WHAT SHOULD A SELF-DISTILLATION TEACHER SEE? SOLVER FEEDBACK FOR LOGICAL REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

A theorem prover can check a language model's program and explain what went wrong. Self-distillation lets a teacher copy of the model see privileged context during training while the student never sees it at test time. What should the teacher see? We test three answers on ProverQA with Qwen2.5-7B-Instruct, using Self-Distillation Policy Optimization (SDPO) after a GRPO warm start. A sibling's verified program ties GRPO in a matched 30-step pilot, +0.67 points [−3.17, +4.50] on a 600-item holdout, and trails it by 4.60 points after 150 steps at the same mini-batch size; moving the program between prompt slots changes accuracy by +0.67 points [−0.98, +2.31] in this seed. Prover diagnostics cost 18.07 points against the program and teach the defect they complain about. An uninformative context collapses program generation: a constant string in a 150-step run, and an empty slot in a fresh matched pilot that scored 0 of 600 on each of two datasets with everything but the payload fixed. A pre-run probe of the teacher's preference margin ranks the verified program first, so a context could be screened before a run, though the ranking depends on showing the teacher the failed attempt. But every added context scores below the plain prompt, so the probe is a relative screen, not a certificate. Every arm has one seed and one model. The pilot holdout is the clean evaluation; the 1,500-item test set was reused in development.

1 INTRODUCTION

Take the premises "Every bird is an animal" and "Ada is a bird." A language model can decide whether Ada is an animal by writing a first-order program and letting a theorem prover derive the answer (Pan et al., 2023). The translation is the weak link. When it fails, two kinds of help exist. The prover can report what broke, such as a malformed rule. Or a sibling sample may have succeeded, giving a worked example.

Self-Distillation Policy Optimization (SDPO) (Hübotter et al., 2026) turns such help into gradient. A teacher copy of the model is re-prompted with the privileged context; the student, which never sees it at test time, is trained toward the teacher's next-token distribution. But "privileged context helps" bundles two hypotheses: the teacher may improve because of what the context says, or merely because it was re-prompted at all. We hold the algorithm fixed and ask: what should the teacher see?

We answer with Qwen2.5-7B-Instruct (Team, 2025) on ProverQA (Qi et al., 2025) with Prover9 (McCune, 2005–2010) in the loop. From one GRPO (Shao et al., 2024) warm start we ran 150-step continuations in which the teacher sees a verified sibling program in a demonstration slot or a feedback slot, prover diagnostics, or a constant failure string, plus continued GRPO (Section 3.3). Those arms differ in more than the teacher's payload, so we added a matched pilot from a fresh warm start: 30 steps each of GRPO, of SDPO with a verified program in the feedback slot of the teacher prompt, and of SDPO with the same teacher prompt, which we call the wrapper, and the slot left empty (Sections 3.3 and 4). A pre-run probe scores each context on fixed response pairs (Section 3.4).

An uninformative context collapses program generation, twice. The 150-step constant-string arm ended at 8.73% accuracy, below the 36.27% starting model, with parse failures on 1,216 of 1,500 items (Table 2). The pilot's empty-slot arm scored 0 of 600 on both the ProverQA holdout and ProofWriter (Tafjord et al., 2021) (Table 3). Its training reward fell from 0.52 at step 12 to 0.009 at step 15 and never recovered. Nothing in the loss, mask, optimizer or warm start differed from the program arm. That is worse than no post-training, and the loss gave no warning.

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.

Diagnostics hurt and teach the defect they describe. Prover diagnostics cost 18.07 points relative to the verified program in the same slot. The diagnostic arm produced 81 symbol clashes, a defect the diagnostics report, against 2 with a program. The mechanism is starvation: a diagnostic rarely yields a verified sibling, so 61.6% of batches contribute no distillation gradient, against 17% with a program.

A verified program ties GRPO, and its slot does not matter. In the pilot, program context and GRPO differ by +0.67 points [−3.17, +4.50]. In the 150-step runs, moving the same program between slots is a null, +0.67 [−0.98, +2.31], the same point estimate by coincidence. On ProofWriter, GRPO wins the 150-step comparison by 18.00 points at unmatched mini-batch sizes, and the pilot arms tie. The probe puts the verified program first, as training does, but does not order the weaker contexts as training did, and every added context scores below the plain prompt. It is a screen, not a certificate.

We make three contributions.

- A controlled comparison of what a self-distillation teacher sees: a verified program in two slots, prover diagnostics, and an uninformative payload, against GRPO (Sections 3.3 and 5).
- A replicated collapse: an uninformative context stops program generation within a few steps, in a 150-step run and again in a matched pilot from a fresh warm start where every other setting equals the arm that learns (Section 5.3).
- A pre-run probe of the teacher's preference margin with a plain-prompt reference: a relative screen for contexts, not a certificate (Section 3.4).

Every arm has one seed and one model. The 1,500-item ProverQA test set was reused for validation; the pilot's 600-item holdout is item-disjoint and is the clean evaluation. The pilot is cheap to replicate (Section 6).

2 Related work

Solvers in the loop. Logic-LM (Pan et al., 2023) prompts a language model to write a program in a solver's language and lets a symbolic backend answer. LINC (Olausson et al., 2023) votes over several translations; SatLM (Ye et al., 2023) targets satisfiability solvers. In this line the translator is prompted, not trained. LogicPO (Viswanadha et al., 2025) trains it by preference optimization on a fixed dataset, and PRoSFI (Chen et al., 2026) adds process rewards over structured intermediates. Neither puts a live solver inside the training loop.

Inference-time feedback. Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) revise an answer from feedback at inference time. When the critic is the model itself, reasoning does not improve (Huang et al., 2024). A theorem prover is an external critic, yet in the Logic-LM pipeline gains come from choosing among candidate programs, not from repairing them (Maheshwari & Sharma, 2026); repair from execution feedback hits the same limit (Chen et al., 2024; Olausson et al., 2024). We ask whether that information does more inside training than inside the prompt.

Training-time feedback. Reinforcement learning from verifiable rewards trains a policy against a checkable outcome. GRPO (Shao et al., 2024) is the default algorithm (DeepSeek-AI, 2025); DAPO (Yu et al., 2025) handles groups where every rollout scores the same. The signal is one number. Distillation instead moves the policy toward a teacher's token distribution (Hinton et al., 2015; Agarwal et al., 2024). In SDPO (Hübotter et al., 2026) the teacher is a copy of the policy that sees privileged context, so the context can be any text. This paper is a controlled study of what that context should be, with a matched empty-context control.

3 Method

3.1 Task and reward

A problem is a set of natural-language premises P and a question q with a gold label $y^\star \in \{\text{True}, \text{False}, \text{Uncertain}\}$. Given the prompt $x = (P, q)$, the policy π_θ writes a program z in the Logic-LM surface syntax (Pan et al., 2023). A deterministic front end extracts it, converts it to

Prover9 syntax (McCune, 2005–2010), and runs the prover to obtain $\hat{y}$. A program that does not execute is scored incorrect, with no language-model fallback.

The training reward is

$$R(z) = \begin{cases} 0 & \text{if extraction, execution, or a validity check fails,} \\ 0.1 & \text{if the program is accepted but } \hat{y} \neq y^\star, \\ 1 & \text{if the program is accepted and } \hat{y} = y^\star. \end{cases} \tag{1}$$

The 0.1 tier separates accepted wrong answers from programs that fail. *Accepted* means a program passes every implemented check and earns reward 0.1 or 1; *verified* means it is accepted and reaches the gold label, earning reward 1. Neither means the translation is faithful: a program can omit a relevant premise and still reach the right label.

Checks against shortcuts. Two kinds of executable program earn a label while encoding nothing: a contradictory premise set proves every formula, and a vacuous one, in which every premise is a logical validity, constrains nothing. The scorer rejects both. The contradiction check asks whether both the conclusion and its negation are provable; it catches some inconsistent premise sets, not all. Neither check tests faithfulness.

3.2 Self-distillation from teacher context

We follow SDPO (Hübotter et al., 2026). For each prompt the policy samples eight responses; the reward marks which are verified. A verified response is a *successful sibling* of the others to the same prompt.

The teacher is an exponential moving average of the student, updated at rate 0.01. Student and teacher both score the same *original sampled response*: the student under the plain prompt x_i, the teacher under a context c_i with privileged text. No repaired response is generated. Training moves the student's next-token distribution toward the teacher's for responses with an eligible context (rules in Section 3.3). With $M_i \in \{0, 1\}$ the eligibility mask, the mini-batch loss is a masked token average,

$$\mathcal{L}_{\mathrm{SD}} = \frac{\sum_{i,t} M_i w_{i,t} D_{\mathrm{KL}}(p_\theta(\cdot \mid x_i, z_{i,<t}) \,\|\, p_{\bar{\theta}}(\cdot \mid c_i, z_{i,<t}))}{\sum_i M_i |z_i|}. \tag{2}$$

Padding tokens are excluded; a mini-batch with no eligible response contributes no distillation gradient. Teacher probabilities are stop-gradient targets, $w_{i,t}$ are importance-sampling weights (Appendix A), and both distributions are approximated by the student's top 20 tokens plus a remaining-mass bucket (Appendix A). At test time the student sees only x, so any gain is a change in the policy, not in the prompt.

3.3 Teacher contexts

Every teacher context in Table 1 continues the Logic-LM few-shot prompt: it ends at the same `###` completion cue as the student prompt and contains no chat markup. Verified programs enter a context only in canonical extracted form.

B vs. D: placement, content fixed. Both show the teacher the same verified sibling program. B appends it as one more worked example and then restates the current problem. D writes it after a `Feedback:` heading, preceded by the current failed program when one extracts cleanly. Any gap is a property of the slot and its surrounding text, not of the payload.

D vs. C vs. E: content, placement fixed. All three write into the feedback slot and differ only in what it says. C carries a short structured summary of the failure, never raw prover output: a parse failure gets a short reason category; other failures get the status, predicted label, and gold label. E replaces the diagnostic with the constant `status = failed`. A deficit shared by C and E belongs to the slot; one specific to C belongs to the diagnostics. After a parse error C omits the failed program, so C and E differ slightly in surrounding text.

Table 1: Training conditions (arms). MB is the PPO mini-batch size; a smaller MB takes more optimizer updates per sampled batch, so arms at different MB are not an algorithmic control for each other. A–F are the historical 150-step runs from one shared warm start: B, C, D, and E share MB 1; A and B8 compare objectives at MB 8; F combines the two objectives at MB 1. PG, PP, and PE are the 30-step pilot arms; they share a second, fresh warm start and MB 8. Slot names where the payload sits in the teacher context.

Arm	Objective	Payload	Slot	MB
A	GRPO	— (scalar reward only)	—	8
B	SDPO	verified program	demonstration	1
B8	SDPO	verified program	demonstration	8
D	SDPO	verified program	feedback	1
C	SDPO	structured diagnostics	feedback	1
E	SDPO	constant `status = failed`	feedback	1
F	SDPO + GRPO	verified program	demonstration	1
PG	GRPO	— (scalar reward only)	—	8
PP	SDPO	verified program	feedback, matched wrapper	8
PE	SDPO	empty	feedback, matched wrapper	8

Eligibility differs across the historical arms. B and D make a response eligible when a successful sibling exists; C and E, when feedback exists. A failed response with no successful sibling therefore trains in C or E but not in B or D: the historical comparison changes which tokens are distilled as well as what the teacher sees. F mixes the normalized distillation and GRPO losses with equal weights and uses the B context.

The matched wrapper: PP and PE. The pilot removes this difference. Both arms use one constructor, `matched_feedback_context`, which assembles the context in order: the few-shot history; the current failed program, included by a rule that depends only on the response, never on the payload; a `Feedback:` heading; the payload; the separator; and the repeated target ending at the cue. PP fills the payload with the verified sibling program; PE leaves it empty, heading and separator intact. One eligibility rule, `matched_sibling_programs`, serves both: a response is eligible if it failed and a sibling to the same prompt is verified with a cleanly extracted program; verified responses are never distilled. The payloads differ in length; nothing else differs. PG is GRPO with no privileged context. All three share a fresh warm start and MB 8 (Section 4), so PP versus PE isolates the payload and PP versus PG repeats the A versus B8 comparison at a short budget.

3.4 A PROBE OF TEACHER-CONTEXT QUALITY

Training is an expensive way to test a context, so we also score fixed responses under each context without gradient steps. For a verified response z^+ and a failed response z^- to the same prompt, define

$$m(c) = \frac{\log \pi_{\bar{\theta}}(z^+ \mid c)}{|z^+|} - \frac{\log \pi_{\bar{\theta}}(z^- \mid c)}{|z^-|}, \qquad g(c) = m(c) - m(c_{\text{plain}}). \tag{3}$$

A positive $g(c)$ means the context widens the teacher's length-normalized preference for the verified response; it is not an accuracy. The probe uses the matched wrapper above and varies only the slot: verified program, diagnostic, constant, empty, or a verified program from a different problem, assigned by a permutation with no self-donors. The scored z^+ is never the program shown in context; eight-gram overlap checks text reuse. Every context is measured against the plain prompt; the probe ranks contexts and certifies none. A factorial follow-up also varies the failed-attempt text (Appendix B.6).

4 EXPERIMENTAL SETUP

Data. All training uses ProverQA (Qi et al., 2025). The historical arms report on its 1,500-item test set. Transfer uses the 204-question FOLIO development set (Han et al., 2024) and the 600-question ProofWriter subset (Tafjord et al., 2021) distributed with Logic-LM (Pan et al., 2023); these contain

73 and 457 distinct premise groups. The compact pilot trains on a separate 4,066-item ProverQA training split and evaluates on a frozen 600-item ProverQA holdout (600 premise groups) and on the same 600 ProofWriter items. The holdout shares no item with the 1,500-item test set, the 300 development items, or the probe prompts.

Historical arms. All runs use Qwen2.5-7B-Instruct (Team, 2025) and a verl-based trainer (Sheng et al., 2025). Arms A–F start from one shared archived 30-step GRPO warm start and continue for 150 steps with a fresh optimizer. Each step samples 32 prompts and eight responses per prompt at learning rate 10^{-6} on two H200 GPUs, with a 2,048-token training cap. A and B8 use PPO mini-batch size 8; B, C, D, E, and F use size 1 (Table 1). We report the final checkpoint of each arm. The 300 development items used for validation during training were drawn from the ProverQA test set. They received no gradient updates, but the test numbers are exploratory, not confirmatory.

Compact pilot. A fresh 30-step GRPO warm start (seed 1) is trained from Qwen2.5-7B-Instruct on that training split. Three continuations then run 30 steps each from the same warm weights with a fresh optimizer and a fresh EMA teacher: PG is GRPO, PP places a verified sibling program in the feedback slot of the matched wrapper, and PE keeps the same wrapper with an empty slot. All three use 32 prompts and eight responses per step, learning rate 10^{-6}, mini-batch 8, and a 2,048-token training cap. The warm start and the three final checkpoints are evaluated on the ProverQA holdout and on ProofWriter: greedy, seed 0, 16-token minimum, 1,024-token cap, vLLM 0.12.0, one H200, same strict scorer. No checkpoint selection used either evaluation set. The whole pilot, with two smoke checks and eight evaluations, ran in 6.2 hours on two H200 GPUs, so a further seed costs hours, not weeks. Appendix B.4 records provenance. Absolute pilot accuracies are not comparable to the 150-step arms.

Metrics. The primary metric is accuracy against the gold label, with every non-executable output counted as wrong. The accepted rate is the fraction of outputs that parse, execute under the prover, and pass the validity checks of Section 3. Rejected outputs are classed as parse failure, contradiction, or no answer; Table 6 gives finer classes. Truncated counts outputs that reach the 1,024-token cap; a truncated output can still contain an accepted program. Some historical evaluations ran on other GPU models; the common-device transfer evaluations re-score the fixed checkpoints A–E on FOLIO and A–C on ProofWriter under the same evaluation settings. Training curves (Figure 1) sample four responses per prompt on the 300 development items and are not greedy final scores.

Uncertainty. Historical ProverQA contrasts use paired normal intervals over the 1,500 items (Appendix B.2). FOLIO and ProofWriter contrasts resample premise groups rather than questions: 10,000 paired draws, seed 20260924. Pilot contrasts use the same premise-group bootstrap with 4,000 draws and seed 20260924; the first teacher probe uses 4,000 prompt-bootstrap draws with seed 20260925. All intervals are nominal 95% intervals over items, unadjusted for multiple comparisons. No interval covers training-seed variation: every arm, historical or pilot, is one seed.

5 RESULTS

Table 2 reports the historical 150-step arms on the 1,500-item ProverQA test set. That set was reused during development, so we treat its numbers as validation numbers. The 600-item pilot holdout of Section 5.3 is the clean set. We say which set each number comes from.

5.1 VERIFIED PROGRAMS TIE GRPO AT 30 STEPS AND TRAIL IT AT 150

Post-training works. The base model answers 36.27% of ProverQA correctly and the shared 30-step warm start 66.13%. Continued GRPO (A) reaches 84.67%.

Program context reaches the same level. With a verified sibling program in the feedback slot (D) the student scores 85.20%; in the demonstration slot (B) it scores 84.53%. Both sit within a point of A, but at a different mini-batch size. At GRPO's mini-batch size of 8 and the full 150 steps, B8 scores 80.07%, and A leads it by 4.60 points $[2.42, 6.78]$. In the pilot of Section 5.3, which shares the warm start, mini-batch size and step budget across arms, program context and GRPO tie. Self-distillation with the best context matches scalar-reward GRPO and does not beat it.

Table 2: Historical single-seed ProverQA results, in percent, on 1,500 items. Accepted rate includes the strict validity checks. All continuation arms use 150 steps after a shared 30-step warm start. ProverQA was partly reused during development. All results use a 1,024-token cap; evaluation hardware differed in some runs.

Training condition	MB	Accuracy	Accepted rate
Starting model	–	36.27	49.27
GRPO warm start	8	66.13	85.20
GRPO (A)	8	84.67	92.40
Program demo (B)	1	84.53	90.40
Program demo (B8)	8	80.07	88.53
Program feedback (D)	1	85.20	92.27
Diagnostic (C)	1	67.13	81.07
Constant (E)	1	8.73	9.47
SDPO + GRPO (F)	1	84.73	94.53

Placement is a null. B and D see the same program in different prompt slots; D minus B is $+0.67$ points $[-0.98, +2.31]$.

Adding a GRPO term to the distillation loss (F) changes nothing: 84.73%, $+0.20$ over B.

5.2 Diagnostics teach the defect they describe

Diagnostics hurt. With the prover's structured diagnostics in the same feedback slot (C), the student scores 67.13%, 18.07 points below D and barely above the warm start. Figure 1 shows the shape: C keeps producing accepted programs through training while its validation accuracy falls, so the loss is in what the programs say, not in whether they parse.

The defect is specific. Symbol clashes rise from 2 in B and D to 81 in C (Table 6, Appendix B). Training on the prover's complaints about malformed symbols produced a policy that emits malformed symbols.

Part of the mechanism is coverage. The distillation loss is masked to responses with a verified sibling. Diagnostics rarely yield one, so 61.6% of C's batches contribute no distillation gradient, against 17% for B. The student in C learns from fewer examples, and the teacher that produces them has been reading descriptions of failures.

Length does not explain the gap. D reaches the 1,024-token cap on 1,277 items and C on 1,334, yet D is the most accurate arm.

5.3 An uninformative context collapses program generation

This is the sharpest result in the paper, and it appears twice.

Historical arm E. The constant `status = failed` in the feedback slot (E) ends at 8.73% accuracy with a 9.47% accepted rate, far below the base model's 36.27%. The model did not stop reasoning; it stopped writing programs the parser accepts. E fails to parse on 1,216 of 1,500 items. It emits only 142 accepted programs, and 131 of them are correct (92.25% conditional). Figure 1 shows the timing: E loses accuracy and accepted programs early in the continuation and never recovers, while the other arms keep a high accepted rate. D and E share the slot, the mini-batch size and the step budget, but they also differ in the eligibility rule (Section 3.3); only the pilot isolates the payload.

Pilot replication. The pilot (Section 4) removes the remaining differences. Three 30-step continuations start from one fresh warm start at mini-batch 8: PG is GRPO, PP places a verified sibling program in the feedback slot of the matched wrapper, and PE is the same wrapper with the slot empty. PP and PE share the loss, the eligibility mask, the failed-attempt rule, the optimizer and the warm start. Only the payload differs.

PP ties PG on both sets. On the 600-item ProverQA holdout PP answers 423 correctly and PG 419, $+0.67$ points $[-3.17, +4.50]$; on ProofWriter 419 against 437, -3.00 points $[-7.59, +1.52]$.

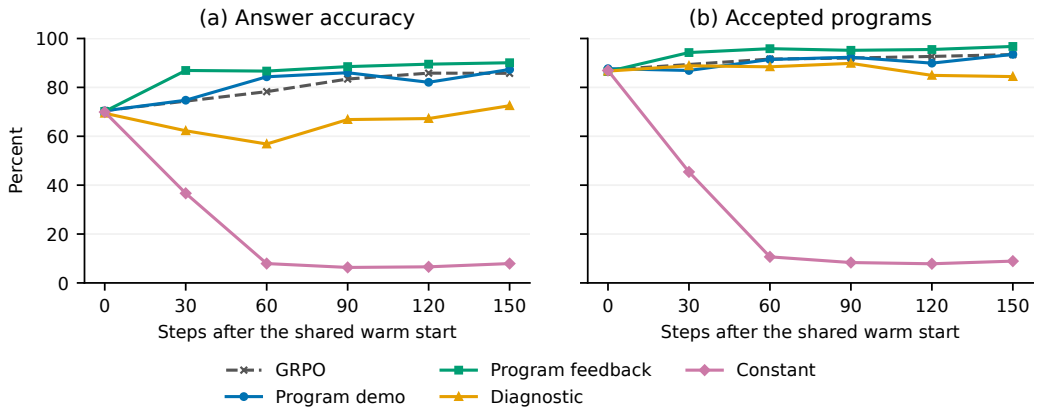


Figure 1: Validation during the 150-step continuation. Each point summarizes four sampled responses on 300 prompts; lines connect recorded checkpoints. Constant feedback (E) loses both accuracy and accepted programs early and never recovers. Diagnostics (C) lose accuracy while keeping a high accepted rate. The curves are from one training seed, use test-derived validation data, and are not directly comparable to greedy final scores.

Table 3: Compact pilot. One training seed; three 30-step continuations from one fresh 30-step GRPO warm start, all at mini-batch 8; greedy decoding with a 1,024-token cap on 600 items per dataset. Differences are percentage points with paired premise-group bootstrap 95% intervals (4,000 draws). Absolute levels are not comparable to the 150-step runs of Table 2. Full counts and provenance: Appendix B.4.

	ProverQA holdout		ProofWriter	
Model	Correct	Accepted	Correct	Accepted
Warm start (30 GRPO steps)	385	545	440	523
PG: GRPO	419	545	437	520
PP: program in slot	423	527	419	480
PE: empty slot	0	0	0	0
PP − PG	+0.67 [-3.17, +4.50]		-3.00 [-7.59, +1.52]	
PP − PE	+70.50 [+66.83, +74.17]		+69.83 [+65.54, +74.07]	
PE − PG	-69.83 [-73.33, -66.17]		-72.83 [-76.71, -68.67]	

PE collapses on both. It answers 0 of 600 on ProverQA and 0 of 600 on ProofWriter. PP minus PE is $+70.50$ points $[+66.83, +74.17]$ and $+69.83$ $[+65.54, +74.07]$. PE finishes below the base model, which answers 183 and 131 of the same 600 items.

The collapse is fast. PE's mean training reward is 0.55 at step 3, 0.71 at step 6, 0.52 at step 12, 0.009 at step 15, and 0.0 at step 30; it never recovers. On the ProverQA holdout its outputs share one signature: 600 of 600 parse failures, 575 of 600 at the 1,024-token cap, and all 600 written in a markdown bullet format with backticks instead of the Logic-LM `: : :` grammar, and many drift into text copied from a few-shot example.

The same loss, mask, optimizer and warm start produced PP, the arm that ties GRPO, and the loss gave no warning. In this seed the payload was the only thing that differed. We report the timing and the signature of the collapse; we do not claim to know its mechanism.

5.4 A CHEAP PROBE RANKS THE PROGRAM CONTEXT FIRST

The probe of Section 3.4 scores the same correct and failed response pair under each teacher context on 63 fixed development prompts and records the teacher's margin: the correct target's mean token log probability minus the failed target's (tables in Appendix B.5).

Program context wins every pairwise comparison. Program minus diagnostic is $+0.0583$ $[+0.0468, +0.0693]$, program minus constant $+0.0567$ $[+0.0450, +0.0681]$, program minus empty $+0.0496$ $[+0.0388, +0.0604]$, and program minus a foreign program from another problem $+0.0353$ $[+0.0251, +0.0454]$. The probe agrees with training on the contrast that matters most: program far ahead of diagnostics, constant and empty. It does not reproduce the large training gap between diagnostics (C) and constant (E): on the probe the two are level, -0.0016 $[-0.0037, +0.0006]$, and diagnostic minus empty is -0.0087 $[-0.0118, -0.0058]$. The probe is consistent with the training outcomes on three contexts; it has not been tested as a predictor.

Every added context scores below the plain prompt. The plain margin is -0.0448; with a program in the wrapper it is -0.1369, and the diagnostic, constant, empty and foreign contexts sit lower still (Appendix B.5, Table 15). The probe is a relative screen among contexts inside one wrapper, not a certificate that a context helps the teacher.

The follow-up (Appendix B.6) removes the failed attempt from the wrapper. The program-versus-empty advantage depends on that text, interaction $+0.0324$ $[0.0218, 0.0437]$, and program minus foreign shrinks to $+0.00571$ $[-0.00522, +0.01742]$ without it, so the program-specific advantage is uncertain once the failed attempt is gone.

5.5 TRANSFER

Table 4: Transfer evaluated sequentially on one H200 with a common 1,024-token cap. Accuracy and accepted rate are percentages; cap reached is a count. FOLIO has 204 questions and ProofWriter has 600. These are new evaluations of the historical single-seed checkpoints, not new training runs.

Dataset	Condition	Accuracy	Accepted	Cap reached
FOLIO	GRPO (A)	47.55	71.08	3
FOLIO	Program demo (B)	50.00	65.20	0
FOLIO	Diagnostic (C)	48.53	73.53	159
FOLIO	Program feedback (D)	47.06	75.00	178
FOLIO	Constant (E)	32.84	50.98	99
ProofWriter	GRPO (A)	78.17	88.67	0
ProofWriter	Program demo (B)	60.17	65.17	1
ProofWriter	Diagnostic (C)	56.67	64.50	495

GRPO transfers better for the historical arms. On 600 ProofWriter items evaluated on one device, A answers 469 correctly (78.17%), B 361 (60.17%) and C 340 (56.67%). B minus A is -18.00 points $[-23.55, -12.67]$ and C minus A -21.50 $[-26.89, -16.13]$. A and B differ in mini-batch size, and D and B8 were not evaluated on ProofWriter, so this compares historical checkpoints, not objectives.

The pilot ties. At matched settings PP minus PG on ProofWriter is -3.00 points $[-7.59, +1.52]$. The best ProofWriter model in the pilot is the warm start itself: 440 of 600, against 437 for PG and 419 for PP. Both continuations reach the 1,024-token cap far more often than the warm start, 377 and 483 outputs against 0 (Table 12), and neither is better out of domain. Thirty more steps of either method lengthened outputs and did not help on ProofWriter in this seed.

FOLIO stays within noise. On 204 items B scores 50.00%, A 47.55%, C 48.53% and D 47.06%; B minus A is $+2.45$ points $[-7.77, +12.96]$. E, the collapsed arm, scores 32.84%. All ten paired contrasts are in Appendix B.3.

6 LIMITATIONS

Every arm is one seed. The historical arms share one archived warm start and the pilot arms share a second, so no effect is shown to be seed-robust. Every arm trains one model, Qwen2.5-7B-Instruct.

The 1,500-item ProverQA test set was reused during development, so the historical accuracies are not clean held-out measurements. The pilot's 600-item holdout was frozen before training and is item-disjoint from the test set, the development items and the probe prompts; its numbers are the clean ones.

The historical SDPO arms differ in more than payload: in which responses are eligible for distillation and in whether the teacher prompt shows the failed attempt. The pilot removes this: PP and PE share both rules and differ only in payload.

We have no mechanism for the collapse. Arm E mostly and arm PE entirely stop producing parseable programs; we report the timing and the signature, not the cause.

The probe is a screen. It ranks the program context first, as training did, but not the weaker contexts among themselves; every added context scores below the plain prompt, and no context has yet been screened before its run.

The reward checks establish acceptance and label agreement, not faithful translation. The guard audit blocks several uninformative programs and exposes one surviving shortcut (Appendix B.11).

Four things would change the paper's confidence: more seeds per arm, a second model family, a mechanism for the collapse, and a prospective test of the probe. The whole pilot, warm start, three arms and eight evaluations, ran in 6.2 hours on two H200 GPUs, so the first two are a few days of compute.

7 Conclusion

What should a self-distillation teacher see? Show it a verified program from a sibling rollout; the slot showed no measurable effect. Do not show it prover diagnostics; the student learns worse and learns the defect they describe. Show it nothing informative, a constant string or an empty slot, and program generation collapses within a few steps.

The collapse is the practical lesson. It happened twice, in independent runs; the loss gave no warning. A practitioner who wires up self-distillation with a weak or empty context gets a model worse than no post-training at all.

With the best context, self-distillation ties GRPO. It does not beat it, in domain or in transfer.

Reproducibility statement

Appendix A documents prompts, training settings, and scoring. Appendix B.2 reports the paired comparisons and bootstrap settings computed from saved predictions. Appendix B.4 records the pilot's job identifiers and archives its resolved specification, source snapshot, and data and checkpoint hashes. Every arm is one training seed, and the historical test set was reused during development; independent replication remains necessary.

Ethics statement

A theorem prover checks consequences of a formal program. It cannot establish that the program faithfully represents a person's words. The reward checks reduce specific invalid shortcuts, so a correct solver answer remains insufficient evidence of reliable natural-language reasoning.

AI use statement

During the preparation of this manuscript, the authors used AI tools solely for grammar correction, language editing, and proofreading. The authors reviewed and verified all AI-assisted edits and take full responsibility for the final manuscript.

References

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *Proc. International Conference on Learning Representations (ICLR)*, 2024.

Luoxin Chen, Yichi Zhou, and Huishuai Zhang. Learning to generate formally verifiable step-by-step logic reasoning via structured formal intermediaries. *arXiv preprint arXiv:2603.29500*, 2026. URL https://arxiv.org/abs/2603.29500.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *Proc. International Conference on Learning Representations (ICLR)*, 2024.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, et al. FOLIO: Natural language reasoning with first-order logic. In *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

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 *Proc. International Conference on Learning Representations (ICLR)*, 2024.

Jonas Hübotter, Frederike Lübeck, Lejs Deen Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026. URL https://arxiv.org/abs/2601.20802.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Devesh Maheshwari and Surbhi Sharma. Beyond self-refinement: Ensembling and chaining for neurosymbolic reasoning. In *ICLR 2026 Workshop on Logical Reasoning of Large Language Models*, 2026. URL https://openreview.net/forum?id=ISQWwRWd0z.

William McCune. Prover9 and Mace4. https://www.cs.unm.edu/~mccune/prover9/, 2005–2010.

Theo X. Olausson, Alex Gu, Benjamin Lipkin, Cedegao E. Zhang, Armando Solar-Lezama, Joshua B. Tenenbaum, and Roger Levy. LINC: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. In *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 5153–5176, 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 *Proc. International Conference on Learning Representations (ICLR)*, 2024.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP*, pp. 3806–3824, 2023.

Chengwen Qi, Ren Ma, Bowen Li, He Du, Binyuan Hui, Jinwang Wu, Yuanjun Laili, and Conghui He. Large language models meet symbolic provers for logical reasoning evaluation. In *Proc. International Conference on Learning Representations (ICLR)*, 2025.

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.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. In *Proc. European Conference on Computer Systems (EuroSys)*, 2025.

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

Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. ProofWriter: Generating implications, proofs, and abductive statements over natural language. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 3621–3634, 2021. doi: 10.18653/v1/2021.findings-acl.317. URL https://aclanthology.org/2021.findings-acl.317/.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2025.

Koushik Viswanadha, Deepanway Ghosal, and Somak Aditya. LogicPO: Efficient translation of NL-based logical problems to FOL using LLMs and preference optimization, 2025. URL https://arxiv.org/abs/2506.18383.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. SatLM: Satisfiability-aided language models using declarative prompting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

A Implementation and reproducibility details

A.1 Prompt construction

All student prompts use the completion-style Logic-LM format and end in `###`. The tokenizer uses a pass-through template rather than adding chat role markers. Teacher contexts end with the target problem and the same completion cue; some also include an earlier target block.

The following schematic describes the actual distinction between the main contexts. History contains the original few-shot examples; target contains the current natural-language premises and question.

Context	Text sequence before scoring the original response
Plain	History; target; completion cue.
B	History; target; successful program; separator; target; completion cue.
D	History; optional target and bounded failed attempt; feedback heading; successful program; separator; target; completion cue.
C	History; optional target and bounded failed attempt; feedback heading; structured status fields; separator; target; completion cue.
E	History; optional target and bounded failed attempt; feedback heading; `status = failed`; separator; target; completion cue.
PP	History; optional target and bounded failed attempt; feedback heading; successful program; separator; target; completion cue.
PE	Same as PP with an empty payload.

C omits the failed program when its diagnostic begins with `status = parse_error`; D and E do not use that same diagnostic-based omission rule. Unavailable or unsafe failed programs are omitted as well. The new fixed-response probe therefore uses a common failed-attempt rule and surrounding text across its added-context conditions. This changes the probe protocol; it does not retroactively change the historical runs.

A.2 Training settings

Table 5: Settings used by the training launcher and distillation implementation. The step count includes a shared warm start and a separate continuation.

Setting	Value
Initial model	Qwen2.5-7B-Instruct
Warm start	30 GRPO steps (archived; pilot trains its own from the base model)
Continuation	Archived: 150 steps per arm; pilot: 30 steps; fresh optimizer in both
Prompt batch / responses per prompt	32 / 8
Learning rate / warmup	10^{-6} / none
Student prompt / response limit	3,072 / 2,048 tokens
Teacher prompt limit	10,240 tokens
PPO mini-batch size	A, B8, and pilot arms: 8; B, C, D, E, F: 1
Teacher update	EMA, rate 0.01
Distillation divergence	Reverse KL ($\alpha = 1$)
Distillation support	Student top 20 tokens plus tail bucket
Success reward threshold	1.0
Own successful response as sibling	Excluded
Distillation importance-ratio clip	2.0
Rollout importance correction	Token-level, threshold 2.0
F objective weights	0.5 GRPO, 0.5 distillation
Validation	Every 10 steps; four samples per prompt
Training seeds	One per arm; pilot arms share one warm start (seed 1)
Hardware	Two H200 GPUs per training job

The description in Section 3.2 follows the trainer's context construction, the actor's teacher forward pass, and the self-distillation loss. A teacher forward pass receives the original sampled response tokens; there is no repaired-response acceptance stage in this path. The historical runs did not archive a resolved configuration or source snapshot; the pilot does (Appendix B.4).

A.3 Scoring and failure labels

The strict scorer assigns each generation one outcome:

- `success`: an accepted program with a returned label; the label may still be wrong.
- `extract_error`: no usable formal program can be extracted.
- `parse_error`: the extracted program is syntactically invalid.
- `symbol_clash`: incompatible arities or symbol roles.
- `solver_error`: the backend rejects or fails on the program.
- `timeout`: the execution or proof-search budget expires.
- `contradictory_kb`: both the conclusion and its negation are proved.
- `vacuous_kb`: the implemented premise-side vacuity check rejects the program.
- `internal_harness_error`: an evaluator failure rather than a model error.

A nonzero internal-harness error requires investigation. It should not be interpreted as evidence about the model. The ProverQA records used here have zero timeout, vacuity, and internal-harness counts. A zero observed count does not establish that the corresponding failure is impossible.

The scorer isolates program execution in child processes and bounds runtime, memory, and output buffering. These safeguards matter when model-generated inputs are repeatedly passed to a solver. They should be tested separately from model accuracy.

A.4 Earlier teacher-probe controls and coverage

The earlier saved probe evaluates 100 prompts with eight initial rollouts per prompt. The successful-sibling generation comparison uses 57 prompts; seven do not supply a distinct additional successful sibling for the preference comparison, leaving 50 response pairs. The foreign-example control has mean margin gain 0.0122, compared with 0.0490 for the same-problem sibling.

Reported mean 8-gram coverage is 0.4125 for successful-sibling context, 0.1662 for foreign context, and 0.3145 for structured feedback. This is a copying diagnostic, not proof that the contexts contain no answer information. The successful-sibling context intentionally contains privileged information about the same problem. The new probe fixes response pairs across conditions and reports overlap sensitivity (Appendix B.5). Its development prompts also have potential historical training exposure. Predictive value on held-out contexts remains untested.

Table 6: ProverQA outcome counts. Accepted programs may have wrong answers. Timeout, vacuity, and internal-harness counts are zero in these records. Each row sums to 1,500.

Arm	Accepted	Extract	Parse	Clash	Solver	Contradict.
Base	739	223	476	14	2	46
Warm start	1278	65	34	9	3	111
A	1386	24	2	1	0	87
B	1356	56	10	2	0	76
B8	1328	0	39	15	5	113
D	1384	61	12	2	1	40
C	1216	105	24	81	6	68
E	142	123	1216	1	6	12
F	1418	12	13	1	1	55

B Additional results

B.1 Archived training and transfer results

The following tables retain the original ProverQA and FOLIO evaluations. The historical FOLIO sweep used mixed GPU types and gave D a larger output cap. It is separate from the common-device evaluation in the main text.

Table 7: ProverQA accuracy by difficulty and gold label, in percent. Each difficulty has 500 items. True, False, and Uncertain have 470, 491, and 539 items, respectively. Macro is the unweighted mean over the three labels.

Arm	Easy	Medium	Hard	True	False	Uncertain	Macro
Base	56.80	37.00	15.00	30.21	29.12	48.05	35.80
Warm start	87.60	74.20	36.60	61.49	57.84	77.74	65.69
A	95.80	92.60	65.60	81.91	81.47	89.98	84.45
B	97.20	92.40	64.00	81.91	80.45	90.54	84.30
B8	94.00	84.00	62.20	78.09	78.62	83.12	79.94
D	96.80	92.00	66.80	82.34	80.45	92.02	84.94
C	85.00	74.60	41.80	62.77	64.56	73.28	66.87
E	8.40	9.40	8.40	8.30	8.96	8.91	8.72
F	94.80	88.80	70.60	80.00	80.04	93.14	84.39

Table 8: FOLIO dev results, in percent ($N = 204$). D uses a 2,048-token output cap; the other arms use 1,024. D comparisons also differ in decoding budget.

Arm	Overall acc.	Accepted rate	Accepted acc.
Base	32.84	50.49	65.05
Warm start	48.53	66.67	72.79
A	46.08	71.08	64.83
B	50.00	65.20	76.69
B8	48.04	68.63	70.00
D	47.06	75.49	62.34
C	46.08	70.59	65.28
E	32.84	50.49	65.05
F	40.20	69.61	57.75

B.2 Paired comparisons of saved predictions

Each comparison aligns the two saved outputs by problem identifier. For item j, let d_j be the difference in correctness indicators. The reported difference is $100\bar{d}$ percentage points. For ProverQA, the

Table 9: ProverQA generations reaching the 1,024-token evaluation cap. A capped output can still contain an extractable, accepted program.

Arm	Cap reached (count)	Cap reached (percent)
Base	719	47.93
Warm start	99	6.60
A	58	3.87
B	139	9.27
B8	4	0.27
D	1277	85.13
C	1334	88.93
E	1088	72.53
F	15	1.00

nominal 95% paired normal interval is

$$100\left[\bar{d} - 1.96\, s_d/\sqrt{N},\ \bar{d} + 1.96\, s_d/\sqrt{N}\right],$$

where s_d is the sample standard deviation of the item-level differences. ProverQA has 1,500 distinct premise texts in this evaluation. The pilot's 600-item ProverQA holdout has 600 premise groups, so its intervals use the group bootstrap described below, with 4,000 draws (Table 13).

FOLIO's 204 questions share only 73 premise texts, so treating every question as independent understates uncertainty. We instead sample 73 premise groups with replacement, keeping all questions in each sampled group and the paired predictions from both systems. We repeat this 10,000 times (seed 20260924), calculate the item-weighted accuracy difference each time, and report the 2.5th and 97.5th percentiles. Groups are defined by identical premise text after whitespace normalization; the question is not part of the group key.

Neither method, nor the pilot's group bootstrap, measures training-seed uncertainty, corrects for repeated development decisions or multiple comparisons, or accounts for shared logical templates across different premise texts. Because some test items informed development, these intervals are descriptive rather than confirmatory. We do not use them as an equivalence test or as proof that a layout has no effect.

Table 10: Paired descriptive comparisons, in percentage points. Only-first and only-second count items answered correctly by one arm but not the other. ProverQA uses paired normal intervals; FOLIO resamples its 73 premise groups (10,000 bootstrap draws). Intervals exclude training-seed uncertainty and development-selection effects.

Dataset	Contrast	Difference	Nominal 95% interval	Only-first	Only-second
ProverQA	D minus B	+0.67	[-0.98, +2.31]	84	74
ProverQA	D minus C	+18.07	[+15.79, +20.35]	312	41
ProverQA	D minus E	+76.47	[+74.30, +78.63]	1149	2
ProverQA	A minus B8	+4.60	[+2.42, +6.78]	175	106
ProverQA	F minus B	+0.20	[-1.73, +2.13]	110	107
FOLIO	B minus A	+3.92	[-6.67, +14.87]	42	34
FOLIO	D minus B	-2.94	[-13.24, +7.08]	33	39
FOLIO	F minus B	-9.80	[-22.56, +2.80]	34	54

B.3 New transfer comparisons

All eight new transfer evaluations use the same allocated H200 and recorded engine settings. The scorer uses a 10-second wall timeout and eight workers; the stop string is the Logic-LM example separator. Output files preserve per-item stopping reasons and token counts. No internal-harness failures are recorded. Result-file hashes and the frozen dataset hashes were checked against the completed execution records.

FOLIO has 73 normalized premise groups across 204 questions. ProofWriter has 457 groups across 600 questions: 348 singleton groups, 79 pairs, 26 triples, and four groups of four. Removing the question suffix from ProofWriter identifiers independently gives the same grouping. We resample whole groups using the procedure above; broader dependence between different but similar premise texts is not modeled. Table 11 includes every available comparison against the two reference arms, GRPO and program demonstrations. The bundle did not evaluate ProofWriter D or E, or either dataset's B8 or F. No absent result is inferred from the historical sweep.

Table 11: All GRPO- and program-demo-reference contrasts in the new transfer evaluations, in percentage points. Paired percentile intervals use 10,000 premise-group bootstrap draws: 73 groups for FOLIO and 457 for ProofWriter. Intervals are nominal and exclude training-seed uncertainty.

Dataset	Contrast	Difference	95% interval
FOLIO	B minus A	+2.45	[-7.77, +12.96]
FOLIO	C minus A	+0.98	[-9.22, +11.50]
FOLIO	D minus A	-0.49	[-10.61, +9.62]
FOLIO	E minus A	-14.71	[-27.92, -1.40]
FOLIO	C minus B	-1.47	[-10.45, +7.39]
FOLIO	D minus B	-2.94	[-12.87, +6.81]
FOLIO	E minus B	-17.16	[-27.44, -7.58]
ProofWriter	B minus A	-18.00	[-23.55, -12.67]
ProofWriter	C minus A	-21.50	[-26.89, -16.13]
ProofWriter	C minus B	-3.50	[-8.25, +1.33]

B.4 Compact pilot: full counts and provenance

Table 12 gives every count behind Table 3, and Table 13 gives the six paired contrasts with their item and group counts. The base-model rows are archived references evaluated under the same protocol; they enter no contrast. The ProverQA holdout has 600 premise groups over 600 items, so its group bootstrap coincides with an item bootstrap. ProofWriter has 457 groups over 600 items.

Table 12: Compact pilot, per-arm counts on 600 items. Base-model rows are archived references evaluated under the same protocol on possibly different hardware and enter no contrast. Truncated counts outputs that reached the 1,024-token cap; a truncated output can still contain an accepted program.

Dataset	Model	Correct	Accepted	Parse fail	Contrad.	No answer	Truncated
ProverQA holdout	Base model (reference)	183	259	322	8	341	321
ProverQA holdout	Warm start (30 GRPO steps)	385	545	29	21	55	17
ProverQA holdout	PG: GRPO	419	545	23	27	55	433
ProverQA holdout	PP: program in slot	423	527	45	25	73	463
ProverQA holdout	PE: empty slot	0	0	600	0	600	575
ProofWriter	Base model (reference)	131	152	429	10	448	192
ProofWriter	Warm start (30 GRPO steps)	440	523	37	24	77	0
ProofWriter	PG: GRPO	437	520	33	26	80	377
ProofWriter	PP: program in slot	419	480	80	26	120	483
ProofWriter	PE: empty slot	0	0	600	0	600	561

Provenance. The pilot ran as HTCondor job 11276065 (DAG 11276060) on two H200 GPUs, allocated at 04:08 CDT on 2026-09-25. The whole pilot ran in 6.2 hours on those two GPUs. It has fourteen stages. Six are training stages: `warm-seed1` (30 GRPO steps from Qwen2.5-7B-Instruct, seed 1); `smoke-program` and `smoke-empty` (two steps each, used only to gate the continuations and never used as an initial model); and `grpo-seed1`, `matched_program-seed1`, and `matched_empty-seed1` (30 steps each from the same warm export, fresh optimizer and fresh EMA teacher, `trainer.resume_mode=disable`). Eight are evaluations: four models times two datasets, one GPU each. All six training stages completed and all eight evaluations completed; no contrast is missing. Every stage ran from one frozen source tarball (sha256 prefix `2883fbd0`) against frozen data files: ProverQA campaign train `e0babd6c`, dev `4745178a`, holdout `800a5316`, and

Table 13: Compact pilot, paired contrasts. Paired premise-group percentile bootstrap, 4,000 draws, seed 20260924; nominal 95% intervals, unadjusted for the six contrasts, within one training seed.

Dataset	Contrast	Difference (pp)	95% interval	Items / groups
ProverQA holdout	PP − PG	+0.67	[-3.17, +4.50]	600 / 600
ProverQA holdout	PP − PE	+70.50	[+66.83, +74.17]	600 / 600
ProverQA holdout	PE − PG	-69.83	[-73.33, -66.17]	600 / 600
ProofWriter	PP − PG	-3.00	[-7.59, +1.52]	600 / 457
ProofWriter	PP − PE	+69.83	[+65.54, +74.07]	600 / 457
ProofWriter	PE − PG	-72.83	[-76.71, -68.67]	600 / 457

ProofWriter test `74baccb8`. The archived model checkpoints have sha256 prefixes `83401302` (warm start), `1beea293` (PG), `155b57dc` (PP), and `e6fe10e6` (PE). The report generator has prefix `2a0b73e3`, the campaign manifest `b4d1514d`, and the report it produced `bce0ae0e`. Evaluation used vLLM 0.12.0 and torch 2.9.0+cu129 with tensor parallel 1, eager mode, prefix caching disabled, and seed 0. This is the only run in the paper whose checkpoint hashes, data hashes, resolved specification, and execution record are archived together.

What PE produced. The empty-slot arm (PE) has zero correct and zero accepted outputs on both datasets (Table 12). All 600 of its ProverQA holdout outputs are parse failures, and 575 reach the 1,024-token cap. All 600 write markdown bullets with backtick-quoted predicates in place of the Logic-LM `:::` grammar, and many continue into text copied from a few-shot example in the prompt. On ProofWriter the pattern repeats: 600 parse failures and 561 capped outputs. The training log shows when this began. Mean batch reward was 0.52 at step 12 and 0.009 at step 15, and it never recovered (0.0 at step 30). PP used the same loss, mask, optimizer settings, and warm start, and kept producing accepted programs. We report the timing and the signature; we do not know the mechanism.

B.5 Matched teacher probe: controls and sensitivity

The matched probe uses one frozen warm-start checkpoint, one H200, and vLLM 0.12.0. Its initial 800 responses contain 517 correct outputs and 649 accepted programs. These sampled rollout rates are not greedy accuracy and do not treat eight responses to one prompt as independent examples. The fixed-response stage scores the actual target-token probabilities; it generates no evaluated revised answers.

Among the 63 failed targets, 24 are accepted but wrong, 19 have contradictory premises, 11 have parse errors, four have extraction errors, four have symbol clashes, and one has a solver error. The failed program appears in 59 contexts and is unavailable in four. Diagnostic payloads contain the gold label in 48 cases; this is privileged development information, not label-free inference feedback.

Correct targets are cleaned programs with a leading newline, while failed targets retain the original generated tokens. Their mean lengths are 587.56 and 749.75 tokens. Each target stays unchanged across contexts, but this asymmetry limits interpretation of an absolute margin. The largest prefix is 4,305 tokens; the largest scored prefix-plus-target input is 5,448, below the 12,288-token model limit. No scoring input is truncated. Four selected failed generations did reach the initial generation cap. Dropping them leaves 59 pairs: program minus diagnostic is $+0.0625$ ($[0.0526, 0.0728]$), and program minus foreign is $+0.0382$ ($[0.0286, 0.0479]$).

The program and foreign-program prefixes have identical mean length (2,869.92 tokens), since each program serves as its own problem's example and another problem's foreign example. Individual paired lengths differ. Diagnostic and empty prefixes average 2,288.52 and 2,277.37 tokens. Mean correct-target eight-gram coverage is 44.69% in program context and 33.06% in each other added context. On the 39 pairs with program-context coverage at most 50%, program minus foreign remains positive: $+0.0280$ ($[0.0153, 0.0405]$). This sensitivity check reduces the concern about direct copying but does not eliminate shared wording or semantics.

Table 14 includes all context contrasts. For diagnostics versus constant feedback, the difference is -0.0016 ($[-0.0037, +0.0006]$). This does not establish that diagnostics are better or worse than the

constant signal. The intervals do not account for alternative sampled pairs, training seeds, or shared foreign donors. No shuffled-diagnostic condition was run.

Table 14: All paired context-margin differences in the new probe. Intervals use 4,000 prompt-bootstrap draws (seed 20260925). The three contrasts specified by the probe are program minus foreign, program minus diagnostic, and diagnostic minus empty; the remaining contrasts are retrospective descriptive checks. Shared foreign donors can induce dependence not modeled by the prompt bootstrap.

Contrast	Mean	95% interval
Plain prompt minus Successful program	+0.0921	[+0.0759, +0.1088]
Plain prompt minus Diagnostic	+0.1504	[+0.1322, +0.1682]
Plain prompt minus Constant	+0.1488	[+0.1300, +0.1674]
Plain prompt minus Empty feedback slot	+0.1417	[+0.1236, +0.1596]
Plain prompt minus Foreign program	+0.1274	[+0.1103, +0.1438]
Successful program minus Diagnostic	+0.0583	[+0.0468, +0.0693]
Successful program minus Constant	+0.0567	[+0.0450, +0.0681]
Successful program minus Empty feedback slot	+0.0496	[+0.0388, +0.0604]
Successful program minus Foreign program	+0.0353	[+0.0251, +0.0454]
Diagnostic minus Constant	-0.0016	[-0.0037, +0.0006]
Diagnostic minus Empty feedback slot	-0.0087	[-0.0118, -0.0058]
Diagnostic minus Foreign program	-0.0230	[-0.0290, -0.0172]
Constant minus Empty feedback slot	-0.0071	[-0.0096, -0.0047]
Constant minus Foreign program	-0.0214	[-0.0281, -0.0155]
Empty feedback slot minus Foreign program	-0.0143	[-0.0205, -0.0087]

Table 15: Fixed-response teacher preferences on the same 63 development prompts. The margin is mean token log probability of the correct target minus that of the failed target. Larger is better. Every added context in this first probe reduces the margin relative to the plain prompt. These numbers are not accuracy or percentage points.

Teacher context	Mean margin	Change from plain
Plain prompt	-0.0448	+0.0000
Successful program	-0.1369	-0.0921
Diagnostic	-0.1952	-0.1504
Constant	-0.1936	-0.1488
Empty feedback slot	-0.1865	-0.1417
Foreign program	-0.1722	-0.1274

Development exposure and provenance. All 100 selected prompts come from the historical training pool; this does not prove that each was sampled during the warm start's 30 updates. Their identifiers and normalized premise blocks do not overlap the historical ProverQA test or the separately partitioned campaign train and holdout sets. The pilot evaluates on that campaign holdout. The old checkpoint's potential training exposure remains. Audits verified selected-ID order, saved aggregate arithmetic, intervals, tokenized context lengths, overlap measurements, and executed source hashes. The job recorded model-file hashes, but the first-probe CPU audit did not rehash the weights or rerun the solver. Per-token log probabilities were not archived; the saved finite response means can be checked for arithmetic consistency, but not independently reconstructed without another model forward pass.

B.6 Exploratory failed-text intervention

The follow-up crosses program versus empty payload with failed-attempt text included versus omitted. Foreign payloads at both levels and the plain prompt complete seven conditions. The primary interaction is

$$[m(\text{program}, \text{with}) - m(\text{empty}, \text{with})] - [m(\text{program}, \text{without}) - m(\text{empty}, \text{without})].$$

The protocol specified this contrast and all 13 secondary contrasts before the follow-up run, after inspecting the first probe. This is a targeted exploratory intervention, not an independent confirmation. All 63 original pairs, successful context programs, target tokens, and foreign donors are retained; no new answers are generated or reselected.

Table 16: Exploratory failed-text intervention on the same 63 frozen response pairs. Correct and failed columns are mean token log probabilities; margin is their difference. All seven conditions were scored together on one H100. These are preference measurements, not generated-answer accuracy.

Context	Correct	Failed	Margin
Plain prompt	-0.1137	-0.0690	-0.0448
Program, with failed text	-0.2137	-0.0769	-0.1368
Empty, with failed text	-0.2296	-0.0439	-0.1857
Foreign, with failed text	-0.2126	-0.0405	-0.1720
Program, without failed text	-0.2374	-0.2236	-0.0138
Empty, without failed text	-0.1487	-0.1184	-0.0303
Foreign, without failed text	-0.1613	-0.1417	-0.0195

The wrapper always keeps the original prompt, feedback heading, separator, and repeated target. The failed-text factor changes only the bounded failed-program text. For 59 pairs that text is available. For four it is unavailable, so the with/without contexts are identical and contribute zero to the interaction. Unlike the earlier constructor, this wrapper retains the first target block in those four cases. It is therefore not an exact reproduction of every old context. The predefined 59-pair sensitivity removes these no-change cases: its interaction is $+0.0346$ ($[0.0231, 0.0469]$). Table 17 reports every prespecified contrast on both samples.

Table 17: Every prespecified factorial contrast, on all 63 pairs and the predefined 59-pair subset with available failed text. With/without refers to failed-attempt text. The primary interaction is (program − empty) with text minus (program − empty) without text. Nominal paired intervals use 4,000 prompt-bootstrap draws, seed 20260924. The follow-up was motivated by the first probe; intervals exclude seed variation, shared-donor dependence, and multiplicity correction.

Contrast	All 63	95% interval	Available 59	95% interval
Program–empty interaction	+0.0324	[+0.0218, +0.0437]	+0.0346	[+0.0231, +0.0469]
Program − foreign, with	+0.0352	[+0.0254, +0.0452]	+0.0394	[+0.0300, +0.0492]
Program − empty, with	+0.0489	[+0.0378, +0.0596]	+0.0542	[+0.0441, +0.0645]
Program − foreign, without	+0.0057	[-0.0052, +0.0174]	+0.0079	[-0.0039, +0.0198]
Program − empty, without	+0.0165	[+0.0051, +0.0282]	+0.0196	[+0.0075, +0.0318]
Program, without − with	+0.1230	[+0.1069, +0.1391]	+0.1313	[+0.1167, +0.1460]
Empty, without − with	+0.1554	[+0.1369, +0.1753]	+0.1659	[+0.1502, +0.1841]
Foreign, without − with	+0.1525	[+0.1346, +0.1711]	+0.1628	[+0.1473, +0.1786]
Program with − plain	-0.0920	[-0.1088, -0.0762]	-0.0968	[-0.1137, -0.0812]
Empty with − plain	-0.1409	[-0.1596, -0.1230]	-0.1511	[-0.1681, -0.1359]
Foreign with − plain	-0.1272	[-0.1441, -0.1107]	-0.1362	[-0.1512, -0.1215]
Program without − plain	+0.0310	[+0.0174, +0.0454]	+0.0345	[+0.0198, +0.0491]
Empty without − plain	+0.0145	[+0.0061, +0.0231]	+0.0149	[+0.0059, +0.0238]
Foreign without − plain	+0.0253	[+0.0157, +0.0354]	+0.0266	[+0.0160, +0.0374]

The follow-up scores 882 target/context combinations on one H100, with vLLM 0.12.0, eager mode, prefix caching disabled, and seed 20260924. The original frozen rollouts still use seed 20260925. GPU type, execution seed, and four wrappers differ from the first probe on an H200; we do not pool scores across the runs or treat small cross-run differences as effects. Comparisons within the follow-up use the same execution environment. The CPU audit reconstructed all 441 contexts and 126 target token sequences, recomputed all 28 full-sample and sensitivity intervals, and rehashed all 13 model files, including four weight shards. They match both jobs' saved manifests. Maximum scored length is 5,448 tokens, with no scoring truncation. Four failed targets and two correct-target source rollouts originally reached the generation cap; the frozen targets remain unchanged. Per-token

Table 18: Generation results from the warm-start teacher probe, in percent. Each row uses its own eligible subset; rows are not a matched cross-context comparison.

Context	Prompts	Plain acc.	Context acc.
Verified program	57	70.18	96.49
Structured diagnostic	67	59.70	28.36

log probabilities were not saved, so the audit checks the response means and execution records rather than independently reproducing model forwards.

The descriptive component differences in the main text were calculated after the run; they are an accounting of the recorded margin rather than additional prespecified hypothesis tests. Removing text also changes context length, so this design does not separate repetition, length, and meaning. In particular, the intervention removes the very failed response whose likelihood is scored. Higher relative margin therefore cannot be read as higher correct-answer probability or better trained-policy accuracy. Correct-target canonicalization, original failed-target tokenization, training-pool exposure, and foreign-donor dependence remain as before.

B.7 Earlier teacher probe

The earlier probe used 100 prompts from the test-derived validation set and the historical context constructors. Its generation comparisons have different eligible subsets (Table 18). On 50 retained preference pairs, successful-program demonstrations gave a mean gain over plain of $+0.0490$, versus $+0.0122$ for an unrelated example and -0.0975 for diagnostics. The program-minus-foreign difference was $+0.0369$. The new matched probe changes the examples, pair selection, and surrounding layout; in particular, it commonly retains a failed attempt. Its negative changes relative to plain must not be averaged with this older positive result or presented as a replication of it.

The historical mixture. On ProverQA, F scores 84.73%, only 0.20 points above B. It accepts 62 more programs, adding three correct and 59 accepted wrong answers. Its hard-item accuracy rises from 64.00% to 70.60%, while easy and medium accuracy fall by 2.40 and 3.60 points. The historical FOLIO F–B difference is -9.80 points with interval $[-22.56, +2.80]$; no new F evaluation was run.

B.8 Items used for validation

The sidecar identifiers for the 300-item validation subset allow the saved ProverQA predictions to be separated into validation-seen and remaining items. Table 19 reports that split. The remaining 1,200 items are not automatically a fresh holdout: their aggregate evaluation results have also been available during development. The split is an overlap audit, not a repair for test-set reuse.

Table 19: Accuracy on validation-seen and remaining ProverQA items. The remaining items are not a new holdout; their aggregate results have already been inspected.

Arm	Validation-seen (300)	Remaining (1,200)
Base	41.67	34.92
Warm start	68.33	65.58
A	86.33	84.25
B	84.67	84.50
B8	79.33	80.25
D	86.67	84.83
C	70.67	66.25
E	8.33	8.83
F	88.00	83.92

B.9 Comparisons on shared accepted items

For each pair of systems, we retain the intersection of the items accepted by both. We then compute each system's accuracy on exactly that set. This makes the examples comparable within a row, but the subset remains selected by the two models' outputs. It does not estimate accuracy over all test items or establish a causal effect on reasoning.

Table 20: Accuracy on the same ProverQA items accepted by both systems in each pair. Each row defines a different, selected subset; these are descriptive comparisons, not causal effects.

First / second system	Items	First acc.	Second acc.
Program feedback / diagnostic	1194	93.63	83.17
Program demo / diagnostic	1190	94.45	83.45
Program demo / constant	141	92.20	92.20
Program demo / program feedback	1322	93.72	92.66
Program demo / mixture	1324	93.81	90.41

B.10 Earlier prompting evaluations

The archived prompting runs use Qwen2.5-7B-Instruct on the same 204 FOLIO items. Direct answering and chain of thought use chat formatting, while the Logic-LM run uses completion formatting. All use greedy decoding; the output caps are shown in Table 21. These runs use a separate prompting/scoring pipeline and are not pooled with the strict checkpoint sweep. They establish useful context for the transfer results: training a formal translator does not automatically outperform the model's natural-language reasoning. The two chain-of-thought runs both answer 139 items correctly, but have different per-label counts; identical aggregate accuracy does not mean identical predictions.

Table 21: Earlier FOLIO prompting evaluations with the same starting model ($N = 204$, greedy decoding). They use their own prompt and scoring pipeline and are kept separate from the strict checkpoint sweep. The repeated chain-of-thought evaluation is a repeated decoding run, not an independent training seed.

Prompt	Format	Token cap	Correct	Accuracy
Direct answer	Chat	16	113	55.39
Chain of thought	Chat	1024	139	68.14
Chain of thought (repeat)	Chat	1024	139	68.14
Logic-LM prompt	Completion	1024	69	33.82

B.11 Existing reward-guard audit

An archived 300-item audit compares reference formalizations with deliberately uninformative programs. Enabling the premise-side guard leaves all 300 reference outcomes unchanged and accepted; 295 return the correct label under both settings. The guard blocks the tested vacuous, tautological, and excluded-middle shortcuts. However, an irrelevant contingent premise still earns reward 1 on all 109 uncertain-label items. This audit shows both the guard's usefulness and its boundary: avoiding vacuity does not establish faithful translation.

B.12 Reading the training curves

The curves are regenerated from the recorded validation metrics in each training log. The available GRPO log has no step-30 point; the figure connects its recorded step-0 and step-60 values without adding a measurement. The other six conditions have recorded points at steps 0, 30, 60, 90, 120, and 150. We show means without uncertainty bands because the within-prompt sampling statistics are not training-seed uncertainty. The larger-mini-batch SDPO run (B8) reaches 88.42% sampled validation accuracy at step 60 and ends at 79.83%. Its final greedy score therefore describes the fixed final checkpoint, not the best possible checkpoint under a separately chosen selection rule.

Table 22: Existing reward-guard audit: counts receiving reward 1. Ref. denotes reference formalizations. Off/on compare the uninformative-program shortcut without/with the guard. Taut. and Excl. use tautologies and excluded middle; Thin uses an irrelevant contingent premise. The last three columns have the guard enabled.

Label	N	Ref.	Off	On	Taut.	Excl.	Thin
True	91	89	0	0	0	0	0
False	100	97	0	0	0	0	0
Uncertain	109	109	109	0	0	0	109

Distillation coverage metrics. The logged `empty_target_batch` value indicates a micro-batch with no eligible distillation responses. Its aggregate is not automatically the fraction of training prompts discarded. Diagnostic eligibility depends on feedback availability, while successful-program contexts depend on a suitable sibling. Coverage must therefore be interpreted together with the response mask and token normalization.

B.13 Source records

Numerical result tables are regenerated from saved evaluation JSON files; the training figure is regenerated from saved logs. Training settings and data counts are documented in the protocol. The audit checks unique identifiers, label agreement across arms, correctness counts, accepted-program counts, failure counts, and the metric identity

$$\text{overall accuracy} = \text{accepted rate} \times \text{accepted accuracy}.$$

A separate manifest records source-file hashes. The audit does not rerun generation or establish that a saved output was produced by the declared checkpoint. That requires checkpoint hashes and archived execution records. The compact pilot (Appendix B.4) archives these.