FOREIGN EXCHANGE AND PROOF DIVERSITY: WHICH FAILURES TO BORROW AND WHICH RETRIES TO BUY

Anonymous authors
Paper under double-blind review

ABSTRACT

After a Lean prover fails a theorem, what should its next attempt see, and how much computation should it get? We study both questions with frozen, 4-bit provers on selected miniF2F theorems. Goedel-Prover-V2 recovers far more often when shown another prover's whole failed reply than its own: 21/84 versus 6/84 verified retries with DeepSeek-Prover-V2 as the donor (Holm $p = .014$), and 25/68 versus 4/68 with Kimina-Prover (Holm $p = .0015$). Its own failed reply, which it mostly copies back, is also worse than a fresh retry, while foreign replies are almost never copied; whether a foreign failure beats a fresh retry remains unresolved. Telling the model that its own reply came from another prover does not help. How a failure was made matters too: shown as code, heavily corrupted correct proofs are recovered from far more often than genuine failures (37/84 versus 21/84, Holm $p = .024$), so repair benchmarks built on corrupted proofs can overstate recovery. Finally, failure types are more useful for budgeting than as prompt text: in our earlier experiments, failure-aware retry allocation saved 27.5–33.9% of retry tokens in the primary draw order, and a targeted output cap saved 11.2–13.3% of measured GPU energy, without losing solves. Which failed proof to show and how much compute to spend are separate decisions, and each should be judged against a fresh retry. Confirmatory analysis plans for the new studies were frozen before generation.

1 INTRODUCTION

When a language-model prover fails a theorem, the natural next step is to try again with the failure in view. Goedel-Prover-V2's correction pipeline, for example, appends the model's raw previous reply to the conversation and follows it with Lean's error message (Lin et al., 2026; Goedel-LM, 2025); related loops appear in provers trained on verifier feedback (Xin et al., 2025; Wang et al., 2026) and in general self-repair and self-refinement methods (Chen et al., 2024; Madaan et al., 2023; Shinn et al., 2023). The debate has mostly compared retrying with a failure against retrying without one; a recent placebo-controlled study found that small code models shown their own failed program largely reproduce it, so that blind resampling beats self-repair (Verma, 2026). In most of this work, the failed attempt is treated as a single kind of context.

We ask a question that this framing leaves open: does it matter *whose* failed reply the prover is shown? Holding the receiving model, the theorem, the sampling seed, the prompt frame and the output budget fixed, we return either the receiver's own whole failed reply or another prover's whole failed reply for the same theorem, and measure how often the retry produces a proof that Lean verifies. To our knowledge, no prior study has made this comparison directly (§2).

The answer is a large and consistent difference (Figure 1). For Goedel-Prover-V2-8B (Lin et al., 2026), its own whole failed reply was the least useful context, or tied for least, in every study in which it appeared: 6/84 retries recovered a proof, against 32/84 for a fresh retry. Whole failed replies from two other provers did markedly better: 21/84 for DeepSeek-Prover-V2 (Ren et al., 2025), and 25/68 for Kimina-Prover (Wang et al., 2025) against 4/68 for the receiver's own reply in that study. Behaviour mirrors the gap: shown its own reply, the receiver largely resubmits it; shown

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

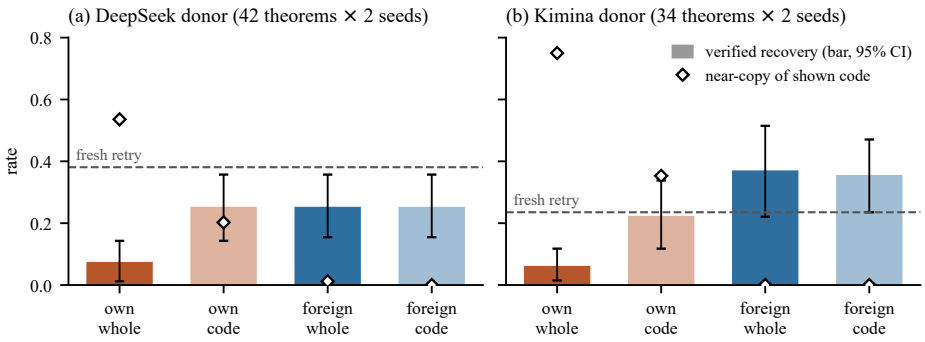


Figure 1: Verified recovery of Goedel-Prover-V2-8B by the failed reply it is shown. Bars: theorem-level recovery rate with pointwise 95% bootstrap intervals; diamonds: share of retries that nearly copy the shown code; dashed line: fresh retry on the same theorems and seeds. "Code" bars return only the extracted final code (§5). The two panels are separate studies with different theorem sets and are not pooled.

another prover's, it almost never does; and telling it that its own reply came from another prover does not change the outcome.

The result has a simple practical consequence. Returning only the final code of the receiver's own failure instead of its whole reply recovered a substantial part of the loss and reduced the observed gap between own and foreign failures; because own whole-reply retries so rarely verify, it also cut the generated tokens spent per verified proof from 42,062 to 15,383 (§5). None of the contexts we tested significantly outperformed a fresh retry.

Contributions. We make three contributions, developed over five results sections.

1. **The failed artifact a prover is shown changes recovery** (§4, §5, §7). With two unrelated donors, another prover's whole failed reply supports more recovery than the receiver's own (+17.9 and +30.9 points; Holm $p = .014$ and .0015), in every seed block. The receiver frequently resubmits its own failed reply and almost never a foreign one, and simple length differences and the tested attribution labels do not explain the gap. Showing only the final checked code of the same own failure recovers a substantial part of the loss at lower cost and roughly halves the copying; hiding the step Lean rejected helps one receiver but not a second.
2. **Synthetic failures misrepresent recovery from genuine ones** (§6). With presentation matched, heavily corrupted correct proofs are recovered from far more often than genuine failures with comparable median line similarity to a correct proof (+19.1 points, Holm $p = .024$).
3. **Failure information is better used to allocate computation than as prompt text** (§8). In our earlier experiments, reported here with their methods, failure-type hints gave no robust gain as prompt content, whereas failure-aware allocation rules beat the tested label-free caps, and a targeted output cap saved tokens and measured GPU energy without changing the verified set.

Confirmatory analyses were frozen before generation, a Lean identity audit removed artifacts whose shown code did not match its diagnostic, and a single command re-derives the principal analyses from receipts and recorded trajectories and regenerates every number; the remaining inputs are identified, integrity-checked summaries and earlier manuscripts' number files (Appendix A).

2 Related Work

Self-correction and repair. Feedback-driven repair improves code and reasoning in some settings (Chen et al., 2024; Madaan et al., 2023; Shinn et al., 2023), but its gains are often modest once the cost of repair is counted (Olausson et al., 2024), models struggle to correct their reasoning without external feedback (Huang et al., 2024), and a survey finds that self-correction works mainly with reliable external feedback or large-scale fine-tuning (Kamoi et al., 2024). Repeated sampling alone is a strong baseline (Brown et al., 2024; Snell et al., 2025), and features of failed traces can predict which intervention rescues a failure (Islah et al., 2026). Placebo-controlled studies of code repair

find that small code models shown their own failed program largely reproduce it, and that blind resampling beats self-repair (Verma, 2026). Re-exposure to the failing code helps less than blind resampling, and feedback helps through executable counterexamples rather than through the failing code itself (Iscan, 2026). Fault localisation does not beat a fresh attempt either (Jha, 2026). Retained reasoning can also hurt: agents gain from keeping recent reasoning in full and older reasoning only as compressed digests (Lian et al., 2026), earlier chains of thought act as shortcuts (Fan et al., 2026), and models err more with their own earlier errors in context (Sinha et al., 2026). These studies vary whether, and how much of, a failure is shown. To our knowledge, no prior work has directly compared recovery from a model's own genuine failed output with recovery from another model's genuine failed output on the same task while holding the receiving model and retry procedure fixed.

Theorem proving with failures. Lean (de Moura & Ullrich, 2021) and Mathlib (The mathlib Community, 2020) make recovery machine-checkable, and miniF2F (Zheng et al., 2022) is a standard testbed. Provers and agents use verifier feedback for whole-proof repair, dual correction, in-context agents and retrieval (First et al., 2023; Zheng et al., 2024; Thakur et al., 2024; Yang et al., 2023), and recent systems train self-correction or repair from compiler feedback (Lin et al., 2026; Xin et al., 2025; Ma et al., 2026). Compiler feedback also drives proof repair (Ospanov et al., 2025) and failure-mode-guided search (Li et al., 2026); we instead run controlled comparisons of what one retry sees and how many retries to buy. Other work repairs failed F* proofs written by other models (Zhang et al., 2025), or combines and refines candidates from a specialist and a generalist prover, finding that the starting proof affects recovery (Liu et al., 2026); we instead hold the receiver and retry fixed and vary only the failed artifact. Interfaces differ in what they return: Goedel-Prover-V2's correction loop keeps the whole previous reply in the conversation (Goedel-LM, 2025), while APRIL (Wang et al., 2026) pairs failing proof code with compiler diagnostics and builds its failures by mutating correct proofs. PoPilot compares synthetic and model-generated repair training data in F* and finds that, in their main ablation, only the latter improves repair (Zhang et al., 2025). We instead compare recovery from synthetic and genuine failures with the receiving model fixed, asking how faithfully synthetic failures stand in for genuine ones at evaluation time; this does not dispute training gains.

Repetition, attribution and synthetic faults. Repetition is self-reinforcing: the more often a sentence already appears in the context, the more likely a model is to generate it again (Xu et al., 2022). Models can also distinguish their own outputs from others' (Panickssery et al., 2024), and relabelling the chat role of an erroneous claim changes whether a model flags it (Chen et al., 2026). Models also correct errors attributed to an external source that they miss when the identical error is their own (Tsui, 2025); that work varies attribution with content fixed, whereas we vary the source with the frame fixed. In software testing, whether artificial mutants substitute for real faults is a long-standing question (Just et al., 2014), learned bug detectors trained on artificial bugs underperform on real ones (Richter & Wehrheim, 2023), and how synthetic bugs are constructed changes how hard they are to detect and repair (Ibrahimzada et al., 2025); §6 asks the analogous question for proof recovery.

3 Experimental setup

Scope and evidence. We investigate two decisions after a proof fails: what the next attempt should see, and how much computation it should receive. We analyse experiments with different serving setups separately, never pooling them. Our earlier experiments (*Protocol A*; methods in Appendix F) use single-slot serving and attempt-blind retry prompts, and examine perseveration on the rejected step and failure signals as prompt content or retry control. The new recovery experiments (*Protocol B*) use continuous batching and the identity rule below, and examine how the source, presentation and construction of a failed proof affect recovery. Portions of the earlier experiments also appear in unpublished manuscripts submitted to non-archival workshops; this paper integrates and extends them, marking shared measurements as *previously reported* and new analyses as new (Appendix A). Unless stated otherwise, results are from Protocol B.

Models and serving. The receiver in most studies is Goedel-Prover-V2-8B (Lin et al., 2026); DeepSeek-Prover-V2-7B (Ren et al., 2025) serves as a donor and as a second receiver, and Kimina-Prover-Preview-Distill-7B (Wang et al., 2025) as a second donor, generated with its native system prompt and sampler. All models run as GGUF Q4_K_M quantisations in llama.cpp (Gerganov & llama.cpp contributors, 2023) on one NVIDIA RTX 4090 (24 GB), with continuous batching (Goedel: 3 slots of 32,768 tokens; DeepSeek: 2 slots of 24,576 tokens) and prompt caching disabled.

Receivers use each model's published whole-proof prompt, temperature 0.7, top-p 0.95, top-k 40, min-p 0.05 and a 12,288-token output cap. Proofs are checked by the Lean 4 REPL (Lean v4.9.0-rc1 with a pinned Mathlib) under a 180 s limit; a retry succeeds when the target theorem verifies.

Theorems. All theorems come from miniF2F (Zheng et al., 2022). The Goedel cohort consists of 46 theorems, selected in earlier work from a pool of 49 as those with recorded genuine failures from both Goedel and DeepSeek; 42 remain after the identity audit, and the masking studies add five more from the pool (47 in all). The Kimina study uses the 34 of the 46 for which a fresh Kimina screen produced an eligible failure. The DeepSeek cohort consists of 60 theorems with a recorded genuine DeepSeek failure and a DeepSeek-verified proof. Selection never used retry outcomes, but the cohorts are chosen to contain failures, so rates do not estimate benchmark performance.

Failure artifacts and presentation. Every non-blind prompt wraps one artifact in a fixed neutral frame ("An attempt was not accepted by the Lean 4 compiler…"), followed by the Lean diagnostic of that artifact and the unchanged task prompt. Exceptions: the prose-only arm has no diagnostic, the label study rewords the frame's first sentence, and the masking studies keep the unmasked code's diagnostic (Appendix B). *Own* artifacts are saved genuine failures of the receiver; *foreign* artifacts are another prover's genuine failures on the same theorem. *Whole* returns the full reply, reasoning and code; the processing variants return only its final checked Lean block (*code*) or only its prose. *Synthetic* artifacts corrupt a correct proof of the same theorem: SYN1 by one local edit, SYN8 by eight span replacements that overwrite most of the proof body. The frame presents the failure as supplied context rather than as a prior assistant turn, so it approximates rather than reproduces native chat-based correction.

Identity rule and audit. The code shown must be exactly the code Lean checked, and the diagnostic must be that check's output; the masking arms are a deliberate exception (§7). Re-checking every artifact found 4 DeepSeek donor artifacts and 15 of 60 saved DeepSeek own failures whose diagnostic came from Lean checking the proof under the benchmark's theorem statement, while the prompt showed the model's own restated version. Lean reproduces each of these diagnostics on the substituted text, so they are genuine, but they do not match the code as shown; all were excluded before the new studies (Appendix C).

Design and statistics. Each new Protocol B study crosses theorems with arms in two seed blocks; within a block, all arms of a theorem share one sampling seed. Outcomes are averaged over seeds within each theorem, and each contrast is a theorem-level mean difference with a two-sided sign-flip randomisation test (200,000 draws) and a pointwise theorem bootstrap interval (10,000 resamples; Efron & Tibshirani, 1993). Holm correction (Holm, 1979) is applied within each study's frozen family (Table 1); other contrasts are labelled secondary. Plans, seeds and job hashes were frozen before generation, and in sensitivity analyses that re-score Lean timeouts in both directions or restrict to matched seed cells, the whole-reply source contrasts and the Goedel masking benefit stay significant, while the construction contrast stays positive but would not survive Holm correction if every uncertain timeout were scored against it (Appendix D). A retry *near-copies* the shown code when its checked code, with comments removed and whitespace normalised, has a line-level similarity of at least 0.9 to it (the repeated-tactic control uses its own frozen token-level measure; Appendix E).

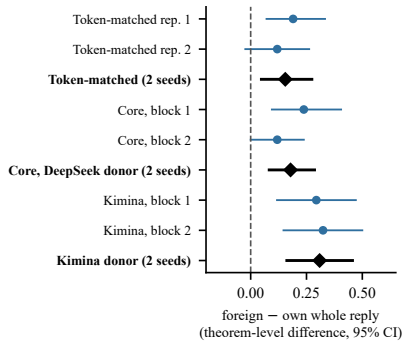
4 Source: another prover's failure is better context than your own

Whole failed replies from another prover supported more recovery than the receiver's own whole failed reply, with both donors (Figure 2). With DeepSeek failures the difference was +17.9 points ([+8.3, +28.6], Holm $p = .014$), and 14 theorems favoured the foreign reply against 2 that favoured the own. With Kimina failures it was +30.9 points ([+16.2, +45.6], Holm $p = .0015$), with 14 theorems against 1. Both arms use the same whole-reply format, frame and kind of diagnostic, so the contrast is whose reply it is; the replies' lengths and contents still differ (below). The direction held in every seed block. The advantage is broad: across 6 seed blocks, 57 theorem-block pairs were solved only with the foreign reply and 8 only with the own (descriptive; Figure 4).

Simple length differences and the tested labels do not explain the gap. Kimina's whole replies are longer than the receiver's own (median 7,479 versus 4,697 characters) and DeepSeek's are shorter (3,297), yet both beat the own reply. Two earlier seed blocks that matched the full prompt

Table 1: Studies. Seeds are shared across arms within a block; "family" lists the confirmatory tests corrected together. Protocol A rows were previously reported and are included with their methods (Appendix F); the label-free allocation baselines are a new analysis of their recorded retries.

Study	Prot.	Receiver	Arms (failure shown)	Thm. × seeds	Family
Core	B	Goedel	fresh; own and DeepSeek × whole, code; own prose; SYN1; SYN8	42×2	Holm, 5
Kimina donor	B	Goedel	fresh; own and Kimina × whole, code	34×2	Holm, 3
Rebuild	B	Goedel	fresh; own whole; own code; own or other prose + own code	32×2	Holm, 3
Labels	B	Goedel	fresh; own whole under a neutral, "your" or "another prover's" label	42×2	1 primary
Repeated tactic	B	Goedel	fresh; own whole; own code; length-matched repeated `linarith` or `omega`	42×2	Holm, 2
Masking	B	Goedel	fresh; intact; rejected step masked; placebo span masked	34×2 (49 failures)	1 primary
Masking	B	DeepSeek	fresh; intact; rejected step masked; placebo span masked	31×2	1 primary
DeepSeek format	B	DeepSeek	fresh; own whole; own code; synthetic code	60×1	earlier
Perseveration	A	Goedel	fresh; own previous attempt; another theorem's attempt	40×2	reported
Allocation	A	all three	four recorded plain retries per failed theorem, replayed under allocation policies	40×2 (Goedel)	reported
Output cap	A	Goedel	full versus targeted output cap on previously truncated theorems	40×3	reported



Study, block	Foreign whole	Own whole	Fresh retry
DeepSeek donor, 1	14/42	4/42	16/42
DeepSeek donor, 2	7/42	2/42	16/42
Kimina donor, 1	13/34	3/34	8/34
Kimina donor, 2	12/34	1/34	8/34

Median characters shown: own whole 4,697; DeepSeek whole 3,297; Kimina whole 7,479.

Figure 2: Left: foreign minus own whole-reply recovery for every seed block (circles) and every study (diamonds), with 95% theorem-bootstrap intervals; no estimate is pooled across studies. Right: per-block counts. The token-matched blocks exclude the four audited artifacts.

length in Goedel tokens, frozen as a replication of an exploratory first block (Table 9), point the same way: 23/84 versus 10/84 verified retries (+15.5 points, [+4.8, +27.4]; exploratory), with a median length gap of 321.5 tokens. All source contrasts use one neutral frame for both replies. A frozen replication then showed the receiver its own whole failed reply under three frames: neutral, "Your previous attempt…" and "Another prover's attempt…". Relabelling did not help. The "another prover" label gave 6/84 verified retries and the "your" label gave 6/84 (0.0 points, [−6.0, +6.0]; $p = 1.00$), against 10/84 with the neutral frame and 27/84 for a fresh retry. Near-copying stayed high under every label (45/84, 54/84 and 49/84). An earlier exploratory study with the receiver's longer original failures had suggested a benefit of the "another prover" label (10/66 versus 3/66; Appendix E); the replication on the core failures does not support it.

The receiver frequently resubmits its own failed reply, almost never a foreign one. The clearest behavioural difference is what the receiver does with the shown code (Table 2). Retries nearly copied the own whole reply in 45/84 (core) and 51/68 (Kimina study) cases, and only 1 of these 96 near-copies verified. In 40/84 and 40/68 retries, the receiver resubmitted the shown code character for character. Foreign whole replies were nearly copied in 1/152 retries. A frozen control shows that this reuse is selective, not indiscriminate. When the receiver was shown length-matched code

Table 2: What the receiver does with a whole failed reply. Near-copy: retry code with line similarity ≥ 0.9 to the shown code. Protocol A showed the attempt without its diagnostic and reports tactic-sequence similarity instead (previously reported). The DeepSeek row excludes its 15 audit-failed theorems.

Study (receiver)	Shown	Fresh retry	With failure	Near-copies	of which verified
Protocol A (Goedel)	own attempt	9/40; 9/40	2/40; 1/40	similarity 0.703 (fresh 0.397)	
Protocol A (Goedel)	other theorem's	9/40; 9/40	7/40; 11/40	similarity 0.375	
Core (Goedel)	own whole	32/84	6/84	45/84	0
Kimina (Goedel)	own whole	16/68	4/68	51/68	1
Rebuild (Goedel)	own whole	23/64	9/64	35/64	1
Core, Kimina (Goedel)	foreign whole	32/84; 16/68	21/84; 25/68	1/152	0
Format (DeepSeek)	own whole	21/45	13/45	1/45	0

Table 3: Returning only the extracted final code (theorem-level points, pointwise 95% CI). The interaction is (foreign − own, whole) − (foreign − own, code).

Contrast	DeepSeek donor (42 theorems)	Kimina donor (34 theorems)
Own code − own whole (secondary)	+17.9 [+7.1, +28.6]	+16.2 [+7.4, +26.5]
Foreign code − foreign whole (secondary)	0.0 [−9.5, +9.5]	−1.5 [−16.2, +11.8]
Foreign code − own code	0.0 [−13.1, +11.9]; Holm 1.00	+13.2 [+1.5, +25.0]; Holm .083
Interaction	+17.9 [+3.6, +32.1]; Holm .089	+17.6 [+2.9, +32.4]; Holm .083

that repeats one tactic (`linarith` or `omega`), no retry met the whole-proof near-copy threshold (0/168). When it was shown its own failed code, 27/84 retries did (Holm $p < .001$). Recovery with the repeated-tactic code was at least as high as with a fresh retry (26/84 and 31/84 versus 26/84) and +12.5 points above recovery with own code (Holm $p = .047$). That recovery difference is sensitive to how Lean timeouts are handled (Appendix E). Resubmission accompanies the recovery loss but is not shown to cause all of it: On the theorems that passed the identity audit, DeepSeek as a receiver nearly copied its own whole failure in only 1/45 retries, yet did worse with its own failure than with a fresh retry in both of its studies (13/45 versus 21/45, $p = .077$; 10/45 versus 25/45, $p = .0015$).

Against a fresh retry. In short, in both studies the own whole reply did worse than both a foreign reply and a fresh retry, but whether a foreign reply beats a fresh retry is unresolved. The own whole reply fell well below a fresh retry (core −30.9 points, [−44.0, −17.9]; Kimina −17.6 points, [−30.9, −4.4], $p = .026$). With DeepSeek failures, foreign replies fell below a fresh retry (−13.1 points, [−25.0, −1.2], $p = .072$). With Kimina failures, they rose above it (+13.2 points, [−1.5, +27.9], $p = .12$), on a harder subset where fresh retries succeeded only 16/68 times. Neither difference is significant, and neither charges the donor generation a foreign failure requires. An earlier single-slot study on a harder population found a same-theorem DeepSeek failure below a fresh retry (Appendix F).

Why DeepSeek is not a source receiver. Both donors were tested with Goedel as the receiver. Using DeepSeek as the receiver requires theorems on which both DeepSeek and a donor genuinely fail, and few exist: the stronger Goedel solved 56 of DeepSeek's 60 failed theorems in a fresh two-draw screen and gave a usable failure on only 7, of which 4 also had an audited DeepSeek failure, and only 22 of the 60 had a usable Kimina failure. Both reverse-direction studies therefore stopped at their frozen eligibility gates of 30 theorems; we report the gates rather than screen further.

5 Presentation: extracting the code reduces the own-reply penalty

Code extraction as a processing step. Returning failed code with compiler feedback is standard in repair (First et al., 2023; Olausson et al., 2024; Wang et al., 2026); what our design adds is a matched comparison on the same genuine failure. Returning only the final checked code of the receiver's own failure raised recovery from 6/84 to 21/84 in the core study and from 4/68 to 15/68 in the Kimina study (+17.9 and +16.2 points, secondary; Table 3). It also roughly halved own near-copying, to 17/84 and 24/68. For foreign failures, extraction barely moved the point estimates. Extraction therefore reduces the observed gap between own and foreign failures: with code alone, DeepSeek and own failures had equal observed counts (21/84 each), and Kimina code stayed ahead

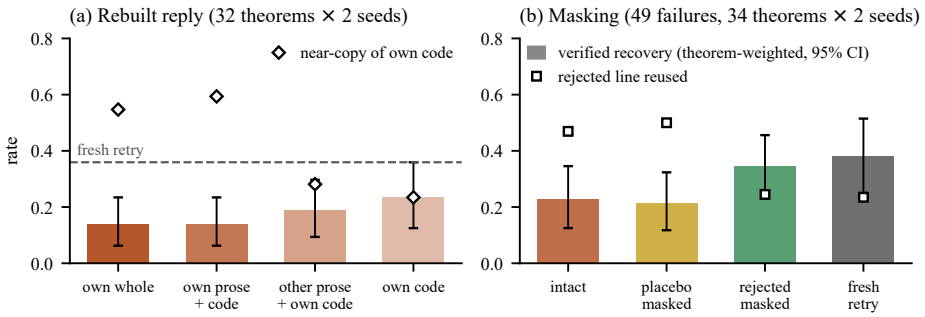


Figure 3: (a) Rebuilt replies: the receiver's own prose placed before its final code gave the same observed recovery count as the whole reply and brought back its copying, whereas another theorem's prose barely changed copying; the recovery contrasts are not conclusive. (b) Masking the lines of Lean's first error versus an equal-size placebo span (Goedel receiver); squares show how often the retry reuses a rejected line. Intervals are pointwise 95% theorem-bootstrap intervals.

of own code by +13.2 points, not significant after correction. The source-by-presentation interaction estimates are nearly identical in the two studies (+17.9 and +17.6 points), but neither survives Holm correction, and we do not pool the studies because they share theorems. Extraction does not make a failure better than a fresh retry: own code still recovered less often than a fresh retry in the core study (21/84 versus 32/84).

Extraction is also cheaper. Own whole-reply retries are short, because the receiver largely resubmits the shown proof: they averaged 3,004 generated tokens against 5,508 for a fresh retry, and 3/84 against 16/84 reached the output cap. But they so rarely verify that they spent 42,062 generated tokens per verified proof, 2.9 times a fresh retry's 14,460. Own code spent 15,383 (Kimina study: Table 15). These are observed costs on our cohorts, not an equal-budget policy comparison.

Which part of the whole reply matters. A frozen rebuild study (32 theorems) separated the parts of the own reply (Figure 3a). Placing the reply's own prose before its final code, without the intermediate code blocks, matched the whole reply's observed counts: 9/64 versus 9/64 verified retries (0.0 points, [−7.8, +9.4]), with 38/64 versus 35/64 near-copies. Adding the own prose to the code more than doubled near-copying relative to code alone (38/64 versus 15/64), whereas length-matched prose from another theorem barely moved it (18/64). The recovery contrasts pointed the same way but were not conclusive: own prose plus code versus code alone −9.4 points ([−20.3, +3.1]; Holm $p = .65$), and own versus other-theorem prose −4.7 points ([−14.1, +4.7]; Holm $p = 1.00$). Prose alone, without code or diagnostic, was not clearly harmful (26/84 versus 32/84; −7.1 points, Holm $p = .82$). The receiver's own explanation next to its own code is thus associated with resubmission; whether it causes the recovery loss remains open.

6 Construction: synthetic failures overstate recovery

Repair methods are often trained and evaluated on failures made by corrupting correct proofs (Wang et al., 2026). With presentation matched (all extracted code), the receiver recovered from its own genuine failures in 21/84 retries, from one-edit corruptions (SYN1) in 47/84, and from heavy corruptions (SYN8) in 37/84; a fresh retry succeeded in 32/84. The prespecified contrast SYN8 − own code was +19.1 points ([+7.1, +31.0]; Holm $p = .024$). With the 13 Lean timeouts scored least favourably, it stayed positive but weaker (+15.5 points, unadjusted $p = .034$) and would fail Holm.

The gap persists despite comparable median line similarity to the available correct proofs, a surface measure rather than a match in mathematical distance. Genuine failures are almost never near-misses: 0/345 Goedel failures and 1/192 DeepSeek failures had a line similarity of at least 0.9 to the nearest verified proof of the same theorem, against 25/48 one-edit corruptions from an earlier study (Appendix C). Copying is not harmful in itself: near-copies of one-edit corruptions (23/84 retries) verified 11 times, near-copies of own whole replies (45/84) never. SYN8 was built to reduce this surface-similarity difference: its median similarity to the nearest verified proof (0.18) is as low as that of genuine failures (0.21), yet it was recovered from far more often. SYN8 matches genuine

Table 4: Failure-aware retry allocation (Protocol A; replay of recorded plain retries, four per failed theorem). Savings are in retry-stage completion tokens at the plain-retry solve count, primary draw order. The label-free cap is a new analysis of the same draws; the other rows were previously reported.

Policy on the same recorded draws	Block 1 (14 solves)	Block 2 (15 solves)
Cap every theorem at two retries (no labels; new analysis)	13 solves	13 solves
Skip retries after timeouts	−10.8%	−11.3%
Cap retries of truncated attempts at two	−13.7%	−23.3%
Both rules	−24.5%	−34.6%
Both rules plus class ordering	−27.5%	−33.9%

failures in format and ordered similarity, not content: it is built from the correct proof's lines, so the right tactics and lemmas remain in view. In an observational analysis across different proofs, we detected no trend with the position of a one-line corruption (Spearman $\rho = +.04$ between the edit's relative position and success over 49 one-edit corruptions, $p = .77$; DeepSeek $\rho = +.18$, $p = .17$; exploratory). We therefore found no evidence that edits after a long correct prefix explain why synthetic failures are easier, although a moderate position effect is not excluded. DeepSeek as a receiver points the same way on its own cohort (synthetic code 20/45 versus own code 15/45 on audited theorems; 27/60 versus 19/60 before the audit), without reaching significance. The point concerns evaluation fidelity: recovery measured on corrupted correct proofs overstates recovery from a prover's genuine failures. It does not show that synthetic data cannot help training.

7 Intervention: hiding the step Lean rejected

Under Protocol A, the line behind Lean's first error reappeared in 20/21 and 20/21 retries shown their own attempt, against 5/21 and 8/21 fresh retries (previously reported). We turned this observation into an intervention. From 49 genuine Goedel failures on 34 theorems whose first Lean error spans one to three lines, we built four prompts: the intact failed code; the same code with the rejected lines replaced by the comment "`-- (line removed)`" at the same indentation; the same code with an equal-size contiguous placebo span replaced (never the statement, never adjacent to the error); and a fresh retry. The diagnostic is identical in all non-blind arms.

An exploratory block showed only a small effect over the placebo (+4.4 points, [−11.8, +20.6]), so we froze a two-block replication with rejected-versus-placebo as its single primary contrast (Figure 3b). Masking the rejected step recovered 36/98 retries against 22/98 for the placebo: +13.2 points weighting theorems equally ([+2.9, +23.5]; $p = .028$). Reuse of a rejected line fell from 49/98 to 24/98, the level of fresh retries (23/98). The comparison with intact code was uncertain (+11.8 points, $p = .055$), and masked prompts did not beat a fresh retry (−3.7 points, [−14.7, +7.4]). The effect is relative to the placebo; reduced reuse accompanies it but is not shown to mediate it.

A second receiver. The same frozen design with DeepSeek-Prover-V2 as the receiver (31 theorems, two blocks) did not show the effect. Masking the rejected step recovered 18/62 retries against 25/62 for the placebo (−11.3 points, [−24.2, +1.6]; $p = .17$), with 19/62 for intact code and 28/62 for a fresh retry; 3 theorems favoured masking and 10 the placebo. DeepSeek rarely reused its rejected line even when shown the intact code (10/62, against 7/62 for fresh retries), so there was little for masking to remove; this explanation is post hoc. In our data, masking is receiver-specific, not a general remedy, in line with placebo-controlled studies of localised repair in code (Jha, 2026).

8 Efficiency: failure signals as retry control

Failure types: prompt text versus allocation. The previous sections ask what a retry should see, and no context we tested significantly beat a fresh retry; this section asks how much computation retries should receive. Placed in the prompt, failure-class hints gave no robust gain over reseeding. Across 19 retry draws and 12 prompt variants, a nominally significant win ($p = .039$) did not survive matched seeds ($p = .549$), while on a larger population four plain reseeded draws lifted the solved count from 119/202 to 146/202 (previously reported). The same failure types were useful for allocating retries. On the recorded retries (Table 4), two rules, skipping retries after a verifier timeout and capping retries of truncated attempts at two, reached the plain-retry solve count while

saving 24.5% and 34.6% of retry tokens in the primary draw order; adding class ordering gave 27.5% and 33.9% (previously reported). Their numerical parameters were fitted on a separate development split and frozen before fresh draws on the held-out evaluation theorems, although the full failure pool informed the rule family. These savings depend on draw order: parity held in only 12 and 20 of the 24 orders. The timeout rule alone kept parity in 24 and 24 of 24 and saved 10.8% and 11.3% (previously reported). A new analysis tested label-free caps of one, two or three retries per theorem. They either lost solves or saved nothing; at two retries they solved 13 and 13 theorems against 14 and 15. The savings also need costly failure modes, which DeepSeek's pool lacked.

Output caps and energy. Long failed generations drive the cost: output-capped branches were 25.0% of branches but 38.0% of measured time. A targeted output cap on previously truncated theorems kept the verified set unchanged while saving 9.8–11.4% of completion tokens and 11.2–13.3% of measured GPU energy (previously reported). A failure was more useful for deciding whether and how much to retry than as text to paste back. The two uses were studied separately, not as a combined policy.

9 Discussion and limitations

Recommendations. On the tested Goedel cohorts, the prover's own whole failed reply, which Goedel-Prover-V2's native loop also returns (there as a prior assistant turn, which we did not test), was among the least useful and costliest contexts examined, in generated tokens per verified proof, and another prover's whole reply was consistently better. For systems that retry with feedback, this suggests three inexpensive steps: extract the code of a failed attempt instead of returning the whole reply; use the failure type to decide whether and how much to retry; and compare every choice of context against a fresh retry, which none of our contexts significantly outperformed.

Why might the own reply anchor? A hypothesis consistent with our data, though not tested by it: own failures may better match the receiver's learned continuations, and text repeated in context becomes more likely to repeat (Xu et al., 2022), whereas a foreign failure is rarely reproduced.

What the results do not show. Source differences come with differences in content, style and error type, and whether the receiver recognises its own style is not tested. The source comparison has one receiver (§4), and the frozen Goedel studies cover 47 distinct theorems from an original pool of 49 selected miniF2F theorems: the Kimina study adds a second donor, not new theorems. DeepSeek evidence as a receiver is weaker after the audit (§4). The new studies each have two seed blocks and 31–42 theorems, earlier DeepSeek studies one block, and the interaction and rebuild contrasts are not confirmed. All receivers ran at 4-bit precision (Q4_K_M) with one sampler; comparisons are internally matched, but quantisation can change reasoning behaviour (Liu et al., 2025), although a related self-repair penalty in small code models was unchanged at full precision (Verma, 2026). Protocol A's allocation results from two earlier manuscripts share recorded trajectories, and its output-cap experiment resampled the same theorems, so none is an independent replication.

Future work. A wider failure search, with more draws per theorem across more provers, more miniF2F theorems and other benchmarks, would supply the matched failures that other receivers need, including DeepSeek, and test how far the source effect generalizes. Repeating the source comparison with Q8 and full-precision weights would test whether the effect depends on quantisation.

10 Conclusion

Whose failed reply a prover is shown matters. On the tested cohorts, Goedel-Prover-V2 recovered far more often from another prover's whole failed reply than from its own, which it largely resubmitted, and relabelling its own reply as another prover's did not help. Extracting the code of a failed attempt recovered a substantial part of that loss, cut the generated tokens per verified proof and reduced the observed gap between own and foreign failures. Format-matched synthetic failures were easier to recover from than genuine ones, so synthetic evaluations overstate recovery; masking the rejected step helped one receiver but not a second; and failure signals were more useful for deciding whether and how much to retry than as text in the prompt. Recovery systems and their evaluations should report whose failure they return, in what form and how it was produced, and compare each choice against a fresh retry.

AI USE STATEMENT

In this work, we used generative AI tools (LLM-based coding and writing assistants) to propose and refine hypotheses and experiments, to give feedback on experimental design and statistical analysis plans, to implement and debug the experiment, verification and analysis scripts, including the scripts that construct the synthetic corrupted proofs, to run and monitor experiments, to help interpret results, to draft planning documents, figures and parts of this paper, and to search, check and format references. We have not used generative AI tools to generate the theorem statements (taken from miniF2F) or to formulate mathematical claims; the genuine proofs and failures studied are the outputs of the language-model provers under study, and the synthetic failures are rule-based corruptions of verified proofs made by those scripts. Every reported number is generated by scripts from saved, hash-checked receipts, recorded trajectories or identified summary files; every proof counted as a success was verified by Lean; and references were checked against their primary records with AI assistance. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies automated theorem provers on a public benchmark and involves no human subjects or personal data. All models and data are publicly released under their respective licences. Experiments ran on a single NVIDIA RTX 4090 GPU.

REPRODUCIBILITY STATEMENT

Appendix A gives the provenance and status of every result; Appendix B the prompts and wrapper verbatim; Appendix C the artifact construction and identity audit; Appendix D the statistical procedures and sensitivity analyses; Appendix E complete per-arm, per-block and per-theorem results; and Appendix F the Protocol A methods and results. An anonymous repository (`https://anonymous.4open.science/r/foreign-proof-lean-artifact-FD72`) contains the frozen plans with seeds and job-file hashes, the generation and verification receipts of every study marked re-derived in Appendix A, the recorded Protocol A retry trajectories, and the scripts that regenerate every number, table and figure in this paper. A single command (`python reproduce.py`, CPU only) checks every file hash, checks each receipt against its frozen job, re-derives the principal Protocol B analyses from the receipts and the Protocol A allocation results from the recorded trajectories, and rebuilds every number exactly; the remaining inputs are integrity-checked summary files and the earlier manuscripts' number files, each identified in Appendix A.

REFERENCES

Sam Bowyer, Laurence Aitchison, and Desi R. Ivanova. Position: Don't use the CLT in LLM evals with fewer than a few hundred datapoints. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 81143–81184. PMLR, 2025.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Kuan-Yen Chen, Fang-Yi Su, Shih-Yen Lin, Bao Li, and Jung-Hsien Chiang. The self-correction illusion: Role relabeling gates explicit error flagging in large language models. *arXiv preprint arXiv:2606.05976*, 2026.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=KuPixIqPiq`.

Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, pp. 625–635. Springer, 2021. doi: 10.1007/978-3-030-79876-5_37.

Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall/CRC, 1993.

Wenxiao Fan, Jingling Fu, Fang Li, Luohang Liu, Yu He, Lichen Ma, Zhiyang Yu, Weishan Bi, Junshi Huang, Yan Li, Gu Simiu, and Kan Li. Recompute or reuse? diagnosing and mitigating textual shortcuts in VLM self-reflection. *arXiv preprint arXiv:2608.01930*, 2026.

Emily First, Markus N. Rabe, Talia Ringer, and Yuriy Brun. Baldur: Whole-proof generation and repair with large language models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1229–1241, 2023. doi: 10.1145/3611643.3616243.

Georgi Gerganov and llama.cpp contributors. llama.cpp: LLM inference in C/C++. `https://github.com/ggml-org/llama.cpp`, 2023. Build 2969d6d.

Goedel-LM. Goedel-Prover-V2: official code repository. `https://github.com/Goedel-LM/Goedel-Prover-V2`, 2025. Correction rounds: `src/utils.py`, `DeepSeekCoTHandler.generate_correction_prompt` appends the previous raw model output as an assistant turn followed by the compiler error; commit 2e9036e, accessed 25 Sep 2026.

Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=IkmD3fKBPQ`.

Ali Reza Ibrahimzada, Yang Chen, Ryan Rong, and Reyhaneh Jabbarvand. Challenging bug prediction and repair models with synthetic bugs. In *IEEE International Conference on Source Code Analysis and Manipulation (SCAM)*, 2025.

Mehmet Iscan. Falsification, not exposure: An internally preregistered placebo-controlled decomposition of self-repair feedback in frozen small code models. *arXiv preprint arXiv:2606.31511*, 2026.

Nizar Islah, Istabrak Abbes, Irina Rish, Sarath Chandar, and Eilif B. Muller. Failed reasoning traces tell you what is fixable (but not by reading them). *arXiv preprint arXiv:2606.05145*, 2026.

Anik Jha. Does fault localization beat a fresh attempt? a placebo-controlled study of test-guided code repair. *arXiv preprint arXiv:2609.00854*, 2026.

René Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, and Gordon Fraser. Are mutants a valid substitute for real faults in software testing? In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014. doi: 10.1145/2635868.2635929.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When can LLMs actually correct their own mistakes? A critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1417–1440, 2024.

Guchan Li, Rui Tian, and Hongning Wang. Compile to compress: Boosting formal theorem provers by compiler outputs. *arXiv preprint arXiv:2604.18587*, 2026.

Shuquan Lian, Juncheng Liu, Yazhe Chen, Yuhong Chen, and Hui Li. SWE-AGILE: A software agent framework for efficiently managing dynamic reasoning context. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17536–17550, 2026. doi: 10.18653/v1/2026.findings-acl.868.

Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, Jiayun Wu, Jiri Gesi, Ximing Lu, David Acuna, Kaiyu Yang, Hongzhou Lin, Yejin Choi, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-Prover-V2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=j4C0nALrgK.

Ruikang Liu, Yuxuan Sun, Manyi Zhang, Haoli Bai, Xianzhi Yu, Tiezheng Yu, Chun Yuan, and Lu Hou. Quantization hurts reasoning? An empirical study on quantized reasoning models. In *Second Conference on Language Modeling*, 2025.

Zhuo Liu, Ding Yu, and Hangfeng He. Compiler-guided adaptive proof search with cross-model synergy on context-dependent theorem proving. *arXiv preprint arXiv:2608.18084*, 2026.

David Ma, Kaijing Ma, Shawn Guo, Yunfeng Shi, Enduo Zhao, Jiajun Shi, Zhaoxiang Zhang, Gavin Cheung, Jiaheng Liu, and Zili Wang. OProver: A unified framework for agentic formal theorem proving. *arXiv preprint arXiv:2605.17283*, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. Is self-repair a silver bullet for code generation? In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=y0GJXRungR.

Azim Ospanov, Farzan Farnia, and Roozbeh Yousefzadeh. APOLLO: Automated LLM and Lean collaboration for advanced formal reasoning. *arXiv preprint arXiv:2505.05758*, 2025.

Arjun Panickssery, Samuel R. Bowman, and Shi Feng. LLM evaluators recognize and favor their own generations. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

Cedric Richter and Heike Wehrheim. How to train your neural bug detector: Artificial vs real bugs. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Akshit Sinha, Arvindh Arun, Shashwat Goel, Steffen Staab, and Jonas Geiping. The illusion of diminishing returns: Measuring long horizon execution in LLMs. In *International Conference on Learning Representations (ICLR)*, 2026.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025.

Amitayush Thakur, George Tsoukalas, Yeming Wen, Jimmy Xin, and Swarat Chaudhuri. An in-context learning agent for formal theorem-proving. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=V7HRrxXUhN.

The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pp. 367–381. ACM, 2020. doi: 10.1145/3372885.3373824.

Ken Tsui. Self-correction bench: Uncovering and addressing the self-correction blind spot in large language models. *arXiv preprint arXiv:2507.02778*, 2025.

Yuvraj Verma. Try again, don't look back: Blind resampling outperforms self-repair in small code models. *arXiv preprint arXiv:2607.26117*, 2026.

Evan Wang, Simon Chess, Daniel Lee, Siyuan Ge, Ajit Mallavarapu, Jarod Alper, and Vasily Ilin. Learning to repair Lean proofs from compiler feedback. *arXiv preprint arXiv:2602.02990*, 2026.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, et al. Kimina-Prover preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*, 2025.

Huajian Xin, Z. Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Haowei Zhang, Qihao Zhu, Dejian Yang, Zhibin Gou, Z. F. Wu, Fuli Luo, and Chong Ruan. DeepSeek-Prover-V1.5: Harnessing proof assistant feedback for reinforcement learning and Monte-Carlo tree search. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=I4YAIwrsXa.

Jin Xu, Xiaojiang Liu, Jianhao Yan, Deng Cai, Huayang Li, and Jian Li. Learning to break the loop: Analyzing and mitigating repetitions for neural text generation. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J. Prenger, and Animashree Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 21573–21612, 2023.

Dylan Zhang, Justin Wang, and Tianran Sun. Building a proof-oriented programmer that is 64% better than GPT-4o under data scarcity. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 23101–23118, 2025. doi: 10.18653/v1/2025.findings-acl.1186.

Chuanyang Zheng, Haiming Wang, Enze Xie, Zhengying Liu, Jiankai Sun, Huajian Xin, Jianhao Shen, Zhenguo Li, and Yu Li. Lyra: Orchestrating dual correction in automated theorem proving. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=Svt75kotzs.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. miniF2F: A cross-system benchmark for formal Olympiad-level mathematics. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=9ZPegFuFTFv.

A Provenance and status of results

Every result in the paper belongs to one of two protocols (§3) and has one of three statuses: a *new experiment* run for this paper, a *new analysis* of data recorded earlier, or a *previously reported* Protocol A result included with its methods. Table 5 lists each result, its status and what regenerates it. In the anonymous repository, a single command checks every file hash, checks each receipt against the job hash frozen before generation, re-runs every analysis marked "re-derived" from raw receipts or recorded trajectories, and rebuilds every number in this paper; each regenerated result file matches the shipped one in content, and every number and appendix table is reproduced exactly.

B Prompts

Task prompt (receiver). Every arm ends with the receiver's published whole-proof prompt for the target theorem, unchanged; the fresh-retry arm consists of this prompt alone. For Goedel-Prover-V2 and DeepSeek-Prover-V2 it reads (the last two sentences are single lines in the actual prompt):

Table 5: Provenance of the results in this paper. "Re-derived": regenerated from raw receipts or recorded trajectories by the reproduction command; "summary": the shipped result file is checked for integrity only; "earlier artifact": the raw runs are held by the earlier study's own anonymous artifact, and this repository ships the number files the paper uses.

Result	Protocol	Status	Regenerated
Own versus foreign whole reply, both donors (§4)	B	new experiment	re-derived
Token-matched source blocks (§4)	B	new experiment	receipts; summary
Attribution labels, replication and earlier study (§4)	B	new experiment	re-derived; summary
Repeated-tactic control (§4, Appendix E)	B	new experiment	receipts; summary
Per-theorem consistency; generated tokens per verified retry (§4, §5)	B	new analysis	re-derived
Whole reply versus code, prose and rebuild (§5)	B	new experiment	re-derived
Synthetic versus genuine failures, both receivers (§6)	B	new experiment	re-derived; receipts
Near-miss, distance and edit-position audits (§6)	B	new analysis	summary
Masking the rejected step, Goedel and DeepSeek (§7)	B	new experiment	re-derived
Identity audits (§3, Appendix C)	B	new analysis	summary
Perseveration on the rejected step (§4, §7)	A	previously reported	earlier artifact
Prompt hints versus plain retries (§8)	A	previously reported	earlier artifact
Earlier foreign failure versus fresh retry, single slot (§4, Appendix F)	A	new experiment	summary
Failure-aware allocation rules and their ablations (§8)	A	previously reported	re-derived
Label-free allocation baselines (§8)	A data	new analysis	re-derived
Serving-time decomposition, output cap and GPU energy (§8)	A	previously reported	earlier artifact

````
Complete the following Lean 4 code:

```lean4
<theorem statement> := by

```
Before producing the Lean 4 code to formally prove the given theorem,
provide a detailed proof plan outlining the main proof steps and
strategies.
The plan should highlight key ideas, intermediate lemmas, and proof
structures that will guide the construction of the final formal proof.
````

Failure wrapper. Every non-blind arm places one artifact and its Lean diagnostic before the task prompt (the first paragraph is a single line in the actual prompt):

```
An attempt was not accepted by the Lean 4 compiler. It is reproduced
below exactly as it was produced, together with the compiler output it
produced.

<previous_attempt>
<artifact: whole reply | final code | rebuilt reply | masked code>
</previous_attempt>

<Lean REPL output for exactly that code (JSON messages)>

<task prompt>
```

The prose-only arm uses the same frame with "Its written reasoning is reproduced below exactly as it was produced, with all Lean code blocks removed and without the compiler output" and no diagnostic. In the rebuild study, the artifact is the reply's prose followed by a blank line and its final code block in its original fence. In the masking studies, each masked line is replaced by `-- (line removed)` at its original indentation, and the diagnostic is that of the original, unmasked code. In the label study, only the first sentence differs: "Your previous attempt was not accepted by the Lean 4 compiler. It is reproduced below exactly as you produced it, ..." or "Another prover's attempt was not accepted by the Lean 4 compiler. It is reproduced below exactly as that prover produced it, ...".

Donor generation. DeepSeek failures were generated with the task prompt above. Kimina failures were generated with Kimina's native policy: system message "You are an expert in mathematics and Lean 4.", its native user prompt, temperature 0.6, top-p 0.95, no top-k or min-p truncation, 8,192 output tokens and one server slot.

C ARTIFACTS, ELIGIBILITY AND THE IDENTITY AUDIT

Core artifacts. For each theorem, the foreign artifact is a fixed genuine DeepSeek failure. The own artifact is a genuine Goedel failure selected from recorded failures to minimise the difference in full-prompt Goedel tokens to the foreign prompt, and re-checked by Lean before use; retry outcomes never entered selection. The *code* arm shows the last fenced Lean block, which equals the output of the verifier's extractor for every artifact; the *prose* arm removes every fenced block (26/84 verified; six of the 42 prose artifacts reduce to a heading and are retained and disclosed). Median lengths: own whole 4,697, own code 2,191, own prose 1,845, DeepSeek whole 3,297 and DeepSeek code 552 characters.

Synthetic artifacts. SYN1 applies the first Lean-failing local edit to a verified Goedel proof of the same theorem. SYN8 applies eight disjoint span replacements, each using a line from the same proof, replacing about 97% of the proof body. The original eight-edit rule left SYN8 too close to the correct proof; it was strengthened before any generation so that SYN8's distance to the nearest verified proof overlaps that of genuine failures. Median line similarity to the nearest verified proof over the 42 core theorems: own 0.21, SYN8 0.18, SYN1 0.99. Across recorded context-free failures, 0/345 Goedel and 1/192 DeepSeek failures have similarity ≥ 0.9 to the nearest verified proof, against 35/549 verified proofs relative to another verified proof and 25/48 one-edit synthetic failures. The one-edit failures come from an earlier study on the 49-theorem pool; like the other counts, they are limited to theorems with a verified proof on record.

Kimina donors. A fresh screen generated two Kimina draws for each of 99 theorems (the Goedel and DeepSeek cohorts; 198 draws). A draw was eligible if (E1) Lean rejected it without a timeout, (E2) the extracted code declares the target theorem, (E3) it contains no `sorry` or `admit`, (E4) at least one tactic follows `:= by`, and (E5) a fresh Lean re-check of exactly that code still fails; the shown diagnostic is that re-check's output. The first eligible draw in seed order was used. 34 Goedel-cohort theorems were eligible (frozen gate 30: passed) and 22 DeepSeek-cohort theorems (gate failed; no reverse study was run). Median Kimina lengths: whole 7,479 and code 777 characters.

Goedel donors for DeepSeek. A fresh Goedel screen (two draws for each of DeepSeek's 60 theorems, under the same eligibility rules) verified 105 of 120 draws and solved 56 theorems. Only 7 theorems yielded an eligible Goedel failure, and 4 of those also had an audited DeepSeek own failure, far below the frozen gate of 30. No further screening was done.

Rebuild cohort. Starting from the 42 core theorems, the six heading-only prose artifacts were removed. Other-theorem prose was assigned one to one by sorting the remaining theorems by own-prose token count and swapping prose within adjacent pairs; a pair was kept only if both donor/receiver token ratios lay in $[0.75, 1.333]$. Two pairs failed, leaving 32 theorems; no theorem received its own prose, and there was no re-pairing.

Masking pools. Genuine failures were eligible if Lean's first error spans one to three lines, the error lines map exactly into the shown code, the error is not on the theorem statement, the code has at least ten lines, and an equal-size contiguous block of non-blank body lines exists that avoids the statement and the error lines ± 1. For Goedel this gave 49 failures from 34 theorems; for DeepSeek, one audited failure for each of 31 theorems. The placebo block was drawn with a fixed seeded generator.

Identity audit. Every shown code was re-extracted and re-checked by Lean, and a pass required the fresh error messages (severity, positions and text) to equal the shown diagnostic. All own, SYN1 and SYN8 artifacts passed. 4 of 46 DeepSeek donor artifacts and 15 of 60 saved DeepSeek own failures failed. In every failing case the diagnostic is reproduced exactly when Lean checks the proof body under the benchmark's theorem statement, which is how the original runs verified proofs, whereas the prompt showed the model's own restated statement; line positions therefore do not match the shown text. These artifacts were excluded from all new studies, and Table 9 lists every earlier block that contained the four donor artifacts; its DeepSeek-receiver rows remove only the theorems

of those four donor artifacts, a different rule from the receiver's own-failure audit below. For the DeepSeek-receiver studies, removing the 15 affected theorems changes the own-failure deficit from 14/60 versus 31/60 ($p < .001$) to 13/45 versus 21/45 ($p = .077$) in one study and from 11/60 versus 33/60 ($p < .001$) to 10/45 versus 25/45 ($p = .0015$) in the other; synthetic versus own code moves from 27/60 versus 19/60 to 20/45 versus 15/45.

Copy measure. For each retry, the code the verifier extracted is compared with the shown code after removing comments and blank lines and collapsing whitespace, using a line-level sequence-matching ratio; a near-copy has ratio ≥ 0.9. For whole replies, the shown code is the reply's final Lean block, whatever its fence label. An earlier version of the extractor did not recognise fences labelled `tactics` (used by Kimina) and read prose as code for Kimina whole replies; after the correction, no near-copy count, solve count or test result changed. Table 10 gives all copy statistics. Its identical column compares code after removing comments and normalising whitespace, so it can exceed the character-for-character counts in §4.

D Statistical procedures and sensitivity

Within each seed block, every arm of a theorem shares one sampling seed; blocks use disjoint seeds that were checked against every seed used in earlier runs. For each contrast, each theorem's outcome in an arm is the mean over its available seed blocks, and the statistic is the mean over theorems of the paired difference. The two-sided sign-flip randomisation test draws 200,000 random sign assignments of the theorem-level differences (seed 11); intervals are pointwise percentile bootstraps over theorems (10,000 resamples, seed 7). Holm correction (Holm, 1979) is applied within each study's frozen family: five tests in the core study, three in the Kimina study and three in the rebuild study; the label study and both masking replications each have a single frozen primary contrast. In the masking studies each theorem's units are averaged before testing. Following Bowyer et al. (2025), we avoid normal approximations at these sample sizes. The repeated-tactic control follows its own frozen plan, with a token-level copy measure and different resampling settings (Appendix E). Sensitivity analyses (Table 8) restrict to seed cells where every compared arm is present, remove Lean timeouts (exit code 124) and re-match, or score timeouts in favour of either arm. The whole-reply source contrasts and the Goedel masking benefit remain significant in every scenario. The construction contrast stays positive in every scenario, but when every uncertain timeout is scored against it (+15.5 points, unadjusted $p = .034$) it would not survive Holm correction across the five core tests, so its adjusted significance depends on how those timeouts are resolved. Descriptive analyses (per-theorem consistency, generated tokens per verified retry, edit position) report counts or rank correlations without confirmatory tests.

E Complete results

Table 6 gives every arm by seed block, Table 7 every planned contrast, Table 10 the copy statistics, and Tables 12 and 13 the per-theorem outcomes behind the source and extraction contrasts. Table 14 gives the DeepSeek masking study and the two attribution-label studies, Table 15 the generated tokens per retry and per verified retry, and Table 16 with Figure 4 the per-theorem consistency of the source effect. Every generation and verification receipt was present; no cell is missing in any study.

Attribution labels. The earlier, exploratory label study (one seed block; fresh retry 16/66, after excluding the four audited DeepSeek artifacts) showed the receiver either its own original failure or DeepSeek's failure, each framed as "your previous attempt" or as "another prover's attempt". Under the "your" frame, DeepSeek's failure beat the receiver's own (13/66 versus 3/66, $p = .013$); under the "another prover" frame the two were level (9/66 versus 10/66). That pattern suggested that the label, not only the content, might matter. The frozen replication on the core study's failures did not support it: relabelling the receiver's own failure left recovery unchanged (6/84 versus 6/84). The two studies differ in the failures shown, the earlier one using the receiver's original, longer failures and the replication the length-matched failures of the core study, so a label effect specific to longer failures is not excluded.

Repeated-tactic control. A frozen control on the same 42 theorems, with two new seed blocks, showed the receiver length-matched code that repeats a single tactic (`linarith` or `omega`; each tactic alone fails on every theorem), its own code, its own whole reply, or nothing. Verified retries:

fresh 26/84, own code 18/84, own whole 7/84, repeated `linarith` 26/84 and repeated `omega` 31/84. Near-copies of the shown code: own code 27/84, own whole 55/84 and repeated tactics 0/168. The two frozen tests, Holm-corrected together, compare the repeated-tactic arms with own code (+12.5 points, [+1.8, +23.8]; Holm $p = .047$) and the excess copying of the own code over copying of the repeated tactics (+32.1 points, [+20.2, +44.0]; Holm $p < .001$). With new seeds, the control also reproduces the core ordering own whole $<$ own code $<$ fresh retry. This study used its own frozen measures: a near-copy is a sequence similarity of at least 0.9 between comment-stripped proof bodies at the level of lexical tokens, rather than the line similarity used elsewhere, and its tests used 100,000 sign flips and 20,000 bootstrap resamples (seed 20260925). No near-copy does not mean no repetition: a run of three or more identical tactic lines appeared on 2/42 theorems with repeated `linarith` and 1/42 with repeated `omega`. Excluding the theorems with Lean timeouts (40 remain), the recovery contrast is +12.5 points ([+1.2, +24.4]; unadjusted $p = .058$), so its significance depends on how timeouts are handled; the copying contrast does not depend on verification.

Cost and consistency. Both analyses are descriptive and computed from the same receipts. Generated tokens dominate serving time in this stack: in a phase-resolved replay under Protocol A, decoding took 73.0% of measured time (previously reported; Appendix F). The consistency counts use blocks that share theorems and artifacts, so they are not independent replications, and no test is computed; the net advantage of the foreign reply stays positive (+13 theorem-block cells) after removing the ten theorems that favour it most.

Table 6: Verified retries per arm and seed block; the last column is the theorem-level rate (seeds averaged within theorem) with a pointwise 95% bootstrap interval.

Study	Arm	Block 1	Block 2	Rate % [95% CI]
Core	fresh retry	16/42	16/42	38.1 [26.2, 50.0]
	own whole	4/42	2/42	7.1 [1.2, 14.3]
	own code	10/42	11/42	25.0 [14.3, 35.7]
	own prose	14/42	12/42	31.0 [19.0, 42.9]
	DeepSeek whole	14/42	7/42	25.0 [15.5, 35.7]
	DeepSeek code	8/42	13/42	25.0 [15.5, 35.7]
	SYN1 code	25/42	22/42	56.0 [44.0, 67.9]
	SYN8 code	19/42	18/42	44.0 [32.1, 56.0]
Kimina donor	fresh retry	8/34	8/34	23.5 [11.8, 36.8]
	own whole	3/34	1/34	5.9 [1.5, 11.8]
	own code	10/34	5/34	22.1 [11.8, 33.8]
	Kimina whole	13/34	12/34	36.8 [22.1, 51.5]
	Kimina code	10/34	14/34	35.3 [23.5, 47.1]
Rebuild	fresh retry	9/32	14/32	35.9 [21.9, 50.0]
	own whole	5/32	4/32	14.1 [6.2, 23.4]
	own code	10/32	5/32	23.4 [12.5, 35.9]
	own prose + own code	5/32	4/32	14.1 [6.2, 23.4]
	other prose + own code	6/32	6/32	18.8 [9.4, 29.7]
Masking	fresh retry	19/49	20/49	38.2 [25.0, 51.5]
	own code	14/49	10/49	22.8 [12.5, 34.6]
	rejected masked	19/49	17/49	34.6 [24.3, 45.6]
	placebo masked	9/49	13/49	21.3 [11.8, 32.4]
Masking, exploratory	fresh retry	23/49	–	–
	own code	9/49	–	–
	rejected masked	17/49	–	–
	placebo masked	13/49	–	–

F PROTOCOL A: METHODS AND RESULTS

Status and provenance. Protocol A comprises the earlier experiments of this study. Selected results also appear in the unpublished workshop manuscripts noted in §3; we include their methods and results here and mark reused measurements as previously reported. Two analyses are new: the label-free allocation baselines, and the re-derivation of the previously reported allocation ablations



Figure 4: Per-theorem consistency of the source effect across 6 seed blocks (Goedel receiver; DeepSeek donor in the first four blocks, Kimina donor in the last two). Blue: only the foreign whole reply led to a verified retry; orange: only the own whole reply did. Theorems are sorted by net advantage. The 43 theorems are the 42 core theorems and one Kimina-study theorem whose DeepSeek artifact failed the identity audit and which is also one of the five masking theorems.

from the recorded retry trajectories, both reproduced by the artifact. The allocation results of the two earlier manuscripts replay the same recorded retry trajectories, so they form one body of evidence, not independent replications; the output-cap experiment instead drew three new seed blocks on the same 40 theorems, so it is not an independent theorem population either. Protocol A is never pooled with Protocol B.

Serving and prompts. Protocol A served Goedel-Prover-V2-8B (the same Q4 weights as Protocol B) with a single server slot and attempt-blind retry prompts; the perseveration experiment used an older base prompt containing the clause "commit to your first derivation".

Perseveration (previously reported). The main experiment crossed exposure to the theorem's exact previous completion, shown *without* its diagnostic, with fixed failure-conditioned guidance, on 40 held-out theorems in two seed blocks. The previous completion lowered verification from 9/40 to

Table 7: All planned contrasts (theorem-level points; pointwise 95% bootstrap CI; two-sided sign-flip p; Holm within each frozen family; theorems favouring the first/second arm). Rows without a Holm value are secondary.

Contrast	Effect	95% CI	p	Holm	$+/-$
Core (DeepSeek donor), family of five					
T1 DeepSeek whole − own whole	+17.9	[+8.3, +28.6]	.0028	.014	14/2
T2 DeepSeek code − own code	0.0	[−13.1, +11.9]	1.00	1.00	10/9
T3 source × presentation	+17.9	[+3.6, +32.1]	.030	.089	–/–
T4 SYN8 code − own code	+19.1	[+7.1, +31.0]	.0061	.024	16/4
T5 own prose − fresh retry	−7.1	[−21.4, +6.0]	.41	.82	11/13
SYN1 code − own code	+30.9	[+15.5, +46.4]	<.001	–	23/5
own code − own whole	+17.9	[+7.1, +28.6]	.0043	–	13/2
DeepSeek code − DeepSeek whole	0.0	[−9.5, +9.5]	1.00	–	8/8
own prose − own whole	+23.8	[+8.3, +38.1]	.0053	–	17/3
SYN8 code − SYN1 code	−11.9	[−29.8, +4.8]	.24	–	12/19
own whole − fresh retry	−30.9	[−44.0, −17.9]	<.001	–	2/21
own code − fresh retry	−13.1	[−26.2, 0.0]	.080	–	6/12
own prose − fresh retry	−7.1	[−21.4, +6.0]	.41	–	11/13
DeepSeek whole − fresh retry	−13.1	[−25.0, −1.2]	.072	–	7/15
DeepSeek code − fresh retry	−13.1	[−26.2, 0.0]	.080	–	8/16
SYN1 code − fresh retry	+17.9	[−1.2, +36.9]	.10	–	22/12
SYN8 code − fresh retry	+5.9	[−3.6, +15.5]	.33	–	11/6
Kimina donor, family of three					
Kimina whole − own whole (primary)	+30.9	[+16.2, +45.6]	<.001	.0015	14/1
Kimina code − own code	+13.2	[+1.5, +25.0]	.078	.083	11/4
source × presentation	+17.6	[+2.9, +32.4]	.041	.083	–/–
own code − own whole	+16.2	[+7.4, +26.5]	.0039	–	9/0
Kimina code − Kimina whole	−1.5	[−16.2, +11.8]	1.00	–	9/8
own whole − fresh retry	−17.6	[−30.9, −4.4]	.026	–	3/11
own code − fresh retry	−1.5	[−17.6, +14.7]	1.00	–	9/7
Kimina whole − fresh retry	+13.2	[−1.5, +27.9]	.12	–	9/3
Kimina code − fresh retry	+11.8	[+1.5, +22.1]	.076	–	10/3
Rebuild, family of three					
M1 own prose + code − own code	−9.4	[−20.3, +3.1]	.22	.65	2/8
M2 own − other prose (with own code)	−4.7	[−14.1, +4.7]	.55	1.00	4/7
M3 own whole − rebuilt reply	0.0	[−7.8, +9.4]	1.00	1.00	4/4
own whole − own code	−9.4	[−21.9, +1.6]	.21	–	4/9
other prose + own code − own code	−4.7	[−18.8, +9.4]	.67	–	6/9
own whole − other prose + own code	−4.7	[−15.6, +6.2]	.60	–	2/5
own whole − fresh retry	−21.9	[−34.4, −9.4]	.0033	–	2/14
own code − fresh retry	−12.5	[−26.6, +3.1]	.17	–	4/10
own prose + own code − fresh retry	−21.9	[−37.5, −6.2]	.019	–	3/14
other prose + own code − fresh retry	−17.2	[−31.2, −3.1]	.042	–	4/12
Masking replication, single primary (theorems weighted equally)					
rejected masked − placebo masked (primary)	+13.2	[+2.9, +23.5]	.028	–	12/4
rejected masked − own code	+11.8	[+1.5, +22.8]	.055	–	12/4
placebo masked − own code	−1.5	[−11.8, +8.8]	.89	–	5/6
own code − fresh retry	−15.4	[−27.9, −2.9]	.032	–	5/14
rejected masked − fresh retry	−3.7	[−14.7, +7.4]	.61	–	7/10
placebo masked − fresh retry	−16.9	[−27.9, −7.4]	.0024	–	2/13

2/40 and from 9/40 to 1/40; a post-hoc control that placed another theorem's length-matched failed attempt in the same frame verified 7/40 and 11/40. Tactic-sequence similarity to the shown proof was 0.703 (own attempt), 0.397 (fresh) and 0.375 (another theorem's attempt); 6 retries resubmitted the rejected attempt byte for byte; and the source line of Lean's first error reappeared in 20/21 and 20/21 own-attempt retries against 5/21 and 8/21 fresh retries.

An earlier contrary result. In an earlier exploratory single-slot study on a harder population of 44 theorems, a same-theorem foreign (DeepSeek) failure lowered recovery relative to no failure (4 versus 11 of 44 theorems with one draw; 11 versus 20 with 8 draws, 0 versus 9 discordant theorems); that study also instructed the receiver not to reuse the supplied failure, and had no own-failure arm. Together with §4, the evidence shows foreign failures outperforming own failures, but no established benefit of foreign failures over a fresh retry.

Retry allocation (previously reported; label-free baselines new). We recorded four plain retries for each failed theorem in two held-out blocks of 40 theorems, and for Kimina and DeepSeek failure pools, with byte-reproducible generations, so any allocation policy can be replayed exactly on the same attempts. Table 11 adds label-free caps (new) to the previously reported ablation. A label-free cap either follows plain retrying exactly, and saves nothing, or stops early and loses solves; in these draws, the savings required knowing which failures timed out or were truncated. Plain retrying's

Table 8: Sensitivity of the frozen contrasts: matched seed cells only; Lean timeouts (exit 124) set missing and re-matched; timeouts scored in favour of either arm (bounds).

Contrast and scenario	Effect	95% CI	p
Core (13 uncertain timeouts)			
DeepSeek whole − own whole: matched cells	+17.9	[+8.3, +28.6]	.0029
DeepSeek whole − own whole: timeouts excluded	+19.2	[+9.0, +29.5]	.0016
DeepSeek whole − own whole: timeouts favour DeepSeek whole	+21.4	[+11.9, +32.1]	<.001
DeepSeek whole − own whole: timeouts favour own whole	+17.9	[+8.3, +28.6]	.0028
DeepSeek code − own code: matched cells	0.0	[−13.1, +11.9]	1.00
DeepSeek code − own code: timeouts excluded	0.0	[−12.8, +12.8]	1.00
DeepSeek code − own code: timeouts favour DeepSeek code	0.0	[−13.1, +11.9]	1.00
DeepSeek code − own code: timeouts favour own code	−3.6	[−16.7, +9.5]	.72
SYN8 code − own code: matched cells	+19.1	[+7.1, +31.0]	.0057
SYN8 code − own code: timeouts excluded	+19.2	[+7.7, +30.8]	.0046
SYN8 code − own code: timeouts favour SYN8 code	+21.4	[+9.5, +33.3]	.0015
SYN8 code − own code: timeouts favour own code	+15.5	[+2.4, +28.6]	.034
own prose − fresh retry: matched cells	−7.1	[−21.4, +6.0]	.41
own prose − fresh retry: timeouts excluded	−6.4	[−21.8, +7.7]	.50
own prose − fresh retry: timeouts favour own prose	−3.6	[−17.9, +10.7]	.75
own prose − fresh retry: timeouts favour fresh retry	−7.1	[−21.4, +6.0]	.41
interaction: matched cells	+17.9	[+3.6, +32.1]	.030
interaction: timeouts excluded	+19.7	[+5.3, +35.5]	.025
Kimina donor (3 uncertain timeouts)			
Kimina whole − own whole: matched cells	+30.9	[+16.2, +45.6]	<.001
Kimina whole − own whole: timeouts excluded	+31.8	[+16.7, +47.0]	<.001
Kimina whole − own whole: timeouts favour Kimina whole	+32.4	[+17.6, +47.1]	<.001
Kimina whole − own whole: timeouts favour own whole	+30.9	[+16.2, +45.6]	<.001
Kimina code − own code: matched cells	+13.2	[+1.5, +25.0]	.078
Kimina code − own code: timeouts excluded	+13.6	[+1.5, +25.8]	.078
Kimina code − own code: timeouts favour Kimina code	+13.2	[+1.5, +25.0]	.078
Kimina code − own code: timeouts favour own code	+11.8	[−1.5, +25.0]	.14
interaction: matched cells	+17.6	[+2.9, +32.4]	.041
interaction: timeouts excluded	+18.2	[+3.0, +33.3]	.041
Rebuild (3 uncertain timeouts)			
own prose + own code − own code: matched cells	−9.4	[−20.3, +3.1]	.22
own prose + own code − own code: timeouts excluded	−9.7	[−22.6, +3.2]	.22
own prose + own code − own code: timeouts favour own prose + own code	−9.4	[−20.3, +3.1]	.22
own prose + own code − own code: timeouts favour own code	−10.9	[−23.4, +1.6]	.15
own prose + own code − other prose + own code: matched cells	−4.7	[−14.1, +4.7]	.55
own prose + own code − other prose + own code: timeouts excluded	−4.8	[−14.5, +4.8]	.55
own prose + own code − other prose + own code: timeouts favour own prose + own code	−4.7	[−14.1, +4.7]	.55
own prose + own code − other prose + own code: timeouts favour other prose + own code	−6.2	[−17.2, +4.7]	.39
own whole − own prose + own code: matched cells	0.0	[−7.8, +9.4]	1.00
own whole − own prose + own code: timeouts excluded	0.0	[−7.8, +9.4]	1.00
own whole − own prose + own code: timeouts favour own whole	0.0	[−7.8, +9.4]	1.00
own whole − own prose + own code: timeouts favour own prose + own code	0.0	[−7.8, +9.4]	1.00
Masking (2 uncertain timeouts)			
rejected masked − placebo masked: matched cells	+13.2	[+2.9, +23.5]	.028
rejected masked − placebo masked: timeouts excluded	+13.2	[+2.9, +23.5]	.028
rejected masked − placebo masked: timeouts favour rejected masked	+14.0	[+3.7, +24.3]	.021
rejected masked − placebo masked: timeouts favour placebo masked	+13.2	[+2.9, +23.5]	.028

four-draw union lifted the first-stage count from 119/202 to 146/202; including the shared first stage, the full rules' primary-order saving is 19.5% of end-to-end completion tokens.

Serving cost and output cap (previously reported). In a phase-resolved replay of recovery branches, decoding accounted for 73.0% of measured time and Lean checking for 26.3%. In three new seed blocks on the same 40 held-out theorems, capping the output of previously truncated theorems at 8,192 instead of 12,288 tokens kept the verified set unchanged while saving 9.8–11.4% of completion tokens and 11.2–13.3% of measured GPU energy.

G SERVING CONFIGURATION

All Protocol B generation used llama.cpp build 2969d6d (Gerganov & llama.cpp contributors, 2023) with continuous batching, prompt caching disabled, context shifting disabled (the default) and full GPU offload on one NVIDIA RTX 4090 (24 GB). Goedel-Prover-V2-8B (Q4_K_M) ran with 3 slots of 32,768 tokens; DeepSeek-Prover-V2-7B (Q4_K_M) with 2 slots of 24,576 tokens and streamed decoding; Kimina-Prover-Preview-Distill-7B (Q4_K_M) with one slot. Before each block, a preflight on the live server confirmed that every prompt plus the full 12,288-token output allowance

Table 9: Earlier blocks that contained the four statement-substituted DeepSeek artifacts, with and without the affected theorems (per block, paired by theorem; wins:losses; exact sign test). Designs differ across rows; rows are listed for disclosure and are never pooled.

Block	Contrast	All theorems	Affected theorems removed
tokenmatch46_g	DeepSeek − own (token-matched)	18/46 vs 5/46 (15:2, $p = .0023$)	16/42 vs 5/42 (13:2, $p = .0074$)
tokenmatch46_g	DeepSeek − fresh retry	18/46 vs 17/46 (9:8, $p = 1.00$)	16/42 vs 16/42 (8:8, $p = 1.00$)
tokenmatch46_g	own (token-matched) − fresh retry	5/46 vs 17/46 (3:15, $p = .0075$)	5/42 vs 16/42 (3:14, $p = .013$)
tokenmatch46_g_rep2	DeepSeek − own (token-matched)	13/46 vs 5/46 (9:1, $p = .021$)	13/42 vs 5/42 (9:1, $p = .021$)
tokenmatch46_g_rep2	DeepSeek − fresh retry	13/46 vs 18/46 (6:11, $p = .33$)	13/42 vs 16/42 (6:9, $p = .61$)
tokenmatch46_g_rep2	own (token-matched) − fresh retry	5/46 vs 18/46 (1:14, $p = .0010$)	5/42 vs 16/42 (1:12, $p = .0034$)
tokenmatch46_g_rep3	DeepSeek − own (token-matched)	11/46 vs 5/46 (8:2, $p = .11$)	10/42 vs 5/42 (7:2, $p = .18$)
tokenmatch46_g_rep3	DeepSeek − fresh retry	11/46 vs 14/46 (5:8, $p = .58$)	10/42 vs 12/42 (5:7, $p = .77$)
tokenmatch46_g_rep3	own (token-matched) − fresh retry	5/46 vs 14/46 (3:12, $p = .035$)	5/42 vs 12/42 (3:10, $p = .092$)
nodiag46_g	DeepSeek − own (token-matched)	12/46 vs 3/46 (10:1, $p = .012$)	11/42 vs 3/42 (9:1, $p = .021$)
nodiag46_g	DeepSeek − fresh retry	12/46 vs 16/46 (4:8, $p = .39$)	11/42 vs 15/42 (4:8, $p = .39$)
foreignown	foreign − own	9/46 vs 3/46 (9:3, $p = .15$)	8/42 vs 3/42 (8:3, $p = .23$)
foreignown	foreign − fresh retry	9/46 vs 15/46 (2:8, $p = .11$)	8/42 vs 14/42 (1:7, $p = .070$)
foreignown	own − fresh retry	3/46 vs 15/46 (2:14, $p = .0042$)	3/42 vs 14/42 (2:13, $p = .0074$)
foreignlength	foreign − own (line-matched)	10/46 vs 5/46 (8:3, $p = .23$)	10/42 vs 4/42 (8:2, $p = .11$)
foreignlength	foreign − fresh retry	10/46 vs 17/46 (2:9, $p = .065$)	10/42 vs 16/42 (2:8, $p = .11$)
foreignlength_rep2	foreign − own (line-matched)	12/46 vs 6/46 (11:5, $p = .21$)	11/42 vs 6/42 (10:5, $p = .30$)
foreignlength_rep2	foreign − fresh retry	12/46 vs 16/46 (4:8, $p = .39$)	11/42 vs 15/42 (3:7, $p = .34$)
provlabel	foreign (self label) − own (self label)	13/70 vs 4/70 (12:3, $p = .035$)	13/66 vs 3/66 (12:2, $p = .013$)
provlabel	foreign (other label) − own (other label)	10/70 vs 10/70 (5:5, $p = 1.00$)	9/66 vs 10/66 (4:5, $p = 1.00$)
provlabel	foreign (self label) − foreign (other label)	13/70 vs 10/70 (10:7, $p = .63$)	13/66 vs 9/66 (10:6, $p = .45$)
reciprocal46_ds	Goedel − DeepSeek	0/45 vs 1/45 (0:1, $p = 1.00$)	0/41 vs 1/41 (0:1, $p = 1.00$)
reciprocal46_ds	DeepSeek − fresh retry	1/45 vs 3/45 (1:3, $p = .62$)	1/41 vs 1/41 (1:1, $p = 1.00$)
dsclean	own − fresh retry	11/60 vs 33/60 (3:25, $p < .001$)	11/58 vs 33/58 (3:25, $p < .001$)
dsclean	synthetic − own	32/60 vs 11/60 (25:4, $p < .001$)	32/58 vs 11/58 (25:4, $p < .001$)
dsformatctl	own whole − fresh retry	14/60 vs 31/60 (4:21, $p < .001$)	14/58 vs 31/58 (4:21, $p < .001$)
dsformatctl	own code − own whole	19/60 vs 14/60 (13:8, $p = .38$)	19/58 vs 14/58 (13:8, $p = .38$)
dsformatctl	synthetic code − own code	27/60 vs 19/60 (14:6, $p = .12$)	26/58 vs 19/58 (13:6, $p = .17$)

Table 10: Near-copies of the shown code (line similarity ≥ 0.9 after removing comments and normalising whitespace; for the rebuild study the reference is the own final code), how many near-copies verified, and retries identical to the shown code after normalisation.

Study	Arm	Near-copies	Verified near-copies	Identical
Core	own whole	45/84	0	41/84
	own code	17/84	0	7/84
	DeepSeek whole	1/84	0	1/84
	DeepSeek code	0/84	0	0/84
	SYN1 code	23/84	11	6/84
	SYN8 code	0/84	0	0/84
Kimina donor	own whole	51/68	1	42/68
	own code	24/68	2	15/68
	Kimina whole	0/68	0	0/68
	Kimina code	0/68	0	0/68
Rebuild	own whole	35/64	1	29/64
	own prose + own code	38/64	0	33/64
	other prose + own code	18/64	0	12/64
	own code	15/64	0	10/64
DeepSeek receiver (audited)	own whole	1/45	0	1/45

fits its slot; no theorem was dropped. Batched serving is self-reproducible but not token-identical to single-slot serving, so Protocol A and Protocol B results are never pooled. Lean verification used the Lean 4 REPL (Lean v4.9.0-rc1; Mathlib commit 2f65ba7) with a 180 s limit.

Table 11: Replay of recorded Protocol A retries (four draws per theorem, all 24 draw orders). Entries are retry-stage completion-token savings at the plain-retry solve count under the primary order, or the solves reached when a policy never reaches that count; the last column counts draw orders in which both coarse rules reach it.

Block (solves)	Max 2 retries	Max 3 retries	Skip timeouts	Both coarse rules	Full rules	Orders
Goedel 1 (14)	13 solves	0.0%	10.8%	24.5%	27.5%	12/24
Goedel 2 (15)	13 solves	13 solves	11.3%	34.6%	33.9%	20/24
Kimina 1 (5)	4 solves	0.0%	0.0%	16.4%	26.7%	24/24
Kimina 2 (4)	0.0%	0.0%	0.0%	0.0%	44.2%	24/24
DeepSeek (10)	8 solves	0.0%	0.0%	0.0%	0.0%	24/24

Table 12: Core study, per theorem: verified retries out of two seeds for each arm (DeepSeek donor). Rows are the theorem-level pairs behind every source and extraction contrast.

Theorem	Fresh	Own whole	DeepSeek whole	Own code	DeepSeek code
aime_1983_p3	0	0	0	0	0
aime_1984_p1	1	0	1	1	0
aime_1988_p4	0	0	1	0	0
algebra_2varlineareq_fp3zeq11_3tfm1m5zeqn68_feqn10_zeq7	2	0	2	1	2
algebra_apbon2pownleqapownpbpowon2	1	0	0	0	0
amc12_2000_p15	0	0	0	0	1
amc12_2000_p20	0	0	0	1	1
amc12_2001_p21	2	0	1	0	0
amc12a_2002_p1	1	0	0	0	0
amc12a_2002_p12	0	2	1	1	1
amc12a_2009_p7	1	0	1	1	0
amc12a_2017_p7	0	0	0	0	0
amc12a_2020_p15	0	0	0	0	0
amc12a_2021_p8	0	0	0	0	0
amc12b_2020_p6	0	0	0	0	0
amc12b_2021_p3	1	0	0	0	1
imo_1960_p2	1	1	0	1	0
imo_1962_p2	1	0	0	0	0
imo_1964_p1_1	0	0	1	0	1
imo_1973_p3	0	0	0	0	0
imo_1984_p2	0	0	1	1	1
induction_nfactltnexpnm1ngt3	0	0	0	0	0
induction_seq_mul2pnp1	2	0	1	0	1
induction_sumkexp3eqsumksq	1	0	0	0	0
mathd_algebra_149	1	0	0	1	0
mathd_algebra_184	0	0	0	0	0
mathd_algebra_188	2	0	1	0	1
mathd_algebra_209	1	2	2	1	1
mathd_algebra_313	1	0	1	2	0
mathd_algebra_482	2	0	0	2	1
mathd_algebra_536	1	0	2	2	2
mathd_algebra_69	1	0	2	0	1
mathd_algebra_756	2	1	1	2	2
mathd_numbertheory_227	0	0	0	0	0
mathd_numbertheory_303	0	0	0	1	0
mathd_numbertheory_412	2	0	1	0	2
mathd_numbertheory_42	0	0	0	0	1
mathd_numbertheory_495	0	0	0	0	0
mathd_numbertheory_780	0	0	0	0	1
numbertheory_notequiv2i2jasqbsqdiv8	1	0	1	1	0
numbertheory_prmdvsneqnsqmodpeq0	2	0	0	2	0
numbertheory_sumkmulnckeqnmul2pownm1	2	0	0	0	0
Total	32/84	6/84	21/84	21/84	21/84

Table 13: Kimina donor study, per theorem: verified retries out of two seeds for each arm.

Theorem	Fresh	Own whole	Kimina whole	Own code	Kimina code
aime_1984_p1	0	0	0	1	0
aime_1988_p4	0	0	0	0	0
algebra_2varlineareq_fp3zeq11_3tfm1m5zeqn68_feqn10_zeq7	2	0	2	0	2
algebra_apbon2pownleqapownpbpowon2	0	0	0	0	0
amc12_2000_p15	0	0	0	0	1
amc12_2000_p20	0	0	0	0	1
amc12_2001_p21	1	0	1	2	2
amc12a_2002_p1	0	0	1	0	0
amc12a_2009_p7	1	0	0	1	1
amc12a_2017_p7	0	0	0	0	0
amc12a_2021_p8	0	0	0	1	0
amc12b_2020_p6	0	0	0	0	1
amc12b_2021_p3	0	0	2	0	0
imo_1962_p2	1	0	1	0	0
imo_1964_p1_1	0	0	0	2	1
imo_1973_p3	0	0	0	0	0
imo_1984_p2	1	0	2	1	1
induction_nfactltnexpnm1ngt3	0	0	2	0	0
induction_seq_mul2pnp1	2	0	2	0	2
induction_sumkexp3eqsumksq	0	0	0	0	0
mathd_algebra_149	2	0	1	0	1
mathd_algebra_158	0	0	0	0	0
mathd_algebra_184	0	0	0	0	0
mathd_algebra_188	1	0	2	0	1
mathd_algebra_209	1	0	2	0	1
mathd_algebra_313	0	0	2	1	1
mathd_algebra_482	0	1	2	1	2
mathd_algebra_69	0	1	1	1	1
mathd_numbertheory_227	0	0	0	0	0
mathd_numbertheory_303	0	1	0	1	0
mathd_numbertheory_42	0	0	0	0	1
mathd_numbertheory_495	0	0	0	1	1
numbertheory_prmdvsneqnsqmodpeq0	2	1	2	2	2
numbertheory_sumkmulnckeqnmul2pownm1	2	0	0	0	1
Total	16/68	4/68	25/68	15/68	24/68

Table 14: The frozen DeepSeek masking and label-replication studies, plus the earlier exploratory label comparison. Top: masking with DeepSeek-Prover-V2 as the receiver (31 theorems $\times$ 2 seeds). Bottom: attribution labels on Goedel's own whole failed reply; the earlier exploratory study used the receiver's original failures, the frozen replication the core study's failures.

Study	Arm	Block 1	Block 2	Total
DeepSeek masking	fresh retry	16/31	12/31	28/62
	intact code	10/31	9/31	19/62
	rejected masked	9/31	9/31	18/62
	placebo masked	11/31	14/31	25/62
Rejected − placebo: −11.3 points [−24.2, +1.6], $p = .17$; theorems 3 vs 10				
Labels (replication)	fresh retry	11/42	16/42	27/84
	own whole, neutral frame	6/42	4/42	10/84
	own whole, "your previous attempt"	3/42	3/42	6/84
	own whole, "another prover's attempt"	2/42	4/42	6/84
Another prover's − your label: 0.0 points [−6.0, +6.0], $p = 1.00$ (frozen primary)				
Labels (earlier)	fresh retry	one block		16/66
	own, "your previous attempt"	one block		3/66
	own, "another prover's attempt"	one block		10/66
	DeepSeek, "your previous attempt"	one block		13/66
	DeepSeek, "another prover's attempt"	one block		9/66

Table 15: Generated tokens per retry and per verified retry (descriptive; Goedel receiver). Own whole-reply retries are short because they largely resubmit the shown proof, but they rarely verify.

Study	Arm	Verified	Mean tokens	Hit cap	Tokens per verified
Core	fresh retry	32/84	5,508	16/84	14,460
	own whole	6/84	3,004	3/84	42,062
	own code	21/84	3,846	3/84	15,383
	own prose	26/84	3,354	3/84	10,838
	DeepSeek whole	21/84	3,751	6/84	15,004
	DeepSeek code	21/84	4,874	9/84	19,496
Kimina donor	fresh retry	16/68	5,050	4/68	21,461
	own whole	4/68	3,053	3/68	51,895
	own code	15/68	4,223	5/68	19,143
	Kimina whole	25/68	3,245	4/68	8,827
	Kimina code	24/68	4,428	4/68	12,546
Rebuild	fresh retry	23/64	4,930	5/64	13,717
	own whole	9/64	2,807	2/64	19,958
	own code	15/64	3,528	2/64	15,054
	own prose + own code	9/64	2,833	3/64	20,144
	other prose + own code	12/64	3,555	4/64	18,959

Table 16: Per-theorem consistency of the source effect (descriptive; blocks share theorems and artifacts, so no test is computed). A discordant theorem-block cell is one where exactly one of the two whole replies led to a verified retry.

Block	Donor	Theorems	Foreign only	Own only
Core 1	DeepSeek	42	12	2
Core 2	DeepSeek	42	6	1
Token- matched 1	DeepSeek (token-matched)	42	9	1
Token- matched 2	DeepSeek (token-matched)	42	7	2
Kimina 1	Kimina	34	11	1
Kimina 2	Kimina	34	12	1
All		43	57	8