LEARNING TO SOLVE STEP-LEVEL PROOF GAPS VIA POST-TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

ProofGap (xielihan, 2026) introduces a new step-level formal reasoning task: solving local proof obligations aligned with semantic steps of natural-language solutions. By separating local proof construction from end-to-end theorem proving, this task provides finer-grained assessment of formal reasoning and motivates dedicated post-training research. We develop gap-specific post-training, aiming to solve these obligations more cost-effectively than transferring theorem-level training recipes to the same task. We also study how post-training can teach models the Domain Specific Language (DSL) introduced with ProofGap, a formal proof language unseen by the base models during pretraining. Our method uses a three-role architecture: natural-language drafting for formal proofs, DSL translation that can revise unreliable drafts, and model correction using verifier diagnostics. The drafting model remains frozen, while the translation and correction models receive task-specific post-training. These roles connect local mathematical intent to executable evidence and target efficient learning of the new language. On 4,000 gaps sampled from the ProofGap benchmark, none of which are solved by direct autosolve, our full system achieves a gap-level success rate of 35.70%, compared with 25.0% for the Codex baseline under the same theorem library and verifier, an improvement of 10.70 percentage points.

1 INTRODUCTION

Formal mathematical reasoning is often evaluated by whether a model produces a complete proof of a theorem. Yet applications such as inspecting a natural-language solution, selecting reasoning trajectories, and supervising intermediate reasoning need feedback on individual steps. A theorem-level failure does not reveal which local inferences a model can justify or where it needs help.

ProofGap introduces a new problem of natural-language-aligned step-level formal reasoning, with proof gaps derived from the semantic steps of coherent textbook solutions in natural language. Given an already formalized local context and goal, the task is to justify the corresponding local inference by generating a derivation accepted by the designated checker. Each gap is evaluated independently under its supplied context.

This alignment connects step-level success to individual inferences in the source argument. Gaps extracted from existing formal proofs benefit from an organized proof structure and explicit intermediate states. A natural-language step that appears obvious to a human may require many formal steps, including intermediate claims and side-condition checks. Even with its context and goal formalized, constructing this implicit derivation presents a distinct learning problem. We therefore ask how models should be post-trained specifically to solve natural-language-aligned step-level proof gaps.

A central design choice is how to organize supervision around the obligations being solved. Theorem-level systems such as DeepSeek-Prover-V2 and Goedel-Prover-V2 learn to construct complete Lean proofs through large-scale proof-data synthesis and post-training (Ren et al., 2025; Lin et al., 2025). For natural-language-aligned step-level gaps, we organize training data and learning objectives around the supplied local obligations. Our aim is to concentrate training on the capabilities required for this task, investigating whether gap-specific supervision can support effective gap solving with less training data and computation than transferring theorem-level training recipes.

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.

Designing such supervision requires accounting for the formal language in which these obligations must be solved. The DSL introduced with ProofGap was not encountered by the base models during pretraining. Models must therefore learn to connect local mathematical reasoning with the construction of executable derivations in this language. This requires supervision that links how a goal follows from its premises to how that inference is expressed and checked in the DSL. We build supervision for its operations and proof conventions, using ProofGap's native checker to validate generated derivations.

To connect these learning requirements within each gap, we introduce a drafting–translation–correction architecture with a frozen drafting model and supervision tailored to translation and correction. Model 1 uses a fixed system prompt to produce a structured natural-language draft for a formal proof, making the proposed local argument explicit as guidance for DSL construction. Model 2 learns to turn this guidance into an executable DSL derivation, assessing and revising unreliable drafts as needed. Model 3 learns to correct rejected DSL candidates using verifier diagnostics. We post-train Models 2 and 3 around the same local obligation, assigning each a distinct supervision target while connecting their outputs through the shared task. This organization makes both local proof construction and learning the ProofGap DSL explicit targets of our post-training scheme.

Our contributions are:

1. **A post-training scheme for natural-language-aligned step-level proof gaps.** We develop a post-training scheme centered on the local proof obligations supplied by ProofGap. The scheme addresses the distinct learning problem posed by natural-language-aligned step-level reasoning and targets a better trade-off between gap-solving performance and total training and inference cost than transferring theorem-level training recipes.
2. **A supervision construction pipeline for learning the ProofGap DSL.** We construct gap-level supervision from structured natural-language drafts, checker-accepted DSL derivations, and failed programs paired with verifier feedback and accepted corrections. This supervision connects local mathematical reasoning to executable proofs, targeting data-efficient acquisition of the DSL introduced with ProofGap, which the base models did not encounter during pretraining.
3. **A three-model architecture for solving gaps with a frozen drafter.** We integrate natural-language drafting for formal proofs, translation under unreliable drafts, and verifier-driven model correction, with separate supervision targets for translation and correction and a common local obligation throughout execution. The drafting model uses its existing natural-language reasoning capabilities without task-specific fine-tuning, avoiding a separate supervision-construction and training stage for this role. This separation also offers a design to investigate when adapting models to other unfamiliar formal languages.

2 Related work

Theorem proving and post-training. Benchmarks such as miniF2F evaluate formal proofs of complete mathematical problems (Zheng et al., 2022). DeepSeek-Prover-V1.5 combines proof-assistant feedback with reinforcement learning and search (Xin et al., 2024); DeepSeek-Prover-V2 combines informal reasoning with recursive subgoal decomposition. Kimina-Prover combines lemma-based search with verifier-guided correction. STP expands training through iterative conjecturing and proving (Dong & Ma, 2025). These systems primarily target end-to-end theorem proving, with intermediate reasoning, subgoal solving, and correction supporting the construction of complete proofs. We study gap-level post-training: each task asks the model to fill an individual proof gap under a supplied local context and goal. We organize supervision and evaluation around these independently solved gaps.

Local obligations and natural-language steps. FormalML extracts subgoal-completion tasks from existing Lean proofs and studies expert iteration on them (Yang et al., 2025). Its gaps inherit an organized formal proof, explicit intermediate states, and reference derivations, which facilitate completion and supervision. SAFE's FormalStep formalizes model-generated reasoning steps for retrospective verification and trajectory assessment (Liu et al., 2025). ProofGap derives obligations from semantic steps of textbook solutions. A step obvious to a human may require many formal

steps, auxiliary claims, and side-condition checks. Its boundaries follow the natural-language argument, without an executable source proof to extract as supervision. Completing a formal proof fragment and constructing the implicit derivation behind a natural-language step thus pose different learning problems. We study the latter through checker-validated supervision and gap-specific post-training, with the local context and goal already formalized.

Proof languages. Minilang draws heavily on Isabelle/Isar: it reworks Isar's declarative proof constructs, retains Isabelle's term language, and relies on Isabelle/HOL for proof checking (Xu et al., 2025). Its overall proof style thus remains *Isar-like*, despite a simplified command set and normalized proof structure. Its training corpus is also translated from existing Isar proofs. AoA further explores agent interaction through the abstract syntax tree of a redesigned language (Xu et al., 2026). This establishes substantial continuity with the Isabelle/Isar ecosystem, whereas the DSL introduced with ProofGap has its own representation and checker, without an inherited corpus. We study post-training for this DSL.

Staged and multi-agent formalization. Draft, Sketch, and Prove (DSP) maps informal proofs to formal sketches, leaving intermediate conjectures for automated provers (Jiang et al., 2023). LEGO-Prover couples a prover and an evolver through a growing library of verified lemmas (Wang et al., 2024). MASA composes formalization, critique, and refinement agents for mathematical statements (Zhang et al., 2025). ToMap decomposes natural-language proofs into local units, then formalizes and proves them in Lean; it optimizes decomposition prompts at test time (Liu et al., 2026). LeanDojo studies theorem retrieval (Yang et al., 2023); Goedel-Prover-V2 jointly trains proof generation and self-correction within one prover. Our role-specific post-training keeps the drafter frozen. For natural-language-aligned gaps in ProofGap's DSL, Model 2's SFT teaches draft execution and revision; paired-view RL optimizes joint coverage across draft-conditioned and no-draft generation. Model 3 receives independent supervision from failed programs, verifier diagnostics, and accepted replacements.

3 Post-training a drafting–translation–correction agent

3.1 Preliminary

Task and scope. Following ProofGap, an upstream formalization procedure derives one or more obligations from each semantic step of a natural-language solution. A gap is $\mathcal{G}_i = (\Gamma_i, g_i)$, where Γ_i contains the local assumptions and intermediate propositions supplied by earlier steps, and g_i is the goal. Given this already formalized input, a model must produce a DSL program accepted by verifier $\mathcal{V}$ under theorem library $\mathcal{T}$, establishing $\Gamma_i \vdash g_i$. Each gap is evaluated independently. The source step supplies alignment during construction; the model receives the formal context and goal. Acceptance establishes local derivability under the supplied premises. One semantic step may require several DSL commands.

Formalization and environment. Even an explicit mathematical move requires selecting an applicable theorem, binding its variables, and satisfying side conditions. The native DSL supports theorem instantiation, rewriting, auxiliary assertions, structured subproofs, and solver calls over local proof states; the current library index contains 525 named entries.

3.2 Interfaces and division of work

Figure 1 separates a frozen Qwen3-14B drafter (Model 1), a Qwen3-14B writer trained by SFT and paired-view coverage optimization (Model 2), and an independently supervised Qwen3-14B corrector (Model 3). With initial theorem context $R = \mathrm{Retrieve}(\mathcal{G}, \mathcal{T})$ and fixed drafting prompt σ_1,

$$z \sim P_{\theta_1}(\cdot \mid \mathcal{G}, R, \sigma_1),$$
$$p^{(0)} \sim W_{\theta_2}(\cdot \mid \mathcal{G}, R, \phi(z)), \tag{1}$$
$$p^{(t+1)} \sim C_{\theta_3}(\cdot \mid \mathcal{G}, R^{(t+1)}, \phi(z), p^{(t)}, e^{(t)}). \tag{2}$$

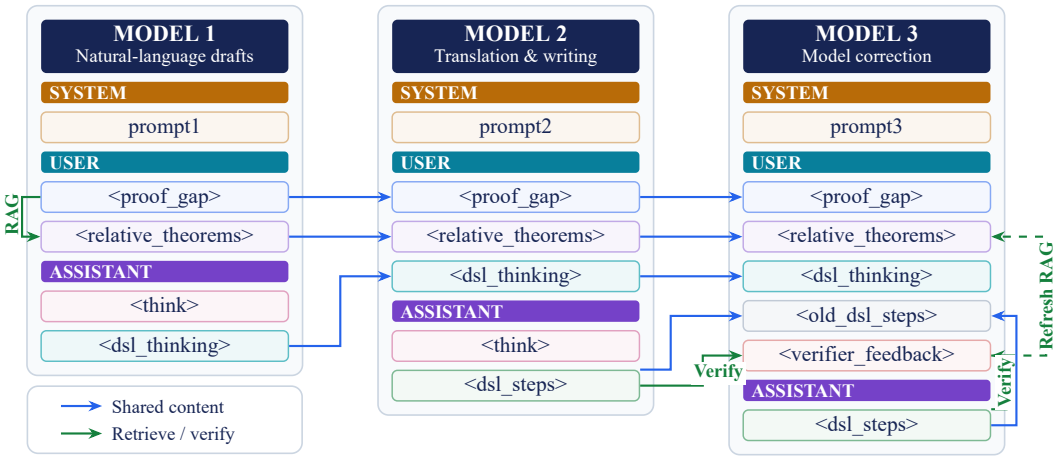

Figure 1: Frozen natural-language drafting with post-trained DSL translation and model correction. The corrector uses verifier feedback and fixed or refreshed theorem context.

Here θ_1 remains fixed, $e^{(t)}$ denotes verifier diagnostics, and ϕ renders action tags as operation hints and expands a retrieved theorem's first citation. No-draft calls omit $\phi(z)$. Model correction produces a complete replacement, checked from the original gap; only executable DSL establishes acceptance.

3.3 Model 1: Eliciting a Structured Natural-Language Proof Draft

Model 1 uses the fixed `new_v1` system prompt with thinking enabled, without task-specific fine-tuning. It may use the given hypotheses, retrieved Top16 theorems, established draft steps, and immediately evident elementary mathematics; invented premises, theorems, witnesses, or results are prohibited. The model-selection comparison is in Section 4.2.

Structure and operational correspondence. After its reasoning, the model returns a delimited `dsl_thinking` block. Goal states the direction and exact conclusion, numbered Routes describe completed actions, and Endpoint explains how the remaining goal closes. The prompt supplies 17 Route tags and two Endpoint tags. Each selects a natural-language action template with slots for mathematical objects, bindings, and results. Templates follow the DSL's operational semantics, including processing the outermost goal structure before its interior. This makes the guidance explicit enough for execution while leaving DSL construction to Model 2. The deterministic adapter supplies command hints, for example B3 to `destruct_and` and G2 to `apply`.

Actual output excerpt. The following retains one Route verbatim from the unadapted 14B model; subsequent theorem instantiation, rewriting, and the Endpoint are omitted.

> **Input excerpt:** hypothesis 3: $x \in \mathrm{RealSet} \wedge 0 \leq x \wedge x \leq 3$; hypothesis 4: $t \in \mathrm{RealSet} \wedge t = \mathrm{sqrtn}(2, x)$; goal: $0 \leq t$.
> **Draft excerpt:** route-1: From hypothesis 3, project $x \geq 0$ from its outermost conjunction, obtaining $x \geq 0$. # B3
> **Rendered hint:** B3 $\mapsto$ `destruct_and`.

The complete deployed prompt, all 19 tag mappings, decoding settings, and full example are included in the accompanying reproducibility material. Model 1 expresses mathematical guidance in natural language, while Model 2 learns to revise erroneous guidance and construct executable proofs. This division avoids a separate supervision-construction and fine-tuning stage for the drafter.

3.4 Model 2: Learning Reliable Execution and Complementary Proof Search

Model 2 learns to generate executable DSL with or without a draft, revising unsupported guidance. SFT teaches execution and draft revision; RL then optimizes joint proof-gap coverage.

SFT views. The corpus contains 8,423 gap–program identities with native-checker-accepted targets and 17,313 presentations before epoch repetition (Table 1). Each response supplies revision or execution reasoning in `think`, followed by a complete `dsl_steps` program; only the latter is executable evidence. Clean drafts and accepted targets support data construction and are unavailable to the deployed writer. Appendix A records sources and training details.

Table 1: Model 2 SFT views, assigned selectively across identities.

Input view	Records	Learning target
No Model 1 draft	8,423	Direct DSL execution
Clean draft	3,367	Execute a valid route
Real-draft repair	5,523	Revise unsupported draft steps

The *bad-plan* repair view uses 5,526 erroneous Model 1 drafts covering 3,514 identities, each supplemented with revision reasoning in `think`. Collapsing three duplicate inputs yields the 5,523 repair presentations. Synthetic gap–DSL expansion labels data provenance; bad-plan supervision labels an input view, and the two can overlap.

Paired-view PKPO. A gap is solved if any budgeted candidate passes. Draft-conditioned and no-draft execution can complement each other: a draft supplies a route, while direct execution can avoid misleading guidance. We extend pass@k policy optimization (PKPO) (Walder & Karkhanis, 2025) to credit a success by its contribution to joint coverage, discounting it when the other view is already likely to succeed.

Let $x = (\mathcal{G}, R)$ and $q_v(x)$ be the writer's acceptance probability under a fixed, unverified draft ($v = \mathrm{p}$) or no draft ($v = \mathrm{n}$). For k conditionally independent attempts per view,

$$J_{\mathrm{pair}}(x) = 1 - (1 - q_{\mathrm{p}}(x))^k (1 - q_{\mathrm{n}}(x))^k. \tag{3}$$

From n sampled responses per view with binary verifier rewards $r_{v,i}$, define $c_v = \sum_{i=1}^{n} r_{v,i}$ and the size-k failure estimate

$$F_v(k) = \frac{\binom{n-c_v}{k}}{\binom{n}{k}}, \qquad \widehat{J}_{\mathrm{pair}} = 1 - F_{\mathrm{p}}(k) F_{\mathrm{n}}(k). \tag{4}$$

PV-PKPO multiplies the binary PKPO SLOO-minus-one coefficient by the other view's estimated failure probability:

$$A_{v,i} = r_{v,i} \frac{\binom{n-c_v}{k-1}}{\binom{n}{k}} F_{\bar{v}}(k), \qquad \bar{\mathrm{p}} = \mathrm{n}, \quad \bar{\mathrm{n}} = \mathrm{p}. \tag{5}$$

We use $\binom{a}{b} = 0$ for $a < b$. The first factor credits within-view contribution; the second introduces cross-view credit. Under independent sampling and a fixed draft, the sequence-score estimator recovers the gradient of joint coverage in expectation (Appendix B).

Training uses $n = 8$ samples and target budget $k = 2$ per view; component evaluation uses 5×2 candidates across four drafts and one no-draft view. Thus the paired objective is a surrogate for the broader evaluation portfolio. The implemented recipe also includes nonzero-signal pair selection, clipped token-level updates, and auxiliary replay; Appendix B distinguishes these from the ideal estimator. Sections 4.3–4.4 evaluate the complete training configurations.

3.5 Model 3: Learning to Correct Failed Proof Programs

Model 3 takes a failed program and its actual verifier feedback and emits a complete replacement. It can revise commands, bindings, or witnesses throughout the program, including earlier decisions that caused the reported failure.

Correction supervision. The corpus contains 18,785 records spanning 8,439 normalized gaps. It combines curated repair pairs, accepted iterative repairs, and persistent failures paired with independently verified same-gap targets. This last source teaches correction of states absent from the earlier corrector's successful trajectories. Selection deduplicates observable inputs and caps records per

training gap. The response contains only `dsl_steps`, with assistant reasoning disabled; an optional original draft remains fallible guidance. Appendix C gives the source accounting, selection rules, and response-normalized loss (Equation 10).

Retrieval and correction loop. Theorems needed for a repair depend on the attempted proof. Trained embedding and reranking models use the gap, optional original draft, current failed DSL, and diagnostics to select Top16 context. Their supervision uses theorem labels associated with verified proofs, but reference proofs and labels are unavailable to inference queries. Appendix C specifies retrieval training and query construction.

Dynamic correction refreshes context before each round. Component experiments use three rounds; the final system combines a five-round schedule with an independent single-round schedule (Section 5.1). The gap and draft remain fixed while the program, diagnostics, and retrieved context evolve. A strict-PASS trajectory stops; attributable failures continue, while inconclusive or unusable results are quarantined. The fixed-context ablation keeps the original Model 2 Top16 throughout the same loop.

4 Component Experiments

4.1 Experimental setup

Data and subsets. We sample 4,909 gaps from ProofGap: native DSL `autosolve` solves 909, leaving 4,000 for evaluation. Unless stated otherwise, component ablations draw from this pool. All 1,000-gap Model 2 ablations use the same D_2, sampled before the compared models' outcomes are known and enriched for easier gaps using historical Codex solvability (58% versus 25% in the source pool). We reuse D_2 for component development and evaluation. The supplied Model 2 SFT/RL and Model 3 training manifests have no gap-ID or source-exercise overlap with this pool. Appendix A.3 gives the manifest audit. Our native DSL evaluation uses the theorem library and verifier binary distributed with ProofGap at commit `7ee2281`.

Protocols. The common writer allocation is 5×2: four draft-conditioned views and one no-draft view, each with two candidates. Bad-plan ablations instead use 4×2. Writer acceptance is union-strict: raw DSL or its declared deterministic preprocessing may pass, counting each candidate once. Correction checks raw DSL and starts from the same frozen SFT failures, comparing fresh writer sampling, fixed-context correction, and dynamic correction. The fresh continuation's raw and preprocessed success sets coincide. Appendix A.5 details checking; Appendix C.4 accounts for correction trajectories.

Metrics. For N gaps, K allocated slots per gap, and m_i accepted candidates, acceptance is $\sum_i m_i/(NK)$ and coverage is $\sum_i \mathbf{1}[m_i > 0]/N$. Missing outputs are failures. On D_2, candidate denominators are 10,000 (5×2) or 8,000 (4×2); the gap denominator is 1,000. Heterogeneous-view unions and adaptive correction report observed coverage, not i.i.d. pass@k. Correction retains the 717 baseline successes and counts newly recovered gaps separately.

4.2 Drafting-model selection

With the same Model 2, four drafts and four candidates per draft (4×4), and no correction, frozen Qwen3-14B reaches 1,137/4,000 (28.425%) versus Qwen3.8-27B's 1,149/4,000 (28.725%). Both enable thinking; the 27B uses medium. The 0.3-point coverage difference supports choosing the frozen 14B drafter in this configuration.

4.3 Model 2 supervision ablations

Synthetic gap–DSL data. Removing 3,448 project-constructed identities reduces coverage from 717 to 687 gaps, despite accepting 15 more candidates (Table 2). Both SFT recipes traverse their datasets twice from the same base model, with update counts scaling with dataset size (Appendix A.2).

Table 2: Writer training on D_2 under the same 5×2 allocation and union-strict acceptance, without Model 3. RL starts from full-data SFT.

Configuration	Covered / 1,000	Accepted / 10,000
SFT, original + external	687 (68.7%)	4,282 (42.82%)
SFT, full data	717 (71.7%)	4,267 (42.67%)
Full SFT + PV-PKPO	768 (76.8%)	5,150 (51.50%)

Targeted bad-plan supervision. Table 3 compares Model 2 SFT with and without erroneous-draft supervision with shared optimizer and adapter settings, using both drafter sizes on D_2. Each gap receives four drafts and two candidates per draft (4×2), excluding no-draft fallbacks, RL, and correction. Training schedules are detailed in Appendix A.2. This ablation removes the 5,523 repair presentations; the synthetic-data ablation instead removes 3,448 gap–program identities.

Table 3: Bad-plan SFT on the same D_2 with 4×2 candidates. The first column identifies the drafter; every writer uses Qwen3-14B.

Draft source	Bad-plan	Covered / 1,000	Accepted / 8,000
Qwen3.8-27B	Retained	690 (69.0%)	3,376 (42.20%)
Qwen3.8-27B	Removed	589 (58.9%)	3,031 (37.89%)
Qwen3-14B	Retained	683 (68.3%)	3,707 (46.34%)
Qwen3-14B	Removed	372 (37.2%)	1,120 (14.00%)

Bad-plan supervision improves coverage by 10.1 points with 27B drafts and 31.1 with 14B drafts. The gap between draft sources narrows from 21.7 to 0.7 points, supporting robustness to misleading guidance and reduced dependence on a larger drafter for these two sources.

4.4 From SFT to paired-view coverage optimization

Under the common 5×2 budget, PV-PKPO increases coverage from 71.7% to 76.8% and candidate acceptance from 42.67% to 51.50% (Table 2). The comparison evaluates the complete PV-PKPO training recipe. Supplementary retrieval transfer appears in Appendix A.4.

4.5 Correction versus fresh generation

The SFT baseline leaves 283 gaps unresolved, supplying 2,510 attributable failed programs. From these frozen failures, we compare six fresh SFT-writer samples per available route with up to three correction rounds (Table 4).

Table 4: Continuation from the same SFT failures. The table reports additional proof programs and recovered gaps.

Continuation	Candidates	Recovered	Coverage / 1,000
None	0	0	717
Fresh writer sampling	7,572	47	764
Correction, fixed Top16	6,901	63	780
Correction, dynamic retrieval	6,898	67	784

Fixed-context correction reaches 78.0% coverage versus fresh sampling's 76.4%; dynamic correction reaches 78.4%, recovering 23.7% of baseline failures. Against fresh sampling, dynamic correction recovers 22 unique gaps and misses two fresh-only successes, a net gain of 20 gaps with 674 fewer programs. Thus correction improves recovery even with fixed context.

Dynamic context adds four recovered gaps over fixed Top16 after three rounds. The round-level comparison is in Table 16, alongside training-duration and workload details (Appendix C.5). An actual whole-program correction that revises an earlier witness choice appears in Appendix C.6. These experiments evaluate continuation from the SFT writer.

5 FULL-SYSTEM EVALUATION

5.1 NATIVE-DSL COMPARISON WITH CODEX ON 4,000 GAPS

Codex (gpt-5.6-sol, medium) and our system use the same DSL environment, library, and verifier on 4,000 non-autosolve gaps. Eight drafts plus two no-draft views receive four SFT step 1084 and four PV-PKPO step 20 candidates each (10×8), solving 1,295 gaps. Model 3 processes the remaining 2,705 with two schedules: up to five rounds of four failed inputs and four candidates each, stopping solved gaps; and an independent single round of eight inputs and four candidates each. These recover 103 and 95 gaps, with 65 shared successes. Their union reaches 1,428/4,000 (35.70%), 10.70 points above Codex.

Table 5: Coverage and mean tokens on 4,000 non-autosolve gaps. Reduced budgets average 40 global position selections. Tokens include all selected drafts and programs, with input counted per candidate.

System	Writer allocation	Coverage (%)	Tokens/gap
Codex (gpt-5.6-sol, medium)	Agent run	25.000	491,750
Draft + writer	5×1	25.812	30,989
Draft + writer	5×2	27.891	45,535
Draft + writer	5×4	29.533	74,644
Draft + writer	10×8	32.375	265,500
Five correction rounds	10×8	34.950	477,359
Five rounds + independent single round	10×8	35.700	608,230

Budget needed to exceed Codex. We retrospectively select four of eight drafts, one of two no-draft views, and k of eight writer positions, shared across gaps and views. Forty hash-ranked selections (seed 20260926) at 5×1 all exceed Codex: mean coverage is 25.812% (range 25.2%–26.2%), using 30,989 tokens per gap, 93.70% below Codex's 491,750. The supplement specifies this frozen-output analysis.

5.2 COMPARISON WITH LEAN PROVERS ON 4,794 GAPS

We exclude 115 `True` placeholders without corresponding Lean formalizations from the 4,909 sampled gaps. Each prover generates eight candidates per retained target, capped at 8,192 tokens. Verification uses `import Mathlib` with a 300-second limit per candidate, rejecting admissions, extra axioms, proof escapes, invalid outputs, and timeouts. Prompts, decoding, and versions are documented in the reproducibility material. For c_i accepted candidates of eight,

$$\widehat{\text{pass@}}\,k = \frac{1}{T}\sum_{i=1}^{T}\left[1 - \frac{\binom{8-c_i}{k}}{\binom{8}{k}}\right], \qquad k \in \{1,3,5,8\}. \tag{6}$$

Table 6 uses $T = 4{,}794$. The cross-environment comparison evaluates each model together with its proof representation and checker.

Table 6: Pass@k (%) on 4,794 gaps with Lean formalizations. DSL rows use the same IDs and report coverage. The retrospective row combines reduced writer outputs with archived corrections from the full 80-candidate pool.

Lean model	pass@1	pass@3	pass@5	pass@8
Goedel-Prover-V2-32B	32.76	40.54	43.04	44.85
DeepSeek-Prover-V2-7B	26.26	35.09	38.27	40.84
Goedel-Prover-V2-8B	24.92	34.67	37.95	40.34
Kimina-Prover-72B	26.36	33.59	36.12	38.03
Kimina-Prover-Distill-8B	23.49	30.96	33.54	35.57
Full system (DSL), both correction schedules				47.48
4×2 writer + archived Model 3 recovery (retrospective)				43.21

Matching IDs retains 887 autosolve and 3,907 non-autosolve gaps, excluding 22 and 93. The full system solves 1,389 of the latter: $(887+1{,}389)/4{,}794 = 47.48\%$, 2.63 points above Goedel-Prover-V2-32B's pass@8 under the respective budgets. The retrospective row averages 40 selections of four drafts and two writer positions, unioned with the 128 matched recoveries from the original 80-candidate workflow. Its 43.21% uses those archived repairs, without rerunning correction from the reduced candidate pool. Codex evaluation retains all 4,000 non-autosolve gaps.

Output truncation. Table 7 covers all 4,909 gaps, including historical reuse and the 115 omitted targets: 39,272 outputs per model. We count `finish_reason="length"` at the 8,192-token cap. DeepSeek-Prover-V2-7B has 19.23% truncated outputs; the others have 39.00%–41.89%.

Table 7: Truncation in the full generation archive. Cells give counts (percentages): output rates use 39,272 candidates; both gap columns use 4,909 gaps, including the 115 omitted from the pass@k table.

Model	Truncated outputs	Gaps: at least one	Gaps: all eight
Goedel-Prover-V2-32B	15,936 (40.58)	3,080 (62.74)	1,024 (20.86)
DeepSeek-Prover-V2-7B	7,551 (19.23)	2,646 (53.90)	164 (3.34)
Goedel-Prover-V2-8B	15,863 (40.39)	3,978 (81.03)	225 (4.58)
Kimina-Prover-72B	16,453 (41.89)	3,140 (63.96)	921 (18.76)
Kimina-Prover-Distill-8B	15,317 (39.00)	3,600 (73.33)	344 (7.01)

5.3 Resource costs and comparison scope

Table 8 covers completed Model 2/3 runs over 9,654 source gap IDs; Model 1 remains frozen.

Table 8: Full training runs. Dataset counts precede repetition; RL records are prompts. Update tokens count non-padding input and output; hours cover training loops on eight NPUs.

Component	Identities	Records	Update tokens	NPU-hours
Model 1 (frozen)	0	0	0	0
Model 2 SFT (3 epochs)	8,423	17,313	102,342,961	82.08
Model 2 RL (20 steps)	3,594	7,188	30,412,138	48.99
Model 3 (3 epochs)	8,439	18,785	210,960,141	175.81

For training-data scale, Goedel-Prover-V2 releases 1.745M SFT records (Goedel-LM, n.d.); Kimina-Prover-72B reports a 6B-token continued-pretraining corpus and roughly 90K RL problems (Numina and Kimi Team, 2025).

Table 9: Mean Codex tokens per gap over all calls (rounded). Cached input/reasoning are subsets of input/output; overall means weight the two group sizes.

Outcome	Gaps	Input	Cached input	Output	Reasoning	Total
Solved	1,000	426,148	413,727	2,321	337	428,469
Unsolved	3,000	510,555	495,103	2,289	451	512,844
Overall	4,000	489,453	474,759	2,297	423	491,750

Tokens use each model's tokenizer; request-shared input accounting is in the supplement. Monetary cost depends on pricing and caching.

6 Conclusion

We study task-specific post-training for natural-language-aligned step-level proof gaps. Reusing a frozen drafter's natural-language reasoning, we train DSL translation and verifier-guided correction, reaching 35.70% on 4,000 non-autosolve ProofGap gaps. This division focuses learning on the proof language and offers a direction for adapting models to unfamiliar formal languages.

AI USE STATEMENT

AI assisted manuscript drafting and editing, literature and project-material summaries, experimental-comparison suggestions and packaging scripts. The research pipeline also uses model-generated drafts, repair explanations, and proof candidates; the deployed generative roles and their training are specified in Section 3. Generated programs are evaluated by the designated checker, and reported aggregates are recomputed from frozen records. The authors retain responsibility for the scientific claims, citations, and submitted artifacts.

REPRODUCIBILITY STATEMENT

Section 3 defines the agent and objectives; Appendices A–C give training and evaluation details. The accompanying supplement contains current-model code snapshots, prompts, run configurations, checkpoint identifiers, aggregate results and token totals, and offline verification scripts. A SHA-256 inventory binds these files. The README maps materials to paper sections and describes arithmetic checks on the released aggregates and the external data, weights, pinned ProofGap environment, and accelerator dependencies needed for training or inference. Historical run identities and remaining configuration gaps are documented explicitly.

ETHICS STATEMENT

The intended use is assistance with mathematical reasoning through checker-validated local derivations. Acceptance is relative to the supplied assumptions, formalization, and checker semantics; mathematical use therefore requires attention to those inputs. The supplementary release contains research code, prompts, software and configuration hashes, and aggregate evaluation results. Source benchmark content and pretrained model weights are obtained from their upstream providers under the applicable distribution terms. Personal workspace paths and server identifiers are removed from the submission copy, while third-party notices and data-source distinctions are preserved.

REFERENCES

Kefan Dong and Tengyu Ma. STP: Self-play LLM theorem provers with iterative conjecturing and proving. *arXiv preprint arXiv:2502.00212*, 2025. doi: 10.48550/arXiv.2502.00212. URL https://arxiv.org/abs/2502.00212.

Goedel-LM. Goedel-Prover-V2 SFT Dataset. Hugging Face dataset card, n.d. URL https://huggingface.co/datasets/Goedel-LM/SFT_dataset_v2. Accessed September 26, 2026; 1,745,010 released records.

Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, Sketch, and Prove: Guiding formal theorem provers with informal proofs. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.12283.

Yilong Li, Suman Banerjee, and Tong Che. Cast a wider net: Coordinated pass@k policy optimization for code reasoning, 2026. URL https://arxiv.org/abs/2605.27000.

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. *arXiv preprint arXiv:2508.03613*, 2025. doi: 10.48550/arXiv.2508.03613. URL https://arxiv.org/abs/2508.03613.

Chengwu Liu, Ye Yuan, Yichun Yin, Yan Xu, Xin Xu, Zaoyu Chen, Yasheng Wang, Lifeng Shang, Qun Liu, and Ming Zhang. Safe: Enhancing mathematical reasoning in large language models via retrospective step-aware formal verification. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12171–12186. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.594. URL https://aclanthology.org/2025.acl-long.594/.

Tian-Shuo Liu, Shiyuan Zhang, Zijie Geng, Haoyu Liu, Runjie Xu, Pengyuan Wang, Lei Yuan, and Yang Yu. Efficient test-time optimization for multi-agent proof autoformalization. *arXiv preprint arXiv:2607.11307*, 2026. doi: 10.48550/arXiv.2607.11307. URL https://arxiv.org/abs/2607.11307.

Numina and Kimi Team. Kimina-Prover: Applying test-time RL search on large formal reasoning models. Hugging Face team article, 2025. URL https://huggingface.co/blog/AI-MO/kimina-prover. Published July 10, 2025.

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. doi: 10.48550/arXiv.2504.21801. URL https://arxiv.org/abs/2504.21801.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. doi: 10.48550/arXiv.2402.03300. URL https://arxiv.org/abs/2402.03300.

Christian Walder and Deep Karkhanis. Pass@k policy optimization: Solving harder reinforcement learning problems. *arXiv preprint arXiv:2505.15201*, 2025. doi: 10.48550/arXiv.2505.15201. URL https://arxiv.org/abs/2505.15201.

Haiming Wang, Huajian Xin, Chuanyang Zheng, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, Zhenguo Li, and Xiaodan Liang. LEGO-Prover: Neural theorem proving with growing libraries. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/85dca46374dc0f27b4bb5f265b3d17f0-Abstract-Conference.html.

xielihan. ProofGap-Benchmark. GitHub repository, 2026. URL https://github.com/xielihan/ProofGap-Benchmark. Main branch, commit 7ee2281146786dfbe20d896f68826c122b1ca484, accessed September 26, 2026.

Huajian Xin, Z. Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, 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. *arXiv preprint arXiv:2408.08152*, 2024. doi: 10.48550/arXiv.2408.08152. URL https://arxiv.org/abs/2408.08152.

Qiyuan Xu, Renxi Wang, Peixin Wang, Haonan Li, and Conrad Watt. A minimalist proof language for neural theorem proving over Isabelle/HOL. *arXiv preprint arXiv:2507.18885*, 2025. doi: 10.48550/arXiv.2507.18885. URL https://arxiv.org/abs/2507.18885.

Qiyuan Xu, Joshua Jun Leang Ong, Renxi Wang, Wenda Li, Haonan Li, Luke Ong, and Conrad Watt. AoA: Theorem proving agent over abstract syntax tree of redesigned language. *arXiv preprint arXiv:2607.16372*, 2026. doi: 10.48550/arXiv.2607.16372. URL https://arxiv.org/abs/2607.16372.

Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2306.15626.

Xiao-Wen Yang, Zihao Zhang, Jianuo Cao, Zhi Zhou, Zenan Li, Lan-Zhe Guo, Yuan Yao, Taolue Chen, Yu-Feng Li, and Xiaoxing Ma. FormalML: A benchmark for evaluating formal subgoal completion in machine learning theory. *arXiv preprint arXiv:2510.02335*, 2025. doi: 10.48550/arXiv.2510.02335. URL https://arxiv.org/abs/2510.02335.

Lan Zhang, Marco Valentino, and Andre Freitas. MASA: LLM-driven multi-agent systems for autoformalization. In *Proceedings of the 2025 Conference on Empirical Methods in Natural*

Language Processing: System Demonstrations, pp. 615–624. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-demos.44. URL https://aclanthology.org/2025.emnlp-demos.44/.

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://arxiv.org/abs/2109.00110.

A MODEL 2 SUPERVISION AND EVALUATION DETAILS

A.1 DATA CONSTRUCTION AND INPUT VIEWS

We distinguish a canonical gap–program identity from a training presentation under a particular input view. The final SFT pool contains 8,423 identities, 8,421 exact gap texts, and 8,416 gap texts after numbering and whitespace normalization. Each identity has a no-draft presentation; clean-draft and real-draft repair presentations are assigned selectively. The 5,523 repair presentations cover 3,514 identities and may contain two different generated drafts for one identity.

The source pool comprises 4,511 original, 464 external-increment, and 3,448 Codex-generated synthetic identities.

Table 10: Training-data sources and the presentations retained by the synthetic-data removal experiment. Synthetic labels the source of a gap–program identity; each source contributes multiple input views.

Source	Identities	No-draft	Clean-draft	Real-draft repair
Original + external	4,975	4,975	1,785	3,067
Project-constructed synthetic	3,448	3,448	1,582	2,456
Full pool	8,423	8,423	3,367	5,523

Accepted targets undergo format, theorem-access, and native-execution checks, together with duplicate and instructional-quality review. Checker acceptance filters the executable target. Clean drafts are constructed from accepted targets. Repair inputs instead use actual Model 1 drafts, so supervision addresses errors produced by the upstream model rather than only artificial corruptions. For example, a draft may request theorem instantiation without its required premises even when another route using available assumptions and a native solver succeeds. The target rationale explains the alternative and the target DSL executes it. This training example illustrates how the writer can replace an unsupported draft route.

Training theorem contexts can be target-derived: in 8,420 of the 8,423 no-draft records, their visible theorem-ID set equals the set referenced by the accepted target. A total of 6,640 contexts are empty. This construction teaches conditional execution under supplied facts. At inference, the writer receives the context materialized by the evaluated pipeline, without adding references from an unavailable reference proof. No-draft means absence of an additional Model 1 draft: 109 historical records also contain a problem-level `METHOD:` hint.

Bad-plan screening and repair reasoning. We review 16,589 available Model 1 drafts and identify 5,526 erroneous attempts covering 3,514 identities. Review follows each proposed route using the full gap, theorem context, draft, and verified target DSL, locating substantive errors in mathematical claims, theorem premises, DSL operations, solver scope, bindings, or proof-state transitions. Valid alternative routes are retained. Each selected attempt receives a new `think` explanation of the error, reusable ideas, a valid alternative, and the relevant state transitions and stopping point. The construction includes self-review, structural checks, and targeted spot checks. The training user contains the gap, theorem context, and original draft; review annotations are used to write the target rationale.

Materialization collapses three duplicate inputs with the same gap, draft, and theorem context, leaving 5,523 repair presentations. The original Top16 order is preserved; missing theorems used by the verified target are appended in 278 of the 5,526 authored attempts. These 278 attempts cover 163 source identities; all target DSL programs and augmented theorem contexts exactly match their selected training presentations. Their acceptance evidence comes from the verified source targets, and the supplemental correspondence ledger records this provenance separately from post-augmentation execution. At inference, theorem context comes from retrieval. Of the retained repair presentations, 2,456 belong to synthetic identities and 3,067 to original or external identities (Table 10). Thus Section 4.3 removes a repair view, while the synthetic-data ablation removes a source category.

A.2 SFT RECIPE AND CHECKPOINT SELECTION

The full-data and data-removal runs independently adapt Qwen3-14B. Their SFT recipe uses LoRA rank/scaling 32/32, all-linear adaptation, AdamW with learning rate 10^{-4}, betas $(0.9, 0.95)$, weight decay 0.01, cosine decay, 0.05 warmup fraction, and gradient clipping at 1.0. The global batch is 32, with microbatch two per NPU on eight Ascend NPUs, BF16 FSDP training, and gradient checkpointing. The full-sequence limit is 24,576 tokens; overlength records are rejected rather than truncated. The supervised output is the assistant rationale, complete DSL, and EOS. The exported implementation masks prompt tokens. The training seed is 42.

Full-data training has 17,313 presentations per epoch, padded by 31 deterministic repeated readings to 17,344 runtime positions, giving 542 updates per epoch. The selected checkpoint at step 1,084 is therefore an epoch-two model. It is selected using the historical 400-gap development evaluation (Table 11). The removal run has 9,827 presentations, padded by 29 to 9,856, and ends after $2\times308 = 616$ updates. The fixed-epoch design compares two traversals of each dataset, allowing the amount of training to scale with the available data.

Table 11: Full-data SFT checkpoint development: four no-draft candidates on 400 development gaps, without Model 3. This protocol differs from the 5×2 component benchmark.

Step	Epochs	Accepted candidates	Covered gaps
1,084	2.0	325/1,600	97/400
1,355	2.5	323/1,600	95/400
1,626	3.0	326/1,600	96/400

Bad-plan removal recipe. Removing the 5,523 repair presentations leaves 11,790 records and preserves their underlying gap–target pool. This run independently adapts Qwen3-14B for two epochs, selecting step 738; both drafter-size evaluations use this same adapter. Full-data SFT plans three epochs and selects epoch two (step 1,084), so the cosine schedules have different total-step horizons. The sequence limits are 24,576 and 16,384, respectively; observed maxima are 17,484 and 13,769, with no truncation. Optimizer, LoRA, batch, and training-seed settings are shared.

A.3 BENCHMARK CONSTRUCTION AND COMPARISON SCOPE

All 1,000-gap Model 2 ablations use this same set, including the bad-plan experiment. The final component benchmark D_2 is randomly sampled from the 4,000-gap non-autosolve test pool defined in Section 4 before the compared models' results are known. Sampling increases the representation of easier obligations, yielding 1,000 frozen identities for subsequent component experiments. The sampling is designed to provide more successful attempts for component comparison. Of the source pool's 4,000 gaps, 1,000 have a historical recorded Codex success (25%). The fixed subset contains 580 such gaps (58%); all 580 have corresponding native PASS records for the same formal assumptions and goals. These historical PASS records provide the difficulty reference for sampling.

Manifest audit. The 4,000 evaluation IDs cover 700 source exercises and 3,991 distinct normalized execution contexts, retaining nine alias IDs. The frozen D_2 contains 1,000 IDs from 489 exercises and agrees with the 5,000 logical writer views and 10,000 corrector candidate slots. For the supplied full-data SFT, synthetic-removal SFT, PV-PKPO pair pool, and Model 3 training manifests, overlap with both evaluation sets is zero by gap ID, source-exercise ID, normalized gap text, and assumption–goal text after removing `METHOD:`. Normalization removes leading gap numbering and whitespace variation.

Retrieval-list checks are reported in Appendix C.3.

Checkpoint and decoding records. The supplied SFT, PV-PKPO, and data-ablation configurations use temperature 0.6, top-p 0.95, an 8,192-token output cap, and thinking enabled. Full-data SFT uses step 1,084 and historical seeds 43/44. PV-PKPO uses step 20 from that parent and retains choices 0/1 from native $n = 4$ sampling with seed 42. The synthetic-removal and new-RAG launchers specify seeds 42/43, using step 616 and step 1,084, respectively. The no-bad-plan runs

use step 738, seeds 42/43, top-$k = 0$, and repetition penalty 1; their actual decoding and engine fields match across the 14B and 27B draft sources. The supplement distinguishes actual generation manifests from launcher-only evidence.

The 4×2 bad-plan table counts draft-conditioned slots. The 27B no-bad-plan receipt additionally contains 311 no-draft fallbacks: its recorded draft-position union of 614 becomes 589 when those fallback slots are excluded. Including the dedicated no-draft route and fallbacks gives 671/1,000 and 3,911/10,000; the corresponding 14B 5×2 run gives 590 and 1,843, with no missing-draft fallbacks. These receipts account for different allocations and do not replace the draft-only table.

Four draft-conditioned views and one no-draft view define ten candidate slots per gap. The component comparisons share the ten-slot allocation and union-strict scoring. Historical full-data SFT lacks generation for 311 missing-draft routes, which count as failures; the no-bad-plan and new-RAG runs materialize all 5,000 views, using no-draft fallback where needed. Seed assignments follow the run records above. Component scores use the union-strict acceptance protocol described in Appendix A.5: each candidate is counted at most once across its raw and permitted-preprocessed checking paths.

Interpretation of the comparisons. The SFT–PV-PKPO comparison evaluates the complete PV-PKPO training configuration on the fixed difficulty-focused development benchmark. GRPO supplies an additional training-configuration reference. The benchmark's sampling design defines the difficulty profile for these component results.

The data-removal experiment compares two dataset traversals under a common evaluation protocol. The fixed-epoch design evaluates the full and reduced training recipes with update counts scaling with dataset size.

The tables report observed scores from the recorded component runs.

GRPO implementation and checkpoint selection. The GRPO configuration (Shao et al., 2024) uses group-mean-centered rewards with standard-deviation normalization disabled. The reported step-20 checkpoint is identified by the authors as the best evaluated within the first 30 updates. Its exact initialization, selection set and metric, and complete training configuration are not specified in the available record.

The recorded GRPO configuration covers 666/1,000 gaps and accepts 4,485/10,000 allocated candidates, compared with SFT's 717 and 4,267. This configuration yields higher candidate acceptance but lower gap coverage than SFT.

A.4 Reusing correction-oriented retrieval for the writer

The updated retrieval module was originally developed for Model 3. We additionally evaluate its use with Model 2 to test whether the retrieved theorem context benefits proof generation without retraining the writer. This supplementary experiment keeps the full-data SFT checkpoint fixed and disables Model 3 correction, evaluating only the writer's outputs under the common 5×2 protocol. The experiment changes the inference theorem context.

New RAG yields 719 covered gaps and 4,698 accepted candidates, compared with 717 and 4,267 for the baseline (Table 12). The new-RAG run accepts 431 more candidates and covers two more gaps. It uses seeds 42/43 and complete view materialization, while the historical baseline uses seeds 43/44 with missing routes counted as failures.

Table 12: Supplementary retrieval transfer to the frozen full-data SFT writer on D_2. Both configurations use five views and two candidates per view, under union-strict scoring; route materialization and seeds follow Appendix A.3. RL and Model 3 correction are disabled.

Configuration	Covered gaps (%)	Accepted candidates (%)
SFT + baseline retrieval	717/1,000 (71.7)	4,267/10,000 (42.67)
SFT + new RAG	719/1,000 (71.9)	4,698/10,000 (46.98)

A.5 NATIVE ACCEPTANCE AND EXECUTION PROTOCOL

Acceptance concerns an executable derivation of the supplied local obligation under the designated native checker. Each candidate is replayed from the original gap; an accepted result requires a completed checking invocation, the strict-success signal, no admits, and closure of all nested obligations. Where a declared preprocessing path is used, the transformed program is recorded separately from the raw response and checked against the same original obligation. We distinguish generated candidates, checked program variants, and covered gaps when reporting results.

Scoring the Model 2 component configurations. The 5×2 Model 2 component tables use recorded union-strict acceptance: a candidate passes if either its raw DSL or its deterministically preprocessed DSL passes the native checker and theorem-access gate. Preprocessing canonicalizes referenced hypothesis descriptors against the tracked state and theorem descriptors against explicitly allowed library entries. It can also remove an explicit named-solver suffix from eligible `autosolve` commands; for an empty scope with an unknown solver name it supplies `all_hypothesis`. The implementation preserves the special `limitcalcsolver` and `sgnpuresolver` suffixes. This fallback can change the effective solver search and is part of the evaluated execution pipeline. The procedure uses the supplied gap, available theorem entries, and generated program, without a reference proof or a replacement obligation. Raw and transformed programs are checked separately when both paths are evaluated, and the union counts each generated candidate at most once. Evaluating both paths can require multiple checker invocations per generated candidate. This definition applies to the component tables; final-system and correction results retain their separately specified evaluation protocols.

Checker and task boundaries. Prompt-visible facts and facts accessible to backend automation are distinct resources. Theorem-access rules and solver behavior are part of the execution pipeline. The reported result is native-checker acceptance of the local goal under its supplied assumptions. ProofGap supplies the task, checking environment, and benchmark construction; this study investigates post-training for that task.

B PAIRED-VIEW COVERAGE OPTIMIZATION

B.1 FROM SINGLE-VIEW PKPO TO CROSS-VIEW CREDIT

For a fixed prompt, the binary SLOO-minus-one PKPO coefficient is

$$A_i^{\mathrm{SV}} = r_i \frac{\binom{n-c}{k-1}}{\binom{n}{k}}, \qquad c = \sum_i r_i. \tag{7}$$

For a successful candidate, the numerator counts subsets in which that candidate is the only success; for a failed candidate the coefficient is zero. Our extension multiplies this within-view contribution by $F_{\bar{v}}(k)$, the failure estimate of the other view (Equation 5). It therefore distinguishes producing an accepted proof from contributing a proof that is useful to the union of routes. No explicit syntax-distance, route-novelty, or length reward is used in the final run.

For example, with $n = 8$ and $k = 2$, a view containing one successful response gives that response coefficient $1/4$ if all eight responses in the other view fail. If six responses in the other view succeed, the coefficient falls to $1/112$. Both proofs receive reward one; the difference is their marginal value to joint coverage. The cross-view term uses verifier outcomes, without a learned reward model or a hand-written proof-text diversity measure.

Gradient identity for paired coverage. All expectations condition on the gap, theorem context, fixed draft, and current policy parameters; responses are sampled independently within and across the two views. If their acceptance probability in view v is q_v, then

$$\mathbb{E}[F_v(k)] = (1 - q_v)^k,$$

and, for the unfiltered samples,

$$\mathbb{E}\left[\sum_i A_{v,i} \nabla_\theta \log \pi_\theta(y_{v,i} \mid x, v)\right] = k(1 - q_v)^{k-1}(1 - q_{\bar{v}})^k \nabla_\theta q_v. \tag{8}$$

Summing both views gives the gradient of Equation 3, even though they share policy parameters. The draft and theorem context are held fixed: no gradient passes through Model 1 or retrieval. This establishes the sequence-score estimator for the stated objective. The clipped token-level training recipe is specified next.

The weighting has useful, explicit boundaries. If one view's sampled successes cover every size-k subset, its estimated failure probability is zero and the other view receives no marginal coverage credit. If both views have no successes, both coefficients vanish. Shared model parameters can also couple the views' behavior.

B.2 Reward, retained pairs, and clipped updates

The configured reward is binary native acceptance subject to response and theorem-access checks. Attributable model failures receive zero; inconclusive timeout or infrastructure failures are retried once and then surfaced rather than intentionally labeled as mathematical failures.

The final launch pool has 3,594 gap pairs and 7,188 prompt rows: one real-Model1-draft row and one no-draft row for each gap. It combines 3,423 main-pool pairs with 171 exploration pairs that had zero baseline successes in both views. The latter can contribute an update only if fresh sampling finds a positive signal. Both rows preserve the same gap and supplied theorem context. Runtime grouping keeps complete pairs and retains a pair only when its joint coefficients contain a nonzero signal. Rejected generations still consume sampling and validation resources.

Let b index retained pairs and $\rho_{b,v,i,t}$ be the new-to-old policy probability ratio for response token t. The practical policy loss is the negative token-mean clipped surrogate,

$$\mathcal{L}_{\text{policy}} = -\frac{1}{T} \sum_{b,v,i,t} \min(\rho_{b,v,i,t} A_{b,v,i}, \text{clip}(\rho_{b,v,i,t}, 0.8, 1.28) A_{b,v,i}) \,, \tag{9}$$

where T counts all unmasked response tokens across the aggregation batch. A response-level coefficient is broadcast to its tokens, without group-mean subtraction or standard-deviation normalization. Filtering, token-mean aggregation, clipping, and auxiliary replay distinguish this practical update from an unbiased population-gradient estimator. The GRPO baseline is described separately in Appendix A.3.

Table 13: Actual final PV-PKPO run: completed 20 main updates from the full-data SFT step-1,084 model. Settings are bound to its launch contract and completion receipt.

Setting	Value
Training pairs / prompt rows	3,594 / 7,188
Samples / target budget	$n = 8$ per view; $k = 2$ per view
Retained trajectories per update	$32 \times (8 + 8) = 512$
Generation batch / maximum refill batches	96 prompt rows / 8
Main updates / PPO epochs per update	20 / 1
Learning rate / LoRA rank / scaling	10^{-6} / 32 / 64
Probability-ratio bounds	$[0.8, 1.28]$
KL penalty	Disabled in reward and actor loss; reference KL monitored
Sampling	Temperature 1.0; top-p 1.0; maximum 4,096 new tokens
Prompt limit / training response tensor limit	8,192 / 8,192 tokens
Training seed	20260911
Coverage replay	Enabled; dynamic-first/static-fill; interval 20 main updates
Novelty bonus	Disabled

Realized sampling volume. Across 20 main updates, each generation batch samples 48 pairs with 16 candidates per pair, giving 960 pair exposures and 15,360 online candidates. There are 723 nonzero-signal pair exposures and 237 zero-signal exposures. The updates retain 640 pairs, or 10,240 trajectories; the remaining 83 active pairs exceed the retained-batch quota. These counts measure repeated sampling exposures from the frozen pool.

B.3 AUXILIARY REPLAY OF FRAGILE SUCCESSES

The run binds a bank of 2,818 verified protector responses spanning 2,344 gap IDs, drawn from low-frequency baseline successes. The replay policy prioritizes dynamically regressed cases and fills remaining eligible selections from the static bank. For cases with one or two baseline successes in eight attempts, dynamic activation requires respectively 24 or 16 consecutive fresh failures; an online success resets the failure evidence. Static fill means replay is not restricted to dynamically triggered regressions.

At a scheduled replay opportunity, let M denote the accumulated token-weighted mean absolute advantage from the main updates. The configured replay coefficient is $a_{\text{rep}} = 0.05M/0.95$, so that $a_{\text{rep}}/(M + a_{\text{rep}}) = 0.05$. This sets the relative replay contribution in the advantage-mass proxy. Replay evaluates fresh log probabilities for stored accepted responses and caps multiplicity at one. At main update 20, one additional replay actor update uses 512 distinct static protectors, with no dynamic selections. The recorded scale is $a_{\text{rep}} \approx 0.0223324$ from $M \approx 0.4243160$.

Attribution and extension scope. Single-view pass@k coefficients are inherited from PKPO; our formulation couples fixed draft-conditioned and no-draft execution through their joint coverage. A related coordinated pass@k method trains strategy planning and code solving jointly (Li et al., 2026). Here the drafting model is fixed during writer RL, and the two views test whether to rely on its fallible guidance. Our experiments assess the implemented PV-PKPO writer configuration.

C MODEL 3: TRAINING DATA, CORRECTION PROTOCOL, AND ABLATIONS

C.1 SELECTED TRAINING CORPUS

The selected corrector is trained independently from pretrained Qwen3-14B. Table 14 describes its 18,785 canonical records. These supervised presentations cover 8,439 normalized gaps, with 2,236 appearing once, 2,060 twice, and 4,143 three times. Selection never pads an individual gap to meet its maximum of three records. The source manifest maps these records to 8,451 gap IDs.

Table 14: Sources of the selected Model 3 correction corpus. Every category supplies a failed program, actual feedback, and a target with native acceptance evidence. Counts precede training-epoch repetition.

Source	Records	Construction
Curated repair pairs	10,536	Retained failure–target supervision from the existing repair pool.
Accepted iterative repairs	6,471	Failed states paired with accepted outputs from an earlier correction model.
Persistent failures with verified targets	1,778	Terminal failed states paired with separately validated same-gap targets.
Total	18,785	8,439 normalized gaps; at most three records per gap.

Accepted iterative examples cover correction rounds 1–6, with 2,310, 1,578, 974, 710, 609, and 290 records, respectively. Persistent-failure examples are drawn from existing terminal states after five unsuccessful correction rounds. Their target programs must pass raw native validation with the exact theorem context supplied to the failed state. Of 20,000 candidate pairings checked during construction, 2,599 pass this validation; 637 whose preprocessing view differs are conservatively excluded, leaving 1,962 eligible pairings before final selection. The selected corpus retains 1,778 of them. This source supervises responses to failures that are absent from the earlier corrector's accepted-output pool. Inclusion requires a compatible, verified target.

For 227 groups with identical observable inputs and multiple target programs, the builder selects one accepted target. It excludes 419 candidate iterative records marked as mechanical repairs. It preferentially retains complementary source categories within each gap. There are 10,856 no-draft and 7,929 draft-conditioned records. The dataset contains no auxiliary Model 2 execution rows and

no supervised assistant reasoning. This removal concerns the standalone execution task; writer-generated failures remain part of repair supervision.

C.2 Interface and optimization

Each user message contains `proof_gap`, `relative_theorems`, an optional `dsl_thinking` draft, `old_dsl_steps`, and `verifier_feedback`. Structured feedback records execution status, error text, the reported instruction and its position, and the preceding proof state when available. The position locates the reported execution failure. Supplied drafts are treated as unverified proposals. The corrector emits the entire replacement inside one `dsl_steps` block, permitting changes before the reported failure and replay from the original context.

The Qwen chat template uses `enable_thinking=false`. Its standard empty reasoning prefix belongs to the masked prompt; the supervised response contains DSL and EOS only. Let x_i contain the input fields, A_i the supervised DSL-envelope and EOS positions, and $\mathcal{B}$ the accumulated examples.

$$\mathcal{L}_3 = -\frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}\frac{1}{|A_i|}\sum_{t\in A_i}\log C_{\theta_3}(y_{it} \mid x_i, y_{i,<t}). \tag{10}$$

The one-example microbatch implementation normalizes loss by that example's supervised token count, then averages the accumulated microbatch losses, as in Equation 10.

Table 15: Configuration of the selected three-epoch Model 3 run.

Setting	Value
Initialization	Pretrained Qwen3-14B; fresh LoRA
Adaptation	Rank 32; alpha 32; all linear target modules
Training targets	DSL envelope and EOS; prompt tokens masked
Optimizer	AdamW; learning rate 10^{-4}; betas $(0.9, 0.95)$; weight decay 0.01
Schedule	Configured cosine decay; warmup fraction 0.05; gradient clipping 1.0
Parallelism	Eight Ascend 910B3 NPUs; BF16; FSDP; sequence parallelism 2
Batch	Global batch 32; one example per data-parallel microbatch
Sequence handling	Limit 28,672; gradient checkpointing; padding removal; overlength rejection
Duration	Three epochs; 588 updates per epoch; 1,764 updates
Seed	20260920
Retained checkpoints	Epoch 2/2.5/3: steps 1,176/1,470/1,764

Each epoch uses 18,816 runtime presentations, comprising 18,785 canonical records and 31 deterministic padding copies. These copies do not add new training identities. Full-sequence token counts have median 3,169, 95th percentile 7,768, 99th percentile 11,582, and maximum 26,396. The run therefore retains its frozen examples without sequence truncation. The 32-example technical-fit panel consists of training records and provides loss-mask and fit diagnostics. There is no separate held-out corrector validation-loss set for this run; checkpoint behavior is evaluated through the correction development experiments below.

C.3 Output-conditioned theorem retrieval

The dynamic arm uses a separately trained Qwen3-Embedding-0.6B retriever and Qwen3-Reranker-0.6B reranker over the fixed 525-entry theorem library. Let q_g contain the gap alone and q_t contain the complete current query, including the optional draft, failed DSL, and verifier feedback. The retriever uses normalized last-token embeddings to score a query and theorem τ by their dot product, $s_E(q, \tau)$. Candidate fusion uses

$$s_{\text{fuse}}(\tau) = \tfrac{1}{2}s_E(q_g, \tau) + \tfrac{1}{2}s_E(q_t, \tau). \tag{11}$$

The two queries share the same embedding model. Their Top64 candidate sets are merged, ranked by s_{fuse}, and reduced to 64 candidates. The reranker scores the top 32 candidates plus explicit draft references that resolve to entries in the library. Up to three such draft references receive guaranteed inclusion; the other slots are filled by reranker score, yielding 16 distinct theorems. When more than three such references occur in the draft, the highest-scoring three receive guaranteed inclusion. The

final list remains in reranker order. Additional draft references may enter through ordinary ranking. Resolved draft references serve as retrieval cues.

Training uses theorem sets extracted from previously accepted proofs as labels. Neither those targets nor their theorem labels enter an inference query. Positives are drawn from the union of known accepted-proof theorem sets for the same query. Retriever negatives are lexically similar library entries outside this union; reranker negatives come from the selected retriever's candidate list. These negatives encode ranking preferences based on observed proofs; other theorems may also support alternative proofs. The reranker relevance score is the difference between its yes and no logits, $s_R(q, \tau)$. For a positive τ^+ and negative τ^-, the losses are

$$\ell_E = \mathrm{softplus}\left(\frac{s_E(q,\tau^-) - s_E(q,\tau^+)}{0.07}\right), \tag{12}$$

$$\ell_R = \mathrm{softplus}\big(s_R(q,\tau^-) - s_R(q,\tau^+)\big). \tag{13}$$

Training weights give each supervised gap equal aggregate weight; reviewed bad-draft views receive twice the within-gap weight of other views. These weights are assigned offline from the annotations.

The retriever and reranker are fully fine-tuned separately from the three generative roles. The recorded training setup uses 31,164 queries with nonempty theorem supervision, pairwise preference losses, global batch 16, learning rate 2×10^{-5}, and three epochs per model. Checkpoint selection on a separate retrieval calibration set chooses retriever epoch 2 and reranker epoch 1. Query-field budgets are 1,536 tokens for the goal, 3,072 for assumptions, 2,048 for the draft, 4,096 for old DSL, and 3,072 for feedback, within a 16,384-token sequence limit. These are retrieval budgets, distinct from the corrector's context limit. The exported retrieval train, calibration, and test lists have no normalized-gap or assumption–goal text overlap with either evaluation pool. Their native gap-group and exercise-group identifiers are also disjoint across retrieval splits. The frozen source mapping covers all 88,604 training, 826 calibration, and 1,756 test query records, corresponding to 7,817, 423, and 909 source gap IDs. Of these, 31,164, 219, and 485 queries have nonempty ranking supervision. The training records comprise 49,883 original-ProofGap and 38,721 project-generated presentations. Source IDs and original numeric exercise IDs are disjoint across retrieval splits and from both proof-evaluation pools. Synthetic generation-batch names are shared across splits; splitting uses the recorded gap and exercise groups. Adding retrieval training and calibration IDs leaves the Model 2/3 source-ID union at 9,654.

The retriever base is Qwen3-Embedding-0.6B at revision `97b0c61`; the reranker base is Qwen3-Reranker-0.6B at `e61197e`. Full revisions and checkpoint identities are recorded in the supplement.

Dynamic retrieval is performed independently for every active trajectory before every repair round. All replacements are checked from the original gap, with theorem-reference permissions taken from that round's selected context. Successful trajectories stop; semantic failures with usable diagnostics provide the next old program and feedback. Timeouts and unusable results are retained in the denominator but quarantined from further semantic correction. Gap-level coverage preserves all previous successes, including successful original writer candidates.

The fixed arm retains the original Model 2 Top16, while the dynamic arm runs the trained retrieval procedure before every round. The two policies recover 62 gaps in common, five unique to dynamic retrieval, and one unique to fixed Top16.

C.4 Development benchmark and accounting

We use the same fixed 1,000-gap component-development set described in Appendix A.3. Initial Codex solving outcomes provide its empirical difficulty reference, with increased representation of gaps solved by Codex. The set remains fixed for subsequent component development and evaluation. The supplied correction-training manifest has zero overlap with both the 1,000-gap and 4,000-gap evaluation pools under the four identity and text checks in Appendix A.3.

The frozen original writer portfolio permits up to four draft-conditioned routes and one no-draft route, with two candidates per available route. Among the 283 gaps without a successful original candidate, the nominal $283 \times 10 = 2,830$ slots contain 2,510 attributable failures with DSL, 306 missing Model 1-route slots, nine outputs without extractable DSL, and five timeouts. We repair all 2,510 usable starts without a per-gap inference cap. The training cap of three records per gap does

Table 16: Cumulative recovered gaps with the selected corrector. All rows start from the same 2,510 failed programs. Training-duration comparisons and candidate-level workloads are in Appendix C.5.

Context	Round 1	Round 2	Round 3
Fixed Top16	49	60	63
Dynamic retrieval	45	64	67

not apply to inference. A gap is recovered when any of its starting trajectories obtains a strict-PASS replacement.

All current correction comparisons use temperature 0.6, top-p 0.95, an 8,192-token output limit, and seeds 42, 43, and 44 for rounds 1–3. The native checker timeout is 30 seconds. Raw extracted DSL must pass the response, theorem-access, and native execution checks; no mechanical program repair contributes to the reported corrector scores. The current comparison uses the same checker and theorem library across the correction arms and the fresh-writer arm. Their recorded binary/library fingerprints begin with `6915e4559040` and `25715009a50a`, respectively.

The fresh-writer continuation uses the same original SFT Model 2 checkpoint at step 1,084, temperature 0.6, top-p 0.95, an 8,192-token output limit, thinking enabled, and a recorded decode seed of 46 with six candidates per available route. Of the nominal 283×5 routes, 1,262 exist; missing drafts are not replaced with new routes. The resulting 7,572 outputs include 157 raw strict-PASS programs and 21 checker timeouts. Preprocessed and raw PASS sets are identical in this run. At least one new candidate succeeds for 101 routes and 47 distinct gaps. We use observed coverage throughout: heterogeneous routes and adaptive repairs are not i.i.d. samples for an unbiased pass@k estimator.

C.5 Training duration and the correction budget

The selected corrector recovers additional gaps over successive rounds (Table 16). Dynamic retrieval trails fixed context after round one and leads after round two. The comparison evaluates the complete retrieval-and-refresh procedure.

Table 17: Retained checkpoints of the selected training run, evaluated with fixed Top16. Entries show cumulative repaired trajectories / recovered gaps. A dash denotes an evaluation that was not run.

Training epochs	Step	Repair round 1	Repair round 2	Repair round 3
2.0	1,176	191/42	—	—
2.5	1,470	219/50	344/58	385/61
3.0	1,764	216/49	340/60	380/63

Epoch 2.5 leads epoch 3 by one recovered gap after a single repair round, whereas epoch 3 leads by two after three rounds. We retain the three-epoch checkpoint for the selected three-round procedure.

Table 18: Actual evaluation workload for the selected checkpoint. Quarantined trajectories remain in the original denominator but do not enter the next round. Counts refer to generated proof candidates.

Context	Round	Inputs	New PASS	Quarantined	Cumulative gaps
Fixed Top16	1	2,510	216	22	49
Fixed Top16	2	2,272	124	29	60
Fixed Top16	3	2,119	40	22	63
Dynamic RAG	1	2,510	215	21	45
Dynamic RAG	2	2,274	135	25	64
Dynamic RAG	3	2,114	45	19	67

The dynamic procedure generates 6,898 replacements and accepts 395 distinct starting trajectories. The trajectory repair fraction is therefore $395/2510 = 15.7\%$; $395/6898 = 5.7\%$ instead measures accepted generated replacements across the evolving pool. Gap recovery is $67/283 = 23.7\%$.

Compared with 7,572 fresh writer candidates, the procedure generates 674 fewer proof candidates while recovering 20 more gaps. Retrieval/reranker execution, prompt lengths, and assistant reasoning vary across these strategies.

C.6 An observed correction before the reported failing instruction

An actual third-round output illustrates why the model emits a complete replacement. In a development gap with $a, b, c > 0$, the relevant goal is

$$\exists x, y, z > 0 : \quad \frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1.$$

This display abbreviates the original formal context. The preceding candidate chose $(x, y, z) = (a, b, c)$ and then attempted to assert the required equation. The native diagnostic reports failure at the assertion, but those witnesses already make the left-hand side equal to three. The corrector changes the earlier witness choice and emits the following complete two-command program:

```
exists (x := {a / 3}) (y := {2 * b / 3}) (z := {2 * c / 3})
autosolve all_hypothesis
```

The new witnesses satisfy $1/9 + 4/9 + 4/9 = 1$ and positivity. The recorded program passes raw native validation without descriptor rewriting or mechanical repair. It illustrates revision of a mathematical choice preceding the detected error and the model–solver division of work. This is an observed model output on the fixed development benchmark.

C.7 Correction evaluation protocol

Conditional recovery and paired accounting. Let D be the fixed component-development set, B its frozen baseline success set, and $U = D \setminus B$. For an evaluated continuation strategy a, let $S_a \subseteq U$ contain gaps with at least one accepted continuation. We report $|S_a|/|U|$ as conditional recovery and $(|B| + |S_a|)/|D|$ as cumulative coverage. A paired comparison with strategy b has net gain

$$|S_a| - |S_b| = |S_a \setminus S_b| - |S_b \setminus S_a|.$$

Thus the dynamic-versus-fresh difference is $22 - 2 = 20$ gaps and the dynamic-versus-fixed difference is $5 - 1 = 4$. These paired set differences compare continuation strategies on the same frozen baseline failures.

Evaluation protocol. All correction configurations use the same 2,510 frozen failed programs from the 283 gaps unsolved by the initial SFT writer. The selected corrector is the epoch-3 checkpoint (step 1,764). We use temperature 0.6, top-p 0.95, an 8,192-token output limit, and a 30-second native-checker timeout. The original gap and draft remain fixed; only the failed program, feedback, and, in the dynamic arm, retrieved theorem context are updated. Every replacement is checked from the original gap and must pass the response-envelope, theorem-access, and native execution checks. Corrections are scored on raw extracted DSL without mechanical repair. Accepted trajectories stop and remain in cumulative coverage; timeouts and unusable results remain in the denominator but do not continue to the next round.

The dynamic run uses two disjoint tensor-parallel shards of size four, a 32,768-token model context, at most 16 concurrent sequences per shard, memory utilization 0.9, and 48 validation workers.

The fresh writer uses the original drafts and theorem contexts, without new retrieval; unavailable routes are not replaced by newly generated drafts. Its raw and preprocessed success sets coincide in this run. Other Model 2 component tables use their declared union-strict scoring protocol, which counts a candidate if its raw or deterministically preprocessed program passes. These policies are reported separately. We preserve the 717-gap baseline success set and add only recovered baseline failures. The continuation budgets are defined by candidate allocations and correction rounds. The dynamic arm additionally performs retrieval and reranking before each round.

Table 19: Inference settings for continuations of the frozen SFT baseline. All three arms use the same checker and theorem library. Candidate counts refer to generated proof programs.

Setting	Fresh writer	Fixed-context correction	Dynamic correction
Checkpoint	M2 SFT step 1,084	M3 step 1,764	M3 step 1,764
Theorem context	Original	Original Top16	Refreshed Top16
Continuation limit	Six per available route	Three rounds	Three rounds
Assistant thinking	Enabled	Disabled	Disabled
Decode seed	46	42/43/44	42/43/44
Generated programs	7,572	6,901	6,898
Raw accepted programs	157	380	395
Newly covered gaps	47	63	67