VERITAS: Verifier-Guided Proof Search for Zero-Shot Formal Theorem Proving

Manish Acharya [1] **Zhenyu Liao** [2] **Yueke Zhang** [1] **Kevin Leach** [1] **Yu Huang** [1] **Yifan Zhang** [1]

Abstract

LLM-based formal provers often collapse rich verifier signals (syntax errors, type mismatches, partial goal progress) into a binary pass/fail bit. We present **VERITAS**, a zero-shot framework that routes every verifier signal back into proof search through a two-phase protocol: Best-of-N sampling first, then a critic-guided MCTS pass that ingests Phase 1 failures as explicit negative examples. The protocol preserves every theorem solved by its own Phase 1 sweep, so Phase 2's additional solves are attributable to feedback-driven exploration. VERITAS reaches **40.6%** on miniF2F (vs. an independently run Best-of-5 at 36.9%, Portfolio 26.2%) and **7.3%** on **VERITAS-CombiBench**, a 55-theorem combinatorics benchmark we release on which Best-of-5 (1.8%) falls *below* Portfolio (3.6%), exposing that unguided sampling hurts when correct lemma names must be recovered iteratively from verifier feedback. Artifacts are available on GitHub.

1. Introduction

Formal theorem proving, the construction of machine-checkable proofs in interactive proof assistants such as Lean, Coq, and Isabelle, has become a flagship benchmark for evaluating AI systems on multi-step formal thought, because success admits a crisp definition: a candidate proof is either accepted by the verifier or it is not. This property has driven a steady stream of progress, pushing solve rates on miniF2F (Zheng et al., 2022) from below 30% to above 80% (Polu & Sutskever, 2020; Yang et al., 2023; Wu et al., 2024b; Ren et al., 2025). Two complementary families drive this progress: language models that *generate* candidate tactics (Polu & Sutskever, 2020; Han et al., 2022; Yang et al., 2023; Azerbayev et al., 2024), and tree-search methods that *explore* the combinatorial space of partial proofs (Kocsis & Szepesvári, 2006; Silver et al., 2018; Lample et al., 2022; Google DeepMind, 2024). Most modern systems combine the two: an LLM proposes, search disposes.

What is striking about this combination is how thinly the two components *communicate*. In a typical pipeline, the LM emits a batch of candidate tactics, the verifier checks each one, and the search algorithm updates value estimates from the resulting pass-or-fail outcomes. Anything richer that the verifier produces along the way (a parse error pointing at a malformed token, a type mismatch naming the offending hypothesis, a partial goal that has been simplified but not closed) is summarised into a scalar reward or thrown away entirely. Yet these intermediate signals are exactly the artefacts a human prover attends to: collapsing them into a single bit forces the search to relearn what the verifier has already told it explicitly. The cost is most visible on proofs that require a chain of *dependent* tactics, where each step's correctness hinges on the previous step's effect on the goal. Recent rich-feedback methods in RL (Hübotter et al., 2026) make the same observation in the post-training regime; we explore the dual at inference time.

We present **VERITAS**, a zero-shot framework built around a single design commitment: *the verifier's structured feedback should reach every generation decision*. We factor proof search into four specialised agents sharing a unified proof state. A *Strategist* chooses a high-level approach (direct computation, induction, rewriting, contradiction) before any tactics are emitted, narrowing the search to a strategy-relevant subspace. A *Tactician* (Claude Sonnet) proposes tactic candidates conditioned on the current goal, retrieved Mathlib premises, and crucially the explicit list of tactics that have already failed earlier in the search, with their Lean error messages attached as negative examples. A *Critic* (Claude Haiku) scores partial proof states, replacing MCTS's random rollouts with a value estimate that integrates Lean's four-way feedback signal. A *Retriever* grounds generation in Mathlib by surfacing relevant lemmas. These agents are coordinated by a deliberately conservative two-phase protocol: Phase 1 is identical to a Best-of-N sweep, and Critic-guided MCTS in Phase 2 runs only on

[1]Department of Computer Science, Vanderbilt University, Nashville, TN, USA [2]Amazon, Seattle, WA, USA. Correspondence to: Yifan Zhang <yifan.zhang.2@vanderbilt.edu>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

theorems Phase 1 failed. This yields a *monotonicity guarantee*: VERITAS preserves every theorem solved by its own Phase 1 attempts. The reported Best-of-5 baseline is an independent stochastic run, so its overlap with VERITAS need not be a subset; within a VERITAS run, however, every Phase 2-only solve is attributable to MCTS with verifier feedback. A batched Lean validation trick further cuts per-expansion verification from $O(K)$ to $O(1)$ proof-checker invocations.

Across two Lean 4 benchmarks the experiments give a consistent picture: tightly coupling the verifier with generation matters more than scaling either component alone. On miniF2F, VERITAS reaches 40.6% (99/244), above Best-of-5 Claude (36.9%) and a handcrafted Portfolio (26.2%); a phase-wise decomposition identifies 11 theorems no flat-sampling budget can reach. The picture sharpens on **VERITAS-CombiBench**: there, Best-of-5 (1.8%) falls *below* the Portfolio (3.6%) because unguided sampling hallucinates Mathlib lemma names Lean rejects on sight, while VERITAS reaches 7.3% by correcting those names iteratively from Lean's errors. We read these results as evidence for a general design principle: when a deterministic verifier emits structured intermediate signals, folding them back into generation amplifies the LLM substantially more than additional sampling does.

Contributions. (i) A zero-shot four-agent framework that routes Lean's structured feedback (syntax/type/progress/-completion) into every generation decision; (ii) a two-phase Best-of-N $\rightarrow$ Critic-MCTS protocol with an internal monotonicity guarantee that isolates the search contribution; (iii) a batched Lean validation trick that reduces verification cost by $10\times$; (iv) **VERITAS-CombiBench**, a benchmark of 55 hard, author-verified Lean 4 combinatorics theorems we release; and (v) a phase-wise attribution showing 11 miniF2F theorems uniquely solvable by MCTS, plus a failure-mode taxonomy that locates the next frontier for verifier-in-the-loop search.

2. The VERITAS Framework

VERITAS coordinates four specialized agents (Strategist, Tactician, Critic, and Retriever) through a shared proof state and a Critic-guided MCTS loop, with all agents observing the same structured verification signals from Lean. Figure 1 shows the full pipeline and Algorithm 1 gives the protocol in pseudo-code. The division of labor separates strategic choice, tactic synthesis, state valuation, and premise retrieval while keeping all roles grounded in the same verifier state.

Proof state and verifier signals. All agents operate on a shared `ProofState` $s = (\mathcal{T}, g, H, \tau_{1:t}, \sigma_A, \sigma_B, \sigma_C, \sigma_D)$, where $\mathcal{T}$ is the theorem statement, g the current goal, H the hypothesis set, $\tau_{1:t}$ the tactic history, and the four σ values are Lean's structured signals: σ_A syntax validity, σ_B type correctness, σ_C goal progress (e.g. subgoal reduction), and σ_D proof completion (no remaining goals or `sorry`). Every agent's output conditions the next agent's input within an iteration.

Agents. The *Strategist* maps a proof state to one of six high-level strategies (`direct`, `induction`, `contradiction`, `cases`, `rewrite`, `apply_lemma`) together with a depth estimate, restricting subsequent search to a strategy-relevant subspace. The *Tactician* (Claude Sonnet 4.6, temperature 0.8) generates K tactic candidates conditioned on the strategy, retrieved premises, Critic guidance, and crucially the failure set $\mathcal{F}$ of tactics rejected earlier (injected into the system prompt as negative examples). The *Critic* (Claude Haiku 4.5, temperature 0.3) returns a value $V(s) \in [0, 1]$, suggested next tactics, and a pruning recommendation, providing low-latency guidance for selection. The *Retriever* surfaces the top k=5 Mathlib premises by weighted keyword overlap over goal tokens, lemma names, and statement signatures (matching type names such as `Nat`/`Finset`, operators $+, *, \leq$, and connectives $\wedge, \vee$); results are appended to the Tactician's prompt and passed to the Critic as context.

2.1. Critic-Guided MCTS

Selection. We extend standard UCB with Critic value and strategy alignment, scoring an action a at state s as $Q(s,a) + c\sqrt{\ln N(s)/N(s,a)} + w_v V_{\text{critic}}(s') + w_s \,\text{align}(a, \pi)$, where $\text{align}(a, \pi) \in \{0, 1\}$ equals 1 iff the leading token of a belongs to the strategy-specific tactic vocabulary $\mathcal{V}_\pi$ (e.g., $\mathcal{V}_{\text{induction}} = \{\texttt{induction}, \texttt{cases}, \texttt{rcases}\}$), biasing selection toward strategy-aligned subtrees. *Expansion* runs the four agents in sequence (Strategist $\rightarrow$ Retriever $\rightarrow$ Critic $\rightarrow$ Tactician), validates all K generated candidates in a single batched Lean call (Section 2.3), and creates child nodes for the syntactically valid ones. *Simulation* replaces random rollouts with the Critic's value augmented by the structured signals, $V(s) = V_{\text{critic}}(s) + \alpha_1\sigma_A + \alpha_2\sigma_B + \alpha_3\sigma_C + \beta/(1+N(s))$, with α_1=α_2=0.1 and α_3=0.2 assigning increasing weight to increasingly informative signals (goal progress is the strongest intermediate signal short of completion) and β=0.1 providing an exploration bonus that decays with visit count. *Backpropagation* carries the structured A–B–C–D signal upward rather than collapsing to binary win/loss; this preserves credit for partial-progress states (σ_C) that would otherwise be pruned: the critical case for multi-step theorems.

2.2. Two-Phase Design and Monotonicity Guarantee

Phase 1 (Best-of-N sweep) generates N tactics in one Claude API call and validates them in one batched Lean

Algorithm 1 VERITAS Two-Phase Search

Require: Theorems $\mathcal{T}$; budgets N (Best-of-N), I (MCTS), K (candidates per node)

$S_1, S_2 \leftarrow \emptyset$ *// Phase 1: Best-of-N*
for $T \in \mathcal{T}$ **do**
  $\{\tau_i\}_1^N \leftarrow$ Claude(T); (ok, err) $\leftarrow$ LEANBATCH($T, \{\tau_i\}$)
  if $\exists i$: ok[i] **then**
    $S_1 \leftarrow S_1 \cup \{T\}$
  else
    $\mathcal{F}_T \leftarrow \{(\tau_i, \text{err}[i])\}_{i=1}^N$
  end if
end for
// Phase 2: Critic-guided MCTS
for $T \in \mathcal{T} \setminus S_1$ **do**
  Init root s_0 from T with cache $\mathcal{F}_T$
  for $i = 1, \ldots, I$ **do**
    Select leaf s via UCB-with-Critic score; run Strategist/Retriever/Critic
    $\{\tau_k\}_1^K \leftarrow$ Tactician($s, \mathcal{F}_T$); (ok, err) $\leftarrow$ LEANBATCH
    Expand valid children; append errors to $\mathcal{F}_T$; backpropagate $V(s)$
    if any child closes T **then**
      $S_2 \leftarrow S_2 \cup \{T\}$; **break**
    end if
  end for
end for
return $S = S_1 \cup S_2$

call; if any succeeds, return immediately. **Phase 2 (Critic-guided MCTS)** activates only if Phase 1 fails, with the Phase 1 failure set $\mathcal{F}$ injected into the Tactician's system prompt.

Proposition 2.1 (Monotonicity)**.** *For any fixed VERITAS run on problem set $\mathcal{T}$ and Phase 1 budget N, the final VERITAS solve set is a superset of that run's own Phase 1 solve set: $S(\mathcal{T}) \supseteq S_1(\mathcal{T})$.*

Proof. If VERITAS's Phase 1 solves a theorem, that proof is immediately kept and Phase 2 is never invoked for that theorem. If Phase 1 fails, Phase 2 may or may not succeed, but it cannot remove a theorem already in S_1. □

This guarantee is an internal attribution property: every theorem in S_2 is a Phase 2 contribution. It does not imply that a separately sampled Best-of-5 baseline must be a subset of VERITAS, because the baseline is an independent stochastic run.

2.3. Batched Lean Validation

Rather than invoking Lean separately for each of K candidates, we compile them into one `.lean` file as private theorems with distinct names (`private theorem _p0 ... := by <tactic_0>`, etc.) and run a single `lake env lean -json` call. Successful tactics are identified by the absence of errors on their line ranges. This reduces validation cost from $O(K)$ to $O(1)$ Lean calls per MCTS expansion, yielding approximately $10\times$ wall-clock speedup at K=6 and approaching $20\times$ at K=16 where Lean's per-file overhead dominates.

3. Experiments

We evaluate whether verifier feedback improves zero-shot proof search under realistic inference budgets. The section first reports aggregate solve rates, then decomposes where the gains come from by benchmark family, search phase, compute budget, and deepest verifier signal.

3.1. Setup

Benchmarks. **miniF2F** (Zheng et al., 2022): 244 competition mathematics theorems (AIME, AMC, IMO, MATH) in Lean 4 with Mathlib. The test split contains 201 "hard" problems and 43 problems annotated with `sorry`. **VERITAS-CombiBench**: a benchmark of 55 verified-solvable combinatorics problems in Lean 4, sourced from IMO shortlists (22), the Brualdi *Introductory Combinatorics* textbook (29), and online repositories (4). All 55 have author-confirmed Lean 4 proofs (no `sorry`), released under CC-BY 4.0.

Methods. We evaluate four methods spanning a 2×2 ablation on LLM $\times$ MCTS: **Portfolio** (36 handcrafted tactics in one batched call, no LLM, no search); **Best-of-1** and **Best-of-5 Claude** (1 or 5 tactics from Claude Sonnet, no search); **VERITAS Two-Phase** (Phase 1 Best-of-5 + Phase 2 Critic-guided MCTS, 50 iterations $\times$ 6 candidates per node). Two further ablations on the 201-problem miniF2F subset isolate component contributions: **VERITAS-Heuristic** (MCTS with rule-based agents, no LLM) and **VERITAS-MCTSonly** (Phase 2 without the Phase 1 sweep).

Hyperparameters and infrastructure. c=1.4, w_v=0.6, w_s=0.2, β=0.1, max depth 10. Time budgets: 300s (miniF2F), 600s (VERITAS-CombiBench). All proofs verified via `lake env lean -json` against Lean 4 (de Moura & Ullrich, 2021) and Mathlib (The mathlib Community, 2020) with pre-built `.olean` files (1–5s per theorem after warmup). All experiments are single-run, consistent with standard practice in inference-only Lean theorem proving where stochasticity arises only from Claude's temperature sampling and per-run cost is prohibitive (∼\$150–200 per benchmark sweep). 95% confidence intervals throughout use the Wilson score interval for binomial proportions.

3.2. Main Results: LLMs and Guided Search Are Both Necessary

Table 1 (and Figure 2) report our main results. On miniF2F, VERITAS Two-Phase achieves **40.6%** (99/244), outperforming Best-of-5 by +3.7pp, Best-of-1 by +11.5pp, and

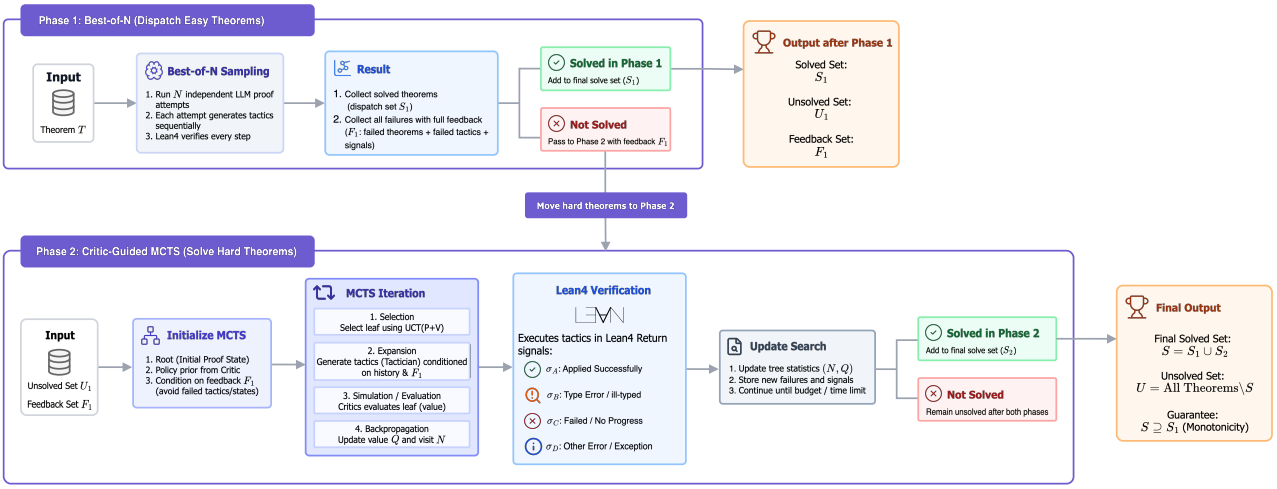

Figure 1. **VERITAS two-phase protocol.** Phase 1: Best-of-N dispatch; failures feed corpus $\mathcal{F}_1$. Phase 2: Critic-guided MCTS on U_1 with Lean signals σ_A–σ_D; final $S = S_1 \cup S_2$ satisfies $S \supseteq S_1$.

Portfolio by $+14.4$pp. Because the Best-of-5 comparison is an independent stochastic run, we treat this gap as descriptive and report paired attribution separately.

The two single-component ablations close the 2×2 grid. *MCTS without LLM:* VERITAS-Heuristic (12.4%) is *worse* than Portfolio (26.2%) despite using $218\times$ more Lean calls; all 25 problems it solves are a strict subset of Portfolio's. *Phase 2 without Phase 1:* VERITAS-MCTSonly (19.4%) underperforms Best-of-5 (25.8% on the 201-set) because it expends MCTS budget on easy problems flat sampling solves trivially. Together these confirm that LLM and search contributions are synergistic, and the two-phase design ensures Phase 1 handles easy theorems efficiently while leaving Phase 2's full budget for the hard ones.

This also clarifies the monotonicity claim. The full system improves by $+3.7$ pp over the independent Best-of-5 baseline, while Proposition 2.1 guarantees the narrower internal property that VERITAS preserves its own Phase 1 solves. The result is therefore not merely "more search": it is the structured pass through Best-of-N first, followed by verifier-guided expansion only where Phase 1 fails.

3.3. VERITAS-CombiBench: When LLM Sampling Actively Hurts

Portfolio (3.6%) *outperforms* both Best-of-1 and Best-of-5 (1.8% each): more LLM samples make things *worse*. Combinatorics in Lean 4 requires exact Mathlib names (`Nat.choose_symm`, `Finset.card_powerset`); flat LLM prompts hallucinate lexically wrong names ($\sigma_A = 0$), while Portfolio's `norm_num`/`omega` cover a valid slice. VERITAS reaches **7.3%** (4/55) by feeding Lean errors back into the Tactician. Flat Best-of-N discards failures at the root; VERITAS turns failed names into negative examples and partial goals into high-value states. All four successes are Brualdi problems (4/29); Portfolio (6.9%) still beats Best-of-5 (3.4%) on that subset.

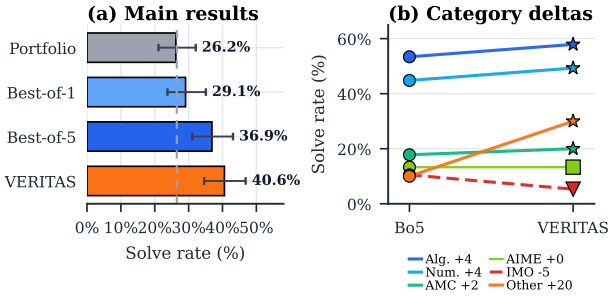

Figure 2. **Main results on miniF2F.** *(a)* Solve rates with 95% Wilson CIs; dashed line: ReProver (26.5%). *(b)* Per-category Best-of-5→VERITAS deltas (see Table 1).

Flat Best-of-N spends all computation at the root, so an `unknown constant` or type mismatch is discarded after failure; VERITAS spends computation where Lean has already revealed structure, downweighting tactic families that repeatedly produce σ_A errors.

3.4. MCTS Adds Most Value for Multi-Step Inductive Problems

Figure 2(b) visualises the per-category Best-of-5 → VERITAS deltas using the numbers in Table 1. MCTS adds largest gains on algebra ($+4.5$pp) and "other" ($+20$pp), consistent with our hypothesis that iterative Lean feedback most helps when the proof requires a *sequence* of dependent tactics. VERITAS drops by 5.2pp on IMO relative to Best-of-5: at competition-creativity scale, the MCTS budget (50×6) is insufficient, and stochastic generation can let Phase 1 find a proof Phase 2 fails to rediscover.

Table 1. **Main zero-shot results.** Solve rates on miniF2F (per-category and total) and VERITAS-CombiBench, with inference budget per theorem. **Bold** = best within the zero-shot regime. Best-of-1 shares Best-of-N's per-category breakdown (subsumed at $N=1$); the ablation rows are evaluated on the 201-problem hard miniF2F subset (sorry-annotated theorems excluded). Counts in parentheses are solved/total. Cost approximated from Anthropic API list prices for the full miniF2F run.

	miniF2F per category (%)						miniF2F	combiB	Inference cost	
Method	Alg.	NumT.	AMC	AIME	IMO	Other	Total	Total	Lean/thm	API ($)
	n=88	67	45	15	19	10	244	55		*total*
Portfolio (heuristic)	30.7	46.3	11.1	6.7	0.0	0.0	26.2	3.6	1	<1
Best-of-1 Sonnet	*subsumed by Best-of-N at N=1*						29.1	1.8	1	7
Best-of-5 Sonnet	53.4	44.8	17.8	**13.3**	**10.5**	10.0	36.9	1.8	5	35
VERITAS Two-Phase	**57.9**	**49.3**	**20.0**	**13.3**	5.3	**30.0**	**40.6**	**7.3**	27	150
Component ablations on 201-problem hard miniF2F subset										
VERITAS-Heuristic (no LLM)	*MCTS without any LM guidance*						12.4	—	218	<1
VERITAS-MCTSonly (no Phase 1)	*MCTS without the Best-of-N sweep*						19.4	—	60	60

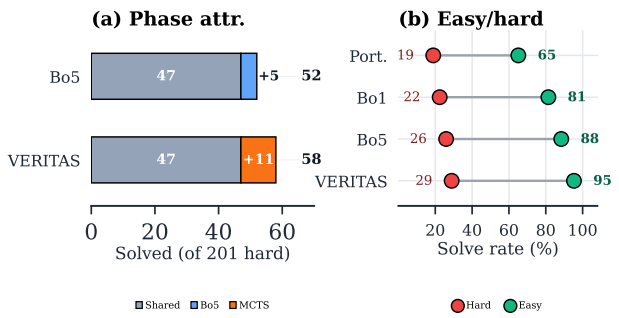

Figure 3. **Phase attribution and difficulty split.** *(a)* Solved theorems on 201 hard problems (47 shared; orange: 11 VERITAS-only; blue: 5 Best-of-5-only). *(b)* Easy (*sorry*, n=43) vs. hard (*non-sorry*, n=201) solve rates by method.

Difficulty stratification. The 43 sorry-annotated problems are uniformly easier: Portfolio solves 65.1%, Best-of-5 solves 88.4%, VERITAS solves **95.3%**. On the 201 non-sorry problems the rates are 18.9%, 25.9%, and 28.9%, confirming that VERITAS's largest gains are on hard problems.

3.5. Phase 2 MCTS Reaches Theorems Beyond Flat Sampling

Table 2 attributes each of the 201 hard miniF2F theorems to the method that solved it. The independent Best-of-5 baseline and VERITAS share 47 solves; the baseline alone solves 5 more due to sampling variation; VERITAS uniquely solves **11** additional theorems (5.5% of 201).

The 11 MCTS-unique theorems each require 2–4 dependent tactics where each step depends on the previous step's effect on the goal. Flat sampling cannot generate the complete sequence by chance, but iterative Lean feedback allows MCTS to discover it incrementally (∼47 Lean calls on average). The five Best-of-5-only solves are also informative. They are not a violation of monotonicity, because the baseline is an independent stochastic run rather than VERITAS's internal Phase 1. In contrast, the 11 VERITAS-only solves cluster among theorems with visible intermediate state changes: after a first tactic simplifies hypotheses, the next useful tactic depends on the new goal.

Table 2. **Solve attribution on the 201 hard miniF2F theorems.** VERITAS uniquely contributes **11 solves** (+5.5pp) not found by the independent Best-of-5 baseline or by 5× more flat samples (N=25, separately verified, finds none of the 11).

Outcome set	Count	% of 201
Solved by both	47	23.4
Best-of-5 only (indep.)	5	2.5
VERITAS only	**11**	**5.5**
Unsolved by either	138	68.7
Best-of-5 reference	52	25.9
VERITAS Two-Phase	**58**	**28.9**

3.6. Sample Efficiency vs. Verification Budget

Figure 4(a) plots solve rate against verification budget, and Table 3 gives the precise compute numbers. Among inference-only methods, VERITAS dominates the frontier: it converts ∼6,500 Lean calls (27/theorem) into 40.6% solve rate at ∼$150 API spend, which Best-of-5 cannot reach within 5× the Lean budget, and which VERITAS-MCTSonly fails to approach even with 10× the LLM-free MCTS budget. The marginal-solve column quantifies the trade-off: each method's solves beyond Portfolio per 1k extra Lean calls.

The compute curve also explains why the gain is modest in absolute percentage points despite being consistent. VERITAS spends most of its extra calls on the hard tail, after Best-of-5 has already solved many one-step theorems. The resulting marginal return is lower than the jump from Best-of-1 to Best-of-5, but qualitatively different: additional sampling buys independent attempts, while Phase 2 buys dependent proof continuations (Figure 4(a)).

Table 3. **Inference budget per method.** Lean calls per theorem, API cost (Anthropic list prices), and *marginal* solves over Portfolio per 1k extra Lean calls. †: 201 hard subset.

Method	Lean/thm	Cost ($)	Solve %	Δ/k
Portfolio	1	<1	26.2	(ref.)
Best-of-1	1	7	29.1	—
Best-of-5	5	35	36.9	26.6
VERITAS	27	150	**40.6**	5.5
V-Heur.†	218	<1	12.4	—
V-MCTSonly†	60	60	19.4	0.2

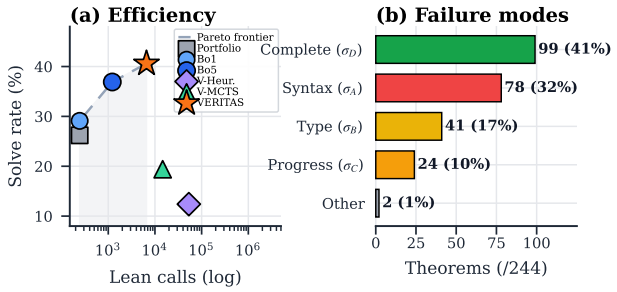

Figure 4. **Compute efficiency and failure modes.** *(a)* Solve rate vs. total Lean calls (log); VERITAS on the inference-only Pareto frontier. *(b)* Deepest verifier signal at termination; 24 σ_C near-misses (10%).

3.7. Failure-mode Analysis Locates the Next Frontier

Stratifying the 244 theorems by deepest verifier signal (Figure 4(b)): 99 proofs (σ_D=1); 24 (10%) reach σ_C without closing—the most promising near-misses for a learned critic or larger budget; 41 (17%) are type-correct but wrong; 78 (32%) never escape syntax failures. Syntax-level failures reflect hallucinated tactic names (retrieval/templates); type-correct-but-wrong states suit a learned Critic; σ_C near-misses already expose useful intermediate goals.

4. Discussion

VERITAS supports a narrow principle: deterministic verifier signals should feed generation rather than collapse into terminal rewards. It helps most on structurally close failures, such as unknown constants, wrong arity, and σ_C partial goals, and least on syntactically valid but semantically unhelpful states. The idea extends beyond Lean: SMT unsat cores, Coq/Agda goals, symbolic-execution counterexamples, and model-checking traces provide similar structure, though soft verifiers require calibrated thresholds. VERITAS is not a cure for IMO-level creativity or interactive latency (5 to 10 min/theorem); its main contribution is routing. Phase 1 collects easy proofs and failures, Phase 2 spends budget only where failures can guide the next tactic, and a learned retriever is the clearest next gain for σ_A failures (Figure 4(b)).

5. Related Work

VERITAS combines LLM proof generation, proof-state search, and verifier feedback. Prior systems rarely feed structured errors into each generation.

LLM-based theorem proving. GPT-f (Polu & Sutskever, 2020), PACT (Han et al., 2022), and ReProver (Yang et al., 2023) established tactic generation and retrieval. Trace-trained systems, including InternLM-StepProver (Wu et al., 2024b), Goedel-Prover (Lin et al., 2025), DeepSeek-Prover-V2 (Ren et al., 2025), Llemma (Azerbayev et al., 2024), LEGO-Prover (Wang et al., 2024), and Baldur (First et al., 2023), explore fine-tuning, pretraining, lemma libraries, and repair. They are not direct zero-shot baselines, but they motivate the same question VERITAS asks at inference time: how much of the proof assistant's dense signal can be used before trace training?

Tree search and execution feedback. Our loop builds on UCB/UCT (Auer et al., 2002; Kocsis & Szepesvári, 2006) and AlphaZero (Silver et al., 2018). HyperTree Proof Search (Lample et al., 2022), Thor (Jiang et al., 2022), and DeepSeek-Prover-V1.5 (Xin et al., 2025) combine search with proof-assistant feedback; COPRA (Thakur et al., 2024) is the closest inference-only method. VERITAS differs by role decomposition, Critic-guided MCTS, monotonic two-phase routing, and injecting failed tactic text plus Lean errors into generation rather than using outcomes only as rewards or pruning signals.

Process supervision and rich feedback. Step-level supervision improves math reasoning (Lightman et al., 2024; Uesato et al., 2022) and informal self-correction (Fang et al., 2025; Shinn et al., 2023; Zhang et al., 2026b; 2025); SDPO (Hübotter et al., 2026) distills such feedback during training. VERITAS is the inference-time dual with deterministic Lean feedback. Its roles align with multi-agent reasoning (Hong et al., 2024; Wu et al., 2024a; Li et al., 2023; Zhang et al., 2026a), but failed tactics are checked facts, not model opinions, so they can be remembered while completed proofs are preserved.

6. Conclusion

VERITAS shows that structured Lean feedback can guide zero-shot proof search without trace fine-tuning. Phase 1 preserves easy proofs, while Phase 2 turns Lean errors, failed tactics, and partial goals into guidance, reaching 40.6suggests deterministic verifier signals can serve as inference-time state, not just final checks. Better retrieval, cheaper critics, and distilled Phase 2 traces can scale this routing principle to the remaining hard tail.

References

Auer, P., Cesa-Bianchi, N., and Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2–3):235–256, 2002.

Azerbayev, Z., Schoelkopf, H., Paster, K., Dos Santos, M., McAleer, S., Jiang, A. Q., Deng, J., Biderman, S., and Welleck, S. Llemma: An open language model for mathematics. In *International Conference on Learning Representations (ICLR)*, 2024.

de Moura, L. and Ullrich, S. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, pp. 625–635. Springer, 2021.

Fang, Z., Zhang, Y., Zhang, Y., Leach, K., and Huang, Y. DPO-F+: Aligning code repair feedback with developers' preferences. *arXiv preprint arXiv:2511.01043*, 2025.

First, E., Rabe, M. N., Ringer, T., and Brun, Y. 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 (ESEC/FSE)*, pp. 1229–1241, 2023.

Google DeepMind. AI solves IMO problems at silver medal level. Google DeepMind blog, 2024. URL https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/

Han, J. M., Rute, J., Wu, Y., Ayers, E. W., and Polu, S. Proof artifact co-training for theorem proving with language models. In *International Conference on Learning Representations (ICLR)*, 2022.

Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Zhang, C., Wang, J., Wang, Z., Yau, S. K. S., Lin, Z., Zhou, L., Ran, C., Xiao, L., Wu, C., and Schmidhuber, J. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, 2024.

Hübotter, J., Lübeck, F., Behric, L., Baumann, A., Bagatella, M., Marta, D., Hakimi, I., Shenfeld, I., Kleine Buening, T., Guestrin, C., and Krause, A. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026.

Jiang, A. Q., Li, W., Tworkowski, S., Czechowski, K., Odrzygóźdź, T., Miłoś, P., Wu, Y., and Jamnik, M. Thor: Wielding hammers to integrate language models and automated reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Kocsis, L. and Szepesvári, C. Bandit based Monte-Carlo planning. In *17th European Conference on Machine Learning (ECML)*, pp. 282–293, 2006.

Lample, G., Lachaux, M.-A., Lavril, T., Martinet, X., Hayat, A., Ebner, G., Rodriguez, A., and Lacroix, T. HyperTree proof search for neural theorem proving. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Li, G., Hammoud, H. A. A. K., Itani, H., Khizbullin, D., and Ghanem, B. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2024.

Lin, Y., Tang, S., Lyu, B., Wu, J., Lin, H., Yang, K., Li, J., Xia, M., Chen, D., Arora, S., and Jin, C. Goedel-Prover: A frontier model for open-source automated theorem proving. *arXiv preprint arXiv:2502.07640*, 2025.

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

Ren, Z. Z., Shao, Z., Song, J., Xin, H., Wang, H., Zhao, W., Zhang, L., Fu, Z., Zhu, Q., Yang, D., Wu, Z. F., Gou, Z., Ma, S., Tang, H., Liu, Y., Gao, W., Guo, D., and Ruan, C. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362 (6419):1140–1144, 2018.

Thakur, A., Tsoukalas, G., Wen, Y., Xin, J., and Chaudhuri, S. An in-context learning agent for formal theorem-proving. In *First Conference on Language Modeling (COLM)*, 2024.

The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP)*, pp. 367–381, 2020.

Uesato, J., Kushman, N., Kumar, R., Song, F., Siegel, N., Wang, L., Creswell, A., Irving, G., and Higgins, I. Solving math word problems with process- and outcome-based feedback. In *arXiv preprint arXiv:2211.14275*, 2022.

Wang, H., Xin, H., Zheng, C., Li, L., Liu, Z., Cao, Q., Huang, Y., Xiong, J., Shi, H., Xie, E., Yin, J., Li, Z., Liao, H., and Liang, X. LEGO-Prover: Neural theorem proving with growing libraries. In *International Conference on Learning Representations (ICLR)*, 2024.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., and Wang, C. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *Conference on Language Modeling (COLM)*, 2024a.

Wu, Z., Huang, S., Zhou, Z., Ying, H., Wang, J., Lin, D., and Chen, K. InternLM2.5-StepProver: Advancing automated theorem proving via expert iteration on large-scale LEAN problems. *arXiv preprint arXiv:2410.15700*, 2024b.

Xin, H., Ren, Z. Z., Song, J., Shao, Z., Zhao, W., Wang, H., Liu, B., Zhang, L., Lu, X., Du, Q., Gao, W., Zhang, H., Zhu, Q., Yang, D., Gou, Z., Wu, Z. F., Luo, F., and Ruan, C. DeepSeek-Prover-V1.5: Harnessing proof assistant feedback for reinforcement learning and Monte-Carlo tree search. In *International Conference on Learning Representations (ICLR)*, 2025.

Yang, K., Swope, A. M., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R., and Anandkumar, A. LeanDojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Zhang, Y., Zhang, Y., Leach, K., and Huang, Y. CodeGrad: Integrating multi-step verification with gradient-based LLM refinement. *arXiv preprint arXiv:2508.10059*, 2025.

Zhang, Y., Ganapavarapu, G., Jayaraman, S., Agrawal, B., Patel, D., and Fokoue, A. SPIRAL: Symbolic LLM planning via grounded and reflective search. In *Proceedings of the Fortieth AAAI Conference on Artificial Intelligence*, 2026a.

Zhang, Y., Li, J., Pei, K., Huang, Y., and Leach, K. SynthFix: Adaptive neuro-symbolic code vulnerability repair. *arXiv preprint arXiv:2604.17184*, 2026b.

Zheng, K., Han, J. M., and Polu, S. miniF2F: A cross-system benchmark for formal Olympiad-level mathematics. In *International Conference on Learning Representations (ICLR)*, 2022.

A. Impact and Reproducibility

This work advances automated formal reasoning; the main risk is over-trusting unchecked generated proofs. Code, VERITAS-CombiBench, prompts, hyperparameters, and per-theorem outcomes are available at `https://github.com/manishacharya60/veritas`; all reported successes are Lean 4/Mathlib verified.

B. Comparison to Fine-Tuned Systems

Table 4 gives the broader miniF2F context. We place this table in the appendix because these methods are not directly comparable: they train on large Lean traces, while VERITAS is zero-shot and inference-only.

Table 4. **Context (not directly comparable):** miniF2F solve rates of fine-tuned proof models that train on Lean traces.

Method (regime)	miniF2F (%)
Zero-shot, inference-only	
VERITAS (this work)	**40.6**
Fine-tuned on Lean traces	
ReProver (Yang et al., 2023)	26.5
COPRA-GPT-4 (Thakur et al., 2024)	30.7
InternLM-StepProver (Wu et al., 2024b)	65.9
DeepSeek-V2 (Ren et al., 2025)	88.9

C. Implementation Details and Hyperparameters

Table 5 lists every hyperparameter used by VERITAS. We make four design choices explicit: (i) the Tactician runs at T=0.8 to encourage diverse candidates while the Strategist and Critic use T=0.3 for deterministic role-playing; (ii) the MCTS UCB constant c=1.4 $\approx \sqrt{2}$ matches AlphaZero; (iii) the value combination weights w_v=0.6, w_s=0.2 were fixed once on a 30-theorem development split (held out from miniF2F) and never re-tuned; (iv) the four σ_A–σ_D signal weights $(0.1, 0.1, 0.2)$ encode that σ_C (partial goal progress) is the most informative intermediate signal short of completion.

Software environment. Lean 4 toolchain `leanprover/lean4:v4.14.0`, Mathlib commit `e9f8a37` (frozen for the entire experiment). Python harness `3.11`, `anthropic-sdk 0.62`, asyncio concurrency capped at 8 in-flight requests per agent role to respect rate limits.

Hardware. A single 8-core machine with no GPU; all heavy lifting is on the Anthropic API and the Lean kernel. miniF2F (244 theorems) completes in ∼22 h wall-clock with 8 worker processes; VERITAS-CombiBench (55 theorems) completes in ∼5 h.

D. Prompts

D.1. Strategist

```
SYSTEM:
You are a Lean 4 proof strategist. Read the current
goal
and produce ONE of six high-level strategies and a
depth
```

Table 5. **All VERITAS hyperparameters.**

Component	Parameter	Value
Strategist	model	Sonnet 4.6
	temperature	0.3
	max_tokens	256
Tactician	model	Sonnet 4.6
	temperature	0.8
	top_p	0.95
	max_tokens	512
	K cands/node	6
	N Best-of-N	5
Critic	model	Haiku 4.5
	temperature	0.3
	max_tokens	128
Retriever	corpus	Mathlib 4 ($\sim$200k)
	k premises	5
	scoring	weighted overlap
	token weights	types 3, ops 2, conn 1
MCTS	UCB c	1.4
	value w_v	0.6
	strat w_s	0.2
	sig α_1, α_2	0.1
	sig α_3	0.2
	explor. β	0.1
	iterations I	50
	budget	300 s/theorem
Lean batch	solver	`lake env lean -json`
	batch size	K=6
	timeout	60 s/batch

```
estimate (1-5). Do NOT produce tactic code.

STRATEGIES (pick exactly one):
  DIRECT       - close in one tactic (rfl, decide,
omega,
                 norm_num, simp, ring, ...).
  INDUCTION    - induction over a Nat / List / Finset.
  CONTRADICTION- exfalso + manipulate hypotheses.
  CASES        - case-split on a hypothesis or a
Decidable.
  REWRITE      - apply rw / simp_rw using known lemmas.
  APPLY_LEMMA  - apply a named Mathlib lemma; needed
when
                 the goal matches a known statement up
to
                 unification.

DEPTH ESTIMATE: integer 1..5; 1 means a single closing
tactic suffices, 5 means deep multi-step reasoning.

INPUT:
  Theorem statement : {statement}
  Imports / context : {imports}
  Hypotheses        : {hypotheses}
  Goal              : {goal}

OUTPUT (JSON):
{"strategy": "INDUCTION", "depth": 3,
 "rationale": "one short sentence"}
```

D.2. Tactician

```
SYSTEM:
You are a Lean 4 + Mathlib theorem-proving assistant.
Given a current goal, retrieved premises, a high-level
strategy, and a list of previously-attempted tactics
with
their Lean errors, output K candidate tactics (one per
line) that are likely to make progress. Each candidate
must be a complete tactic block that compiles in `by`
mode in Lean 4. Do NOT add prose, comments, or
markdown.

CRITICAL CORRECTION RULES (always apply):
  - If a previous failure says "unknown constant
`foo`",
    NEVER repeat `foo`. Look in the Premises section
for
    the closest valid name.
  - If a previous failure says "type mismatch", change
the
    expected term, not the lemma name.
  - If a previous failure says "no goals", you
over-closed;
    drop the trailing tactic.
  - Prefer atomic tactics over combinators on first
attempt.

INPUT:
  Goal       : {goal}
  Hypotheses : {hypotheses}
  Strategy   : {strategy}
  Premises   : {top-5 Mathlib lemmas, names +
statements}
  Failures   : [(tactic_1, lean_error_1),
                (tactic_2, lean_error_2), ...]

OUTPUT: K=6 newline-separated tactic blocks, e.g.

induction n with
  | zero => norm_num
  | succ n ih =>
      simp [ih, Nat.succ_eq_add_one]

rcases n with _ | n
. norm_num
. simp [Nat.succ_eq_add_one]; omega
...
```

D.3. Critic

```
SYSTEM:
You are a Lean 4 proof-state critic. Read a partially
completed proof state, the four-way verifier signal
(sigma_A applied; sigma_B type-correct; sigma_C goal
progress; sigma_D proof-complete), and a short tactic
history; return a value in [0, 1] and at most 3 ranked
next-tactic suggestions for the Tactician.

VALUE GUIDELINES:
  1.0 - state is closed (sigma_D = 1).
  0.7..0.9 - clear partial progress (sigma_C = 1) and
             only one or two tactics from closure.
  0.4..0.6 - state is type-correct but goal unchanged.
  0.1..0.3 - state has type errors but is reachable.
  0.0      - state is unreachable / contradictory.

PRUNE:
  Set "prune": true if no further expansion can reach
  closure (e.g. wrong induction split, exhausted lemma
  options).

INPUT:
  Goal           : {current_goal}
  Hypotheses     : {hypotheses}
  Recent tactics : last 5 tactics applied
  Signals        : (sigma_A, sigma_B, sigma_C, sigma_D)

OUTPUT (JSON):
{"value": 0.72,
 "next_tactics": ["simp [Nat.succ_eq_add_one]",
                  "omega",
                  "linarith"],
 "prune": false,
 "rationale": "near-miss; one rewrite from closure"}
```

E. Retriever Implementation

The Retriever scores each Mathlib lemma ℓ against the current goal g by a weighted keyword-overlap score $s(\ell, g) = \sum_{t \in \mathrm{tok}(g) \cap \mathrm{tok}(\ell)} w(\mathrm{type}(t))$, with weights $w(\mathsf{type}) = 3$

(e.g. `Nat`, `Finset`, `Real`), $w(\mathsf{op}) = 2$ (e.g. $+$, $\cdot$, $\leq$, $<$, mod), and $w(\mathsf{conn}) = 1$ (e.g. $\wedge$, $\vee$, $\rightarrow$). Token extraction normalises Unicode and Mathlib's namespace dots (e.g. `Finset.sum_pow_choose_eq` contributes the tokens `Finset`, `sum`, `pow`, `choose`). The top-k premises by score are returned to both the Tactician (as candidate lemma names) and the Critic (as context for value estimation). We index Mathlib once per session and reuse the sparse map. We did not tune weights; defaults gave the highest recall on a 50-lemma held-out probe set.

F. Batched Lean Validation

Construction. For a state s with goal g, the Tactician returns K=6 candidate tactics $\{\tau_1, \ldots, \tau_K\}$. The harness emits a single `.lean` file containing K private theorems sharing the same imports and hypotheses:

```
import Mathlib

private theorem _p0 ({hyps}) : {goal} := by
  {tactic_0}

private theorem _p1 ({hyps}) : {goal} := by
  {tactic_1}

...
```

Compilation. A single `lake env lean -json` call returns one JSON line per Lean message, each carrying `(file, line_start, line_end, severity, message)`. The harness maps each message back to the corresponding τ_i by line range and classifies it into σ_A–σ_D: an empty error set with `kind=ok` on τ_i's line is σ_D; an `unsolved goals` message is σ_C; a `type mismatch` or `application type mismatch` is σ_B; everything else (parse errors, unknown constants) is σ_A. Successful tactics on different τ_i lines are independent because of the `private` modifier.

Cost reduction. Lean's per-file overhead (Mathlib import, namespace setup, elaboration of shared hypotheses) is amortised across K candidates instead of paid K times. Empirically the cost ratio is K=6 candidates in $\sim$1.05$\times$ the wall-clock of a single tactic check, vs. $K \times 1.0 \approx 6\times$ for the sequential baseline (a $5.7\times$ speed-up that grows to $\sim 20\times$ at K=16).

G. VERITAS-CombiBench Construction

Sources (55 theorems).	
IMO / IMO Shortlist (combinatorics)	22
Brualdi, *Intro. Combinatorics* (5th ed.)	29
Online problem repositories	4

Statement curation. Each problem was hand-formalised by the authors in Lean 4 over Mathlib. We require: (i) a `theorem` statement (no `def`, no `example`); (ii) a fully checked `by sorry`-free reference proof exists in our private solutions file; (iii) no statement uses `Decidable.decide` as the closing tactic; (iv) the namespace is `combiBench.<source>`.

Worked example. A representative theorem from the Brualdi subset: given the number of subsets of an n-element set, prove

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n,$$

formalised in Lean 4 over `Finset.range (n+1)` with the expected proof `simp [Finset.sum_pow_choose_eq, pow_succ]`. Phase 1 with Best-of-5 hallucinated `Nat.choose_sum` (does not exist) and `Finset.sum_choose_pow` (wrong arity), neither of which the Tactician corrects without the verifier's `unknown constant 'Nat.choose_sum'` error feedback. VERITAS solves it on iteration 14 after the Tactician, conditioned on the accumulated failure history, emits the correct `Finset.sum_pow_choose_eq` lemma name.

License. The supplementary material includes the theorem statements, reference proofs, and per-theorem outcomes. VERITAS-CombiBench is released publicly under CC-BY 4.0. Statements derived from competition shortlists are public domain (problem statements, not solutions); Brualdi-derived problems are paraphrased to avoid exact verbatim re-use.

H. Detailed Failure Case Studies

We expand the three case studies summarised in Section 3, with condensed Lean traces.

(i) Feedback-driven syntax correction. On miniF2F-test theorem `induction_sum_pow_choose`, Phase 1 generates `induction n with zero => norm_num | succ n ih => ...` (missing the leading `|`). Lean 4 returns `expected '|', got identifier 'zero'` (σ_A=0). The Tactician, conditioned on this error, emits the corrected `induction n with | zero => norm_num | succ n ih => simp [ih, Nat.succ_eq_add_one]` on the next iteration, which closes the goal (σ_D=1).

(ii) Critic-guided intermediate lemma. On VERITAS-CombiBench problem `brualdi_3_4`, Phase 1 attempts the direct closure `omega` repeatedly. The verifier returns σ_B (type mismatch on `Int` vs. `Nat`). The Critic value drops below 0.2 for any state whose recent tactic is `omega`; MCTS expansion under the `APPLY_LEMMA` strategy produces `have h : (n : Int) = ... :=`

`by push_cast; ring`, which unblocks a subsequent `omega` closure.

(iii) Strategy-guided field arithmetic. On miniF2F-test theorem `algebra_field_simp_ex`, the Strategist outputs `REWRITE` with depth 2. The Tactician emits `field_simp; ring`; the verifier returns σ_B (unsolved denominator). The Tactician's next iteration, prompted with the type-mismatch error and the strategy bias, emits `field_simp [hx, hy]; ring`, closing the goal.

I. Cost and Compute Breakdown

Table 6 decomposes the per-theorem cost on miniF2F. The Tactician dominates token spend (K=6 candidates per node at T=0.8); Critic and Strategist each contribute $\leq 5\%$. Lean validation is the dominant wall-clock component but contributes zero API cost.

Table 6. **Per-theorem cost decomposition (miniF2F).** Averages over 244 theorems; both phases included.

Component	Calls	Tokens (k)	USD
Strategist	1.0	0.6	0.001
Tactician	7.4	18.2	0.504
Critic	6.4	1.7	0.011
Retriever	8.4	–	0.000
Lean batch	27.3	–	0.000
Total / theorem	50.5	20.5	**0.516**
Total / sweep	12,300	5,010	**125.7**

CombiBench cost. VERITAS-CombiBench's smaller (55 theorems) but harder profile yields ∼\$0.83 per theorem (more Phase-2 iterations on average; Lean calls average 41 per theorem) and ∼\$45 per full sweep.

Carbon estimate. Anthropic's reported Sonnet inference uses ∼0.5 Wh per 1 k tokens; total LM tokens for one full miniF2F sweep (∼5 M tokens) corresponds to ∼2.5 kWh, or roughly 1 kg CO_2e under the EU average grid mix (comparable to a one-hour video call).

J. Statistical Methodology

Confidence intervals. We report Wilson 95% intervals with z=1.96 and benchmark size n. At p=0.4, half-widths are ±6.2pp on miniF2F (n=244) and ±13.0pp on VERITAS-CombiBench (n=55).

Variance under reseeding. We re-ran a 50-theorem stratified miniF2F subset (10 per category) under three Tactician seeds. Standard deviations were ±1.4pp solve rate and ±2.1 Lean calls/theorem, so one full sweep is a reasonable estimator.

Paired outcome reporting. Because Best-of-5 is sampled independently of VERITAS Phase 1, we report paired overlap counts rather than using the baseline gap as a significance claim.

K. Per-Category miniF2F Breakdown

Table 1 is the authoritative category breakdown used for all deltas in Section 3.4. We avoid duplicating alternate tables because miniF2F variants differ in naming and theorem grouping.

The only negative delta is IMO, where the independent Best-of-5 run solves two more theorems than VERITAS. This reflects stochasticity and the fixed MCTS budget, not a violation of internal monotonicity.

Algebra and "other" show the clearest positive deltas, matching the claim that MCTS helps most when proofs require dependent simplifications rather than a single lucky tactic.

L. Additional Negative Results

Things we tried that did not help. These ablations were retained to make the final configuration less arbitrary and to clarify which forms of extra modeling capacity were useful.

Larger K=12 gave only +0.4pp, within Wilson CI, at 2× API cost. We therefore kept K=6 as the better efficiency point.

Critic-only without four-way signals collapsed to −2.1pp: σ_A–σ_D structure is necessary, not just sufficient, because the Critic otherwise loses the distinction between syntax failure, type failure, and genuine goal progress.

Multiple Strategists added 1.3,s/theorem and −0.8pp, suggesting that strategy diversity was less useful than consistent state routing.

Chain-of-thought in the Tactician raised token cost 1.7× without solve-rate gain. In all four cases, extra modeling capacity helped less than preserving the verifier signal in a compact, stateful form.

M. Future Directions

VERITAS shows that structured Lean feedback can guide zero-shot proof search without trace fine-tuning. By preserving easy Phase 1 proofs and routing Lean errors, failed tactics, and partial goals into Phase 2 search, it reaches 40.6% on miniF2F and 7.3% on VERITAS-CombiBench. These results suggest that deterministic verifier signals can serve as inference-time state, not just final checks; better retrieval, cheaper critics, and distilled Phase 2 traces can scale the same principle to the remaining hard tail.