# Sequent-Prover: Training Agents for Formal, Checkable SMT-based Reasoning

**Anonymous ACL submission**


## Abstract

We introduce Sequent-Prover, a family of verifier-using language-model agents, and the post-training framework that produces them. Agents reason in an executable SMT-LIB scratchpad: they write complete SMT-LIB programs, query Z3 across turns, inspect solver feedback, repair formalizations, and terminate with an explicit answer. Our recipe, verifier-conditioned trajectory distillation, bootstraps from a noisy teacher proposal policy by sampling multi-turn trajectories, retaining the successful tail that is correct, non-trivial, grounded, and executable, then applying supervised fine-tuning and trajectory-level preference optimization. Because solver success alone does not guarantee semantic faithfulness, we evaluate task accuracy, k-sample reliability, and SMT program quality. Sequent Prover moves language-model reasoning toward answers supported by executable formal artifacts.


## 1 Introduction

Large language models can produce fluent multi-step reasoning, but fluency is not executable evidence that a conclusion follows from the stated assumptions (Wei et al., 2022; Kojima et al., 2022; Turpin et al., 2023; Lanham et al., 2023). A natural alternative is to make reasoning *checkable*: translate an input into explicit assumptions, constraints, and claims, then verify the resulting obligation with a symbolic backend (Pan et al., 2023; Ye et al., 2023; Olausson et al., 2023; Ganguly et al., 2024). SMT solvers such as Z3 are especially attractive because SMT-LIB can encode logical entailment, arithmetic, datatypes, arrays, bit-vectors, and program constraints, while solver execution returns concrete feedback such as `sat`, `unsat`, countermodels, parser errors, and type errors (de Moura and Bjørner, 2008; Barrett and Tinelli, 2018; Barrett et al., 2025).

However, solver access alone does not make reasoning reliable. Recent work by Ganguly et al. (2026) shows that SMT-based autoformalization can improve premise-explicit reasoning but degrade performance when the generated artifact misrepresents the input. The core problem is the *autoformalization gap*: real tasks arrive as natural language, specifications, and informal intent, not as already-correct proof obligations. A solver checks only the artifact it receives; if the model omits a premise, reverses entailment, introduces inconsistent assumptions, or encodes a different problem, the solver may return a valid result for the wrong object. Reliable verifier use therefore requires models that construct faithful, non-vacuous, executable formalizations through multi-turn tool use.

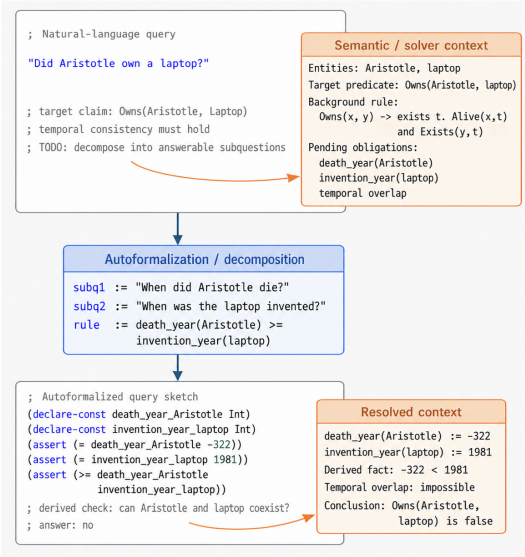


Figure 1: A toy example showing how Sequent Prover turns an underspecified commonsense claim into a grounded, checkable inconsistency rather than relying on fluent explanation alone.

We introduce *Sequent Prover*, a family of post-trained language models for SMT-verifiable reasoning. Rather than using the solver as a test-time wrapper (Ganguly et al., 2024; Singh et al., 2026; Feng et al., 2025), we use solver interaction as a training substrate: we sample candidate formalization traces, execute them with Z3, reject malformed

or vacuous artifacts, and convert supported traces into supervised and preference data. Post-training turns inconsistent SMT tool use into more dependable multi-sample correctness and yields agents that write more grounded, non-trivial, executable SMT programs. In particular, we find that:

1. Post-training substantially improves SMT reasoning: models that previously achieved only inconsistent solver-aided success become more reliable under multi-sample evaluation.
2. Supervised fine-tuning improves *proof discipline*: tuned agents produce higher-quality SMT-LIB programs that are better grounded in the input, less vacuous, and more often aligned with the intended claim.
3. Post-training strengthens more demanding proof strategies, including refutational reasoning, where agents increasingly formulate proof obligations by contradiction and correctly interpret the resulting `unsat` evidence.
4. Preference optimization with DPO further improves stability, grounding, proof quality, and efficiency beyond supervised fine-tuning: DPO-tuned agents solve problems earlier, with fewer tokens and fewer solver-interaction rounds per correct answer.

## 2 Methodology

**Overview.** We frame verifier-guided reasoning as success-conditioned policy bootstrapping. A teacher model need not be reliable on every rollout; it only needs a nonzero probability on trajectories that use the verifier correctly. Sampling the teacher in an SMT-LIB scratchpad exposes a sparse successful tail: rollouts that construct and revise executable formalizations, use Z3 feedback, and terminate with the correct answer. We condition on this tail so the student learns verifier-mediated search behavior that the teacher produces only sporadically.

Our method, *verifier-conditioned trajectory distillation*, has three stages. First, we sample teacher trajectories under the same `z3-solve`/`finish` interface used at evaluation time. Second, we curate them with an acceptance predicate that combines answer correctness with executable scratchpad evidence: solver execution, non-triviality, grounding, and protocol completion. Third, we train by supervised fine-tuning on accepted traces and by DPO on trajectory-level pairs that contrast accepted traces with rejected or weakly supported ones. The resulting policy amortizes rare successful verifier-mediated searches into more reliable inference.

### 2.1 Problem Formulation

Each task instance is $x = (\mathcal{C}, q)$, where $\mathcal{C} = \{p_1, \ldots, p_m\}$ contains the natural-language premises, program fragments, or specifications, and $q$ is the query. The model must produce a final answer $y$ matching $y^\star$ and, when possible, support it with an executable SMT-LIB artifact. We use $\phi$ for the model's formalization map: $\phi_{\mathcal{C}}$ encodes assumptions from $\mathcal{C}$, and $\phi_c$ encodes a candidate claim $c$. Context consistency is checked by $\mathrm{SAT}(\phi_{\mathcal{C}})$; claim compatibility by $\mathrm{SAT}(\phi_{\mathcal{C}} \wedge \phi_c)$; and entailment by refutation, requiring $\mathrm{SAT}(\phi_{\mathcal{C}}) = \mathrm{true}$ and $\mathrm{SAT}(\phi_{\mathcal{C}} \wedge \neg\phi_c) = \mathrm{false}$. Thus, a claim is entailed only when the formalized context is satisfiable and the claim's negation is inconsistent with it, making `unsat` the proof signal for refutational entailment.

### 2.2 The SMT Scratchpad Interface

A central design choice is to expose the SMT-LIB backend from Ganguly et al. (2024) as an *executable scratchpad*: a workspace in which the model constructs, tests, and revises formal artifacts across turns. We define the tool environment as $\mathcal{E}_{\mathrm{SMT}} = (\mathcal{A}, \mathcal{O}), \mathcal{A} = \{\texttt{z3-solve}, \texttt{finish}\}$. The `z3-solve` action executes a self-contained SMT-LIB script with declarations and a `check-sat` command. A single action can express context checks, candidate-claim tests, counterexample searches, model construction, semantic-equivalence checks, or refutational proof obligations. Z3 returns an observation in $\mathcal{O}$, such as `sat`, `unsat`, `unknown`, a model, a parser/type error, or a timeout, which is appended to the interaction history.

The `finish` action is separate from scratchpad execution: it terminates the episode with a final answer after the model judges that the accumulated solver feedback is sufficient. A trajectory is therefore a multi-turn record $\tau = (m_1, o_1, \ldots, m_T, o_T)$, where model messages may contain natural-language reasoning, SMT-LIB code, tool calls, repairs, or the final answer, and the policy $\pi_\theta$ maps the current history to the next token or tool action. We do not require every scratchpad call to constitute a complete proof; early calls may be exploratory or diagnostic. However, accepted training traces must contain meaningful scratchpad evidence: at least one Z3 invocation with an executable, non-trivial, input-grounded SMT-LIB

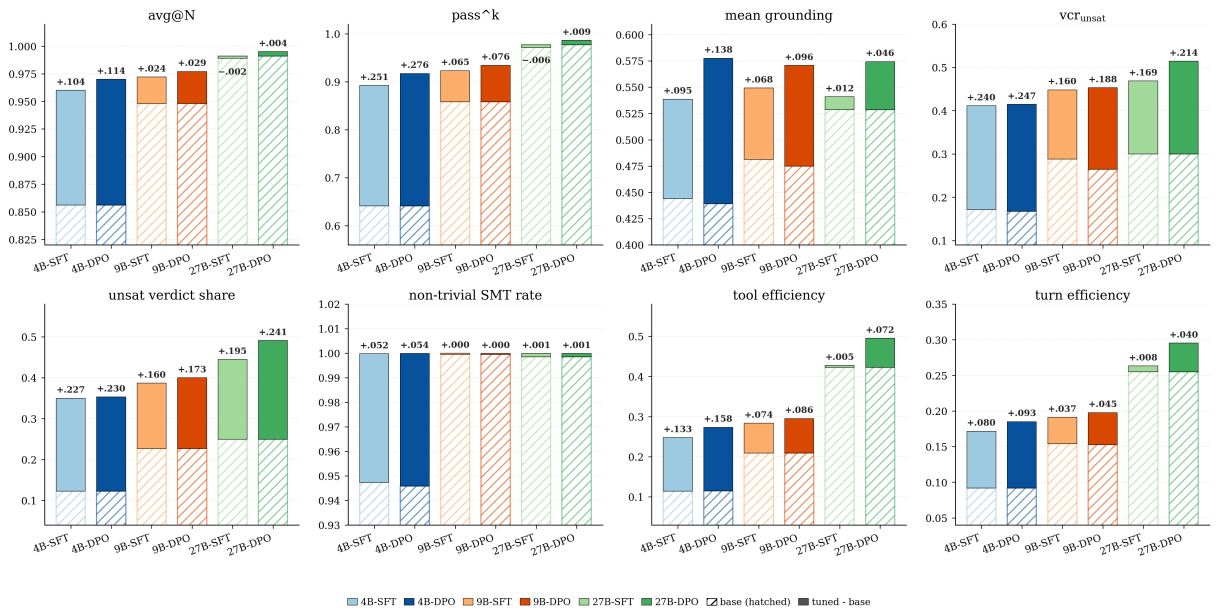


Figure 2: Post-training, especially DPO, shifts the models from merely sampling correct answers toward more reliable solver-grounded proof behavior. Hatched bars show each base checkpoint and colored overlays show the tuned-model gain: avg@N measures per-sample correctness, pass^k measures consistency by requiring all sampled trajectories to be correct, mean grounding measures how much of the problem is represented in SMT, and vcr_unsat measures correct answers backed by non-trivial UNSAT/refutation certificates. The remaining panels show the mechanism behind the gains: tuned models more often drive Z3 to UNSAT verdicts, avoid trivial SMT calls, and obtain more verified progress per tool call and per reasoning turn.

program that obtains a real solver verdict.

### 2.3 Verifier Feedback and Trace Semantics

Each `z3-solve` call converts a generated SMT-LIB artifact into a structured observation that the agent can use to interpret, repair, or commit to a formalization. We write $\mathrm{SAT}(\psi) \in \{\texttt{sat}, \texttt{unsat}, \texttt{unknown}\}$ for the Z3 verdict on a formula or script $\psi$, and treat parser errors, type errors, timeouts, malformed scripts, and solver failures as unresolved observations. For a candidate claim $c_i$ with context encoding $\phi_{\mathcal{C}}$ and claim encoding $\phi_{c_i}$, the verifier first separates inconsistent contexts from meaningful claim statuses: if $\mathrm{SAT}(\phi_{\mathcal{C}}) = \texttt{unsat}$, the artifact receives the inconsistent-context status $\sigma_I$. Otherwise, when $\mathrm{SAT}(\phi_{\mathcal{C}}) = \texttt{sat}$, the claim is entailed ($\sigma_E$) if $\mathrm{SAT}(\phi_{\mathcal{C}} \wedge \neg\phi_{c_i}) = \texttt{unsat}$, contradictory ($\sigma_C$) if $\mathrm{SAT}(\phi_{\mathcal{C}} \wedge \phi_{c_i}) = \texttt{unsat}$, and possible but not forced ($\sigma_P$) if both $\mathrm{SAT}(\phi_{\mathcal{C}} \wedge \phi_{c_i}) = \texttt{sat}$ and $\mathrm{SAT}(\phi_{\mathcal{C}} \wedge \neg\phi_{c_i}) = \texttt{sat}$. We use $\sigma_U$ for states in which no semantic status can be assigned, including unknown, timeout, solver failure, or parse/type errors. The same scratchpad also supports comparison between alternative formalizations: given formulas $\phi_a$ and $\phi_b$ over a shared signature and background theory $\Sigma$, they are equivalent when $\mathrm{SAT}_\Sigma(\neg(\phi_a \leftrightarrow \phi_b)) = \texttt{unsat}$.

Thus, solver feedback gives the trajectory more than a binary success signal: `sat` can expose a witness or countermodel, `unsat` provides refutational evidence, and execution errors identify artifacts that must be repaired. When constraints are labeled or instrumented, this feedback can be augmented with unsat cores or approximate correction sets, but the main training signal comes from executable SMT-LIB interaction and solver feedback.

**Teacher Rollout Generation.** For each training instance $x = (\mathcal{C}, q)$, we sample $K_T$ trajectories from a teacher proposal model $M_T$ under the same SMT-scratchpad interface used at evaluation time. The teacher is given only the problem instance $x$, not the target answer $y^\star$; the target is reserved for offline curation. Thus, the teacher is not an oracle annotator, but a stochastic proposal distribution $p_T(\tau \mid x)$ over executable reasoning traces. Each sampled trajectory is stored as $\tau = (x, h_T, \hat{y}, \mu)$, $h_T = (m_1, o_1, \ldots, m_T, o_T)$, where $h_T$ is the interaction transcript, $\hat{y}$ is the answer emitted through `finish`, and $\mu$ records protocol metadata such as solver-call counts, solver verdicts, execution errors, token usage, and termination status. These rollouts provide the candidate pool from which verifier-conditioned trace curation

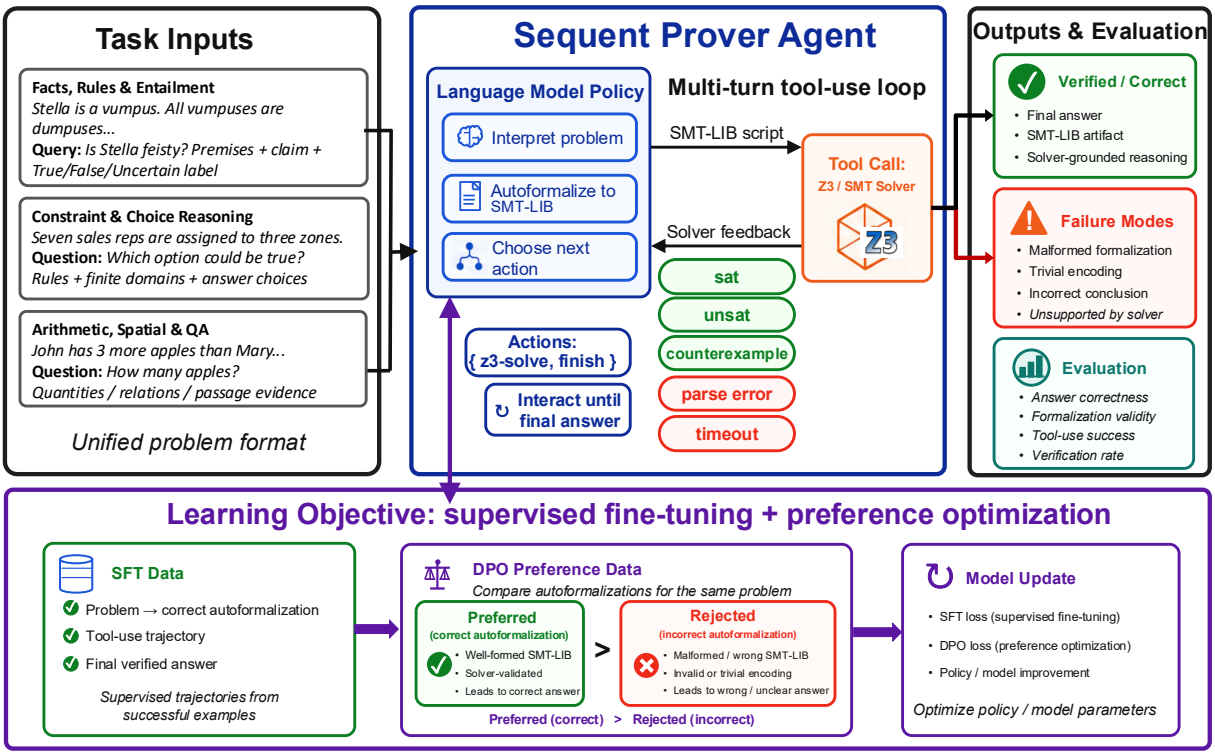


Figure 3: The Sequent Prover Framework

selects training examples.

**Verifier-Conditioned Trace Curation.** The offline curator converts teacher rollouts into training data by applying a trajectory-level acceptance predicate $A(x, y^\star, \tau)$. A rollout is accepted when it reaches the correct task answer through `finish` and contains meaningful scratchpad evidence: an executable `z3-solve` interaction that obtains a solver verdict and passes the structural checks for non-triviality, grounding, and detectable non-vacuity. We label accepted rollouts as *Gold*. Rollouts that reach the correct answer but provide weak formal support are labeled *Silver*, while rollouts that fail the answer, protocol, executability, or scratchpad-quality conditions are labeled *Rejected*. This partition separates task success from verifier-use quality, so that only fully supported traces become demonstrations while weak or failed traces remain available as a contrastive signal.

Formally, if $p_T(\tau \mid x)$ is the teacher proposal distribution and $Z_T(x) = \Pr_{\tau \sim p_T(\cdot|x)}[A(x, y^\star, \tau) = 1]$, then the gold supervision distribution is $p_{\text{gold}}(\tau \mid x) = \frac{p_T(\tau|x) A(x, y^\star, \tau)}{Z_T(x)} = p_T(\tau \mid x, A(x, y^\star, \tau) = 1)$. Trace curation, therefore, defines the success-conditioned target that SFT imitates, with Silver and Rejected traces supplying the negative side of the preference data.

## 2.4 Training

**SFT on Executable Traces.** Gold trajectories define the supervised corpus $\mathcal{D}_{\text{gold}}$. We train the student policy $\pi_\theta$ by maximum likelihood over the model-generated actions in each accepted transcript:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\sum_{\tau \in \mathcal{D}_{\text{gold}}} \sum_{t \in \mathcal{I}(\tau)} \log \pi_\theta(a_t \mid h_t),$$

where $h_t$ is the interaction history, $a_t$ is the next model token or tool-use action, and $\mathcal{I}(\tau)$ indexes positions generated by the model. Solver observations remain in the conditioning history but are masked from the prediction loss. SFT amortizes the curated successful tail: the student learns the action patterns by which accepted traces construct evidence, respond to verifier feedback, and commit to a correct answer.

**Trajectory Preference Optimization.** After SFT, we further refine the student with trajectory-level preferences. For each problem $x$ with at least one Gold trace and at least one contrastive trace, we form triples $(x, \tau^+, \tau^-)$, where $\tau^+$ is an accepted trajectory and $\tau^-$ is either Rejected or Silver. Silver traces serve as hard negatives because they may reach the correct answer with weaker scratchpad support; Rejected traces capture answer, protocol, executability, or verifier-use failures. Thus, Gold

traces define what to favor, while Silver and Rejected traces define nearby behaviors to suppress.

We optimize Direct Preference Optimization with the frozen SFT model $\pi_{\text{ref}}$ as reference. Let $r_\theta(\tau, x) = \log \pi_\theta(\tau \mid x) - \log \pi_{\text{ref}}(\tau \mid x)$. The objective is $\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{(x,\tau^+,\tau^-)} \log \sigma(\beta \left[r_\theta(\tau^+, x) - r_\theta(\tau^-, x)\right])$, where $\beta$ controls the strength of the preference update. As in SFT, trajectory likelihoods are computed over model-generated tokens and tool actions, with solver observations used as conditioning context. This stage raises the probability of complete, verifier-supported traces and lowers the probability of correct-looking but weakly supported or failed ones.

**Policy Bootstrapping.** The success-conditioned bootstrapping mechanism relies on the teacher's *support* under the acceptance predicate. Let $Z_T(x) = \Pr_{\tau \sim p_T(\cdot|x)}[A(x, y^\star, \tau) = 1]$ denote the teacher's probability of producing an accepted trajectory for problem $x$. When $Z_T(x) > 0$, sampling $K_T$ rollouts exposes at least one Gold trace with probability $1 - (1 - Z_T(x))^{K_T}$. Trace curation turns these rare successes into a high-precision target: SFT amortizes accepted search behavior, while DPO uses Silver and Rejected traces to suppress nearby failure modes.

**Inference.** At test time, the student uses the same SMT-scratchpad protocol as teacher sampling and training. Given $x = (\mathcal{C}, q)$, it generates tool-use actions until `finish` or the interaction budget. We score the final answer against $y^\star$ and retain the full trajectory for process analysis: solver invocation, verdict quality, non-triviality, grounding, and protocol completion. For multi-sample evaluation, we draw $K$ independent trajectories per problem and aggregate answer and scratchpad-quality metrics.

## 3 Results

**Evaluation protocol.** We present evaluations of nine Sequent Prover checkpoints: base, supervised fine-tuned (SFT), and further preference-optimized (DPO) variants of the Qwen3.5–4B, Qwen3.5–9B, and Qwen3.6–27B families. Each model is evaluated with $K = 4$ stochastic SMT-scratchpad trajectories per problem. The final paired pool contains 37 benchmarks and 7,254 problems per model family. The suite covers logical and constraint reasoning (ProofWriter, RuleTaker, FOLIO, ProntoQA, AR-LSAT, ZebraLogic, ReClor, StrategyQA), arithmetic and spatial reasoning (GSM8K, SVAMP, MAWPS, ASDiv, AQuA-RAT, DROP, SpartQA-MC, SpartQA-YN, StepGame), NLI and multiple-choice commonsense/knowledge tasks (ANLI, BoolQ, WinoGrande, CommonsenseQA, OpenBookQA, QASC, ARC-Challenge, MMLU, HellaSwag), domain and reading-comprehension QA (MedQA, MedMCQA, PolicyQA, HotpotQA, MuSiQue, SQuAD, TriviaQA), and AGIEval LSAT/SAT subsets (LSAT-LR, LSAT-RC, SAT-Math, SAT-EN).

**Metrics.** We report *avg@$N$*, the per-attempt accuracy over all sampled trajectories; *pass@4*, the probability that at least one of four samples solves the problem; and *pass*ˆ4, the probability that all four samples solve the problem. We use passˆ4 as the main reliability metric because deployed stochastic agents should produce correct behavior consistently, rather than only under best-of-$K$ selection. We also report *stability*=passˆ4/pass@4, which normalizes all-sample consistency by four-sample coverage. For process analysis, we use the SMT-quality metrics from Section 2: first-pass Z3 validity, non-trivial grounded solver use, solver verdict distributions, and token/iteration efficiency.

**Research questions.** The results address four questions. **RQ1**: does verifier-conditioned post-training improve single-sample accuracy and multi-sample reliability? **RQ2**: how do SFT and DPO differ in training stages? **RQ3**: how does post-training change the solver-agent process? **RQ4**: where are the gains largest across tasks and model scales?

### 3.1 Aggregate accuracy and reliability

Table 1 shows the main comparison. Verifier-conditioned post-training yields the largest improvements for the 4B model: DPO raises avg@$N$ from 0.856 to 0.970 and passˆ4 from 0.641 to 0.917, a 27.7-point increase in all-sample reliability. The 9B model follows the same pattern from a stronger starting point, with DPO improving passˆ4 from 0.858 to 0.934. The 27B base model is already close to saturation on the aggregate pool, and DPO still reaches the best value in the family for every reported metric, including passˆ4 $= 0.986$.

The distinction between pass@4 and passˆ4 captures the central effect. The base checkpoints already have high four-sample coverage, especially at 9B and 27B, due to the multi-option nature of most

**Aggregate performance**

| Family | Ckpt. | avg@$N$ ↑ | pass@4 ↑ | pass^4 ↑ | Stab. ↑ | Δpass^4 |
|---|---|---|---|---|---|---|
| 4B | BASE | 0.856 | 0.970 | 0.641 | 0.661 | – |
| 4B | SFT | 0.960 | 0.995 | 0.892 | 0.896 | +0.251 |
| 4B | DPO | **0.970** | **0.998** | **0.917** | **0.919** | **+0.277** |
| 9B | BASE | 0.948 | 0.994 | 0.858 | 0.863 | – |
| 9B | SFT | 0.972 | 0.997 | 0.923 | 0.926 | +0.065 |
| 9B | DPO | **0.977** | **0.998** | **0.934** | **0.936** | **+0.076** |
| 27B | BASE | 0.991 | 0.998 | 0.977 | 0.979 | – |
| 27B | SFT | 0.989 | 0.999 | 0.971 | 0.972 | -0.006 |
| 27B | DPO | **0.995** | **1.000** | **0.986** | **0.986** | **+0.009** |

**pass@$k$ scaling**

| Model | @1 | @2 | @3 | @4 |
|---|---|---|---|---|
| 4B BASE | 0.856 | 0.943 | 0.963 | 0.970 |
| 4B DPO | **0.970** | **0.992** | **0.996** | **0.998** |
| 9B BASE | 0.948 | 0.984 | 0.991 | 0.994 |
| 9B DPO | **0.977** | **0.994** | **0.997** | **0.998** |
| 27B BASE | 0.991 | 0.996 | 0.998 | 0.998 |
| 27B DPO | **0.995** | **0.999** | **1.000** | **1.000** |

Table 1: Aggregate performance and pass@$k$ scaling on the 37-benchmark paired evaluation pool. Each family is compared against its own base checkpoint. pass@4 measures four-sample coverage, while pass^4 requires all four sampled trajectories to be correct. The pass@$k$ panel shows that a single 4B DPO sample matches the 4B base model's four-sample coverage. Bold marks the best checkpoint within each family.

of these tasks. Post-training concentrates probability mass on the correct trajectory mode, producing large gains in pass^4 and stability. For 4B, stability rises from 0.661 to 0.919 under DPO; for 9B, it rises from 0.863 to 0.936. This shows that the trained policies improve single-run reliability as well as best-of-four coverage.

Table 1 presents the pass@$k$ curves for the base and DPO checkpoints. The 4B result is especially clear: 4B DPO reaches 0.970 at pass@1, matching the 4B base model's pass@4 value. At 9B, DPO raises pass@1 from 0.948 to 0.977 and nearly saturates pass@4. At 27B, the curve is near the ceiling throughout, with DPO reaching 1.000 by pass@3.

### 3.2 Problem-level consistency transitions

Table 2a decomposes the reliability gains by the number of correct base-model attempts. Across 4B and 9B, the largest source of improvement is the middle tier: problems with one to three correct base attempts become correct on all four tuned attempts. For 4B DPO, this happens for 86.7% of the 2,379 partially correct base problems. For 9B DPO, it happens for 78.7% of the 970 partially correct base problems. The tuned models also preserve most problems that were already reliable: DPO retains 96.5%, 96.2%, and 99.3% of base 4/4 problems in the 4B, 9B, and 27B families, respectively.

The transition table also clarifies the role of scale. The 4B base model has 2,379 partially correct problems, so post-training has substantial headroom to improve reliability. The 27B base model has only 153 partially correct problems and 13 zero-correct problems, so aggregate gains are naturally smaller. Within that restricted headroom, DPO still improves the middle-tier conversion rate from 53.6% under SFT to 71.9%, and it preserves nearly all already reliable examples.

### 3.3 Training-stage ablation

Table 2b compares DPO directly against SFT on shared datasets. Preference optimization improves every family on avg@$N$, pass^4, stability, and verified-correct rate. The coverage overlap is large: for example, 4B SFT and 4B DPO both solve 7,213 problems at least once, while DPO adds 25 DPO-only problems and SFT adds 8 SFT-only problems. The most consistent DPO gains are therefore in reliability and trace quality. At 27B, the DPO stage is particularly useful: it adds +0.012 macro pass^4, +0.016 stability, +0.046 verified-correct rate, and +0.050 UNSAT-restricted verified-correct rate over SFT.

### 3.4 Solver-agent process quality

Table 3a shows that post-training improves the computational profile of the agent loop. The 4B DPO checkpoint reduces average iterations from 4.97 to 3.51, SMT calls from 4.10 to 2.52, and tokens from 18.97k to 12.72k. Normalizing by successful attempts, tokens per correct answer fall from 26.73k to 13.89k. The 9B and 27B families show the same pattern, with DPO giving the lowest token and solver-call cost within each family.

The first verified-correct SMT signal also arrives earlier after tuning. The mean first-success iteration drops from 2.00 to 1.29 for 4B DPO, from 1.62 to 1.22 for 9B DPO, and from 1.19 to 1.07 for 27B DPO. Under a two-iteration budget, 4B DPO reaches a verified-correct rate of 0.902, above the 4B base model's full-budget rate of 0.765; 9B DPO reaches 0.929 by iteration two, above the 9B base full-budget rate of 0.908. These early-exit curves show that post-training improves both endpoint ac-

(a) Problem-level transitions grouped by base-model correctness.

| Family | Tune | $n_{0/4}$ | $P(\geq 1/4 \mid 0/4)$ | $n_{1-3/4}$ | $P(4/4 \mid 1\text{–}3/4)$ | $n_{4/4}$ | $P(4/4 \mid 4/4)$ |
|---|---|---|---|---|---|---|---|
| 4B | SFT | 215 | 0.930 | 2,379 | 0.839 | 4,660 | 0.942 |
| 4B | DPO | 215 | **0.958** | 2,379 | **0.867** | 4,660 | **0.965** |
| 9B | SFT | 41 | **0.780** | 970 | 0.753 | 6,243 | 0.954 |
| 9B | DPO | 41 | **0.780** | 970 | **0.787** | 6,243 | **0.962** |
| 27B | SFT | 13 | 0.846 | 153 | 0.536 | 7,088 | 0.982 |
| 27B | DPO | 13 | **0.923** | 153 | **0.719** | 7,088 | **0.993** |

(b) Direct DPO–SFT comparison. Deltas are macro-averages over shared datasets; DPO-only/SFT-only paired solves are 25/8, 13/8, and 7/0 for 4B, 9B, and 27B.

| Family | $\Delta$avg@$N$ | $\Delta$pass^4 | $\Delta$stab. | $\Delta$VCR | $\Delta$VCR$_{\text{UNSAT}}$ |
|---|---|---|---|---|---|
| 4B | +0.005 | +0.009 | +0.007 | +0.050 | +0.013 |
| 9B | +0.006 | +0.008 | +0.011 | +0.036 | +0.026 |
| 27B | +0.005 | +0.012 | +0.016 | +0.046 | +0.050 |

Table 2: Performance transition and direct-comparison analyses for tuned checkpoints. In Table 2a, the three rates measure coverage gained on base-unsolved problems, conversion of partially correct problems to 4/4 correctness, and retention of already 4/4 problems. In Table 2b, metric deltas are macro-averages over shared datasets; DPO-only and SFT-only count paired problems solved by one tuned checkpoint and not the other.

(a) Solver-agent efficiency.

| Family | Ckpt. | Iter. ↓ | SMT calls ↓ | Tokens ↓ | Tok./ correct ↓ | First VC iter. ↓ |
|---|---|---|---|---|---|---|
| 4B | BASE | 4.97 | 4.10 | 18.97k | 26.73k | 2.00 |
| 4B | SFT | 3.76 | 2.78 | 13.70k | 15.23k | 1.33 |
| 4B | DPO | **3.51** | **2.52** | **12.72k** | **13.89k** | **1.29** |
| 9B | BASE | 4.37 | 3.41 | 15.69k | 19.53k | 1.62 |
| 9B | SFT | 3.52 | 2.54 | 12.31k | 14.16k | 1.24 |
| 9B | DPO | **3.33** | **2.34** | **11.85k** | **13.27k** | **1.22** |
| 27B | BASE | 2.92 | 1.90 | 9.51k | 10.28k | 1.19 |
| 27B | SFT | 2.85 | 1.86 | 7.92k | 8.52k | 1.08 |
| 27B | DPO | **2.58** | **1.58** | **6.78k** | **7.01k** | **1.07** |

(b) SMT scratchpad diagnostics.

| Family | Ckpt. | Error ↓ | No verdict ↓ | UNSAT ↑ | UNSAT \|correct ↑ | Calls |
|---|---|---|---|---|---|---|
| 4B | BASE | 41.3% | 10.3% | 12.3% | 12.9% | 122.6k |
| 4B | SFT | 19.1% | **0.3%** | 35.0% | 36.2% | 82.0k |
| 4B | DPO | **19.0%** | 0.4% | **35.3%** | **36.3%** | 73.3k |
| 9B | BASE | 26.0% | 0.4% | 22.7% | 23.0% | 101.1k |
| 9B | SFT | **13.9%** | **0.3%** | 38.7% | 39.7% | 75.0k |
| 9B | DPO | 14.4% | 0.4% | **40.0%** | **41.0%** | 68.7k |
| 27B | BASE | 16.9% | 0.7% | 25.0% | 25.2% | 59.8k |
| 27B | SFT | 5.8% | **0.1%** | 44.5% | 45.8% | 55.0k |
| 27B | DPO | **5.5%** | **0.1%** | **49.1%** | **49.7%** | 46.9k |

Table 3: Efficiency and SMT-scratchpad diagnostics. In Table 3a, Iter. is the average number of model iterations, SMT calls is the average number of solver invocations, Tokens is the average token count per problem attempt, Tok./correct normalizes token use by correct-attempt rate, and First VC iter. is the mean iteration index of the first verified-correct SMT signal. In Table 3b, Error and no-verdict measure unusable calls, UNSAT is the share of calls returning an UNSAT verdict, UNSAT|correct restricts to calls inside correct attempts, and Calls is total solver calls.

curacy and the efficiency frontier of the interactive solver loop.

Table 3b shows that the tuned policies emit cleaner SMT programs and obtain stronger solver signals. For 4B, the error rate falls from 41.3% to 19.0% under DPO, and no-verdict calls fall from 10.3% to 0.4%. For 27B, the error rate falls from 16.9% to 5.5%. The same table shows a shift toward refutational solver evidence: UNSAT calls increase from 12.3% to 35.3% for 4B DPO, from 22.7% to 40.0% for 9B DPO, and from 25.0% to 49.1% for 27B DPO. The UNSAT share inside correct attempts rises in parallel, reaching 49.7% for 27B DPO.

### 3.5 Benchmark-level trends

Table 4 reports representative per-dataset improvements for DPO. The highest-gain tasks are those with explicit symbolic or relational structure. At 4B, pass^4 improves by 85.2 points on ProofWriter, 59.0 points on RuleTaker, 52.5 points on AR-LSAT, 59.5 points on ZebraLogic, and 51.0 points on ProntoQA. At 9B, the largest gains include AR-LSAT, ZebraLogic, QASC, WinoGrande, CommonsenseQA, and ARC-Challenge. At 27B, aggregate scores are close to ceiling, and the clearest remaining gains are on ZebraLogic and AR-LSAT.

| Fam | Dataset | Base pass^4 | DPO pass^4 | $\Delta$ |
|---|---|---|---|---|
| 4B | ProofWriter | 0.074 | 0.926 | +0.852 |
| 4B | RuleTaker | 0.395 | 0.985 | +0.590 |
| 4B | AR-LSAT | 0.035 | 0.560 | +0.525 |
| 4B | ZebraLogic | 0.120 | 0.715 | +0.595 |
| 4B | ProntoQA | 0.445 | 0.955 | +0.510 |
| 4B | SpartQA-MC | 0.250 | 0.745 | +0.495 |
| 9B | AR-LSAT | 0.235 | 0.640 | +0.405 |
| 9B | QASC | 0.840 | 1.000 | +0.160 |
| 9B | ZebraLogic | 0.410 | 0.790 | +0.380 |
| 9B | WinoGrande | 0.845 | 0.990 | +0.145 |
| 9B | CommonsenseQA | 0.845 | 0.980 | +0.135 |
| 9B | ARC-Challenge | 0.870 | 0.985 | +0.115 |
| 27B | ZebraLogic | 0.825 | 0.965 | +0.140 |
| 27B | AR-LSAT | 0.785 | 0.920 | +0.135 |
| 27B | ReClor | 0.960 | 0.985 | +0.025 |
| 27B | StepGame | 0.825 | 0.860 | +0.035 |

Table 4: Largest representative dataset-level DPO gains on pass^4 within each model family.

These dataset-level results align with the scratchpad interpretation of the method. The strongest gains occur where the task contains latent constraints, relational structure, arithmetic structure, or entailment structure that can be made executable. Broad QA and domain tasks also benefit in the smaller model families, with improvements on ARC-Challenge, CommonsenseQA, MedQA,

MedMCQA, OpenBookQA, QASC, and WinoGrande. The aggregate effect therefore spans more than formal logic benchmarks while remaining largest on tasks with natural SMT structure.

| Comparison | Small | Large | Tie | Win |
|---|---|---|---|---|
| 4B SFT / 9B BASE | 816 | 518 | 5,920 | 0.612 |
| 4B DPO / 9B BASE | 840 | 361 | 6,053 | **0.699** |
| 9B SFT / 27B BASE | 73 | 490 | 6,691 | 0.130 |
| 9B DPO / 27B BASE | 90 | 420 | 6,744 | **0.176** |
| 4B DPO / 27B BASE | 84 | 532 | 6,638 | 0.136 |

Table 5: Cross-scale paired comparison. Small and Large count paired problems where the smaller tuned checkpoint or larger base checkpoint has more correct attempts out of four. Win is the smaller tuned checkpoint's non-tie win rate.

**Post-training and scale.** Table 5 compares tuned smaller models against larger base models. The 4B tuned agents compare favorably with the 9B base model: 4B SFT wins 61.2% of non-tied paired comparisons, and 4B DPO wins 69.9%. The 27B base model sets a higher aggregate ceiling, and the tuned 9B models win fewer non-tied comparisons against it. Combining scale and post-training gives the strongest overall checkpoints: 27B DPO has the highest aggregate avg@$N$, pass@4, pass$\hat{}$4, and SMT-quality profile in the sweep.

## 4 Related Works

**Verifier-Integrated Formal Reasoning.** Recent progress in formal theorem proving has shown that language models can benefit greatly from emitting machine-checkable artifacts rather than only natural-language explanations. Systems such as Goedel Prover, DeepSeek-Prover, and Kimina-Prover combine long-form reasoning with formal verification and iterative refinement (Lin et al., 2025b; Ren et al., 2025; Wang et al., 2025). Our work is inspired by this paradigm, but shifts the target language from Lean to SMT-LIB and focuses on compact models that must operate across multiple SMT logics.

**Solver-Based Symbolic Reasoning and Synthesis.** SMT solvers such as Z3 and cvc5 are central tools for formal verification, constraint solving, and program synthesis (de Moura and Bjørner, 2008; Barbosa et al., 2022). Classical symbolic systems rely on hand-written encodings or search procedures over formal objects. Our setting is different: the primary generator is a language model trained to emit solver-executable artifacts directly, while a symbolic backend is used to validate outputs and to synthesize additional training data. The use of SymPy as a controllable symbolic front-end is especially important for scaling supervision across diverse logic fragments (Meurer et al., 2017).

**Self-Repair and Tool Feedback.** Outside theorem proving, a large body of work studies how models can improve outputs by reacting to execution feedback, compiler messages, or explicit critics (Yao et al., 2023; Shinn et al., 2023; First et al., 2023; Olausson et al., 2024; Chen et al., 2024; Bouzenia et al., 2025). We adopt the same high-level intuition, but the repair channel here is solver-native: parser errors, sort mismatches, failed assertions, and counterexamples. This makes the feedback more structured than free-form critique and better aligned with symbolic correctness.

**Reinforcement Learning for Verifiable Outputs.** RL has recently become an effective way to improve pass@1 performance in reasoning and formal generation tasks (Shao et al., 2024; Liu et al., 2025; Yu et al., 2025). Our work extends that lesson to SMT-native generation. Because repair is serial and strongly depends on first-round quality, improving the expected solver reward is particularly useful in our setting.

## 5 Conclusion

Sequent-Prover shows that solver access can be learned as a formal-reasoning policy rather than used as a one-off tool call. By filtering trajectories with executable SMT feedback and contrasting successful traces against failed ones, our method amortizes solver-guided search into the model: it learns to formulate proof obligations, encode them in SMT-LIB, interpret Z3 responses, repair formal artifacts, and commit with evidence. Empirically, post-training turns sample-dependent solver successes into stable behavior, produces more grounded and non-vacuous SMT programs, and shifts correct traces toward refutational UNSAT evidence. The scale-dependent results clarify where this training is most useful: when a base model has latent solver-use competence but lacks consistency. Overall, Sequent-Prover reframes theorem prover interaction as a post-training target in its own right, pointing toward language models whose reasoning is not only generated, but checked.

## 6 Limitations

Sequent-Prover inherits a fundamental limitation of formal verification itself: a solver can check a formal artifact, but it cannot decide whether the chosen abstraction is the only, best, or intended one. This boundary is not specific to language models; even expert verification engineers must decide what to model, what to omit, which predicates or theories to introduce, and which specification captures the informal task. This takes months of time. Our goal is therefore not to completely eliminate the autoformalization problem, but to make it executable, inspectable, and learnable: Sequent-Prover trains models to construct grounded, non-trivial SMT artifacts, use solver feedback, and revise failed abstractions through interaction. The broader challenge of learning the right level of formal abstraction remains open.

## References


Haniel Barbosa, Clark Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. 2022. cvc5: A Versatile and Industrial-Strength SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 415–442. Springer.

Clark Barrett, Pascal Fontaine, and Cesare Tinelli. 2025. The SMT-LIB standard: Version 2.7. Technical report, Department of Computer Science, The University of Iowa. Available at `https://smt-lib.org`.

Clark Barrett and Cesare Tinelli. 2018. Satisfiability modulo theories. *Handbook of model checking*, pages 305–343.

Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. 2025. RepairAgent: An autonomous, LLM-based agent for program repair. In *International Conference on Software Engineering*, pages 2188–2200.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

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

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 conference of the north American chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)*, pages 2924–2936.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Peter Clark, Oyvind Tafjord, and Kyle Richardson. 2020. Transformers as soft reasoners over language. *arXiv preprint arXiv:2002.05867*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2368–2378.

Yu Feng, Nathaniel Weir, Kaj Bostrom, Sam Bayless, Darion Cassel, Sapana Chaudhary, Benjamin Kiesl-Reiter, and Huzefa Rangwala. 2025. Vericot: Neuro-symbolic chain-of-thought validation via logical consistency checks. *arXiv preprint arXiv:2511.04662*.

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

Debargha Ganguly, Srinivasan Iyengar, Vipin Chaudhary, and Shivkumar Kalyanaraman. 2024. PROOF OF THOUGHT : Neurosymbolic program synthesis allows robust and interpretable reasoning. In *The First Workshop on System-2 Reasoning at Scale, NeurIPS'24*.

Debargha Ganguly, Vikash Singh, Sreehari Sankar, Biyao Zhang, Xuecen Zhang, Srinivasan Iyengar, Xiaotian Han, Amit Sharma, Shivkumar Kalyanaraman, and Vipin Chaudhary. 2026. Grammars of formal uncertainty: When to trust LLMs in automated reasoning tasks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, and 1 others. 2024. Folio: Natural language reasoning with first-order logic. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 22017–22031.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611.

Tushar Khot, Peter Clark, Michal Guerquin, Peter Jansen, and Ashish Sabharwal. 2020. Qasc: A dataset for question answering via sentence composition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8082–8090.

Takeshi Kojima, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, pages 22199–22213. Curran Associates, Inc.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. Mawps: A math word problem repository. In *Proceedings of the 2016 conference of the north american chapter of the association for computational linguistics: human language technologies*, pages 1152–1157.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, and 1 others. 2023. Measuring faithfulness in chain-of-thought reasoning. *arXiv preprint arXiv:2307.13702*.

Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. 2025a. ZebraLogic: On the scaling limits of LLMs for logical reasoning. *arXiv preprint arXiv:2502.01100*.

Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou Xia, Danqi Chen, Sanjeev Arora, and Chi Jin. 2025b. Goedel-Prover: A frontier model for open-source automated theorem proving. *arXiv preprint arXiv:2502.07640*.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 158–167.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025. Understanding R1-Zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*.

Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, Amit Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, and 8 others. 2017. SymPy: Symbolic computing in Python. *PeerJ Computer Science*, 3:e103.

Shen-Yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A diverse corpus for evaluating and developing english math word problem solvers. In *Proceedings of the 58th annual meeting of the Association for Computational Linguistics*, pages 975–984.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2381–2391.

Roshanak Mirzaee, Hossein Rajaby Faghihi, Qiang Ning, and Parisa Kordjamshidi. 2021. Spartqa: A textual question answering benchmark for spatial reasoning. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4582–4598.

Theo X Olausson, Alex Gu, Benjamin Lipkin, Cedegao E Zhang, Armando Solar-Lezama, Joshua B Tenenbaum, and Roger Levy. 2023. LINC: A Neurosymbolic Approach for Logical Reasoning by Combining Language Models with First-Order Logic Provers. *arXiv preprint arXiv:2310.15164*.

Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. 2024. Is self-repair a silver bullet for code generation? In *International Conference on Learning Representations*.

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

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple

math word problems? In *Proceedings of the 2021 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 2080–2094.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 2383–2392.

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. 2025. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1743–1752.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Abulhair Saparov and He He. 2022. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. *arXiv preprint arXiv:2210.01240*.

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

Zhengxiang Shi, Qiang Zhang, and Aldo Lipani. 2022. Stepgame: A new benchmark for robust multi-hop spatial reasoning in texts. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 11321–11329.

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

Vikash Singh, Darion Cassel, Nathaniel Weir, Nick Feng, and Sam Bayless. 2026. Verge: Formal refinement and guidance engine for verifiable llm reasoning. *arXiv preprint arXiv:2601.20055*.

Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. 2021. Proofwriter: Generating implications, proofs, and abductive statements over natural language. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3621–3634.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. musique: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. 2023. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36:74952–74965.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, Jianqiao Lu, Hugues de Saxcé, Bolton Bailey, Chendong Song, Chenjun Xiao, Dehao Zhang, Ebony Zhang, Frederick Pu, Han Zhu, and 21 others. 2025. Kimina-Prover preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. 2023. SATLM: Satisfiability-Aided Language Models using Declarative Prompting. In *Proceedings of the International Conference on Neural Information Processing Systems*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, and

16 others. 2025. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Weihao Yu, Zihang Jiang, Yanfei Dong, and Jiashi Feng. 2020. Reclor: A reading comprehension dataset requiring logical reasoning. *arXiv preprint arXiv:2002.04326*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 4791–4800.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2024. Agieval: A human-centric benchmark for evaluating foundation models. In *Findings of the association for computational linguistics: NAACL 2024*, pages 2299–2314.

Wanjun Zhong, Siyuan Wang, Duyu Tang, Zenan Xu, Daya Guo, Jiahai Wang, Jian Yin, Ming Zhou, and Nan Duan. 2021. AR-LSAT: Investigating analytical reasoning of text. *arXiv preprint arXiv:2104.06598*.

# A Background: Agent, Solver & Training

Before diving into the evaluation metrics it is essential to fix three pieces of background: (1) *what Z3 actually computes*, (2) *how the language model interacts with it*, and (3) *how the two post-training recipes differ in what they teach*. Each of these choices is reflected in the metric definitions that follow, and the metric definitions only carry their intended meaning under these operational assumptions.

## A.1 The Z3 SMT Solver and the Two Proof Modes

Z3 is a Satisfiability Modulo Theories (SMT) solver: given a quantifier-free first-order formula $\varphi$ over a combination of background theories (linear/non-linear integer or real arithmetic, fixed-size bit-vectors, equality with uninterpreted functions, strings, arrays, datatypes), Z3 returns one of

- SAT: $\varphi$ has a model under the theory combination, and Z3 (optionally) returns it as (`define-fun` $x \ldots$) definitions;
- UNSAT: $\varphi$ is unsatisfiable in the theory combination—no assignment can make it true;
- UNKNOWN: Z3's procedure was incomplete or timed out;
- an ERROR string: the input did not parse, used a symbol Z3 cannot interpret, or violated the declared sort.

Our agent uses Z3 in two operationally distinct modes; the distinction is central to how to read the verdict mix in §B.3.

**Direct constraint-satisfaction (the SAT mode).** The agent encodes the problem's premises as constraints and the candidate answer as an additional assertion, then queries `(check-sat)`. A return of SAT establishes that the candidate answer is *consistent* with the premises and the agent may extract a satisfying model from Z3's `(get-model)` output. For an open-ended constraint-search task (e.g., ZebraLogic, where the agent must find "the house with the cat"), the witness *is* the answer. For a deductive task (e.g., ProofWriter, where the agent must say whether "$p$ entails $q$"), a SAT verdict on $\Gamma \cup \{q\}$ only tells us that $q$ is *not* contradicted by $\Gamma$, which is a strictly weaker claim than $\Gamma \models q$.

**Proof by contradiction (the UNSAT mode).** The deductively-strong way to use Z3 for an entailment question is to assert the premises together with the *negation* of the candidate conclusion, $\Gamma \cup \{\neg q\}$, and check satisfiability. A return of UNSAT establishes that $\Gamma \cup \{\neg q\}$ has no model, i.e. $\Gamma \models q$ in the theory. This is the standard refutational argument that underlies essentially all SMT-based theorem proving. An agent that systematically produces UNSAT verdicts on deductive tasks is using the solver as a *prover*; an agent that systematically produces SAT verdicts on the same tasks is using it as a *model-finder* and is, in the worst case, verifying nothing more than internal consistency.

The pair of metrics vcr and $\text{vcr}_{\text{UNSAT}}$ introduced in §B.3 make this distinction quantitative: a delta in $\text{vcr}_{\text{UNSAT}}$ larger than the delta in vcr indicates that post-training is shifting the agent toward refutational rather than model-finding use of the solver, even when the underlying correctness count is unchanged.

## A.2 The Tool-Calling Agent Loop

The agent harness uses an *OpenAI-compatible* tool-calling protocol exposed by vLLM. At each iteration $\tau$ the model receives the cumulative conversation $\mathcal{H}_\tau = (m_1, m_2, \ldots)$ consisting of the system prompt, the user problem, every prior model turn, and the structured payload from every prior tool call. The model may emit one of two output shapes:

1. a *tool call* to `z3_solve` with arguments $\{\texttt{smt_code} : s_\tau, \texttt{timeout} : \texttt{int}\}$, parsed by Qwen's `qwen3_coder` tool grammar; or
2. a *free-text* reply containing the final answer.

Reasoning content (Qwen's ⟨think⟩…⟨/think⟩ block) is parsed by the `qwen3` reasoning parser and is not re-emitted to subsequent turns; only the model's user-visible reply plus its tool-call schema is carried forward, which prevents the context window from growing quadratically in $T_{\max}$.

When the agent calls `z3_solve`, the harness runs $s_\tau$ under a 30-second wall-clock cap and a 256-MB memory cap, captures the full stdout/stderr, and returns the payload $(\texttt{success}, \texttt{output}, \texttt{sat_result}, \texttt{error})$ as a tool-response message. Crucially, the `sat_result` field is *not* parsed from Z3's stdout post-hoc—it is the exit-line of `(check-sat)`, normalised to one of $\{$SAT, UNSAT, UNKNOWN$\}$, with the original output preserved separately so the agent (and our analyzer) can recover `(get-model)` witnesses or error stacks.

The loop terminates when the model produces an attempt without a tool call (success), when it explicitly emits the special `finish` terminator (success), or when $\tau$ reaches $T_{\max} = 10$ (timeout). The terminating attempt's free-text answer is run through the dataset-specific `check_correct` extractor (which handles option-letter matching for MCQ, multiset matching for ZebraLogic, exact-string for arithmetic, etc.) to produce the correctness bit $c_p^{(k)}$.

The boolean attribute `verified` on each JSONL row is *not* the correctness bit. It records whether the agent terminated by producing an answer (`verified` = TRUE) or by hitting the iteration cap and being forced to extract whatever the model last said (`verified` = FALSE); the metrics *soundness* and *completeness* in §B.2 treat `verified` as a confidence/commitment signal, not as correctness.

### A.3 Datasets.

The training set is intentionally narrow. We sample 1,000 questions each from the training splits of three hard reasoning datasets: AR-LSAT (Zhong et al., 2021), FOLIO (Han et al., 2024), and ZebraLogic (Lin et al., 2025a). These datasets were chosen because they require different forms of symbolic reasoning: AR-LSAT consists of analytical reasoning problems with combinatorial constraints, FOLIO requires reasoning over first-order logical premises, and ZebraLogic contains constraint-satisfaction puzzles with latent entity–attribute assignments. This gives the model supervision on problems where success typically requires constructing and checking an explicit symbolic state, rather than relying only on lexical or parametric associations.

At evaluation time, we enforce strict decontamination: no problem used for training is included in evaluation, and evaluation examples are drawn from held-out splits whenever labels are available. We therefore distinguish between *in-distribution* evaluation, which consists only of held-out examples from the same three dataset families used during training, and *out-of-distribution* evaluation, which covers all other benchmark sources. This distinction is at the dataset-family level rather than the instance level: even the in-distribution setting is instance-disjoint from training, while the out-of-distribution setting tests transfer to new annotation protocols, answer formats, domains, and reasoning styles.

Our full evaluation suite is deliberately broad. In addition to the three training-source benchmarks, we evaluate on logical and rule-based reasoning tasks such as StrategyQA, ProntoQA, ProofWriter, ReClor, and RuleTaker; arithmetic and discrete reasoning benchmarks such as GSM8K, SVAMP, MultiArith, ASDiv, AQuA-RAT, MAWPS, and DROP; spatial reasoning tasks including SpartQA and StepGame; commonsense and science multiple-choice benchmarks including OpenBookQA, QASC, ARC-Challenge, CommonsenseQA, and WinoGrande; broad-coverage multiple-choice benchmarks including MMLU, HellaSwag, BoolQ, MedQA, and MedMCQA; and free-form multi-hop QA datasets including HotpotQA, MuSiQue, SQuAD, and TriviaQA (Geva et al., 2021; Saparov and He, 2022; Tafjord et al., 2021; Han et al., 2024; Zhong et al., 2024; Lin et al., 2025a; Yu et al., 2020; Clark et al., 2020; Cobbe et al., 2021; Patel et al., 2021; Roy and Roth, 2015; Miao et al., 2020; Ling et al., 2017; Koncel-Kedziorski et al., 2016; Dua et al., 2019; Mirzaee et al., 2021; Shi et al., 2022; Mihaylov et al., 2018; Khot et al., 2020; Clark et al., 2018; Talmor et al., 2019; Sakaguchi et al., 2021; Hendrycks et al., 2020; Zellers et al., 2019; Clark et al., 2019; Yang et al., 2018; Trivedi et al., 2022; Rajpurkar et al., 2016; Joshi et al., 2017). The resulting split tests whether post-training on a small set of hard symbolic datasets teaches a general solver-use behavior that transfers beyond the observed training distri-

bution, rather than merely improving accuracy on the benchmark families used for supervision.

### A.4 Reproducibility.

We commit to releasing the SMT harness as a plug-and-play package, curated trajectories for training, prompts and model checkpoints for the academic community and towards the broader AI developer community.

**Training Data.** Our trajectory dataset contains a total of 14307 training samples, containing 10785 gold, 2995 reject and 464 silver trajectories. This data was generated by using our verifier-conditioned trajectory distillation from a Deepseek-v4 Flash model. Temperature during both dataset curation and all evaluation was maintained at 0.9.

**Training details.** We post-train models using full-parameter fine-tuning with DeepSpeed ZeRO-3. For supervised fine-tuning, we optimize the autoregressive language-modeling objective over pre-rendered Qwen3.5 chat-template traces. Each training example is stored as a single text sequence that preserves the exact interleaving of reasoning, tool calls, and solver outputs, including `<think>...</think>`, `<tool_call>...</tool_call>`, and `<tool_response>...</tool_response>` spans. We use this representation directly rather than converting through an OpenAI- or ShareGPT-style message format, since the latter can discard assistant reasoning text when tool calls are also present. This ensures that the model is trained on the same rendered traces that it is expected to produce at inference time.

The supervised run uses the base checkpoint, full fine-tuning, bfloat16 precision, gradient checkpointing, FlashAttention, and a maximum sequence length of 16,384 tokens. We enable sequence packing for throughput. Training is performed for two epochs on 8×A100-80GB GPUs or H200 GPUs according to SLURM availability with per-device batch size 1 and gradient accumulation 8, giving an effective batch size of 64. We use AdamW with learning rate $3 \times 10^{-6}$, cosine learning-rate decay, warmup ratio 0.03, weight decay 0, and gradient clipping at norm 1.0. Checkpoints are saved every 10 optimizer steps, retaining the three most recent checkpoints.

We then apply Direct Preference Optimization (DPO) starting from the last supervised checkpoint, where the supervised loss had plateaued in our run. The preference data consists of chosen/rejected SMT autoformalization traces. We train on the entire assistant trajectory, while preserving the same Qwen3.5 chat-template rendering. We use the standard sigmoid DPO objective with $\beta = 0.1$. DPO is also performed with full-parameter fine-tuning, bfloat16 precision, gradient checkpointing, FlashAttention, and DeepSpeed ZeRO-3. Because DPO evaluates chosen and rejected continuations separately and does not use packing, we set the cutoff length to 8,192 tokens and discard preference pairs whose prompt-plus-response length exceeds this limit, avoiding truncation inside tool-call spans. The DPO run is trained for two epochs on 8×A100-80GB GPUs with per-device batch size 1 and gradient accumulation 4, giving an effective batch size of 32. We use learning rate $5 \times 10^{-7}$, cosine decay, warmup ratio 0.1, weight decay 0, and gradient clipping at norm 1.0.

**Scope, Design Choices and Limitations** Several aspects of our setup reflect deliberate scope decisions rather than oversights, and we make them explicit here. First, Sequent-Prover builds directly on the SMT-LIB backend of Proof-of-Thought (Ganguly et al., 2024), and our contribution is the post-training recipe that turns sporadic solver use into reliable, grounded behavior; we therefore do not re-compare against test-time-scaling approaches such as Grammars of Formal Uncertainty or VERGE (Singh et al., 2026; Ganguly et al., 2026), which sample on the order of 100s of programs per question, because an apples-to-apples comparison would require granting our agents a comparable test-time scaling budget, which is beyond the scope of this work. Second, because Sequent-Prover is a recipe rather than a single tuned configuration, and because each component ablation requires a full post-training-plus-paired-evaluation sweep, isolating the marginal contribution of every design choice is computationally prohibitive; we instead supply broad empirical evidence that the recipe as a whole improves reliability, grounding, and refutational solver use across nine checkpoints and 37 benchmarks, and we expect that further tuning of individual components would yield additional gains in follow-up work. Third, our headline improvements concentrate on the smaller models and on tasks with latent symbolic structure: the 27B Qwen3.6 base model is already a remarkably strong agentic tool-caller and code generator, and our post-training still pushes it to near-ceiling on the aggregate pool, which indi-

cates that the principal remaining headroom now lies in harder, less-saturated tasks rather than in the current suite. Fourth, we report results in aggregate over 7,254 problems per family rather than enumerating per-dataset tables for every metric, not for lack of statistical rigor but because the full breakdown would expand the paper past any reasonable length while the aggregate pool is large enough to make the reported deltas faithful and meaningful; the full metric family of Appendix B is defined for completeness and downstream analysis, and every auxiliary measure—soundness, recovery, calibration, skeleton diversity, and the rest—trends consistently with the reliability and verifier-use metrics highlighted in the main tables, so we omit exhaustive enumeration for the reader's sanity rather than because any measurement disagrees. In particular, the predicted SFT-induced soundness drop does not translate into degraded reliability for our checkpoints. Fifth, the grounding and triviality predicates are shared between curation and evaluation by design: applying the same permissive, substring-based criteria to every checkpoint, including the untrained baselines, levels the playing field and, if anything, biases comparisons toward the baselines rather than against them. Finally, all base models and the DeepSeek-v4-Flash teacher used here are trained and rendered under their respective chat-template format; the mixed point-release versions within a family sweep are intentional and reflect the actual base models used, and we will release the SMT harness, the full set of 14,307 curated trajectories, prompts, and all checkpoints upon publication so that reported numbers are reproducible under the identical harness.

**License, Compute Budget, Risks** All evaluation benchmarks are public datasets used under their original research licenses; Z3 (MIT), and the Qwen base models (under their respective open model licenses) are used in accordance with their terms, and we will release our harness, trajectories, and checkpoints under a permissive open-source license for research use. Across prototyping, repeated iteration, teacher rollout generation, SFT, DPO for all three model families, the full sweep used approximately 1000 GPU-hours on A100-80GB/H200 hardware. Our work studies formal reasoning over public academic benchmarks and introduces no new data, deployment surface, or generative capability that poses foreseeable risk of harm.

# B Evaluation Metrics

Let $\mathcal{D}$ be a dataset with $M = |\mathcal{D}|$ problems, each problem $x \in \mathcal{D}$ having a gold answer $y_x^\star$. For each problem the model produces $K$ i.i.d. samples; the $r$-th attempt yields a trace $\tau_x^{(r)}$ and a final answer $\hat{y}_x^{(r)}$, plus a sequence of SMT calls

$$\big(z_x^{(r,1)}, \ldots, z_x^{(r,S_x^{(r)})}\big)$$

with associated Z3 responses

$$\big(\rho_x^{(r,1)}, \ldots, \rho_x^{(r,S_x^{(r)})}\big).$$

We write

$$\chi_x^{(r)} := \mathbb{1}\Big[\texttt{check correct}\big(\hat{y}_x^{(r)}, y_x^\star\big)\Big]$$

$$\chi_x := \sum_{r=1}^{K} \chi_x^{(r)} \in \{0, 1, \ldots, K\}.$$

$\chi_x$ is the number of correct attempts on problem $x$, and $\bar{\chi}_x := \chi_x / K$ is the per-problem accuracy. The boolean $\kappa_x^{(r)}$ denotes the commitment event, i.e. the `verified` JSONL field of §A.2; $T_x^{(r)}$ denotes the iteration count; and $S_x^{(r)}$ denotes the number of SMT calls. Unless stated otherwise we use $K = 4$.

We use $r$ for the sample index and reserve $n$ for the sample budget inside metrics such as pass@$n$ and maj@$n$. We group metrics into three families: *sample-based correctness* (§B.1), which depend only on the bag of correctness indicators $\{\chi_x^{(r)}\}$; *agent-reliability measures* (§B.2), which characterize the relationship between the agent's internal commitment signal and its actual correctness; and *SMT-program quality* (§B.3), which inspect the structural and semantic properties of the SMT-LIB code that the agent emitted, independent of whether the final answer was correct.

## B.1 Sample-Based Correctness

The metrics in this subsection depend only on the multiset $\{\chi_x^{(r)}\}_{x,r}$; they do not inspect the agent trace. Each is a moment of the per-problem correctness count $\chi_x \in \{0, \ldots, K\}$ under a different functional form, designed to isolate different facets of the underlying success distribution.

**avg@$N$.** The per-attempt expected accuracy, computed as the empirical mean of $\bar{\chi}_x$ across problems:

$$\text{avg@}N = \frac{1}{M} \sum_{x \in \mathcal{D}} \bar{\chi}_x = \frac{1}{MK} \sum_{x,r} \chi_x^{(r)}. \quad (1)$$

avg@$N$ has the operational interpretation of *single-attempt success*: if a user runs the agent once with one temperature draw, avg@$N$ is the expected correctness of that single attempt. It is invariant to which of the $K$ samples we examine, and it is the unique linear functional of the $\{\chi_x^{(r)}\}$ that is unbiased for the underlying per-attempt correctness probability.

**pass@$n$.** The unbiased estimator of Chen et al. (2021) for the probability that at least one of $n$ samples is correct: for $n \le K$,

$$\text{pass@}n = \frac{1}{M}\sum_{x\in\mathcal{D}}\left[1 - \frac{\binom{K-\chi_x}{n}}{\binom{K}{n}}\right]. \tag{2}$$

The estimator is unbiased under random subsampling without replacement from the $K$ observed attempts: $\binom{K-\chi_x}{n}/\binom{K}{n}$ is the exact probability of drawing $n$ *all-incorrect* attempts, and pass@$n$ is one minus its expectation. pass@1 = avg@$N$ exactly. For $n > 1$, the quantity reflects the user-facing utility of taking multiple samples and accepting the answer if any one is correct—an *ensemble-with-oracle-selection* ceiling. We report pass@4 throughout.

**pass$^n$.** The expected probability that *every* one of $n$ samples is correct, under the same random-subsampling estimator:

$$\text{pass}^n = \frac{1}{M}\sum_{x\in\mathcal{D}}\frac{\binom{\chi_x}{n}}{\binom{K}{n}}. \tag{3}$$

$\binom{\chi_x}{n}/\binom{K}{n}$ is the probability that all $n$ randomly drawn attempts are correct. Two checkpoints with the same pass@$n$ may differ dramatically in pass$^n$: pass$^n$ rewards *distributional concentration on the right answer*—high values arise iff the right answer is the modal answer across samples, low values arise when correctness is sample-dependent.

**maj@$n$.** The expected probability that majority vote among $n$ samples agrees with the gold answer. Because `check correct` is binary, majority on correctness is well-defined and gives, under random subsampling without replacement and the conservative tie convention of strict majority, with ties broken against the agent,

$$\text{maj@}n = \frac{1}{M}\sum_{x\in\mathcal{D}}\frac{1}{\binom{K}{n}}\sum_{j=\lfloor n/2\rfloor+1}^{n}\binom{\chi_x}{j} \times \binom{K-\chi_x}{n-j}. \tag{4}$$

We report maj@4 throughout. The gap between maj@$n$ and avg@$N$ measures the benefit of self-consistency aggregation, which assumes wrong answers are more diverse than right ones.

**Stability.** The ratio of consistency to coverage,

$$\text{stability} = \frac{\text{pass}^n}{\text{pass@}n} \in [0,1]. \tag{5}$$

A value of 1 indicates perfect consistency: any problem the model can solve, it solves on *every* attempt. A value of 0 indicates the right answer is purely sample-dependent. The stability delta is the cleanest one-number summary of the post-training phenomenon documented in our results.

**AUC@$n$.** A finer-grained "area-under-the-curve" summary of the full $(\text{pass@}j)_{j=1}^{n}$ trajectory, used internally by the reporting pipeline:

$$\text{AUC@}n = \frac{1}{n}\sum_{j=1}^{n}\text{pass@}j. \tag{6}$$

AUC@$n$ favors checkpoints whose accuracy at small $j$ is already high—i.e., checkpoints that do not depend on the oracle-selection ceiling. Equivalently, it can be read as a discount on later samples and so penalizes sample-greedy ensembles.

**g-pass@$n_\alpha$.** The threshold-pass variant: g-pass@$n_\alpha$ is the expected indicator that at least a fraction $\alpha \in (0,1]$ of $n$ random samples is correct. $\alpha = 1$ recovers pass$^n$, and $\alpha = 1/n$ recovers pass@$n$; we report the standard $\alpha \in \{0.25, 0.5, 0.75, 1.0\}$ ladder in the per-dataset reports of `nl2smt.analyze`, but only the endpoints in the main tables below.

### B.2 Agent-Reliability Measures

These metrics compare the agent's behavior conditional on internal events. As established in §A.2, $\kappa_x^{(r)}$ is the *commitment* bit, namely the JSONL `verified` field: it is 1 iff the agent terminated the loop by producing an answer rather than hitting $T_{\max}$. $\chi_x^{(r)}$ is the correctness bit. $\kappa_x^{(r)} = 1, \chi_x^{(r)} = 0$ is the *overcommit* cell—the agent declared confidence in a wrong answer—and is the operative quantity in the soundness metric.

**Soundness.**

$$\text{soundness} = \mathbb{P}(\chi = 1 \mid \kappa = 1) = \frac{\sum_{x,r}\chi_x^{(r)}\kappa_x^{(r)}}{\sum_{x,r}\kappa_x^{(r)}}. \tag{7}$$

Among attempts where the agent terminated voluntarily—i.e., chose to commit rather than continue refining—what fraction were correct? Soundness is a measure of *decision quality*: it asks whether the agent's internal stopping criterion is well-aligned with ground truth. Low soundness indicates an agent that "thinks it's done" while still wrong; a drop in soundness during post-training is diagnostic of training-induced overconfidence, typically introduced by SFT on demonstration traces that all terminate successfully.

**Completeness.**

$$\text{completeness} = \mathbb{P}(\kappa = 1 \mid \chi = 1). \tag{8}$$

Among correct attempts, what fraction terminated voluntarily? Low completeness is the dual failure to soundness: the agent silently arrives at a correct answer but does not commit, instead burning iterations until it hits $T_{\max}$. In practice every reported configuration in this work has completeness $\geq 0.999$ because Qwen reliably emits a `finish` terminator when satisfied. We retain completeness in the table for diagnostic value and because failure modes on novel datasets could plausibly violate it.

**Recovery.**

$$\text{recovery} = \mathbb{P}\big(\chi = 1 \mid \text{first SMT call errored}\big). \tag{9}$$

The conditioning event is precisely

$$\{\texttt{success}(z_x^{(r,1)}) = \textsc{False}\}$$

or the first call's `sat result` was missing; i.e., the very first Z3 invocation either failed to parse, ran out of resources, or did not return a verdict. *Recovery* measures the agent's robustness to early failure: a high value indicates the agent diagnoses the error from Z3's stderr text, rewrites the SMT, and eventually arrives at the right answer; a low value indicates the agent commits to a wrong answer after the first bad call. This metric is methodologically subtle: the difference recovery $-$ avg@$N$ measures the *conditional* penalty of a first-call failure, with the unconditional accuracy as baseline.

**Expected Calibration Error.** With only a binary commitment signal $\kappa$ and binary correctness $\chi$, ECE collapses to the absolute gap between the per-attempt commitment rate and the per-attempt accuracy:

$$\text{ECE} = \big|\,\overline{\kappa} - \overline{\chi}\,\big|. \tag{10}$$

For a well-calibrated agent, the rate at which it commits matches the rate at which it is right. Lower values are better; the metric becomes a binned ECE if a continuous confidence signal is supplied via `att["confidence"]`, but none of our checkpoints expose one.

**Tool efficiency and turn efficiency.** Let $S_x^{(r)}$ be the number of SMT calls and $T_x^{(r)}$ the number of agent iterations in attempt $(x, r)$. Then

$$\text{tool eff.} = \frac{\text{avg@}N}{\overline{S}}, \qquad \text{turn eff.} = \frac{\text{avg@}N}{\overline{T}},$$
$$\overline{S} = \frac{1}{MK}\sum_{x,r} S_x^{(r)}, \qquad \overline{T} = \frac{1}{MK}\sum_{x,r} T_x^{(r)}. \tag{11}$$

Both ratios are dimensionally CORRECTNESS/COMPUTE-UNIT: tool efficiency is accuracy per SMT call, the marginal value of *querying the solver*; turn efficiency is accuracy per agent iteration, the marginal value of *calling the model*. Because $T_x^{(r)} \geq S_x^{(r)} + 1$, each SMT call consumes an iteration and the final answer takes another, turn efficiency is always the smaller of the two; the gap reflects how much non-solver agent work is being done per problem.

**Wasted turns.** For each correct attempt, let $\iota_x^{(r)}$ denote the smallest index $i$ such that the $i$-th SMT call in the trace has

$$\rho_x^{(r,i)} \in \{\textsc{SAT}, \textsc{UNSAT}\},$$

i.e., the earliest call that emitted a real Z3 verdict. Then

$$\text{wasted turns} = \mathbb{E}_{x,r:\chi_x^{(r)}=1} \Big[\max\{0, S_x^{(r)} - \iota_x^{(r)} - 1\}\Big], \tag{12}$$

the mean number of *additional* SMT calls the agent issued *after* an already-correct call's verdict was returned. The metric captures retroactive exploration: it is invariant to how hard a problem was, only to whether the agent kept poking after the answer was implicitly already known. Lower is better; a value of zero indicates the agent commits the moment it has a real verdict.

**Unique skeletons.** For each problem we compute the SMT *skeleton*—the sequence of top-level command tokens of each call,

$$(\texttt{declare-const}, \texttt{assert}, \texttt{check-sat}, \ldots),$$

| Attribute | Definition |
|---|---|
| `any non trivial` | $\bigvee_i \neg\,\texttt{is trivial}(z_x^{(r,i)})$ |
| `any grounded` | $\bigvee_i \left(\texttt{grounding}(z_x^{(r,i)}, x) \geq \lambda_{\text{gnd}}\right)$ |
| `any real verdict` | $\bigvee_i \left(\texttt{sat result}_i \in \{\text{SAT}, \text{UNSAT}, \text{UNKNOWN}\}\right)$ |
| `any unsat` | $\bigvee_i \left(\texttt{sat result}_i = \text{UNSAT}\right)$ |
| `first pass valid` | $\texttt{sat result}_1 \in \{\text{SAT}, \text{UNSAT}, \text{UNKNOWN}\}$ |
| `max grounding` | $\max_i \texttt{grounding}(z_x^{(r,i)}, x)$ |

Table 6: Per-attempt structural attributes computed from the sequence of SMT calls $(z_x^{(r,1)}, \ldots, z_x^{(r,S_x^{(r)})})$ in trace $\tau_x^{(r)}$. The grounding threshold is $\lambda_{\text{gnd}} = 0.25$. `is trivial` is the predicate defined in §B.3.2.

and count the number of distinct skeletons across the $K$ attempts. The reported value is the mean of this count over $\mathcal{D}$; we also report the mean pairwise Jaccard distance between the skeleton sets of two attempts on the same problem. Both are diversity measures: large values mean the agent explores qualitatively different solution shapes on resampling; small values mean it produces structurally homogeneous attempts.

### B.3 SMT-Program Quality

Sample-based correctness metrics measure *what answer* the agent produced; SMT-program quality metrics measure *how it got there*. Each is a function of the SMT-LIB code and Z3 verdicts in the trace, defined first per-call, then per-attempt via the lift in Table 6, then aggregated over $\mathcal{D}$. The per-attempt lift is materially important: an agent that emits a strong SMT call *somewhere* in its trace should not be penalized for also issuing weak exploratory calls, so we use disjunctions, e.g., "any non-trivial," "any grounded," and "any real verdict," rather than averages over calls.

#### B.3.1 Grounding

The grounding score quantifies how much of the problem's actual content the agent transferred into the formal SMT representation. We extract *problem entities* $E(x)$ from the problem text $x$ via the regular expression `[A-Za-z][A-Za-z0-9 ]{1,}` after lower-casing, then remove a small fixed list of $\sim 50$ English stopwords plus generic task scaffolding tokens, e.g., `question`, `answer`, `true`, `false`, and `option`. For an SMT call $z$, the grounding score is the fraction of problem entities that appear as substrings in the lower-cased SMT body:

$$\text{gnd}(z, x) = \frac{1}{|E(x)|} \sum_{e \in E(x)} \mathbb{1}[e \in \text{lc}(z)]\,, \quad (13)$$

where $\text{lc}(z)$ is the lower-cased SMT body. We use the convention $\text{gnd} = 0$ when $|E(x)| = 0$, which is vanishingly rare in our datasets. Because the predicate "$e$ is a substring of $z$" is permissive—it counts both `declare-const e Int` and any occurrence of $e$ in a comment or sort name—grounding is a *liberal* measure of semantic transfer rather than a strict one, intentionally biased toward false positives so that the metric does not reward *intentional* obfuscation. The reported `mean grounding` is the per-attempt `max grounding` averaged over $\mathcal{D}$.

A grounding score of $0.05$ on a problem with twenty named entities indicates the SMT references one of them: this is the *decorative-SMT* regime in which the agent has constructed a formally valid program that does not actually express the problem. A score of $0.5$ indicates the SMT references roughly half the entities, sufficient for the program to be non-vacuous. The threshold $\lambda_{\text{gnd}} = 0.25$ used in Table 6 is empirically set: it cleanly separates inspection-verified decorative programs from inspection-verified content-loaded ones on a held-out development sample.

#### B.3.2 Triviality

An SMT call $z$ is *trivial* if any of the following hold:

1. $z$ is empty or whitespace.
2. $z$ contains no `declare-const`, `declare-fun`, or `define-fun`, and contains fewer than two `(assert ...)` forms.
3. Every `(assert ...)` form in $z$ matches a literal tautology or contradiction: `(assert true)`, `(assert (= n n))` with $n$ a numeric literal, `(assert (not true))`, or `(assert (not false))`.
4. $z$ has fewer than $40$ non-whitespace characters and at most one assertion.

The first three cases catch *semantically empty* SMT—programs that contain Z3 syntax but no problem content. The fourth case catches *stub* programs that initiate an SMT-LIB block but abandon it before encoding anything substantive. The `trivial smt rate` reported in tables is the fraction of attempts whose *entire* SMT call sequence is trivial:

$$\text{triv}_{\text{SMT}} = \frac{1}{N_{\text{SMT}}} \sum_{(x,r)} \mathbb{1}\left[\bigwedge_i \text{triv}(z_x^{(r,i)})\right], \quad (14)$$

where $N_{\mathrm{SMT}}$ is the number of attempts with at least one SMT call. A non-zero trivial-SMT rate is diagnostic of *performative SMT*: the agent emits Z3 calls to satisfy the tool-calling prior of its training distribution without actually using the solver to reason.

#### B.3.3 First-Pass Z3 Validity

The fraction of attempts, over those with at least one SMT call, whose very first invocation produced a real Z3 verdict:

$$\begin{aligned} \mathcal{V}_{\mathrm{Z3}} &= \{\textsc{sat}, \textsc{unsat}, \textsc{unknown}\}, \\ \mathcal{A}_{\mathrm{rv}} &= \{(x,r) : \rho_x^{(r,1)} \in \mathcal{V}_{\mathrm{Z3}}\}, \\ \mathcal{A}_{\mathrm{SMT}} &= \{(x,r) : S_x^{(r)} \geq 1\}, \\ \mathrm{fpv}_{\mathrm{Z3}} &= \frac{|\mathcal{A}_{\mathrm{rv}}|}{|\mathcal{A}_{\mathrm{SMT}}|}. \end{aligned} \tag{15}$$

This isolates whether the agent's very first SMT-LIB emission is syntactically well-formed under SMT-LIB v2 and well-typed under the declared theory. A delta on this metric during post-training reflects improvements in surface-level SMT syntax—fewer malformed parentheses, fewer undeclared symbols, fewer theory mismatches—rather than improvements in reasoning depth.

#### B.3.4 Verified-Correct Rate and its Unsat Restriction

We now define the strictest accuracy measure used in this work. An attempt is *verified-correct* if the conjunction

$$\begin{aligned} \mathrm{VC}_x^{(r)} := \chi_x^{(r)} \\ &\wedge \texttt{any real verdict}_x^{(r)} \\ &\wedge \texttt{any grounded}_x^{(r)} \\ &\wedge \texttt{any non trivial}_x^{(r)} \end{aligned} \tag{16}$$

holds, and verified-correct under UNSAT if it further satisfies $\texttt{any unsat}_x^{(r)}$. The aggregate rates are

$$\begin{aligned} \mathrm{vcr} &= \frac{1}{MK} \sum_{x,r} \mathrm{VC}_x^{(r)}, \\ \mathrm{vcr}_{\textsc{unsat}} &= \frac{1}{MK} \sum_{x,r} \left( \mathrm{VC}_x^{(r)} \wedge \texttt{any unsat}_x^{(r)} \right). \end{aligned} \tag{17}$$

vcr filters out four distinct failure modes that can all produce a passing $\chi_x^{(r)} = 1$:

- *No real verdict*: the agent answered correctly without ever obtaining a SAT/UNSAT/UNKNOWN verdict from Z3, i.e., all calls errored or produced no verdict.
- *Decorative SMT*: the agent obtained a verdict but the program had grounding below $\lambda_{\mathrm{gnd}}$, i.e., the SMT did not reference the problem.
- *Trivial SMT*: every call was structurally trivial.
- *The conjunction held but on a vacuous verdict*: e.g., a SAT verdict on the empty constraint set.

Of these the most dangerous is the no-real-verdict case: it indicates a model that answers correctly via natural-language reasoning while only *performing* SMT use. The post-training vcr delta therefore quantifies the shift toward genuine—rather than performative—solver use.

$\mathrm{vcr}_{\textsc{unsat}}$ goes further by requiring that at least one SMT call in the trace returned UNSAT. By the discussion in §A.1, this is the *refutational-proof* restriction: the agent must have attempted to derive a contradiction from the negation of its candidate answer, and the solver must have confirmed it. The pair $(\mathrm{vcr}, \mathrm{vcr}_{\textsc{unsat}})$ therefore separates correctness into three regimes: (i) correct but performative; (ii) verified-correct via SAT witness, i.e., model-finding; and (iii) verified-correct via UNSAT, i.e., refutational proof.

#### B.3.5 Z3 Outcome Distribution

For diagnostic and downstream analysis purposes, we additionally report the distribution of Z3 outcomes across every SMT call in the sweep:

$$\begin{aligned} \omega^{\mathrm{Z3}} = \big(&\omega_{\textsc{sat}}, \omega_{\textsc{unsat}}, \omega_{\textsc{unknown}}, \\ &\omega_{\textsc{error}}, \omega_{\textsc{no-verdict}}\big), \end{aligned}$$

with $\omega_u^{\mathrm{Z3}}$ the fraction of all SMT calls whose `sat result` took value $u$. We treat NO VERDICT, where the call succeeded but no `sat result` token appeared in stdout, as distinct from ERROR, where the call raised an exception or returned a non-zero exit.

## C Comparison to Open-Weight Baselines

We additionally compare the DPO-tuned Sequent Prover checkpoints against external open-weight baselines evaluated with the same SMT-scratchpad harness. The external baselines are Gemma-4B, GPT-OSS-20B, and GPT-OSS-120B. All deltas are Qwen-DPO minus the external model. Thus, positive values indicate that the Qwen-DPO checkpoint is higher, except for lower-is-better diagnostic rates,

such as the trivial-SMT rate, where the raw signed delta is still shown.

**Aggregate correctness.** Table 7 reports macro-averaged deltas over the shared canonical datasets. The 4B comparison shows a large separation from Gemma-4B: Qwen-4B-DPO improves avg@$N$ by $+0.647$, pass@4 by $+0.524$, and pass$^4$ by $+0.640$. The GPT-OSS-20B comparisons show the same pattern at a larger external baseline: Qwen-27B-DPO improves avg@$N$ by $+0.326$ and pass$^4$ by $+0.648$, while Qwen-9B-DPO improves avg@$N$ by $+0.287$ and pass$^4$ by $+0.567$. Against GPT-OSS-120B, the comparison is closer and therefore more informative: Qwen-27B-DPO remains positive on all correctness aggregates, including $+0.027$ avg@$N$, $+0.021$ pass@4, and $+0.036$ pass$^4$; Qwen-9B-DPO is close on coverage, with $+0.006$ pass@4 and $+0.002$ AUC@4, while GPT-OSS-120B has the advantage on avg@$N$ and pass$^4$.

**Verifier-use diagnostics.** Table 8 compares solver-facing behavior. The largest gains are in the verified-correct rate, UNSAT-restricted verified-correct rate, and grounding. Qwen-4B-DPO improves verified-correct rate by $+0.615$ relative to Gemma-4B. Against GPT-OSS-20B, both Qwen-27B-DPO and Qwen-9B-DPO improve the verified-correct rate by $+0.487$. The GPT-OSS-120B comparison is particularly useful because it separates answer-level strength from verifier-use quality: Qwen-27B-DPO improves verified-correct rate by $+0.341$ and grounding by $+0.274$, while Qwen-9B-DPO improves verified-correct rate by $+0.349$ and grounding by $+0.299$. Trivial-SMT rate decreases in all five comparisons, indicating that the DPO checkpoints emit fewer structurally empty or decorative SMT programs.

**Per-dataset breadth.** Table 9 reports polarity-adjusted win/loss/tie counts across shared datasets. The Qwen-DPO checkpoints dominate the 4B and 20B external baselines across the main sample-based metrics: pass$^4$ and maj@4 win on every shared dataset in the Gemma-4B and GPT-OSS-20B comparisons. The GPT-OSS-120B comparison is more balanced. Qwen-27B-DPO wins avg@$N$ on 28 of 35 datasets and pass$^4$ on 24 of 35 datasets, with five pass$^4$ ties. Qwen-9B-DPO remains competitive on coverage, winning pass@4 on 22 of 33 datasets, while GPT-OSS-120B has the stronger all-sample consistency profile.

**Category-level trends.** Table 10 groups the avg@$N$ deltas by benchmark category. Qwen-DPO is substantially ahead of Gemma-4B and GPT-OSS-20B in every category. Against GPT-OSS-120B, Qwen-27B-DPO is strongest on AGIEval, NLI/MCQ, and hard-reasoning categories, and remains close on arithmetic and free-form QA. The 9B-DPO comparison to GPT-OSS-120B is near parity for AGIEval, arithmetic, hard reasoning, and NLI/MCQ, with larger gaps on spatial and free-form QA tasks.

Overall, these external comparisons support two conclusions. First, Sequent Prover DPO checkpoints are substantially stronger than the same-scale or moderately larger open-weight baselines under the SMT-scratchpad evaluation, with especially large advantages in pass$^4$ and verifier-grounded correctness. Second, the comparison with GPT-OSS-120B shows that the 27B DPO checkpoint is competitive with a much larger open-weight model on answer-level metrics, while retaining a clear advantage in the use of executable, grounded solvers.

## D Qualitative Improvements

The quantitative improvements we observe in our metrics, such as verified correct rate with unsat (i.e. appropriate proof by contradiction) co-relate to a strong qualitative improvement in semantic faithfulness when inspected manually. Throughout the harness and metric development, as well as during intermediate checkpoints, we took care to ensure there is no reward hacking and quantitative improvements actually manifest via qualitative improvements in formal artifact quality. In this section, we attempt to elucidate a few learnt capabilities in our trained models.

### D.1 Capability 1: SMT-LIB language fluency

The most basic gap is that the 4B Qwen Base model emits malformed SMT-LIB on the first attempt and rarely recovers, whereas Sequent-Prover-4B recovers in one turn. We illustrate with ProntoQA-95 (gold = B): a chain of universal implications resolves to "Stella is not feisty."

**Qwen-4B** The first tool call is itself malformed at the envelope level: the model wraps its payload as `{"arguments":"{\"smt_code\":\"...\"}"}`, which the harness's JSON-strict tool parser cannot decode, so the next `tool` message is empty. Inside the inner payload the SMT-LIB is also incorrect:

| Comparison | Shared | avg@$N$ | pass@4 | pass$^4$ | maj@4 | AUC@4 | Cons. |
|---|---|---|---|---|---|---|---|
| Qwen-4B-DPO − Gemma-4B | 33 | +0.647 | +0.524 | +0.640 | +0.720 | +0.591 | +0.568 |
| Qwen-27B-DPO − GPT-OSS-20B | 35 | +0.326 | +0.067 | +0.648 | +0.408 | +0.151 | +0.699 |
| Qwen-9B-DPO − GPT-OSS-20B | 33 | +0.287 | +0.053 | +0.567 | +0.372 | +0.130 | +0.625 |
| Qwen-27B-DPO − GPT-OSS-120B | 35 | +0.027 | +0.021 | +0.036 | +0.028 | +0.023 | +0.017 |
| Qwen-9B-DPO − GPT-OSS-120B | 33 | -0.010 | +0.006 | -0.039 | -0.008 | +0.002 | -0.054 |

Table 7: Macro-averaged correctness deltas against external open-weight baselines. Deltas are computed over shared canonical datasets and are reported as Qwen-DPO minus the external model. Cons. is pass$^4$/pass@4.

| Comparison | Sound. | Recov. | VCR | VCR-U | FP-Z3 | Triv. SMT | Ground. |
|---|---|---|---|---|---|---|---|
| Qwen-4B-DPO − Gemma-4B | +0.444 | +0.490 | +0.615 | +0.399 | +0.164 | -0.016 | +0.399 |
| Qwen-27B-DPO − GPT-OSS-20B | +0.206 | +0.181 | +0.487 | +0.345 | +0.034 | -0.032 | +0.329 |
| Qwen-9B-DPO − GPT-OSS-20B | +0.172 | +0.152 | +0.487 | +0.321 | +0.023 | -0.031 | +0.353 |
| Qwen-27B-DPO − GPT-OSS-120B | -0.010 | -0.020 | +0.341 | +0.196 | +0.004 | -0.011 | +0.274 |
| Qwen-9B-DPO − GPT-OSS-120B | -0.036 | -0.032 | +0.349 | +0.179 | -0.008 | -0.011 | +0.299 |

Table 8: Macro-averaged verifier-use deltas. Sound. is $P(\text{correct} \mid \text{finish})$; Recov. is correctness after an initial SMT error; VCR is verified-correct rate; VCR-U additionally requires an UNSAT verdict; FP-Z3 is first-pass Z3 validity; Triv. SMT is trivial-SMT rate, where lower is better; Ground. is mean SMT grounding.

```
(set-logic QF_LIA)
(declare-const IsFeisty Pred)
(declare-const IsVumpus Pred)
(declare-const Stella Bool)
(assert (= (IsVumpus Stella) True))
(assert (forall ((x Pred))
  (=> (= (IsVumpus x) True)
      (= (IsDumpus x) True))))
```

Several things are wrong: `Pred` is not a sort, `IsVumpus` is declared as a constant but applied as a function, `Stella` is a `Bool` but used as an individual, and `True` appears as a value rather than the SMT-LIB literal `true`. Over the next eight turns the model loses track of the original problem and cycles through `(set-logic QF_LIA)`, `(set-logic QF_BV)`, `(set-logic QF_NIA)`, and `(set-logic QF_ARITH)`, the last of which is not a recognized logic name. It exhausts its 10-turn budget without emitting `finish`.

**SequentProver-4B.** The first tool call is a first-order encoding with a sort for animals and one predicate per type:

```
(declare-sort Animal)
(declare-fun vumpus (Animal) Bool)
(declare-fun dumpus (Animal) Bool)
(declare-fun rompus (Animal) Bool)
;; ... (one declare-fun per category) ...
(declare-const Stella Animal)

(assert (forall ((x Animal))
  (=> (vumpus x) (dumpus x))))
;; ... the full implication chain ...
(assert (forall ((x Animal))
  (=> (zumpus x) (not (feisty x)))))
(assert (vumpus Stella))
(assert (feisty Stella)) ; proof by
    contradiction
(check-sat) ; --> unsat
```

The first pass surfaces two ordering errors (`yumpus` and `floral` are used before they are declared). The Sequent-Prover-4B model reads the Z3 error, hoists the missing `declare-funs` to the top, re-runs, and finishes in four turns. The qualitative point is not that the base model made syntactic mistakes, it is that, having seen the error, it could not localize the fix and degenerated into unrelated probes.

### D.2 Capability 2: First-order over propositional encoding

When a problem quantifies over individuals, a propositional encoding (one boolean per ground proposition) often forces the model to fill in unstated relations by hand. The base model does this routinely; Sequent-Prover does not.

**Example: FOLIO-1361 (gold = Uncertain).** Premises describe animals near a campus and ask whether "Rockie is a turtle." The critical step is to notice that the premises constrain Rockie to be a rabbit *or* a squirrel but say nothing about whether rabbit and turtle are disjoint categories.

**Qwen-4B** Declares one `Bool` per (Rockie, property) pair, with no animal sort. Lacking a way to express "the only animals spotted are rabbits and squirrels" as a universal, it instead injects two disjointness axioms that are nowhere in the premises:

| Comparison | avg@$N$ | pass@4 | pass$^4$ | maj@4 | FP-Z3 | Sound. | Recov. |
|---|---|---|---|---|---|---|---|
| Qwen-4B-DPO — Gemma-4B | 32/0/1 | 32/1/0 | 33/0/0 | 33/0/0 | 33/0/0 | 27/1/5 | 27/1/5 |
| Qwen-27B-DPO — GPT-OSS-20B | 35/0/0 | 33/1/1 | 35/0/0 | 35/0/0 | 35/0/0 | 29/1/5 | 27/2/6 |
| Qwen-9B-DPO — GPT-OSS-20B | 33/0/0 | 31/1/1 | 33/0/0 | 33/0/0 | 29/3/1 | 26/2/5 | 23/5/5 |
| Qwen-27B-DPO — GPT-OSS-120B | 28/7/0 | 23/9/3 | 24/6/5 | 25/9/1 | 22/9/4 | 16/14/5 | 14/15/6 |
| Qwen-9B-DPO — GPT-OSS-120B | 16/17/0 | 22/8/3 | 6/26/1 | 15/16/2 | 3/29/1 | 8/20/5 | 6/21/6 |

Table 9: Per-dataset win/loss/tie counts. Each entry is W/L/T after metric polarity is applied. Higher is better for all displayed metrics.

| Category | 4B−Gemma | 27B−OSS20 | 9B−OSS20 | 27B−OSS120 | 9B−OSS120 |
|---|---|---|---|---|---|
| AGIEval | +0.795 | +0.376 | +0.331 | +0.069 | +0.024 |
| Arithmetic | +0.686 | +0.304 | +0.283 | +0.021 | +0.001 |
| Free-form QA | +0.611 | +0.363 | +0.312 | +0.002 | -0.049 |
| Hard reasoning | +0.659 | +0.346 | +0.317 | +0.035 | -0.003 |
| NLI/MCQ | +0.647 | +0.346 | +0.301 | +0.044 | +0.003 |
| Other | +0.816 | +0.307 | +0.285 | -0.008 | -0.022 |
| Spatial | +0.341 | +0.143 | +0.110 | -0.037 | -0.071 |

Table 10: Mean avg@$N$ delta by dataset category. Values are macro-averages over shared canonical datasets in each category.

```
; Assumption: Rabbits and turtles are mutually
    exclusive
(assert (=> RockieRabbit (not RockieTurtle)))

; Assumption: Squirrels and turtles are mutually
     exclusive
(assert (=> RockieSquirrel (not RockieTurtle)))
```

With those injected axioms, the negation of the conclusion becomes `unsat` and the base model answers "False", but the false answer is caused by the hallucinated premise, not by the actual problem.

**SequentProver-4B.** Uses a sort `Animal` and universally quantified predicates over animals; the conditions in the prompt translate mechanically. We discuss Sequent-Prover's full reasoning trajectory under Capability 6 below, because it turns on the same diagnostic move used in another problem.

### D.3 Capability 3: Bidirectional satisfiability for ternary verdicts

When the verdict space is ternary (True / False / Uncertain) or four-valued (entailment / contradiction / neutral / self-contradiction), a single satisfiability query is not enough. The textbook idiom is two queries: assert the conclusion, then assert its negation. The base model asks one and stops; Sequent-Prover-4B asks both.

**Example: FOLIO-476 (gold = False).** "Tom will rent at the Olive Garden," under premises constraining the deposit to be at least the monthly rent ($2000) and renting to require deposit ≤$1500.

**Qwen-4B** Spends three turns repairing surface syntax (`Boolean`→`Bool`; `->` to `implies`; `(= oliveWillRent False)`→`(not oliveWillRent)`). Its single satisfiability query is on the negation of the conclusion, which is `sat`, so it concludes "the conclusion does not necessarily follow" and answers **Uncertain**. It never asks the dual question.

**SequentProver-4B.** Runs both queries:

```
;; First: can the conclusion FAIL under the
    premises?
(assert (not (and allowed_to_move_in_with_fluffy
                  (<= deposit_at_olive 1500.0))))
(check-sat) ; sat ==> not forced True

;; Second: can the conclusion HOLD under the
    premises?
(assert (and allowed_to_move_in_with_fluffy
             (<= deposit_at_olive 1500.0)))
(check-sat) ; unsat ==> forced False
```

(sat, unsat) is the signature of a forced-false conclusion. Sequent-Prover-4B finishes with **False** and the explanation reproduces the informal argument: "$2000 > $1500, so the deposit condition cannot be satisfied at the Olive Garden." What changed is not the SMT vocabulary, both models know `check-sat`, but the model's choice of *which experiment to run next*.

### D.4 Capability 4: Declarative encoding via datatypes

For problems with explicit finite enumerations (LSAT analytical reasoning, ZebraLogic), the canonical idiom is `declare-datatypes` for each finite domain and a function from one to another.

**Example: AR-LSAT-82 (gold = E).** Seven sales reps assigned to three zones, with same-zone pair-

ings and an XOR-style either-but-not-both rule. "If more reps work in Zone 1 than in Zone 3, which one of the following *could* be true?"

**Qwen-4B** Encodes zones as unbounded integers and emits a wall of errors over all ten turns. The actual bugs we observe across calls are: bare `<= 1 zone_Kim 3` expressions at top level (outside any `assert`); ad-hoc `[zone_Kim]` bracket syntax; `(select zone_Kim 1)` on a `Bool` as if to extract an indicator; the conjunction `(and (= zone_Kim 1) (= zone_Kim 2) (= zone_Kim 3))` used where the intent is disjunction; `(if ...)` instead of `(ite ...)`; and a multi-variable declare, `(declare-const zK zM zP zQ zS zT zU Int)`, which is not legal SMT-LIB. None of these get repaired; each turn re-issues a near-copy of the previous one. The model never reaches the option-testing stage.

**SequentProver-4B.** First turn uses two algebraic datatypes and an abstract assignment function:

```
(declare-datatypes ()
  ((Rep Kim Mahr Parra Quinn Stuckey Tiao Udall))
      )
(declare-datatypes ()
  ((Zone Zone1 Zone2 Zone3)))
(declare-fun zone (Rep) Zone)

(assert (xor (= (zone Parra) Zone1)
             (= (zone Tiao) Zone1))) ; rule 1
(assert (xor (= (zone Tiao) Zone2)
             (= (zone Udall) Zone2))) ; rule 2
(assert (= (zone Parra) (zone Quinn))) ; rule 3
(assert (= (zone Stuckey) (zone Udall))) ; rule
    4

(define-fun count1 () Int
  (+ (ite (= (zone Kim) Zone1) 1 0)
     (ite (= (zone Mahr) Zone1) 1 0)
     ;; ... ite per rep ...
     ))
```

The first pass writes `(not = =)` (the constraint is "exactly one of X, Y") and gets a Z3 arity error; the next turn replaces the offending subtree with `xor`. This is the same local-edit recovery pattern as in Capability 1.

### D.5 Capability 5: Push/pop for multi-option queries

LSAT analytical-reasoning items ask "which of the following could / must be true?" This is a sequence of independent satisfiability tests on a shared base theory, the SMT idiom is push/pop.

**Example: AR-LSAT-60 (gold = E).** Four riders and four bicycles across two days, with same-day distinctness and cross-day pairing. "If Theresa tests G on day 2, which one of the following *must* be true?"

**Qwen-4B** Across all ten turns the same family of errors recurs. Seven of the ten calls contain `(assert (= T1 H) or (= T2 H))`, with `or` as a top-level token outside the `assert`, which Z3 reports as "invalid assert command, ) expected." The base model treats the error as a request to reformat the body (one call collapses the entire program onto a single line; another swaps in unbounded integer ranges), but never rewrites the offending node as the prefix expression `(or (= T1 H) (= T2 H))`. It never reaches the option-testing stage, and `extraction_method = none`.

**SequentProver-4B.** A single first-turn body emits five push/pop blocks, one per answer choice. Each block asserts the *negation* of the choice: the canonical pattern for "must be true" (if the negation is unsat, the choice must hold). `(echo ...)` labels the output so the five results are self-describing.

```
(declare-datatypes () ((Rider R Seamus T Yuki)))
(declare-datatypes () ((Bike F G H J)))
(declare-fun d1_rider (Bike) Rider)
(declare-fun d2_rider (Bike) Rider)

(assert (distinct (d1_rider F) (d1_rider G)
                  (d1_rider H) (d1_rider J)))
;; ... base theory ...
;; Yuki's day-1 bike is Seamus' day-2 bike:
(assert (exists ((b Bike))
  (and (= (d1_rider b) Yuki)
       (= (d2_rider b) Seamus))))

(echo "Testing (A): Reynaldo tests H, day 1")
(push) (assert (not (= (d1_rider H) R)))
       (check-sat) (pop)
;; ... (B), (C), (D) similarly ...
(echo "Testing (E): Yuki tests H, day 2")
(push) (assert (not (= (d2_rider H) Yuki)))
       (check-sat) (pop)
```

The single Z3 reply ends with "Testing (E) ... unsat," the unique negation that is unsat, so (E) is the choice that must hold. Sequent-Prover finishes in three iterations, having issued one comprehensive query instead of five separate ones.

Three sub-skills are bundled into this first turn: (i) scaffolding with `(echo ...)` so the Z3 output is self-describing; (ii) knowing *which* direction to negate for "must" vs. "could" questions; (iii) knowing that push/pop make the five queries independent. Note also the encoding choice: inverting the natural direction to `d1_rider : Bike → Rider` lets the day-1 distinctness collapse to a single `distinct` call instead of six pairwise inequali-

ties, and lets the cross-day pairing be expressed as a one-variable existential.

### D.6 Capability 6: Diagnosing inconsistent premise sets

The most subtle capability we observe is meta-cognitive: noticing that *both* directions of a hypothesis are unsatisfiable, and inferring from that signature that the premise set itself is inconsistent. This move appears in two superficially unrelated transcripts.

**Example A: Self-Contradiction** We used an example where there were twelve atomic facts and twelve quantified rules over seven named people. The base model spends ten turns rewriting an 80-line SMT body, hitting the same family of errors each time (missing `Harold_brainy` declaration, mis-parenthesized rules 4 and 9). At turn 22 the harness sends a follow-up nudge, "Z3 returned a clean UNSAT result, so now emit the finish tool call", but the base ignores it and issues yet another `z3_solve` on turn 23.

Sequent-Prover-4B arrives at the verdict in three steps. First, it tests both directions of the hypothesis:

```
;; (a) Is the hypothesis decidable?
(push) (assert R_huge)
      (check-sat) (pop) ; unsat
(push) (assert (not R_huge))
      (check-sat) (pop) ; unsat
```

Both `unsat` is the diagnostic signature of an inconsistent premise set. Sequent-Prover-4B confirms by querying the premises alone:

```
;; (b) Are the premises consistent?
;; ... (only premises asserted) ...
(check-sat) ; unsat
```

and finishes with `self_contradiction`. The explanation it produces names the diagnostic explicitly: "Z3 returns UNSAT for the premises alone, and both `R_huge` and `(not R_huge)` are UNSAT when added to the premises. This means the premises contain an inherent contradiction."

**Example B: FOLIO-1361, revisited.** Recall that the base model on this problem (§D.2) injected ad-hoc disjointness between rabbits, squirrels, and turtles and got an unsound "False." Sequent-Prover's trajectory uses the same diagnostic move as Example A:

1. No injected axioms; both directions of `(turtle Rockie)` return `sat`. (Sequent-Prover-4B does not yet realize this *is* the "Uncertain" answer.)
2. Adds disjointness axioms; the negation goes `unsat`, suggesting the conclusion is forced True.
3. Runs both push/pop queries on the disjointness-augmented theory: *both* are `unsat`.
4. Recognizes both-`unsat` as a contradiction signature, checks the augmented premises alone, gets `unsat`.
5. Concludes the disjointness axioms are not licensed; removes them.
6. Re-runs both directions: `sat` and `sat`.
7. Finishes with `Uncertain` and a verbal argument: "The premises do not state that rabbits and turtles are disjoint categories."

The same meta-strategy, both UNSAT ⇒ suspect your own axioms, recovers the correct answer on a problem where the base model is led astray by the very axioms it injected. The strategy is not encoded by any single SMT primitive; it is a policy over multi-step solver interactions.

### D.7 Capability 7: Error text as actionable feedback

The recovery patterns we observe in Sequent-Prover-4B follow a recurring structure: read the Z3 error string, locate the named node in the previous body, and patch it without disturbing the rest of the program. The base model either re-emits a near-identical body or escalates the change in unhelpful directions (switching logics, collapsing onto one line). Three mini-patterns:

**ProntoQA-95.**
Error: "unknown constant `yumpus (Animal)`." Fix: move `(declare-fun yumpus (Animal) Bool)` above its first use. Nothing else changes.

**FOLIO-476.**
Error: "unknown sort `Boolean`." Sequent-Prover-4B patches each occurrence to `Bool` in one turn. The base model takes two extra turns to also re-discover that `->` is not a keyword (use `=>` or `implies`) and that the constant is lower-case `false`.

**AR-LSAT-82.**
Error: "Wrong number of arguments (2) passed to function `not`." Fix: replace the entire `(not (= ...) (= ...))` subtree with `(xor (= ...) (= ...))`, which is both shorter and correct.

The base-model failure mode is not lexical: it knows the tokens. It does not, however, treat the SMT body as code with a parse tree, a symbol table, and named scopes; it treats it as an incantation that either works or doesn't.