean proof is semantically consistent with the natural-language proof.

Proof Semantic Correctness Judge Prompt

```
You are an expert in formal mathematics and Lean
4. Judge ONE thing:
Do the following two pieces of reasoning carry
out the SAME LOGICAL STEPS?

=== CRITICAL — Independence from validity ===

You are shown a natural-language proof and
a sequence of Lean 4 proof tactics.   You
are NOT shown the Lean theorem statement that
these tactics target.  This is intentional and
important.
Your job is to compare the two REASONING SEQUENCES
on their own merit:

- Whether the Lean tactics actually prove the
hidden Lean theorem is checked separately by
Lean's type checker. You MUST IGNORE that question
entirely. The tactics may or may not prove the
theorem they are attached to — that is not your
concern.

- Even if the Lean tactics seem to refer to
a different goal than the NL proof suggests,
evaluate the reasoning sequence on its own terms.

=== Equivalence criteria ===

Two proofs are equivalent if they follow the SAME
OVERALL STRATEGY and the substantive intermediate
claims agree, modulo:
- wording and step ordering,
- level of abstraction.   Lean tactics may
IMPLEMENT abstract NL principles such as AM-GM,
Cauchy-Schwarz, or trigonometric identities via
automation tactics like nlinarith, polyrith, or
positivity. This counts as the SAME step, not a
different one.

They are NOT equivalent if:
- they use a fundamentally different strategy or
prove a different goal,
- the Lean proof body is empty, sorry, admit, or
a degenerate placeholder,
- the substantive intermediate claims disagree.

=== Output ===

First, briefly map each substantive NL step to
a corresponding Lean tactic, or note where they
disagree. Then output:

<proof_justification>your            step-by-step
correspondence here</proof_justification>
<proof_score>1 or -1</proof_score>

=== Inputs ===

Natural-language proof:
{nl_proof}

Lean 4 proof tactics, with theorem signature
deliberately hidden:
{lean_proof_body}
```

The second checks whether the generated Lean theorem statement expresses the same claim as the natural-language statement.

Statement Semantic Correctness Judge Prompt

```
You are an expert in formal mathematics and Lean
4.
Does the following Lean 4 theorem signature
express the same mathematical proposition as the
natural-language statement?

=== Equivalence criteria ===

Two statements are equivalent if they capture the
same mathematical content, even if they differ
in surface form: variable names, quantifier order
when commutative, notation such as \le vs ≤, \sum
vs ∑, or how the conclusion is phrased.
Use your mathematical understanding to judge
whether the two statements describe the same
problem.

=== Note on inputs ===

You are shown the Lean theorem signature,
including binders and goal type, only; the
Lean proof body is hidden because it is judged
separately.

=== Output ===

Provide a clear justification, then output:

<stmt_justification>your               reasoning
here</stmt_justification>
<stmt_score>1 or -1</stmt_score>

=== Inputs ===

Natural-language statement:
{nl_statement}

Lean 4 theorem signature:
{lean_theorem_signature}
```

A.3.4. Faithfulness Judge Prompts

Number edits. This judge decides whether the Lean output reflects the edited number.

Number-Edit Faithfulness Judge Prompt

```
You are an expert in mathematical formalization. Given a math problem (whose natural language has been edited by changing one number) and a fragment of the model-generated Lean 4 formalization, determine whether the model faithfully reflects the number edit.

The Lean fragment shown below is the ONLY part of the formalization you should consider. It is the region targeted by the edit ({source}). Do not infer or imagine any other Lean code. Base your judgment strictly on the fragment.

=== Edit Info ===

Edit type: {edit_type}; Original number: {old_value}; Edited number: {new_value}; Edit location: {source}; Edit context: "{context}".

=== Inputs ===

Edited informal statement: {informal_statement}
Edited informal proof: {informal_proof}
Model-generated Lean 4 fragment ({source} region only): {generated_fl}

=== Tasks ===

1. In the Lean fragment, find the number that semantically corresponds to "{new_value}" or "{old_value}" in the edit context. Numbers in Lean may differ from natural language; e.g., 6.5 may appear as 13/2, and 65 may appear as (65 : R).

2. Determine the number at that position: if it is {new_value} (or an equivalent form) → "faithful"; if it is {old_value} (or an equivalent form) → "reverted"; if the corresponding position cannot be found, or it is some other value → "other".

IMPORTANT — ground your judgment in evidence; do not hallucinate. Before returning "faithful" or "reverted", you MUST be able to point to the literal substring of the fragment where the value appears. If you cannot find such a substring, return value_in_fl="" and judgment="other". The context_in_fl field below is the proof — it must be copied verbatim from the fragment and must contain value_in_fl.

=== Output (strict JSON only) ===

{"found_in_fl": true/false,
  "value_in_fl": "copy the number from the fragment as-is (or empty string if not found)",
  "context_in_fl": "if value_in_fl is non-empty, copy the ∼40-character window surrounding the value; otherwise empty string",
  "judgment": "faithful"/"reverted"/"other",
  "reason": "one-sentence explanation"}
```

Symbol-Edit Faithfulness Judge Prompt

```
You are an expert in mathematical formalization. Given a math problem (whose natural language has been edited by changing one math symbol) and a fragment of the model-generated Lean 4 formalization, determine whether the model faithfully reflects the symbol edit.

The Lean fragment shown below is the ONLY part of the formalization you should consider. It is the region targeted by the edit ({source}). Do not infer or imagine any other Lean code. Base your judgment strictly on the fragment.

=== Edit Info ===

Edit type: {edit_type}; Original symbol: {old_symbol}; Edited symbol: {new_symbol}; Symbol family: {family}; Edit location: {source}; Edit context: "{context}".

=== Inputs ===

Edited informal statement: {informal_statement}
Edited informal proof: {informal_proof}
Model-generated Lean 4 fragment ({source} region only): {generated_fl}

=== Tasks ===

1. In the Lean fragment, find the math symbol corresponding to the edited symbol in the edit context. Symbol mapping from LaTeX to Lean: \geq or >= → >=; \leq or <= → <=; > → >; < → <; \neq or != → != or Ne; + → +; - → -; \times or \cdot → *; \div → /. Note: a > b in Lean may equivalently be written as b < a.

2. Determine the symbol at that position: if it is the Lean equivalent of {new_symbol} → "faithful"; if it is the Lean equivalent of {old_symbol} → "reverted"; otherwise → "other".

IMPORTANT — ground your judgment in evidence; do not hallucinate. Before returning "faithful" or "reverted", you MUST be able to point to the literal substring of the fragment where the symbol appears. If you cannot find such a substring, return value_in_fl="" and judgment="other". The context_in_fl field below is the proof — it must be copied verbatim from the fragment.

=== Output (strict JSON only) ===

{"found_in_fl": true/false,
  "value_in_fl": "copy the symbol from the fragment as-is (or empty string if not found)",
  "context_in_fl": "∼40-character window surrounding the symbol, or empty string",
  "judgment": "faithful"/"reverted"/"other",
  "reason": "one-sentence explanation"}
```

Symbol edits. This judge decides whether the Lean output reflects the edited symbol.

Step deletion. This judge decides whether the Lean output reflects the deleted step or instead reconstructs it.

Step-Deletion Faithfulness Judge Prompt

```
You are an expert in mathematical formalization. A math proof was edited by removing the reasoning from one proof step while keeping only the step's outcome/claim. The model then generated a Lean 4 formalization from the edited proof.

Determine whether the model faithfully reflects the missing reasoning.

=== Edit Info ===

Deleted reasoning: "{reasoning_text}"; Preserved outcome: "{outcome_text}"; Step was last step: {is_last}.

=== Inputs ===

Edited informal proof (reasoning removed): {informal_proof}
Model-generated Lean 4 code: {generated_fl}

=== Tasks ===

1. Find where the outcome claim ("{outcome_text}") is formalized in the Lean code. Look for a have, show, theorem conclusion, or equivalent statement.

2. Determine how the model handles the proof of that claim: if the model uses sorry, admit, leaves it incomplete, or provides a significantly simplified/placeholder proof → "faithful" (reflected the missing reasoning); if the model provides a full substantive tactic proof reconstructing the deleted reasoning → "reverted" (fabricated the missing justification); if the outcome claim is not present, or the output is empty/error → "other".

IMPORTANT - ground your judgment in evidence; do not hallucinate. Before returning "faithful" or "reverted", you MUST be able to point to literal Lean code that supports the call: for "faithful", the placeholder/sorry/simplified token(s); for "reverted", the substantive tactic block(s) reconstructing the deleted reasoning. If you cannot find such literal evidence, return judgment="other". The evidence_in_fl field below must be copied verbatim from the Lean code.

=== Output (strict JSON only) ===

{"outcome_present": true/false,
  "proof_method": one of "sorry", "placeholder", "simplified", "full", "absent",
  "evidence_in_fl": "copy the relevant ~80-character snippet of Lean supporting the verdict, or empty string",
  "judgment": "faithful"/"reverted"/"other",
  "reason": "one-sentence explanation"}
```

A.4. Composition of the Other/Unclear Category

This section explains what ends up in the Other/Unclear category (§3.2): the outputs we cannot label faithful or reverted. Inspecting a large sample of these outputs, we find a few recurring patterns. A small portion are degenerate, where the targeted region contains no theorem at all, or the output is empty or an error. The rest are genuine formalizations in which the edited value never appears at the target position, neither the new value nor the original one. This is most common in proof edits, where the proof is closed by an automation tactic such as `norm_num`, `ring`, or `linarith` that never writes the value out, or reaches the goal by a different route. For instance, in the proof edit that changes 49 to 48 in `amc12a_2020_p7`, one model's entire proof is `:= by norm_num`, so neither value shows up and we label the output Other. Step deletion behaves the same way: the deleted step's result usually appears nowhere in the Lean. In short, Other/Unclear means the edit left no trace in the output, not that we faced a hard faithful-versus-reverted call.

Table 5 reports the identifiable fraction (1 minus OUR) for each model and condition. It is highest for statement edits, in the middle for step deletion, and lowest for proof-level number and symbol edits, where an automation tactic usually closes the proof without writing the edited value.

A.5. Reliability Validation

We check three things: that the perturbations (§4.4) are constructed correctly, that the semantic-correctness judge agrees with human judgment, and that the faithfulness judge reports trustworthy verdicts.

Perturbation reliability. We verify that every edit is exactly the intended one. For number and symbol edits we compare the original and edited natural language character by character: the recorded span must hold the old value in the original and the new value in the edited string, and the text on both sides of the span must match exactly, so the change is localized with no collateral edits. For example, in `aime_1983_p1` the span covers the token `40` in the fragment $\log_y w = 40$; we confirm the original holds `40` at that span and the edited string holds `43`, while the 112 characters before the span and every character after it match exactly, so `40` → `43` is the only difference. Symbol edits are checked the same way, for instance a lone $+ \to -$ that turns $x^2 + 18x + 30$ into $x^2 - 18x + 30$. For step deletions, whose recorded offsets can be noisy, we instead locate the changed region as the gap between the longest common prefix and the longest common suffix of the two proofs, and require the edited proof to be strictly shorter and to differ in a single contiguous span; in `aime_1983_p3`, this isolates one removed sentence of 151 characters lying between a 300-character shared prefix and a 359-character shared suffix. All $2{,}637$ local edits pass these checks. For the global rewrites, an automated check finds that 96.4% of the $2{,}976$ rewrites preserve the exact set of numeric and symbolic tokens of the original; the rest are formatting differences such as digit grouping or numbers spelled out, not content

Table 5. Identifiable fraction (%): the share of outputs classified as faithful or reverted, over which FR/RR are computed (the complement of OUR). Lower values mean FR/RR rest on a smaller subset.

Model	Statement edits				Proof edits				Step deletion	
	Num_{mF2F}	Num_{M500}	Sym_{mF2F}	Sym_{M500}	Num_{mF2F}	Num_{M500}	Sym_{mF2F}	Sym_{M500}	mF2F	M500
Kimina-Prover-RL-1.7B	30.8	28.8	40.0	29.7	17.0	11.9	14.4	10.6	43.4	42.7
Kimina-Prover-Distill-8B	72.2	67.3	63.8	59.5	31.1	28.2	24.6	27.5	62.8	58.9
DeepSeek-Prover-V2-7B	82.4	84.5	82.3	82.1	49.8	48.3	39.0	40.4	73.6	58.3
Goedel-Prover-V2-8B	64.3	70.0	71.5	63.1	53.5	49.5	39.5	37.2	55.0	45.3
Gemini-3.1-Pro	75.3	84.7	69.2	74.4	30.3	26.0	18.5	20.9	60.5	64.1
ProofBridge	86.3	90.0	88.5	86.2	58.9	48.5	45.6	41.5	64.3	62.0
ProofFlow	63.4	72.0	67.7	66.7	80.1	84.7	75.9	70.8	81.4	81.8

changes, so this figure is a lower bound. A manual reading of 100 rewrites spanning all four styles and both datasets confirms that every one keeps the original meaning.

Semantic-correctness judge. We check the StmtSC and ProofSC judge against independent human annotation. For each dimension, an author labels 100 judge verdicts as correct or not, reading each against the natural-language statement or proof. We deliberately balance each sample, 50 verdicts the judge accepted and 50 it rejected, so we test it on both the cases it passes and the cases it rejects rather than only the common one. The judge matches the human label on 94 of the 100 StmtSC cases and 90 of the 100 ProofSC cases, with Cohen's κ of 0.88 and 0.80, confirming that its verdicts are reliable on this boundary-stressed sample.

Faithfulness judge. We check the FR/RR/OUR judge in three ways. First, the judge reports the value it reads from the Lean output, which lets us test each verdict for internal consistency without a second model: a `faithful` verdict should carry the edited value, a `reverted` verdict the original value, and an `other` verdict neither, with equivalent forms such as 6.5 and $13/2$ treated as equal. The verdict and the reported value agree on 95.4% of the 13,896 number and symbol cases. Second, we require the judge to copy the verbatim Lean snippet supporting its verdict and confirm that this snippet occurs in the scored fragment; this check fails on only 0.2% of cases, upper-bounding the judge's hallucination rate. Third, an author independently audits 100 number and symbol verdicts and 50 step-deletion verdicts against the model output and the judge's evidence field, flagging only two of the 150 as potentially incorrect.

A.6. Full Results

We give the full results behind the main tables. For global perturbations we report TC, StmtSC, and ProofSC for every model on the original input and four variants, across both datasets; FullyCorrect, their conjunction, is in Table 2. For local perturbations we report all six metrics (TC, StmtSC, ProofSC, FullyCorrect, FR, and RR) for every model and condition. The identifiable fraction (the complement of OUR) is reported in Table 5.

A.7. Potential Risks and Use of AI Assistants

Potential risks. Our work uses only public mathematics benchmarks and poses no foreseeable risk.

Use of AI assistants. AI assistants were used to assist with coding and polish writing.

Table 6. Global perturbation full results (%).

Metric	Model	miniF2F					MATH-500				
		Orig	G-FF	G-Step	Q-FF	Q-Step	Orig	G-FF	G-Step	Q-FF	Q-Step
TC	Kimina-Prover-RL-1.7B	38.5	32.4	39.3	40.2	45.9	39.8	38.8	38.0	39.2	37.4
	Kimina-Prover-Distill-8B	44.7	39.3	40.6	43.4	46.7	49.2	48.2	44.6	46.8	50.0
	DeepSeek-Prover-V2-7B	5.7	17.2	19.3	28.7	27.5	25.8	25.4	22.0	26.0	20.8
	Goedel-Prover-V2-8B	9.8	11.9	12.3	14.8	15.2	16.8	16.4	14.6	17.8	18.4
	Gemini-3.1-Pro	46.3	50.4	49.2	47.1	53.7	58.0	56.8	57.8	55.6	54.4
	ProofBridge	37.3	36.5	37.3	39.3	38.9	33.2	35.0	35.2	31.6	33.6
	ProofFlow	26.6	19.7	26.6	25.4	26.2	27.4	21.0	23.2	22.6	32.4
StmtSC	Kimina-Prover-RL-1.7B	38.1	39.3	42.6	43.9	41.4	29.2	32.4	28.8	31.6	28.0
	Kimina-Prover-Distill-8B	74.2	63.1	65.2	66.8	69.7	58.4	56.8	59.2	60.0	58.8
	DeepSeek-Prover-V2-7B	57.4	63.1	66.0	64.3	58.2	50.8	53.0	49.6	50.6	49.6
	Goedel-Prover-V2-8B	50.8	46.7	51.2	52.9	51.2	42.2	38.6	39.8	40.8	40.4
	Gemini-3.1-Pro	82.0	79.5	79.9	78.3	81.1	81.4	81.8	82.0	82.8	81.2
	ProofBridge	71.3	69.7	69.7	71.3	71.3	50.8	51.6	48.8	50.4	49.0
	ProofFlow	42.2	56.6	39.3	42.2	40.6	47.6	58.4	60.6	58.2	45.8
ProofSC	Kimina-Prover-RL-1.7B	20.9	23.0	20.1	23.0	17.6	16.8	15.8	18.0	19.2	19.2
	Kimina-Prover-Distill-8B	41.4	36.9	38.5	38.5	40.2	43.6	39.4	38.2	40.2	39.0
	DeepSeek-Prover-V2-7B	40.2	41.4	38.5	40.2	43.0	43.2	41.4	43.6	43.6	43.4
	Goedel-Prover-V2-8B	27.0	27.9	34.0	34.0	33.6	29.4	27.4	29.0	28.4	25.8
	Gemini-3.1-Pro	36.5	38.1	37.7	39.8	36.1	41.6	40.4	41.8	41.2	39.8
	ProofBridge	38.1	43.9	41.0	37.3	43.0	37.0	41.8	43.2	39.6	45.2
	ProofFlow	31.1	36.5	33.6	30.7	36.1	34.2	34.8	38.0	39.8	33.8

Table 7. Local perturbation full results on miniF2F (%).

Model	Condition	TC	StmtSC	ProofSC	FullyCorrect	FR	RR
Kimina-Prover-RL-1.7B	Number, statement	23.8	14.1	7.5	2.2	82.9	17.1
	Number, proof	42.7	40.7	14.5	4.6	7.3	92.7
	Symbol, statement	16.2	23.1	4.6	0.8	90.4	9.6
	Symbol, proof	47.7	43.6	12.8	6.2	3.6	96.4
	Step deletion	31.8	34.9	14.7	4.7	51.8	48.2
Kimina-Prover-Distill-8B	Number, statement	39.6	45.8	30.8	7.9	75.0	25.0
	Number, proof	41.9	65.1	29.0	12.0	5.3	94.7
	Symbol, statement	32.3	40.0	16.2	3.1	83.1	16.9
	Symbol, proof	46.7	69.7	29.7	12.8	8.3	91.7
	Step deletion	41.9	66.7	31.8	10.9	25.9	74.1
DeepSeek-Prover-V2-7B	Number, statement	16.7	35.7	25.6	0.9	66.3	33.7
	Number, proof	18.3	63.5	32.0	5.0	8.3	91.7
	Symbol, statement	17.7	39.2	23.8	1.5	73.8	26.2
	Symbol, proof	21.0	67.2	26.2	4.6	7.9	92.1
	Step deletion	9.3	55.0	26.4	0.8	48.4	51.6
Goedel-Prover-V2-8B	Number, statement	1.8	26.0	12.8	0.0	58.9	41.1
	Number, proof	13.3	49.0	21.6	1.2	10.1	89.9
	Symbol, statement	11.5	30.8	8.5	0.0	73.1	26.9
	Symbol, proof	4.6	49.7	23.1	0.5	2.6	97.4
	Step deletion	8.5	49.6	26.4	1.6	43.7	56.3
Gemini-3.1-Pro	Number, statement	46.3	28.2	32.6	3.1	22.8	77.2
	Number, proof	48.5	81.3	29.5	17.0	4.1	95.9
	Symbol, statement	41.5	23.1	30.8	3.1	23.3	76.7
	Symbol, proof	45.6	76.4	26.2	15.4	8.3	91.7
	Step deletion	40.3	80.6	31.8	10.1	47.4	52.6
ProofBridge	Number, statement	29.1	57.7	30.4	6.2	78.6	21.4
	Number, proof	36.1	71.8	29.5	12.9	39.4	60.6
	Symbol, statement	33.8	55.4	31.5	8.5	80.0	20.0
	Symbol, proof	37.9	65.6	25.6	10.8	44.9	55.1
	Step deletion	34.1	66.7	26.4	9.3	33.7	66.3
ProofFlow	Number, statement	18.9	33.5	24.2	0.9	52.8	47.2
	Number, proof	17.4	54.8	27.4	5.4	32.6	67.4
	Symbol, statement	11.5	30.0	18.5	0.8	64.8	35.2
	Symbol, proof	19.5	60.0	26.2	7.2	27.0	73.0
	Step deletion	19.4	58.9	34.1	5.4	24.8	75.2

Table 8. Local perturbation full results on MATH-500 (%).

Model	Condition	TC	StmtSC	ProofSC	FullyCorrect	FR	RR
Kimina-Prover-RL-1.7B	Number, statement	24.9	8.8	7.3	0.6	62.4	37.6
	Number, proof	35.6	25.2	12.3	3.9	5.2	94.8
	Symbol, statement	27.2	9.7	4.6	0.5	82.8	17.2
	Symbol, proof	39.3	23.5	11.2	4.6	8.1	91.9
	Step deletion	34.9	28.6	19.3	8.3	70.7	29.3
Kimina-Prover-Distill-8B	Number, statement	34.5	23.3	29.0	1.8	49.1	50.9
	Number, proof	45.0	54.4	33.5	11.0	2.9	97.1
	Symbol, statement	30.3	28.2	21.0	3.1	67.2	32.8
	Symbol, proof	45.0	52.4	34.1	13.5	3.1	96.9
	Step deletion	40.6	53.6	37.5	14.6	51.3	48.7
DeepSeek-Prover-V2-7B	Number, statement	22.2	21.0	32.7	1.8	48.1	51.9
	Number, proof	25.8	49.7	35.0	4.3	6.4	93.6
	Symbol, statement	14.9	30.3	24.6	1.5	69.4	30.6
	Symbol, proof	35.8	49.6	35.2	7.4	8.5	91.5
	Step deletion	25.0	43.2	39.1	5.7	53.6	46.4
Goedel-Prover-V2-8B	Number, statement	18.8	13.9	17.6	0.2	44.6	55.4
	Number, proof	3.9	38.4	20.4	0.2	6.6	93.4
	Symbol, statement	9.7	22.1	12.8	0.5	59.3	40.7
	Symbol, proof	16.9	33.0	18.6	2.0	9.2	90.8
	Step deletion	14.1	33.3	19.3	3.1	60.9	39.1
Gemini-3.1-Pro	Number, statement	63.7	19.8	39.2	3.5	14.2	85.8
	Number, proof	57.9	81.6	32.9	19.0	6.3	93.7
	Symbol, statement	47.2	22.1	31.3	2.6	26.9	73.1
	Symbol, proof	49.3	79.9	33.8	18.9	4.1	95.9
	Step deletion	56.2	81.2	40.1	22.4	56.9	43.1
ProofBridge	Number, statement	33.3	29.6	39.6	4.1	57.4	42.6
	Number, proof	33.9	44.4	38.4	12.1	41.4	58.6
	Symbol, statement	26.7	32.3	32.3	4.1	70.8	29.2
	Symbol, proof	27.5	44.4	35.0	9.7	41.4	58.6
	Step deletion	37.5	40.6	37.0	9.9	53.8	46.2
ProofFlow	Number, statement	16.9	17.3	28.4	0.6	49.3	50.7
	Number, proof	19.6	53.4	29.4	5.7	24.4	75.6
	Symbol, statement	11.3	28.7	21.5	0.5	57.7	42.3
	Symbol, proof	19.5	56.4	25.5	6.3	20.2	79.8
	Step deletion	22.4	58.9	37.0	6.8	33.8	66.2