WITALLOC: DEPENDENCY-AWARE CLAIM–WITNESS ALLOCATION FOR NEURAL PROOF REPAIR

Anonymous authors
Paper under double-blind review

ABSTRACT

A sound check can be spent on the wrong proof node: a cheap leaf may pass while one unchecked upstream error invalidates its dependency cone. WITALLOC therefore allocates a claim and its executable witness jointly. It compiles generated traces into typed dependency DAGs, admits context-bound contracts from five frozen witness families, and ranks affordable pairs by source risk, false-pass-adjusted fidelity, inclusive reach, and cost; a failed contract rebuilds the affected subgraph and must pass a rebound audit. Across miniF2F, SV-COMP, and MATH-Combinatorics, this ordering of an identical typed inventory adds 2.1–2.2 accepted-correct points and removes 2.6–4.2 false-acceptance points relative to uniform placement. Against a 355M-parameter learned step verifier, paired gains are 0.7–0.9 accepted-correct points and 2.0–2.1 fewer false-acceptance points at lower realized cost; prospective natural errors show 4.6–6.1 fewer accepted-wrong points and 4.1–5.4 more repaired-to-correct points. The advantage peaks at intermediate caps and in high-coverage traces, where choosing which evidence to execute matters more than increasing audit volume.

1 INTRODUCTION

A generated miniF2F trace weakens $x \geq -2$ to $x \geq -4$ and builds six later claims on that bound. Uncertainty-first checking spends its budget on a cheaper leaf and receives `pass`. A context-bound arithmetic witness on the upstream claim instead returns $x = -3$, identifies the invalid cone, and enables its repair; the rebound contract then passes (Figure 1).

This is a two-address decision: which node should be checked, and which executable judgment should be bound to its assumptions, variables, types, and domains? Scalar process verifiers supply only the first address; uniform execution ignores unequal dependency consequences. On prospective natural failures, the distinction corresponds to 4.6–6.1 fewer accepted-wrong points and 4.1–5.4 more repaired-to-correct points than learned step verification.

WITALLOC compiles candidates into typed dependency DAGs and admits five frozen witness families through semantic signatures. An online score ranks affordable claim–witness pairs by predicted source risk, false-pass-adjusted fidelity, inclusive dependency reach, and cost. Outcomes are fail-closed: `fail` triggers dependency-local reconstruction, while `compile_error` and `timeout` remain unresolved; any repair must pass a rebound instance of the selected witness family within the same cap.

The evaluation separates placement from the complete evidence interface. Holding witnesses, traces, repair, and metering fixed, dependency-aware order adds 2.1–2.2 accepted-correct points and removes 2.6–4.2 false-acceptance points over uniform typed placement. Against a calibrated 355M-parameter learned verifier, the complete interface gains 0.7–0.9 accepted-correct points and removes 2.0–2.1 false-acceptance points across three domains, with paired intervals excluding zero and lower realized cost. Budget frontiers, semantic controls, held-out localization, natural errors, adaptive attacks, and a second generator locate the gain where several contracts compete for limited compute. The contribution is thus an allocation unit that commits target, context, executable semantics, and charge together, then preserves that contract through graph-local repair.

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

2 Related Work

Neural proving and repair. Neural theorem proving has progressed through formal-data training, hyper-tree search, retrieval-augmented environments, proof decomposition, and reusable lemma libraries (Polu & Sutskever, 2020; Polu et al., 2023; Lample et al., 2022; Yang et al., 2023; Jiang et al., 2023; Wang et al., 2024; Xin et al., 2024). THOR integrates language models with automated provers (Jiang et al., 2022); Baldur and APOLLO use compiler feedback for whole-proof or sub-proof repair (First et al., 2023; Ospanov et al., 2025). Exact final checking remains the certification endpoint when a domain provides it. WitAlloc addresses the preceding repair decision: which local diagnostic to execute when several typed contracts compete for a cap. Generation, candidate traces, and the repair controller are held fixed so that the experiment measures this evidence-placement interface.

Learned verification and test-time allocation. Outcome verifiers, process reward models, and process supervision assign scalar evidence to candidate solutions or intermediate steps (Cobbe et al., 2021; Uesato et al., 2022; Lightman et al., 2024). Compute-aware search adapts sampling and path selection under inference budgets (Song et al., 2025), while selective verification routes a frozen solver to an additional verification intervention (Dip et al., 2026). Their control variable is typically a sample, reasoning step, or additional model call. Here the action has two addresses, (v, W_j): the node determines the repair boundary, while the witness determines the executable judgment and any returned diagnostic. The learned-verifier comparison evaluates this complete interface against scalar scoring; the uniform-typed comparison keeps contracts and diagnostics identical and isolates pair ordering.

Executable evidence and local formal checking. Proof-carrying code, verified compilation, Lean, Dafny, Why3, and Z3 demonstrate machine-checkable evidence interfaces (Necula, 1997; Leroy, 2009; de Moura et al., 2015; Leino, 2010; Filliâtre & Paskevich, 2013; de Moura & Bjørner, 2008). Tool-integrated reasoning brings execution into language-model traces (Gao et al., 2023; Chen et al., 2023; Gou et al., 2024; Schick et al., 2023). SAFE formalizes and verifies natural-language reasoning steps (Liu et al., 2025), and FaithSieve tests whether local formal evidence faithfully represents a natural-language step (Wang et al., 2026). These systems establish valuable checker and tool interfaces. The allocation contribution is orthogonal: WitAlloc chooses among a frozen inventory of local evidence interfaces, charges the selected node–contract pair to an end-to-end cap, and carries its semantics through a versioned repair.

3 Semantically Bound Claim–Witness Allocation

Let $G = (V, E)$ be a proof dependency DAG, with an edge $v \to w$ when claim w depends directly on claim v. Node v stores claim ϕ_v, formal context Γ_v, semantic signature σ_v, uncertainty u_v, and descendants $\mathrm{desc}_G(v)$. Its inclusive dependency reach is

$$\mathcal{R}_G(v) = \{v\} \cup \mathrm{desc}_G(v), \qquad \rho_G(v) = |\mathcal{R}_G(v)|.$$

Thus a leaf has reach one. The signature binds quantifiers, free variables, assumptions, types, and finite domains. A witness compiler W_j is compatible with v when $W_j \in \mathcal{W}(v)$ and produces one outcome in

$$\{\texttt{pass}, \texttt{fail}, \texttt{compile_error}, \texttt{timeout}\}$$

at measured cost $\kappa(v, W_j)$.

Definition 1 (Budgeted claim–witness allocation)**.** *For admissible pairs*

$$\mathcal{A}(G) = \{(v, j) : v \in V,\ W_j \in \mathcal{W}(v)\},$$

choose $x_{vj} \in \{0, 1\}$ *to maximize expected false-acceptance reduction*

$$\max_{x}\ \mathbb{E}[\mathrm{FA}(0) - \mathrm{FA}(x)] \quad \textit{s.t.} \quad \sum_{(v,j)\in\mathcal{A}(G)} x_{vj}\kappa(v, W_j) \le B, \qquad \sum_{j} x_{vj} \le 1\ \ \forall v. \tag{1}$$

Equation 1 defines the decision target on one graph version. The one-witness constraint also applies per version: a failure produces a new state with a rebuilt affected subgraph, a new feasible set, residual

budget, and one mandatory reaudit of the repaired node. Reaudit retains the witness family and contract schema but rebinds the repaired $(\phi_v, \Gamma_v, \sigma_v)$ after confirming that the root goal and inherited ancestor context are unchanged. Because detection, repair, and overlapping dependency cones make the full trajectory outcome-dependent, the implemented selector is an online marginal-damage surrogate rather than an offline optimizer of Equation 1. Its quantities are development-estimated or directly measured, and every risk, reach, compatibility, and affordability term is recomputed after a graph transition.

Acceptance semantics. A `pass` validates only its bound local contract. A `fail` flags the node and descendants and invokes the shared repair controller if budget remains. A `compile_error` or `timeout` is unresolved and cannot be converted into a pass. All generation, scoring, compilation, execution, repair, retry, and timeout charges consume the corresponding cap. Runtime acceptance requires a protocol-valid candidate—successful parsing, typing, graph construction, and output-format validation, without access to the benchmark answer oracle—together with passed final-version selected audits, no unresolved selected outcome, and every required post-repair reaudit completed within budget. Unselected nodes retain the status `unaudited`. The benchmark oracle is used only after execution to label accepted-correct rate, false acceptance, and successful repair.

4 WITALLOC

WITALLOC enforces three allocation invariants: actions address typed intermediate DAG nodes, witness semantics are bound before execution, and every unit of verification budget is charged to an affordable claim–witness pair. The graph model, compiler inventory, alignment objective, and repair controller implement these invariants while directing execution toward expected downstream damage.

4.1 TYPED PROOF-DAG COMPILATION

The compiler extracts ϕ_v, Γ_v, support edges, σ_v, and u_v from each prover trace. Lean 4 tactic states make the local goal and context explicit, with edges from premise-producing states to the child subgoals that consume them. SV-COMP verification-condition generation yields typed obligations and their dependence on path conditions and invariants. For MATH-Combinatorics, the adapter emits a node only when the trace states a finite domain or a checkable arithmetic or rewrite relation; support edges follow referenced prior claims. A node with an incomplete or ambiguous signature has no compatible witness and cannot enter the action set. Average graph sizes are $|V| = 12.4$, $|E| = 14.7$ on miniF2F and $|V| = 8.2$ on SV-COMP.

4.2 NODE RISK AND FIXED WITNESS SEMANTICS

The risk model estimates whether a node is the source of a local error from its serialized claim, formal context, semantic signature, uncertainty, and graph neighborhood. Writing their learned node representation as h_v, the predictor is

$$r_v = R_\psi(v) = \sigma(\mathrm{GNN}_\psi(h_v, \{h_w : (v, w) \in E\}, u_v, \rho_G(v))) \,. \tag{2}$$

It uses three message-passing layers and 256-dimensional hidden states. On controlled-error DAGs, the directly modified node is the positive source label; all labels are fixed by the sampler before any policy runs. Source-proof identities are disjoint across training, development, and evaluation.

The compiler inventory contains W_{search} for bounded finite search, W_{arith} for exact rational and interval arithmetic, W_{rewrite} for checked equational certificates, W_{test} for executable program invariants and postconditions, and W_{tactic} for Lean kernel checking of local tactic certificates. This inventory is frozen before evaluation. Benchmark adapters map native objects to the same five contracts; they do not introduce benchmark-specific witness families or retune the generator.

Semantic signatures bind the witness to its intended context. Alignment training contrasts a faithful witness w_v^+ with five corruptions that weaken quantifiers, relax bounds, drop assumptions, substitute a type-compatible variable, or check only a subset of required conditions:

$$\mathcal{L}_{\mathrm{align}} = -\log \frac{\exp(\mathrm{sim}(\phi_v, w_v^+)/\tau)}{\exp(\mathrm{sim}(\phi_v, w_v^+)/\tau) + \sum_k \exp(\mathrm{sim}(\phi_v, w_v^{-,k})/\tau)}, \qquad \tau = 0.07. \tag{3}$$

The similarity encoder consumes the full $(\phi_v, \Gamma_v, \sigma_v)$ serialization on the claim side and the compiled contract on the witness side. It supplies the alignment gate used to construct $\mathcal{W}(v)$: a binding enters the feasible set only when its score exceeds a family-specific threshold selected on source-disjoint development groups. Ambiguous signatures and bindings below threshold are treated as incompatible rather than delegated to execution. The encoder and all thresholds are frozen before evaluation.

4.3 AFFORDABLE PAIR SELECTION AND REPAIR

For each family, development negatives estimate

$$\widehat{\mathrm{FPR}}(W_j) = \frac{\mathrm{FP}_j + 1}{N_j^- + 2}.$$

Here a false positive is a corrupted local claim that receives `pass`, so $1 - \widehat{\mathrm{FPR}}(W_j)$ estimates the family's detection fidelity on compatible development contracts. The Beta-smoothed rates, family thresholds, cost estimates, and compiler completion statistics are frozen before test evaluation. Held-out attack false-pass rates in Section H never enter the score. Among currently affordable pairs on graph version G_t, WITALLOC maximizes

$$s_t(v, W_j) = \frac{r_v(1 - \widehat{\mathrm{FPR}}(W_j))\rho_{G_t}(v)}{\kappa(v, W_j)}. \tag{4}$$

The numerator is the estimated corrupted-node mass exposed by one resolved check under a local source-error approximation: source risk times witness detection fidelity times the number of claims invalidated by that source. Division by κ converts it to value per reserved execution charge. The inclusive current-version reach $\rho_{G_t}(v)$ values what a failure would invalidate; it does not treat a pass as certification of every descendant. Overlapping reaches are not removed after a pass because another local source may remain inside the same cone. After repair, changed descendants become new-version nodes and every risk, reach, compatibility relation, and affordable pair is recomputed, preventing stale scores from crossing graph versions. Ties are resolved by larger numerator, lower cost, topological index, and compiler identifier. The loop stops only when no compatible unaudited pair is affordable.

Algorithm 1 WITALLOC audit, repair, and reaudit

Require: Candidate π, budget B, compilers $\mathcal{W}$, risk model R_ψ
1: Build current graph G and set remaining budget $B_r \leftarrow B$
2: Mark every node `unaudited`
3: **while** an affordable compatible unaudited pair exists **do**
4: $F \leftarrow \{(v, j) \in \mathcal{A}(G) : v$ unaudited, $\kappa(v, W_j) \le B_r\}$
5: Select $(v^*, j^*) \leftarrow \arg\max_{(v,j)\in F} s(v, W_j)$ with fixed tie-breaking
6: Execute $o \leftarrow W_{j^*}(\phi_{v^*}, \Gamma_{v^*})$ and debit realized cost
7: **if** $o =$ `pass` **then**
8: Mark v^* passed on the current graph version
9: **else if** $o =$ `fail` and one repair attempt is affordable **then**
10: Flag $\mathcal{R}_G(v^*)$ and attempt at most two local repairs within the residual cap
11: After each attempt, preserve root/ancestor semantics and rebuild the affected subgraph
12: Rebind and execute W_{j^*} on repaired (ϕ, Γ, σ); stop at the first pass
13: Reject if no repair attempt produces a mandatory reaudit pass within budget
14: Mark the repaired node passed, changed descendants unaudited; refresh R_ψ and $\mathcal{A}(G)$
15: **else**
16: Reject because the selected audit is unresolved
17: **end if**
18: **end while**
19: Accept iff the candidate and all selected final-version audits satisfy Section 3

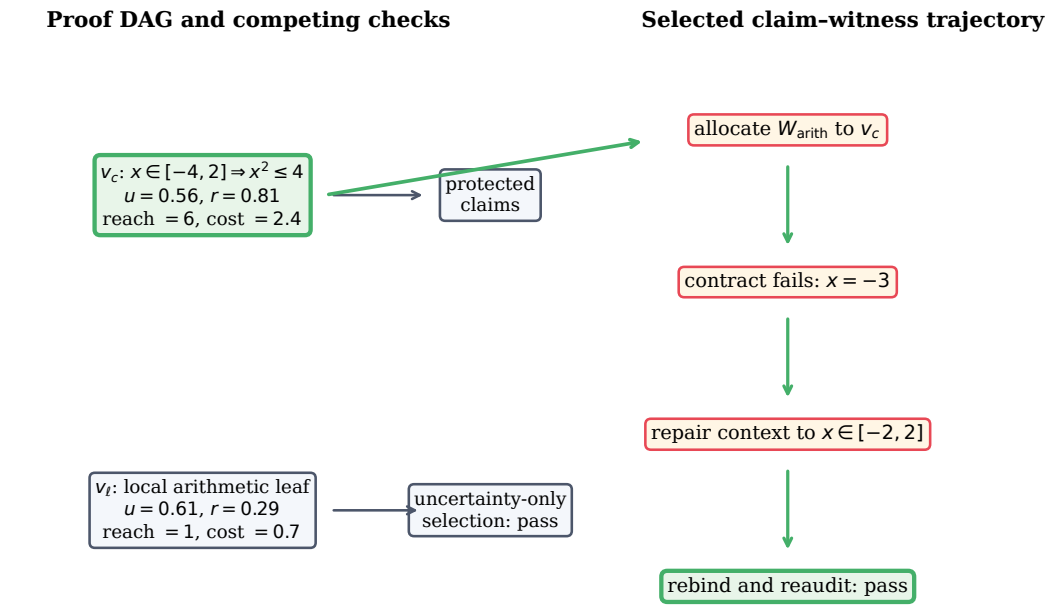


Figure 1: One allocation decision on a proof dependency graph. Uncertainty alone prioritizes the cheaper leaf, whereas WITALLOC values the selected node's risk, witness fidelity, inclusive reach, and cost. The arithmetic contract exposes a counterexample, identifies the affected dependency cone, and rebinds the repaired node to a mandatory typed reaudit.

Why pair selection changes the decision. Figure 1 contrasts two miniF2F nodes with similar uncertainty. The central arithmetic node has uncertainty 0.56, risk 0.81, inclusive reach six, and cost 2.4; a cheap leaf has uncertainty 0.61, risk 0.29, reach one, and cost 0.7. Uncertainty selects the leaf, which passes. Equation 4 selects the higher-impact pair; W_{arith} finds the counterexample $x = -3$ after the generated trace relaxes an inherited $x \geq -2$ premise to $x \geq -4$. Repair restores the ancestor-derived bound without changing the root goal, rebuilds the dependency cone, and passes the rebound arithmetic contract.

5 EXPERIMENTS

Three questions determine the evidence order. Does dependency-aware ordering improve outcomes when the typed witness inventory is identical? Does semantic binding change what an executed local check establishes? Do the resulting localization and repair gains survive budget changes, naturally occurring failures, held-out fault families, and a second generator?

5.1 BENCHMARKS, MODELS, AND SPLITS

The evaluation covers all 488 miniF2F problems (Zheng et al., 2022), 1,142 SV-COMP verification tasks (Beyer, 2022), and 374 MATH combinatorics problems (Hendrycks et al., 2021). Official theorem and program identities define miniF2F and SV-COMP evaluation units; the MATH subset is selected by the published category label and deduplicated by normalized problem text. No theorem, program, or source-proof identity crosses a training, development, or evaluation boundary.

The common generator is a frozen Llama-3.1-70B checkpoint (Grattafiori et al., 2024). Lean goals and tactic states, SV-COMP C programs with verification conditions, and MATH natural-language problems with explicit finite-domain traces use domain-native input and output adapters; generator parameters remain unchanged across domains and methods. DeepSeek-Prover-V1.5 supplies a second-backbone transfer study (Xin et al., 2025). Each trained method receives at most 640 A100-hours of comparator-specific development; evaluation uses eight 80GB NVIDIA A100 accelerators with PyTorch 2.3.1 on CUDA 12.4.

Risk training derives 15,000 controlled-error DAGs from 2,400 correct source proofs. The source-proof split is 1680/360/360; the derived graph split is 10500/2250/2250. These 2,250 evaluation DAGs are model-evaluation units. Headline false-acceptance denominators instead count problem–seed attempts after the frozen corruption sampler and domain eligibility filters, yielding 2,440, 5,710, and 1,870 attempts; they therefore do not represent another partition of the 2,250 DAGs. Alignment uses 8,000 claim–witness groups, split 5600/1200/1200 by source identity and stratified over the five compiler families. Each evaluation corruption modifies one eligible intermediate node; corruption family and severity are sampled independently, and the held-out-family study removes the evaluated family from both risk and alignment training.

5.2 Matched comparison and cost accounting

The controlled comparison uses three alternatives that share WitAlloc's frozen checkpoint, tokenizer, prompts, candidate traces, decoding, generated-token cap, repair controller, executor pool, and timeout policy. Every method receives the same budgeted whole-trace self-revision before its method-specific intervention. *Generator only* has generic repair but no audit module. *Uniform typed* samples without replacement from the same affordable claim–witness pairs and invokes the same typed diagnostics and repair logic as WitAlloc; this is the placement control. *Learned verifier* is a 355M-parameter three-layer transformer trained on the same source-identity split as the risk model, temperature-calibrated on development data, and connected to the shared repair controller after a flagged step. It supplies the flagged node and its serialization, whereas a typed failure additionally supplies its bound counterexample or checker diagnostic. This comparison therefore measures the complete executable-evidence interface, not placement alone.

The free-form factorial controls receive the same serialized claim and context and may synthesize a check using the common executor pool, but receive neither the semantic signature nor the family-specific compatibility gate. Their uniform variant samples affordable nodes, and their risk-directed variant ranks generic node checks by predicted risk, inclusive reach, and estimated execution cost. They share the resource meter, repair controller, caps, and four outcome states with the typed rows. Uniform typed versus WitAlloc holds the witness interfaces fixed to isolate placement. Free-form versus typed tests the complete semantic-interface change—signature, compatibility gate, and typed compiler—rather than attributing the difference to alignment alone.

On miniF2F, Lean kernel verification of a completed theorem remains the exact endpoint certification mechanism. The local-audit layer is evaluated for its additional diagnosis and repair yield before that endpoint: failed contracts expose a repair location and diagnostic under the shared cap. SV-COMP property verdicts and MATH exact answers play the analogous evaluation-only labeling role, keeping selection and repair blind to final correctness in every domain.

We distinguish the generated-token cap, permitted audit/tool cap, and realized end-to-end inference cost. Every problem receives at most 16,384 generated tokens across initial generation, the common whole-trace revision, and localized repair. The resource meter converts summed time to A100-equivalent seconds as

$$c = t_{\mathrm{A100}} + t_{\mathrm{CPU}}/32 + t_{\mathrm{exec}}/8,$$

with coefficients frozen from development throughput; parallel calls count summed rather than wall-clock time. Cost includes generation, risk or verifier scoring, matching, witness compilation, execution, repair, retry, and timeout, then normalizes by the generator-only mean. The main normalized end-to-end cap is $C = 1.55$ and the sweep uses $C \in \{1.10, 1.20, 1.30, 1.40, 1.55\}$. After fixed charges, the remaining amount defines B in Eq. 1; $\kappa(v, W_j)$ is the frozen development 95th-percentile compile-and-execute charge. Realized charges are debited after each action. If variability exhausts the residual cap before a selected outcome or mandatory reaudit resolves, that outcome is unresolved

and the candidate is rejected. Table 9 therefore labels normalized end-to-end caps, while methods may finish below a cap.

5.3 Evaluation protocol

Main results use seeds 17, 29, 41, 53, and 67. We report three distinct populations rather than combining their denominators. On unmodified benchmark problems, *accepted-correct rate* is the fraction whose final output both satisfies the runtime acceptance predicate and passes the benchmark-native oracle; a correct output rejected by an unresolved check is not accepted. Realized cost uses the same unmodified stream. On controlled corruptions, *false acceptance* is the fraction that satisfies runtime acceptance but whose final post-repair output is incorrect under the evaluation-only oracle. Natural-error cohorts, described next, condition on unmodified generator failures and measure accepted-wrong and repaired-to-correct outcomes. The oracle is queried only after each method terminates, never for selection, repair, or runtime acceptance. We report problem-and-seed clustered 95% bootstrap intervals for false acceptance and paired deltas, and seed dispersion for accepted-correct rate and realized cost.

Natural errors are collected prospectively from the frozen checkpoint before any method outcome is observed. An attempt enters the cohort when the native oracle rejects its unmodified result; duplicate final traces from the same problem and seed are collapsed before auditing. The original 96 miniF2F and 228 SV-COMP attempts are retained, then extended with disjoint problem–seed cohorts and 312 MATH-Combinatorics attempts. All methods receive identical attempts and acceptance, repair, timeout, and cost rules.

6 Results

6.1 Mechanism isolation under matched resources

Table 1: Selection and semantic-interface decomposition on miniF2F. Rows share candidate nodes, executor pool, repair controller, checkpoint, and caps. CE/TO denominators are audited or scored claims: 8,846 / 8,212 / 9,077 / 8,174 / 30,256 in row order. Repairs are successful repairs across 2,440 problem–seed attempts.

Selection and semantics	Accepted correct	False acc.	Cost	CE / TO	Repairs
Uniform + free-form	40.3	25.1	1.48	6.7/4.8	64
Risk-directed + free-form	41.6	23.4	1.42	6.2/4.4	78
Uniform + typed	41.0	20.3	1.51	2.9/3.1	88
Risk-directed + typed	43.1	16.8	1.38	2.4/2.7	112
Learned verifier	42.4	18.9	1.41	0.0/1.1	86

The factorial separates two decisions that a scalar verifier merges. Risk-directed ordering improves the free-form interface, and binding checks through the complete typed interface improves both ordering policies. Most importantly, uniform typed and WitAlloc expose the same claim–witness pairs and diagnostics; changing only their order yields +2.1 [+1.4,+2.8] accepted-correct points, -3.5 [-4.7,-2.3] false-acceptance points, and -0.13 [-0.18,-0.09] normalized cost. This is the direct evidence that dependency-aware placement matters. The free-form/typed contrast estimates the semantic-interface bundle rather than alignment alone, and Appendix F reproduces both effects on the other domains.

6.2 Reliability–utility across caps

WitAlloc moves both axes of the frontier on every domain. Against the learned verifier, paired accepted-correct differences are miniF2F +0.7 [+0.2,+1.2], SV-COMP +0.9 [+0.3,+1.5], and MATH-Combinatorics +0.9 [+0.2,+1.6]; false-acceptance differences are -2.1 [-3.2,-1.0], -2.0 [-2.9,-1.1], and -2.1 [-3.5,-0.7] points. Normalized realized cost, which accounts for both methods' scoring as well as downstream execution and repair, is lower by 0.03–0.04. MATH-Combinatorics contributes

Table 2: Same-checkpoint frontier under matched caps. Accepted-correct rate and cost use unmodified benchmark attempts; false acceptance uses controlled corruptions (2,440 / 5,710 / 1,870 problem–seed attempts). Rates and cost report mean ± seed SD where shown; brackets are problem-and-seed clustered 95% intervals. Realized cost includes scoring, matching, compilation, execution, repair, retries, and timeouts.

Benchmark	Method	Accepted correct	False acceptance	Cost
miniF2F	Generator only	39.7±0.8	700/2440, 28.7 [26.9, 30.5]	1.00±0.02
	Uniform typed	41.0±0.9	495/2440, 20.3 [18.7, 21.9]	1.51±0.05
	Learned verifier	42.4±0.6	461/2440, 18.9 [17.4, 20.5]	1.41±0.04
	WitAlloc	43.1±0.7	410/2440, 16.8 [15.4, 18.3]	1.38±0.04
SV-COMP	Generator only	61.3±0.9	1279/5710, 22.4 [21.3, 23.5]	1.00±0.02
	Uniform typed	63.1±0.8	959/5710, 16.8 [15.8, 17.8]	1.47±0.04
	Learned verifier	64.3±0.7	925/5710, 16.2 [15.3, 17.2]	1.39±0.04
	WitAlloc	65.2±0.8	811/5710, 14.2 [13.3, 15.1]	1.35±0.03
MATH-Comb.	Generator only	48.2±1.0	589/1870, 31.5 [29.4, 33.6]	1.00±0.02
	Uniform typed	50.4±1.0	423/1870, 22.6 [20.8, 24.6]	1.50±0.05
	Learned verifier	51.7±0.7	383/1870, 20.5 [18.7, 22.4]	1.45±0.04
	WitAlloc	52.6±0.8	344/1870, 18.4 [16.7, 20.2]	1.42±0.04

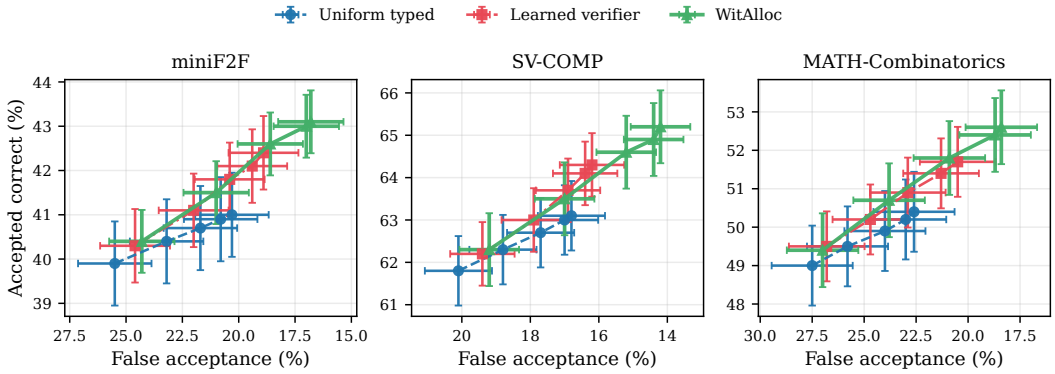


Figure 2: Reliability–utility frontiers across end-to-end caps. Each point is a five-seed aggregate; error bars are problem-and-seed clustered 95% intervals. WitAlloc separates from uniform typed placement and learned verification through the middle range; MATH-Combinatorics crosses at the smallest cap and all methods flatten at the largest.

the largest generator-only shift (+4.4 accepted-correct and −13.1 false-acceptance points), while SV-COMP provides the most direct benchmark-native contract semantics.

Table 3: Held-out natural errors. Cells report accepted wrong and repaired-to-correct counts with rates; every method receives the same pooled cohort.

Method	miniF2F ($N = 340$)	SV-COMP ($N = 714$)	MATH-Comb. ($N = 312$)
Generator only	78/340 (22.9%)	135/714 (18.9%)	79/312 (25.3%)
	34/340 (10.0%)	60/714 (8.4%)	25/312 (8.0%)
Uniform typed	66/340 (19.4%)	119/714 (16.7%)	64/312 (20.5%)
	47/340 (13.8%)	80/714 (11.2%)	38/312 (12.2%)
Learned verifier	60/340 (17.6%)	109/714 (15.3%)	58/312 (18.6%)
	55/340 (16.2%)	88/714 (12.3%)	43/312 (13.8%)
WITALLOC	40/340 (11.8%)	76/714 (10.6%)	39/312 (12.5%)
	73/340 (21.5%)	117/714 (16.4%)	60/312 (19.2%)

Figure 2 shows where the pair-valued action matters. At the smallest cap, all methods can afford only a few interventions and MATH-Combinatorics slightly favors the learned verifier. Between 1.20 and 1.40×, typed dependency-aware placement removes more accepted errors while retaining more accepted-correct outputs. Once high-risk auditable nodes are covered, additional budget has diminishing value and the curves flatten.

6.3 PROSPECTIVE, HELD-OUT, AND ADAPTIVE ERRORS

On unmodified prover failures, WITALLOC accepts 4.6–6.1 points fewer wrong attempts and repairs 4.1–5.4 points more than the learned verifier. Paired accepted-wrong differences are -5.9 [-9.6,-2.3], -4.6 [-7.3,-1.9], and -6.1 [-10.8,-1.5]; paired repaired-to-correct differences are +5.3 [+1.4,+9.1], +4.1 [+1.5,+6.7], and +5.4 [+1.0,+9.8]. The original cohorts and disjoint extensions in Appendix E retain the direction independently.

Node-risk localization remains 8–10 F1 points above the learned verifier and 22–28 points above uncertainty or centrality under held-out families and natural errors (Appendix G). Labels are fixed before execution from the earliest dependency-local contradiction. The MATH natural row uses adapter replay tied to its exact-answer oracle and is interpreted as ranking evidence; miniF2F and SV-COMP provide native proof or obligation semantics.

WITALLOC also has lower descriptive attack success than the learned verifier in every compound, fan-out-evasion, and budget-exhaustion condition (Appendix H); the hardest miniF2F condition is budget exhaustion, at 14.4 versus 22.2 false pass. Coverage localizes the benefit: high-coverage strata gain 5.8–6.5 accepted-correct points and remove 10.1–14.6 false-acceptance points over generator only, while low-coverage gains are 0.7–1.1 and 1.9–2.8 points (Appendix J).

7 DISCUSSION AND CONCLUSION

Uniform typed placement shows that compiler order changes correctness, false acceptance, and cost; the semantics factorial shows why each selected check must remain bound to its quantifiers, assumptions, variables, and domains. Natural-error cohorts further show that failed local contracts drive successful repair rather than rejection alone.

The evaluated regime has explicit local contexts and compatible witnesses: high coverage creates the most valuable choices, while sparse coverage leaves more unaudited uncertainty. Exact endpoint checking remains complementary; MATH natural localization is interpreted through adapter replay; and unresolved outcomes remain fail-closed.

By committing target and executable semantics together through dependency-local reconstruction, WITALLOC turns scarce checking into more correct repaired outputs and fewer accepted errors.

AI USE STATEMENT

Generative AI was used only for manuscript writing, editing, and presentation. The authors reviewed all edits and remain responsible for the final content.

REFERENCES

Dirk Beyer. Progress on software verification: SV-COMP 2022. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 13244 of *Lecture Notes in Computer Science*, pp. 375–402. Springer, 2022. doi: 10.1007/978-3-030-99527-0_20. URL https://doi.org/10.1007/978-3-030-99527-0_20.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=YfZ4ZPt8zd.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Łukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. doi: 10.48550/arXiv.2110.14168. URL https://arxiv.org/abs/2110.14168.

Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pp. 337–340. Springer, 2008. doi: 10.1007/978-3-540-78800-3_24. URL https://doi.org/10.1007/978-3-540-78800-3_24.

Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean theorem prover (system description). In *Automated Deduction – CADE-25*, volume 9195 of *Lecture Notes in Computer Science*, pp. 378–388. Springer, 2015. doi: 10.1007/978-3-319-21401-6_26. URL https://doi.org/10.1007/978-3-319-21401-6_26.

Sajib Acharjee Dip, Dawei Zhou, and Liqing Zhang. Think again or think longer? selective verification for budget-aware reasoning. *arXiv preprint arXiv:2606.19808*, 2026. doi: 10.48550/arXiv.2606.19808. URL https://arxiv.org/abs/2606.19808.

Jean-Christophe Filliâtre and Andrei Paskevich. Why3 – where programs meet provers. In *Programming Languages and Systems*, volume 7792 of *Lecture Notes in Computer Science*, pp. 125–128. Springer, 2013. doi: 10.1007/978-3-642-37036-6_8. URL https://doi.org/10.1007/978-3-642-37036-6_8.

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

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799. PMLR, 2023. URL https://proceedings.mlr.press/v202/gao23f.html.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. ToRA: A tool-integrated reasoning agent for mathematical problem solving. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Ep0TtjVoap.

Aaron Grattafiori, Abhimanyu Dubey, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. doi: 10.48550/arXiv.2407.21783. URL https://arxiv.org/abs/2407.21783.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In Joaquin Vanschoren and Sai-Kit Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1. Curran Associates, Inc., 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Albert Q. Jiang, Wenda Li, Szymon Tworkowski, Konrad Czechowski, Tomasz Odrzygóźdź, Piotr Miłoś, Yuhuai Wu, and Mateja Jamnik. THOR: Wielding hammers to integrate language models and automated theorem provers. In *Advances in Neural Information Processing Systems*, volume 35, pp. 8360–8373. Curran Associates, Inc., 2022. doi: 10.52202/068431-0608. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/377c25312668e48f2e531e2f2c422483-Abstract-Conference.html.

Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, Sketch, and Prove: Guiding formal theorem provers with informal proofs. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=SMa9EAovKMC.

Guillaume Lample, Timothée Lacroix, Marie-Anne Lachaux, Aurélien Rodriguez, Amaury Hayat, Thibaut Lavril, Gabriel Ebner, and Xavier Martinet. HyperTree proof search for neural theorem proving. In *Advances in Neural Information Processing Systems*, volume 35, pp. 26337–26349. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/a8901c5e85fb8e1823bbf0f755053672-Abstract-Conference.html.

K. Rustan M. Leino. Dafny: An automatic program verifier for functional correctness. In *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 6355 of *Lecture Notes in Computer Science*, pp. 348–370. Springer, 2010. doi: 10.1007/978-3-642-17511-4_20. URL https://doi.org/10.1007/978-3-642-17511-4_20.

Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009. doi: 10.1145/1538788.1538814. URL https://doi.org/10.1145/1538788.1538814.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's Verify Step by Step. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

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

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

George C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 106–119, 1997. doi: 10.1145/263699.263712. URL https://doi.org/10.1145/263699.263712.

Azim Ospanov, Farzan Farnia, and Roozbeh Yousefzadeh. APOLLO: Automated LLM and Lean collaboration for advanced formal reasoning. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.48550/arXiv.2505.05758. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/3b77109ad4dd4ba82d07cacd4b24207e-Abstract-Conference.html.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020. doi: 10.48550/arXiv.2009.03393. URL https://arxiv.org/abs/2009.03393.

Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. Formal mathematics statement curriculum learning. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=-P7G-8dmSh4.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Zeen Song, Wenwen Qiang, Siyu Zhao, Changwen Zheng, and Gang Hua. Reward model generalization for compute-aware test-time reasoning. *arXiv preprint arXiv:2505.18065*, 2025. doi: 10.48550/arXiv.2505.18065. URL https://arxiv.org/abs/2505.18065.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022. doi: 10.48550/arXiv.2211.14275. URL https://arxiv.org/abs/2211.14275.

Haiming Wang, Huajian Xin, Chuanyang Zheng, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, Zhenguo Li, and Xiaodan Liang. LEGO-Prover: Neural theorem proving with growing libraries. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=3f5PALef5B.

Ziyu Wang, Qiming Dai, Yishan Wu, and Zaiwen Wen. FaithSieve: Fine-grained evaluation of math proofs with faithful formal evidence. *arXiv preprint arXiv:2608.26310*, 2026. doi: 10.48550/arXiv.2608.26310. URL https://arxiv.org/abs/2608.26310.

Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. Deepseek-prover: Advancing theorem proving in LLMs through large-scale synthetic data. *arXiv preprint arXiv:2405.14333*, 2024. doi: 10.48550/arXiv.2405.14333. URL https://arxiv.org/abs/2405.14333.

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

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J. Prenger, and Animashree Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 21573–21612. Curran Associates, Inc., 2023. doi: 10.52202/075280-0944. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/4441469427094f8873d0fecb0c4e1cee-Abstract-Datasets_and_Benchmarks.html.

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

A Implementation and Protocol Details

Training. The node-risk GNN uses three message-passing layers with 256-dimensional states and AdamW (Loshchilov & Hutter, 2019) at learning rate 5×10^{-4} for 50 epochs. Its node input is the deterministic serialization of $(\phi_v, \Gamma_v, \sigma_v, u_v, \rho_G(v))$ used throughout the paper, and message passing adds the directed support neighborhood. It predicts the sampler-designated source node rather than every descendant invalidated by that source. Alignment training runs for 20 epochs with temperature 0.07 and five negative witnesses per positive. Early stopping and family-specific compatibility thresholds use source-identity-disjoint development graphs. The learned-verifier control is trained on the same source-node targets and split, then temperature calibrated and thresholded before test execution. The frozen generator is never updated by comparator-specific training.

Compiler behavior. W_{search} uses bounded Python enumeration with a ten-second per-node timeout; W_{arith} uses arbitrary-precision rationals and outward-rounded intervals; W_{rewrite} emits equational certificates checked by Lean; W_{test} compiles Dafny/C conditions into executable harnesses; and W_{tactic} invokes Lean on a single tactic-local judgment. Repeated witness attempts are disabled unless a repair changes the graph version. Repair allows at most 2 attempts per flagged node, and each attempt plus mandatory reaudit is debited from both the audit cap and realized cost.

Repair lifecycle. The shared controller receives the failed node, its bound context, the executable counterexample or checker diagnostic, and the affected dependency cone. A repair candidate is eligible only if it preserves the root goal and inherited ancestor context and passes parsing, typing, graph reconstruction, and signature extraction. The affected subgraph is then versioned, changed descendants return to `unaudited`, and the original witness family is rebound to the repaired node. No method can accept a repair that omits the resulting reaudit, exceeds the two-attempt limit, or runs out of residual budget.

Controlled-error sampler. Eligible nodes have a complete semantic signature and at least one compatible compiler. Each attempt changes one node through quantifier weakening, finite-bound relaxation, assumption deletion, type-compatible variable substitution, or subset-only checking. These are the same five families used for alignment negatives and the columns of Table 14. Severity is stratified by the number of affected descendants. Native benchmark oracles determine whether the resulting attempt is corrupted. Each held-out-family fold excludes that family from both risk and alignment training and evaluates it only after models and thresholds are frozen.

Natural-error collection. For every benchmark and seed, the frozen generator is run once under the main decoding cap. Native-oracle failures are logged before an audit method is assigned. Problem identity, normalized final trace, and semantic signature define deduplication. Method outcomes never influence cohort inclusion.

Domain adapters and oracle separation. Table 4 traces each native proof object to the common allocation interface. In every domain, compatibility is decided before execution and the native answer oracle remains unavailable until the method has terminated.

Table 4: Domain-specific construction of the shared claim–witness interface. The listed oracles label evaluation outcomes only; they do not select nodes, contracts, or repairs.

Domain	Claim, context, and edge source	Compatible evidence	Evaluation-only oracle
miniF2F	Lean goal/tactic state; local hypotheses and consumed subgoals define context and support	Local tactic, rewrite, arithmetic, and bounded-search contracts when the signature is complete	Lean kernel acceptance of the final theorem
SV-COMP	Verification conditions; path conditions, invariants, and obligation dependence	Program tests, arithmetic, rewrite, and bounded-search contracts	Benchmark-native property verdict
MATH-Comb.	Explicit finite-domain, arithmetic, and rewrite claims; references to prior claims define support	Bounded search, exact arithmetic, and checked rewrite contracts	Exact normalized final answer

Stress-test map. Headline utility and controlled-error reliability use separate evaluation streams. Table 5 records how the secondary stress tests probe binding, placement, and cap enforcement without changing those estimands.

Table 5: Stress boundaries and supporting evidence. Verdicts come from typed compilers, cap enforcement from the resource meter, and labels from the evaluation oracle after execution.

Protected object	Stress capability	Comparator / setting	Evidence
Quantifiers	Quantifier weakening	No alignment; held-out	Tables 16, 15
Numeric bounds	Bound relaxation	No alignment; compound	Tables 16, 15
Assumptions	Assumption deletion	Learned verifier; compound	Tables 16, 15
Variable identity	Typed-variable swap	Learned verifier; held-out	Tables 16, 14
Conjunct coverage	Subset-only check	No alignment; held-out	Tables 16, 14
Node selection	High-fan-out evasion	Learned verifier; adaptive	Table 15
Resource cap	Timeout / exhaustion	Learned verifier; adaptive	Table 15
Binding and selection	Unseen compound	No alignment; compound	Table 15

The adaptive rows quantify robustness through residual error when the error source, target, or compiler latency shifts, directly testing whether the selected contract remains informative outside the training mixture.

B Positioning by Technical Interface

Table 6: Evidence interfaces of adjacent verification families. The distinguishing conjunction evaluated here is a bound local contract, dependency-aware selection, and joint budget–repair coupling.

Family	Checked object	Evidence interface	Dependency-aware selection	Budget / repair coupling
PRMs	Step plausibility	Unbound score	Sometimes	Search budget
Learned step verifier	Intermediate step	Unbound score or label	Sometimes	Optional repair
Verifier / reranker	Final proof or sample	Score or verdict	No	Candidate budget
Free-form tool agents	Ad hoc computation	Model-chosen tool output	Model-directed	Tool-call budget
Certificate systems	Artifact or program	Bound checker verdict	Usually exhaustive	Workflow-specific
WitAlloc	Typed proof-DAG claim	Bound four-state execution	Yes	Joint allocation and repair

Unlike adjacent interfaces, WitAlloc binds executable local evidence, dependency-aware placement, and repair to the same resource decision.

C SEED-LEVEL STABILITY

Table 7: Accepted-correct stability for seeds 17, 29, and 41. The main five-seed aggregates additionally include the prespecified runs in Table 8; Student-t intervals here use two degrees of freedom.

Benchmark	Method	Seed 17	Seed 29	Seed 41	Mean ± SD	95% interval
miniF2F	Generator only	39.2	39.8	40.1	39.7±0.5	[38.6,40.8]
	WITALLOC	42.6	43.3	43.4	43.1±0.4	[42.0,44.2]
SV-COMP	Generator only	61.0	61.3	61.6	61.3±0.3	[60.6,62.0]
	WITALLOC	64.8	65.4	65.4	65.2±0.3	[64.3,66.1]
MATH-Comb.	Generator only	47.7	48.5	48.4	48.2±0.4	[47.1,49.3]
	WITALLOC	52.4	52.8	52.6	52.6±0.2	[52.1,53.1]

Table 8: Prespecified added seeds. Each cell is accepted-correct rate / accepted corruptions / realized cost. Denominators per seed are 488, 1,142, and 374.

Benchmark	Method	Seed 53	Seed 67
miniF2F	Generator only	38.7/135/0.98	40.9/145/1.02
	Uniform typed	40.1/96/1.45	41.9/102/1.54
	Learned verifier	41.8/93/1.36	43.0/91/1.46
	WITALLOC	42.2/80/1.33	44.0/84/1.43
SV-COMP	Generator only	60.1/250/0.98	62.5/262/1.02
	Uniform typed	62.3/189/1.42	63.9/194/1.52
	Learned verifier	63.6/181/1.34	65.0/189/1.44
	WITALLOC	64.1/158/1.31	66.3/167/1.39
MATH-Comb.	Generator only	46.9/114/0.98	49.5/122/1.02
	Uniform typed	49.4/81/1.44	51.4/88/1.53
	Learned verifier	51.0/74/1.40	52.4/79/1.50
	WITALLOC	51.5/66/1.37	53.7/72/1.47

D CROSS-BUDGET FRONTIER VALUES

Table 9: Five-seed cross-budget frontier over normalized end-to-end caps. Each cell is accepted-correct rate / false acceptance / realized cost; raw false-acceptance denominators are 2,440, 5,710, and 1,870.

Benchmark	Cap	Uniform typed	Learned verifier	WITALLOC
miniF2F	1.10	39.9/25.5/1.09	40.3/24.6/1.08	40.4/24.3/1.08
	1.20	40.4/23.2/1.18	41.1/22.0/1.17	41.5/21.0/1.16
	1.30	40.7/21.7/1.28	41.8/20.4/1.26	42.6/18.6/1.24
	1.40	40.9/20.8/1.38	42.1/19.4/1.34	43.0/17.0/1.32
	1.55	41.0/20.3/1.51	42.4/18.9/1.41	43.1/16.8/1.38
SV-COMP	1.10	61.8/20.1/1.08	62.2/19.4/1.08	62.3/19.2/1.07
	1.20	62.3/18.8/1.18	63.0/17.9/1.17	63.5/17.0/1.16
	1.30	62.7/17.7/1.28	63.7/16.9/1.26	64.6/15.2/1.24
	1.40	63.0/17.0/1.38	64.1/16.4/1.34	64.9/14.4/1.32
	1.55	63.1/16.8/1.47	64.3/16.2/1.39	65.2/14.2/1.35
MATH-Comb.	1.10	49.0/27.5/1.09	49.5/26.8/1.09	49.4/27.0/1.08
	1.20	49.5/25.8/1.18	50.2/24.7/1.18	50.7/23.8/1.16
	1.30	49.9/24.0/1.28	50.9/22.9/1.26	51.8/20.9/1.24
	1.40	50.2/23.0/1.38	51.4/21.3/1.34	52.4/18.7/1.32
	1.55	50.4/22.6/1.50	51.7/20.5/1.45	52.6/18.4/1.42

E NATURAL-ERROR COHORTS

Table 10: Original held-out natural-error cohorts, retained separately from the extension. Each entry is count / cohort size, rate, and 95% bootstrap interval.

Benchmark	Method	Accepted wrong	Repaired to correct
miniF2F	Generator only	23/96, 24.0 [15.6,32.4]	9/96, 9.4 [4.7,14.2]
	Uniform typed	20/96, 20.8 [13.9,30.0]	13/96, 13.5 [8.1,21.8]
	Learned verifier	18/96, 18.8 [12.3,25.0]	15/96, 15.6 [9.2,22.1]
	WITALLOC	11/96, 11.5 [6.6,16.3]	21/96, 21.9 [14.3,29.3]
SV-COMP	Generator only	44/228, 19.3 [14.1,24.4]	18/228, 7.9 [4.4,11.5]
	Uniform typed	39/228, 17.1 [12.8,22.5]	24/228, 10.5 [7.2,15.2]
	Learned verifier	36/228, 15.8 [11.1,20.6]	27/228, 11.8 [7.8,15.8]
	WITALLOC	25/228, 11.0 [7.0,15.1]	37/228, 16.2 [11.5,21.0]

Table 11: Disjoint extension cohorts. Each entry is accepted wrong / repaired to correct; rates appear in Table 3.

Method	miniF2F ($N = 244$)	SV-COMP ($N = 486$)	MATH-Comb. ($N = 312$)
Generator only	55/25	91/42	79/25
Uniform typed	46/34	80/56	64/38
Learned verifier	42/40	73/61	58/43
WITALLOC	29/52	51/80	39/60

Realized costs for miniF2F / SV-COMP / MATH-Combinatorics are 1.00/1.00/1.00 for generator only, 1.48/1.46/1.49 for uniform typed, 1.40/1.38/1.44 for the learned verifier, and 1.37/1.34/1.41 for WITALLOC.

F FACTORIAL AND COVERAGE-CONDITIONED RESULTS

Table 12: Placement–semantics decomposition on SV-COMP and MATH-Combinatorics. Entries are accepted-correct rate / false acceptance / cost / successful repairs.

Configuration	SV-COMP	MATH-Combinatorics
Uniform + free-form	62.5/20.7/1.45/123	49.6/27.8/1.52/49
Risk-directed + free-form	63.8/19.2/1.39/149	51.2/25.9/1.46/63
Uniform + typed	63.1/16.8/1.47/167	50.4/22.6/1.50/74
Risk-directed + typed	65.2/14.2/1.35/221	52.6/18.4/1.42/98
Learned verifier	64.3/16.2/1.39/183	51.7/20.5/1.45/81

Against uniform typed placement, the paired accepted-correct / false-acceptance / cost effects are +2.1 [+1.5,+2.7] / -2.6 [-3.4,-1.8] / -0.12 [-0.16,-0.08] on SV-COMP and +2.2 [+1.2,+3.2] / -4.2 [-5.8,-2.6] / -0.08 [-0.13,-0.03] on MATH-Combinatorics. Intervals use the same problem-and-seed clustered bootstrap as the main comparison.

Cells in Table 13 macro-average seed-specific stratum rates; Table 18 reports pooled problem–seed stratum sizes. Because stratum composition varies by seed, weighting the displayed macro-averages by the pooled sizes is not the main-table estimator.

Table 13: Coverage-conditioned comparison under development-fixed auditable-share strata. Each cell is accepted-correct / false acceptance.

Benchmark	Coverage	Uniform typed	Learned verifier	WITALLOC
miniF2F	High	41.5/20.8	42.8/19.1	43.8/16.6
	Mid	41.3/18.7	42.3/18.3	42.9/17.1
	Low	41.5/19.4	41.7/18.5	41.9/18.5
SV-COMP	High	63.9/16.8	65.2/15.9	66.0/13.2
	Mid	63.8/15.9	64.8/15.6	65.4/14.2
	Low	63.0/19.4	63.5/19.2	63.8/18.8
MATH-Comb.	High	50.5/22.8	52.4/20.3	53.2/18.2
	Mid	50.8/22.3	52.2/21.3	52.8/18.9
	Low	50.4/20.8	50.7/20.8	51.0/19.8

G HELD-OUT LOCALIZATION DETAILS

Table 14: Leave-one-corruption-family-out node-risk F1. Natural-error values use unmodified prover failures.

Domain	Method	Quant.	Bound	Assump.	Variable	Subset	Natural
miniF2F	Learned verifier	.50	.56	.49	.54	.55	.49
	Node risk	.61	.67	.60	.65	.66	.58
SV-COMP	Learned verifier	.47	.53	.46	.51	.52	.46
	Node risk	.57	.62	.56	.60	.61	.55
MATH-Comb.	Learned verifier	.44	.50	.43	.49	.51	.43
	Node risk	.55	.59	.53	.58	.60	.52

H ADAPTIVE AUDIT ATTACKS

Table 15: Held-out adaptive attacks. Each cell is attack success numerator / denominator and rate. Compound attacks combine two corruption families absent as a pair from training; fan-out attacks target high-descendant nodes; budget attacks force slow compatible compilers.

Domain	Attack	No alignment	Learned verifier	WITALLOC
miniF2F	Compound	78/360, 21.7	70/360, 19.4	39/360, 10.8
	Fan-out evasion	69/360, 19.2	65/360, 18.1	31/360, 8.6
	Budget exhaustion	92/360, 25.6	80/360, 22.2	52/360, 14.4
SV-COMP	Compound	96/540, 17.8	88/540, 16.3	51/540, 9.4
	Fan-out evasion	90/540, 16.7	84/540, 15.6	42/540, 7.8
	Budget exhaustion	122/540, 22.6	111/540, 20.6	70/540, 13.0
MATH-Comb.	Compound	75/300, 25.0	68/300, 22.7	37/300, 12.3
	Fan-out evasion	67/300, 22.3	62/300, 20.7	30/300, 10.0
	Budget exhaustion	88/300, 29.3	80/300, 26.7	48/300, 16.0

On miniF2F compound attacks, WITALLOC localizes 0.68 of corrupted nodes, retains 42.5% accepted-correct rate at 1.40× cost, and records 2.8% compile errors and 2.2% timeouts. The learned verifier localizes 0.55, reaches 41.7% accepted-correct, and costs 1.42×. Under budget exhaustion, timeout rates are 7.8% and 8.9%, respectively.

I WITNESS ALIGNMENT AND TRANSFER

Table 16: False-pass rate (%) before → after alignment training for quantifier weakening, bound relaxation, assumption deletion, variable substitution, and subset-only checking.

Witness	Quantifier	Bound	Assumption	Variable	Subset
W_{search}	17.4→8.9	14.2→7.5	19.1→9.8	12.6→6.4	16.3→8.1
W_{arith}	11.8→5.2	9.4→4.1	10.2→4.9	8.1→3.6	9.7→4.4
W_{rewrite}	15.2→8.0	13.1→6.9	14.6→7.7	11.4→5.8	12.9→6.5
W_{test}	13.5→6.8	11.2→5.5	12.8→6.2	9.9→4.8	10.6→5.1
W_{tactic}	8.2→3.9	6.5→3.1	7.4→3.5	5.8→2.7	6.1→2.9

Table 17: DeepSeek-Prover-V1.5 transfer. Each entry is accepted-correct / false acceptance / realized cost, using the same five seeds, task sets, and controlled-error denominators (2,440 / 5,710 / 1,870) as the main comparison.

Method	miniF2F	SV-COMP	MATH-Comb.
Text-only	42.1/24.1/1.00	64.5/19.5/1.00	51.7/28.4/1.00
Uniform typed	43.7/18.0/1.47	66.2/15.7/1.43	53.8/21.6/1.46
Learned verifier	43.9/17.1/1.43	66.4/15.0/1.40	54.0/20.9/1.42
WITALLOC	44.3/15.4/1.34	67.0/13.6/1.31	54.5/19.8/1.37

J COVERAGE AND SCALING

Table 18: Witness coverage and gains relative to generator only. Gain entries are accepted-correct / false-acceptance changes for high, middle, and low coverage strata. Stratum sizes count problem–seed attempts assigned by development-fixed auditable-share thresholds.

Benchmark	Auditable / selected	High / middle / low gain	Stratum sizes
miniF2F	0.64/0.27	+5.8/-13.4; +3.1/-8.6; +0.7/-1.9	1834/498/108
SV-COMP	0.79/0.24	+6.2/-10.1; +4.0/-7.4; +1.1/-2.6	3014/1999/697
MATH-Comb.	0.71/0.31	+6.5/-14.6; +4.2/-10.1; +0.9/-2.8	1436/328/106

For $|V| > 20$, localization F1 is 0.69 versus 0.72 overall. Heap-based allocation costs $O(|V||\mathcal{W}| \log |V|)$. The largest evaluated DAG has $|V| = 47$, and pair allocation takes 0.3 seconds. Combined with the coverage strata, this shows that contract availability, rather than allocation runtime, determines the evaluated operating region.

K ADDITIONAL MECHANISM CASES

SV-COMP context binding. A loop state carries $0 \le i < n$, $n \le \text{len}$, and transition $i' = i + 1$; the local claim requires $i' < n$. A free-form check substitutes the buffer length and validates only $i' \le \text{len}$, which passes at $i = n - 1$. The typed W_{test} contract preserves n, fails on that boundary state, and repairs the loop guard to maintain $i < n - 1$ before the transition; the rebound claim $i' < n$ then passes.

Low-coverage boundary. In a MATH-Combinatorics involution argument, the central parity node has no compatible witness, while a low-impact arithmetic leaf is auditable. The leaf passes, the central node remains unaudited, and no local repair is triggered. This case matches the near-parity between the learned verifier and WITALLOC in the low-coverage stratum.