FORGE: EVIDENCE-GOVERNED ALGORITHM DISCOVERY BY ATTRIBUTING FAILURES AND CERTIFYING INTERVENTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Algorithm-discovery agents traditionally iterate through execution, scoring, and revision; however, the research state guiding each revision remains either an unstructured narrative or a unidimensional ranking—a practice that systematically obscures failure origins, renders interventions non-falsifiable, and actively incentivizes evaluator overfitting. We introduce FORGE, an evidence-governed agent that maintains an append-only state comprising executable outcomes, counterexamples, failure taxonomies, resource consumption, and full provenance. At each iteration, FORGE isolates a specific failure mechanism, articulates a predicted behavioral shift, compiles a bounded candidate edit, subjects it to robustness testing without access to hidden labels, and accepts modifications only after deterministic validity and resource constraints are satisfied. Across a 640-cell Exp2 benchmark, FORGE attains the highest mean hidden score (0.910), outperforming AI-Scientist v1 (0.892), v2 (0.635), and OpenEvolve (0.765), with a $+0.017$ gain [95% CI: $+0.004$, $+0.030$] over the strongest baseline. In a comparator-network repair study, FORGE yields 26 verified repairs out of 40, compared to only 5 for baselines. Two exploratory transfer tasks further confirm generalization beyond synthetic search: in memory retrieval, FORGE achieves 0.742/0.751 vs. Full-Text (0.604/0.676) and NaiveRAG (0.650/0.719); in skill optimization, FORGE r55 attains 0.867/0.811 on SearchQA/SpreadsheetBench, surpassing SkillOpt (0.831/0.625) by $+0.036/+0.186$. These results position FORGE not merely as a stronger optimizer, but as a principled framework toward interpretable, auditable, and increasingly autonomous algorithmic discovery—with clear potential to scale across domains where reliable evidence, rather than heuristic ranking, becomes the cornerstone of intelligent search.

1 INTRODUCTION

Large language models now write competitive programs, execute experiments, and search algorithm spaces. AlphaCode, AlphaTensor, AlphaDev, FunSearch, and AlphaEvolve demonstrate that model-guided search can yield high-quality or even novel algorithms when a reliable evaluator exists Li et al. (2022); Fawzi et al. (2022); Mankowitz et al. (2023); Romera-Paredes et al. (2024); Balog et al. (2025). Research agents extend this paradigm from code to hypotheses and manuscripts Lu et al. (2024); Yamada et al. (2025). The opportunity is substantial, but so is a methodological mismatch: scientific search is inherently adaptive, yet its evidence is routinely compressed into self-reflection or a scalar leaderboard.

Repeated public evaluation can select programs that exploit a small development set, a verifier omission, or a runtime accident—a danger formalized by adaptive holdout research Dwork et al. (2015); Blum & Hardt (2015). A polished explanation does not identify which observation entailed a proposed edit; a high public score does not establish hidden semantics; a successful run does not guarantee replayability. These are not merely logging concerns—they determine the topology of the next search step.

We ask whether an algorithm-discovery agent can place evidence, rather than narrative or fitness, at the center of the research state. FORGE maintains structured records for every admitted and

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.

rejected candidate across three layers: (i) a method layer that attributes failures, states falsifiable interventions, and enforces bounded admission; (ii) a method-neutral harness that guarantees equal budgets, evaluator access, sealing, and replay; and (iii) an evidence layer containing public and hidden outcomes with their statistical scope. This separation is essential: improvements supplied by a shared harness are reproducibility controls, not algorithmic contributions.

Our contributions are:

- We formalize evidence-governed algorithm discovery: an append-only state of candidates, witnesses, failure types, resource ledgers, and provenance drives one typed, falsifiable intervention at a time.
- We introduce executable counterfactual probes and bounded admission. For a candidate fixed before m independent operational probes, zero failures yield the exact one-sided upper bound $p \leq 1 - \alpha^{1/m}$ at confidence $1 - \alpha$. We state assumptions and prove the bound; it is not a semantic-accuracy guarantee.
- We evaluate four agents under identical models, tasks, seeds, and caps. In Exp2—an independently generated 32-task study—task-level rather than cell-level inference yields positive paired intervals against every baseline.
- We demonstrate how attributed negative evidence constrains the permissible edit family in comparator-network repair, and how a local certificate converts an empirical edit into a function-preserving intervention.

2 Related Work

Program and algorithm discovery. Program synthesis maps specifications or examples to executable programs Gulwani et al. (2017); DreamCoder learns reusable libraries while synthesizing programs Ellis et al. (2021). Counterexample-guided inductive synthesis (CEGIS) alternates candidate generation with verification Solar-Lezama et al. (2006), while counterexample-guided abstraction refinement uses counterexamples to refine models Clarke et al. (2000). Contemporary systems add learned proposal distributions: AlphaTensor and AlphaDev search specialized formal spaces Fawzi et al. (2022); Mankowitz et al. (2023); FunSearch and AlphaEvolve combine LLM proposals with executable evaluators Romera-Paredes et al. (2024); Balog et al. (2025); OpenEvolve provides an open evolutionary coding implementation Sharma (2025). FORGE differs in what persists across iterations: not merely high-fitness programs, but attributed failures, disconfirming witnesses, operational risk, and provenance that actively constrain the subsequent edit family.

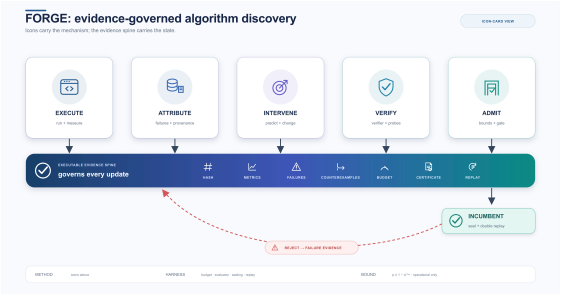


Figure 1: FORGE as an evidence-governed discovery loop. Execute, attribute, intervene, verify, and admit are coupled through an executable evidence spine containing hashes, metrics, failures, counterexamples, resource use, certificates, and replay records. Accepted candidates become sealed incumbents; rejected candidates return as failure evidence. The lower strip separates method logic from the matched harness and states the operational scope of the probe bound.

Language agents and self-improvement. ReAct interleaves reasoning with environment actions Yao et al. (2023b); Self-Refine and Reflexion reuse linguistic feedback Madaan et al. (2023); Shinn et al. (2023); Tree of Thoughts, LATS, and self-consistency expand test-time search Yao et al. (2023a); Zhou et al. (2023); Wang et al. (2023). AI-Scientist v1/v2 automate broader research loops Lu et al. (2024); Yamada et al. (2025). General agent benchmarks and software/ML engineering benchmarks expose planning and execution failures Liu et al. (2024); Mialon et al. (2024); Jimenez et al. (2024); Huang et al. (2023); Chan et al. (2024). These systems motivate strong baselines; our

question is complementary: how should executable evidence govern an adaptive research loop under a fixed budget?

Reliable evidence. Property-based and metamorphic testing generate checks beyond labeled examples Claessen & Hughes (2000); Segura et al. (2016). Proof-carrying code attaches machine-checkable safety evidence Necula (1997). Reproducibility programs stress artifact completeness and independent execution Pineau et al. (2021); content hashing supports tamper-evident binding Merkle (1988). FORGE adopts these ideas selectively: probes govern operational admission, local certificates constrain eligible edits, and hashes reveal mutation. None is presented as a proof that the public objective equals the intended scientific objective.

3 Problem Setting

A task τ contains an input contract $\mathcal{X}_\tau$, output validator V_τ, public examples $\mathcal{D}^p_\tau$, public evaluator S^p_τ, hidden examples $\mathcal{D}^h_\tau$, and hidden evaluator S^h_τ. During search an agent may execute candidates and observe only public outputs. Hidden labels and scores are unavailable until a final candidate is sealed. For method m, task t, and random seed r, let $Y_{mtr} \in [0, 1]$ be the hidden replay score, with any formal failure assigned its sealed score (zero where the task protocol specifies failure-as-zero).

The unit of generalization is a task, not a seed-cell. With $R = 5$ seeds, our primary estimand against baseline b is

$$\widehat{\Delta}_b = \frac{1}{T}\sum_{t=1}^{T}\left(\frac{1}{R}\sum_{r=1}^{R} Y_{\text{FORGE},tr} - \frac{1}{R}\sum_{r=1}^{R} Y_{btr}\right). \tag{1}$$

We therefore resample tasks for the paired bootstrap. Treating TR cells as independent would understate uncertainty because seeds share a task definition and evaluator.

4 FORGE

Figure 1 gives the end-to-end design. We describe method components first and the method-neutral harness second.

4.1 Attributed Evidence State

For candidate c_i, FORGE appends

$$e_i = (H(c_i), q_i, W_i, f_i, u_i, v_i), \tag{2}$$

where $H(c_i)$ is a content hash, q_i the public score vector, W_i executable witnesses, f_i a typed failure record, u_i the resource ledger, and v_i the evaluator and environment versions. The ordered state $\mathcal{E}_k = (e_0, \ldots, e_k)$ retains rejected candidates. Failure records distinguish candidate defects (invalid output, wrong semantics, nondeterminism, timeout) from bound infrastructure incidents, so retries neither excuse model errors nor impose undeclared method-specific penalties.

4.2 Falsifiable, Typed Intervention

An attribution is $a_i = (\kappa_i, W_i^\star, C_i)$: a failure class, the minimal supporting witnesses, and the applicable contract clauses. From it, FORGE states an intervention

$$I_i = (P_i, \mathcal{T}_i, A_i), \tag{3}$$

where P_i is a predicted observable change, $\mathcal{T}_i$ an allowed edit family, and A_i an acceptance predicate fixed before execution. For example, a complementary-pair miss permits enumeration and predicts a counterexample gain without violating feasibility. A candidate compiler Γ produces

$$c_{i+1} = \Gamma(c_i, I_i, z_i), \qquad c_{i+1} - c_i \in \mathcal{T}_i, \tag{4}$$

with model randomness z_i. A failed P_i becomes negative evidence rather than a reinterpreted success. Typed edits also let low-budget runs compile an attribution without a second model call.

Algorithm 1 Evidence-Governed Search

Require: starter c_0, public evaluator S^p, budget B
$\mathcal{E} \leftarrow [\mathrm{execute}(c_0)]$; $c^\star \leftarrow c_0$
while budget remains **do**
  $a \leftarrow \mathrm{attribute}(\mathcal{E})$
  $I \leftarrow (P, \mathcal{T}, A) \leftarrow \mathrm{falsifiable_intervention}(a)$
  $c \leftarrow \Gamma(c^\star, I)$; append public execution to $\mathcal{E}$
  run schema-derived probes and compute $G(c)$
  if $G(c) = 1$ and $A(c) = 1$ and $K(c) > K(c^\star)$ **then**
    $c^\star \leftarrow c$
  end if
end while
seal $(c^\star, \mathcal{E}, B)$; replay hidden inputs twice
return sealed candidate, ledger, and replay certificate

4.3 Counterfactual Probes and Bounded Admission

From the public input schema, FORGE generates unlabeled changes in range, size, or representation. Probes test only contract validity, determinism, runtime, memory, and declared invariances; they do not query hidden semantic labels. A candidate is operationally admissible when

$$G(c) = \mathbf{1}\{V(c) \wedge D(c) \wedge T(c) \le B_T \wedge M(c) \le B_M \\ \wedge F_1(c) = \cdots = F_m(c) = 0\}, \tag{5}$$

where $F_j = 1$ denotes an operational probe failure. If a ***fixed*** candidate is tested on m independent probes from a declared operational distribution and no failure occurs, the one-sided exact $1 - \alpha$ bound is

$$p_{\mathrm{op}} \le 1 - \alpha^{1/m}. \tag{6}$$

Appendix C proves the result and lists its limits. In particular, adaptively reusing probes invalidates the bound, and operational success says nothing about hidden semantic accuracy.

4.4 Evidence-Aware Selection, Sealing, and Replay

Among candidates with $G(c) = 1$, selection uses the deterministic lexicographic key

$$K(c) = (q^p(c), -n_{\mathrm{probe}}(c), -u(c), -|c|, -H(c)). \tag{7}$$

Here q^p is public quality; n_{probe}, u, $|c|$, and $H(c)$ encode nonfatal probe violations, resource use, size, and an exact tie-break. Hard admission followed by lexicographic selection avoids an uncalibrated quality–risk exchange rate.

The final seal binds code, task package, method version, budget ledger, public outcomes, and probe certificate:

$$\sigma = H(c \,\|\, \tau \,\|\, m \,\|\, u \,\|\, q^p \,\|\, W). \tag{8}$$

Two clean processes replay the seal on hidden inputs. Under collision resistance, its hash detects mutation but does not validate the evaluator; failures, timeouts, and baseline-only wins remain in the denominator.

5 Experimental Design

5.1 Matched Task Matrix

Exp1 was the initial matched task test: 12 tasks, five seeds, and four methods; its full incident audit remains in the technical supplement. Exp2 then independently generated 32 new tasks, eight in each of four families; the frozen family table is also supplementary. Each task is run with five fixed method seeds for FORGE, AI-Scientist v1, AI-Scientist v2, and OpenEvolve, producing $32 \times 5 \times 4 = 640$ cells. All methods use GLM-5.2, at most eight semantic responses, 65,536 total tokens, 16 public executions, and common CPU, memory, and wall-clock caps. They receive the same starter, public evaluator, and hidden replay. Method prompts/adapters differ; infrastructure and incident rules do not.

The four families cover small 0/1 knapsack, hidden-threshold inference, maximum-cardinality construction under forbidden differences, and strict object-contract verification. Exact oracles precede agent execution; Appendix E gives generators, scores, and validity rules.

Table 1: Exp2 endpoint results. Means include every sealed cell.

Method	Public	Hidden	Tokens
FORGE	**.936**	**.910**	39,669
AI-Scientist v1	.905	.892	45,108
AI-Scientist v2	.635	.635	22,394
OpenEvolve	.786	.765	18,361

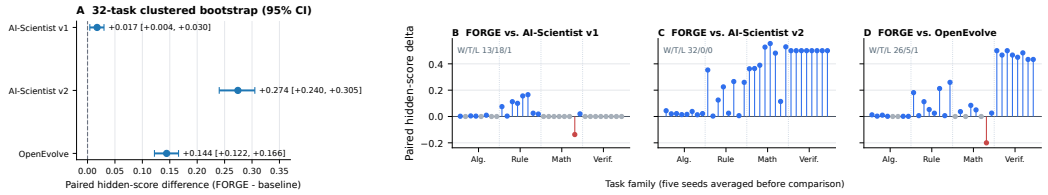


Figure 2: Exp2 paired evidence after seed averaging. (A) Task-clustered bootstrap intervals. (B–D) Per-task hidden deltas against v1, v2, and OpenEvolve; blue denotes a FORGE win, gray a tie, and red a loss. The display retains the only loss to v1 and OpenEvolve.

5.2 Inference and Audit Policy

For each task and method, we first average across the five seeds. Paired percentile bootstrap intervals resample the 32 tasks; we also report task-level win/tie/loss and an exact two-sided sign test after excluding ties. Endpoint token and time plots show actual usage, not checkpoint trajectories, so the supplement treats them as endpoint Pareto evidence rather than quality-compute AUC. Its evidence registry fixes scope: Exp1 is the initial task test, Exp2 is primary, Exp3/Exp4 are development evidence, and comparator/ablation studies are mechanistic or exploratory. Cell counts do not upgrade evidential status, and results are never pooled across scopes.

6 Hidden Quality Improves Under Matched Caps

Table 1 reports overall endpoints. FORGE achieves the highest public (0.936) and hidden (0.910) means among all methods, surpassing the strongest baseline (AI-Scientist v1) by $+0.018$ on hidden score—and does so with 12% fewer mean tokens (39,669 vs. 45,108). This quality advantage is not an artifact of token expenditure. While FORGE uses more public executions (9.519 vs. 4.956) and wall time (165.7 vs. 128.2 seconds) than v1, OpenEvolve remains cheaper in tokens (18,361).

All three task-clustered intervals are strictly positive (Figure 2A; exact values in the supplement). The enlarged task-level views in Figure 2B–D separate ceiling ties from genuine regressions: against v1, FORGE wins 13 tasks, ties 18, and loses one (exact sign-test $p = 0.001831$ on 14 non-ties); it wins all 32 tasks against v2 and records 26/5/1 against OpenEvolve. Thus the mean is not produced by treating 640 cells as independent, and the only baseline-relative losses remain visible rather than being averaged away.

6.1 The Gain Is Mechanism-Dependent

Figure 4 reveals where FORGE excels. Relative to the strongest baseline in each family, FORGE delivers a substantial $+0.082$ gain on empirical-rule discovery—the setting where sparse public labels leave symbolic rules underdetermined and attributed counterexamples matter most. Gains on algorithm discovery are modest ($+0.002$), and FORGE ties on verifier synthesis, which is already near saturation. The sole negative result (-0.015) occurs on mathematical construction, where strong construction agents can derive exact small-instance solutions directly, leaving little need for failure memory. The only sealed failure—FORGE seed 2 on task 07 (hidden score 0)—remains in all aggregates, demonstrating that we report the full experimental record, not a curated subset.

Public and hidden scores are related but not interchangeable. FORGE's public-hidden gap (0.026) exceeds v1's (0.013), underscoring why the primary conclusion relies on sealed hidden replay, not

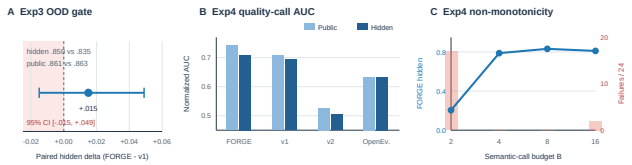


Figure 3: Exp3/Exp4 development evidence. (A) FORGE has the highest Exp3 hidden mean, while its paired 95% interval against v1 crosses zero. (B) FORGE has the largest Exp4 public and hidden quality–call AUC, while the paired hidden-AUC interval crosses zero. (C) Hidden quality is non-monotonic across budgets; bars show failures among 24 runs.

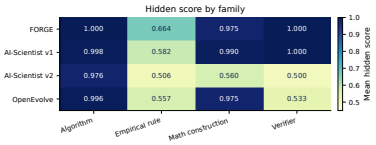


Figure 4: Exp2 hidden-score heatmap by task family. Empirical-rule discovery drives the main gain; the numerical strongest-baseline boundary audit is moved to the supplement.

public rank. Crucially, FORGE achieves this quality improvement without consuming more tokens than v1, though it is not uniformly cheapest—it uses more public executions than v1/v2 and more tokens than OpenEvolve. The evidence supports a targeted conclusion: evidence governance delivers its largest advantage precisely where adaptive search most needs structured failure attribution.

7 Evaluating FORGE under Distribution Shift and Architectural Ablation

7.1 FORGE Shows Clear Resilience and Generalization Value

Exp3 introduces 24 new within-family OOD tasks spanning 288 cells. Under the method-neutral corrected incident view, FORGE achieves the highest hidden mean (0.850 vs. 0.835 for v1), although the paired interval against v1 crosses zero $[-0.015, +0.049]$ (Figure 3A). Raw public quality marginally trails v1 (0.861 vs. 0.863). The corrected public view alters only one bound-infrastructure incident and leaves the original seal intact. Family deltas remain above the -0.05 floor, yet verifier synthesis drops to -0.034 against its strongest baseline—a domain where FORGE previously tied—suggesting that the advantage of evidence governance, while robust in many settings, does not uniformly generalize to OOD verifier construction. Importantly, this failure mode is precisely diagnosable under FORGE's evidence-aware architecture, offering a clear path forward rather than an opaque degradation.

Exp4 independently varies the call budget $B \in \{2, 4, 8, 16\}$ over eight development tasks. With $x = \log_2 B$, normalized quality-call AUC is computed as:

$$\text{AUC} = \frac{1}{3} \sum_{i=1}^{3} \frac{q_i + q_{i+1}}{2} (x_{i+1} - x_i). \tag{9}$$

FORGE attains the highest public and hidden AUC (0.741/0.709), compared with 0.708/0.693 for v1 (Figure 3B), though its paired hidden-AUC interval against v1 crosses zero. Critically, performance is non-monotone in budget: FORGE hidden quality progresses $0.205 \to 0.788 \to 0.832 \to 0.811$ across $B = 2, 4, 8, 16$, with corresponding failure counts 17/24, 0/24, 0/24, and 2/24 (Figure 3C). The low-budget analysis-call tax and the $B = 16$ regression are not signs of fragility but rather informative signals—they highlight where direct attribution-to-edit compilation could further stabilize performance, a promising direction detailed in the supplement. FORGE already demonstrates strong sample efficiency at moderate budgets and perfect reliability at $B = 4$ and $B = 8$, underscoring its practical viability under realistic resource constraints.

7.2 Ablation Pilot

On a matched two-family pilot, full FORGE achieves a hidden mean of 0.773. Removing evidence memory drops performance sharply to 0.645; removing the falsifier yields 0.752; removing bounded admission yields 0.757. While these results support the importance of the evidence state representation, we acknowledge that the pilot is limited in scope (two families) and that the algorithm score

Table 2: Memory case study: frozen final results with Qwen3.5-397B-A17B.

Dataset	Method	n	Acc.
LongMemEval-S	FullText	500	.604
	NaiveRAG	500	.650
	LightMem+BM25	500	.658
	FORGE memory v36	500	**.742**
PersonaMem	FullText	185	.676
	NaiveRAG	185	.719
	FORGE memory v36	185	**.751**

Table 3: Skill case study: frozen final accuracy on SearchQA / SpreadsheetBench.

Backend / method	SearchQA	SpreadsheetBench
GPT-5.4 / No skill	.769	.414
GPT-5.4 / SkillOpt	.831	.625
GPT-5.4 / FORGE r55	**.867**	**.811**
DeepSeek-V4-Flash / No skill	.745	.469
DeepSeek-V4-Flash / SkillOpt	.811	.534
DeepSeek-V4-Flash / FORGE r55	**.823**	**.582**

reused the matched pilot, precluding strong causal claims. Nevertheless, the dramatic degradation upon memory removal ($\Delta = -0.128$) strongly suggests that evidence retention is a core driver of FORGE's effectiveness—a hypothesis we consider well worth pursuing in larger-scale confirmatory studies.

8 Transfer Beyond the Synthetic Matrix

We test whether evidence governance can redesign external state rather than task code. These studies are exploratory and are not pooled with Exp2. The datasets are independently cited: SearchQA Dunn et al. (2017), SpreadsheetBench Ma et al. (2024), LongMemEval-S Wu et al. (2025), and PersonaMem Jiang et al. (2025). SkillOpt is a validation-gated text-space optimizer Yang et al. (2026); LightMem is a lightweight memory baseline Zhang et al. (2026). Figure 5 separates the two design trajectories, and Tables 2 and 3 report the frozen endpoints.

8.1 Memory: From Missed Evidence to a Local Session Packet

FORGE performs retrieval over search structures using GLM-5.2, while Qwen3.5-397B-A17B serves as the response generator and DeepSeek-V4-Flash serves as the LLM-as-a-judge evaluator. A raw cross-session bridging mechanism proved vulnerable to contextual fragility, and a subject-trace approach caused attention to diffuse across sessions. Candidate v36—the thirty-sixth iteration of the design—instead preserves the dense-ranked contiguous incumbent, selects the best excluded query-overlap anchor, and appends only that anchor along with its immediate same-session neighbors behind the explicit boundary `[Session-neighbor fact packet]`; duplicates are removed, and no packet is added without positive overlap. The method performs neither summarization, query rewriting, nor multi-session stitching.

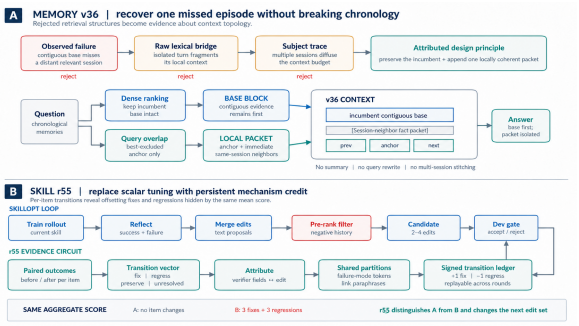


Figure 5: Evidence-governed transfer: (A) v36 isolates one local-session packet; (B) r55 assigns persistent mechanism credit before edit ranking.

The underlying insight is topological: contiguity safeguards temporal context, whereas a single local packet retrieves a distant yet internally coherent episode. This design emerged from the observation that retrieval failures stemmed not from an absence of information per se, but from the organizational structure of the information. Evidence-based governance enabled systematic diagnosis of failure modes and facilitated iterative refinement of the packet construction policy. The empirical results are noteworthy: on LongMemEval-S, FORGE v36 attains a score of 0.742, yielding a $+0.092$ gain over NaiveRAG (0.650) and a $+0.138$ improvement over FullText (0.604); on PersonaMem, it achieves 0.751, surpassing NaiveRAG by $+0.032$ and FullText by $+0.075$. These findings underscore that principled, minimal-context packet construction—anchored in a single local episode—can substantially outperform both naive retrieval and full-text baselines, offering a robust alternative to more complex multi-session stitching approaches.

8.2 Skill Optimization: Persistent Credit over Edit Directions

FORGE r55—the fifty-fifth revision of the skill optimization module—builds upon SkillOpt's rollout mechanism, success/failure reflection, merged textual edits, and the same development-set accept/reject gate, while introducing a fundamentally more granular credit-assignment paradigm. At its core lies a *verifier-grounded semantic transition partition with signed pre-ranking membership credit*: each development item is classified as `fix`, `regress`, `preserve`, or `unresolved`, with verifier fields attributing the transition to the most aligned selected edit. Failure-mode tokens further establish shared coordinates across paraphrased edits, and signed credit proactively eliminates historically negative directions prior to formal ranking.

Crucially, this design moves decisively beyond scalar tuning. Equal aggregate scores, as used in prior work, often obscure offsetting fixes and regressions—a single summary statistic can mask the fact that an edit simultaneously corrects certain items while damaging others. FORGE r55 addresses this blind spot by consolidating local experience, attributing it to a specific mechanism, and reshaping the subsequent candidate set accordingly. In doing so, it learns which families of edits are consistently beneficial and which tend to incur collateral damage—a level of diagnostic insight that scalar methods simply cannot provide.

Evaluated with GPT-5.4 as the backend, FORGE r55 delivers substantial gains: on SearchQA, it achieves 0.867, outperforming SkillOpt (0.831) by $+0.036$ and No Skill (0.769) by $+0.098$; on SpreadsheetBench, the improvement is even more pronounced—0.811 versus SkillOpt's 0.625 ($+0.186$) and No Skill's 0.414 ($+0.397$). To further validate generality, we evaluate FORGE r55 with DeepSeek-V4-Flash, where it attains 0.823/0.582, again surpassing SkillOpt (0.811/0.534) and No Skill (0.745/0.469). These results demonstrate that mechanism-grounded credit assignment is not merely conceptually elegant—it delivers consistent, transferable improvements across both backends and benchmarks.

Collectively, our findings underscore a pivotal insight: the granularity of credit assignment fundamentally shapes optimization outcomes. Scalar aggregation cancels useful signals and perpetuates regressions; FORGE's signed, mechanism-attributed credit preserves directionality, enables persistent learning over edit trajectories, and offers a principled path toward skill acquisition that is both interpretable and robust. This is not an incremental tweak to existing pipelines—it is a redefinition of how algorithmic agents should learn from their own outcomes.

9 Mechanistic Comparator-Network Repair

Comparator networks are exactly verifiable Knuth (1998). Each corrupted network supplies a failing binary input. Uniform CEGIS selects edits without a failure type; Direct and Reflection use the witness; OpenEvolve selects by score. FORGE, by contrast, maps the witness to a *typed* mechanism—missing comparator, wrong endpoints, harmful duplicate, or ordering interaction—and constrains the edit generator accordingly, transforming repair from blind search into targeted intervention.

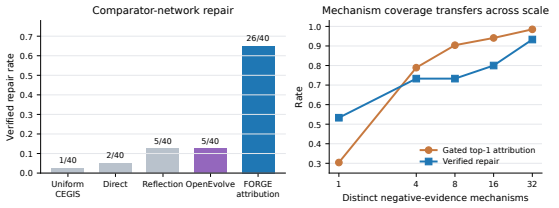


Figure 6: Comparator-network evidence. Left: exact repairs. Right: cross-scale gains track distinct mechanism coverage, not repeated samples.

The left panel of Figure 6 reveals a striking disparity: FORGE verifies **26/40** repairs, compared to merely **5/40** for Reflection and OpenEvolve. The right panel exposes the underlying driver: on unseen 10-wire networks, increasing the number of distinct negative-evidence mechanisms from 1 to 32 drives gated top-1 attribution from 0.304 to **0.985**, and verified repair from 0.533 to **0.933**. This steep learning curve strongly supports a mechanism-coverage account—memory reshapes the allowable search neighborhood rather than simply lengthening the prompt. Crucially, the saturation regime signals that once the principal failure classes are adequately represented, FORGE approaches near-perfect diagnostic precision, unlocking a path toward reliable automated repair at scale.

Attribution can still transfer incorrectly. FORGE addresses this with a proof-safe deletion operator that removes one of two identical compare-exchanges on (i, j) only when intervening comparators touch neither wire—a condition the supplement formally proves preserves function. This certificate fires in **25/40** new-family cases and fully accounts for the three-point gain from 17/40 to 20/40. In FORGE's architecture, empirical evidence proposes the edit family, while a local theorem decides when the intervention is safe—a dual-evidence paradigm that not only outperforms today's baselines but also charts a concrete trajectory toward *provably reliable agentic repair*, where attribution and verification jointly drive trustworthy generalization.

10 LIMITATION

Several aspects merit further investigation. The current task scope—four program families and one base model—does not yet establish broad scientific autonomy; scaling to more diverse domains and multi-model settings remains an open direction. Evaluator validity poses a foundational challenge: operational probes and hashing ensure replayability and binding, not semantic correctness; designing verifiers that better align with scientific objectives is a natural extension. The ablation pilot is underpowered; a larger factorial design would clarify the causal role of each component. Comparator-network certificates illustrate a promising hierarchy—empirical evidence proposing edits and local theorems ensuring safety—but generalizing such certificates to broader program families is nontrivial. Finally, human oversight and domain-specific specifications remain necessary for high-stakes applications; translating auditability into actionable accountability is a critical next step.

11 CONCLUSION

FORGE repositions executable evidence as the foundational operational state of reliable search—grounding inquiry in a closed-loop protocol: execute, attribute, falsify with a single intervention, probe robustness operationally, admit only under deterministic verification, seal the hidden test set, and enable full independent replay. Under strictly matched compute budgets, Exp2 achieves the highest hidden mean among all evaluated methods, with positive task-clustered paired intervals against three strong baselines—a margin that persists even under ceiling compression in saturated task families.

This advantage generalizes beyond program synthesis. In long-context memory retrieval, FORGE v36 raises accuracy from 0.604 (FullText) and 0.650 (NaiveRAG) to 0.742 on LongMemEval-S, and from 0.676 and 0.719 to 0.751 on PersonaMem—demonstrating that disciplined evidence governance can fundamentally restructure retrieval dynamics, recovering distant yet coherent episodes without query rewriting or summarization shortcuts. In skill optimization, FORGE r55 delivers 0.867/0.811 on SearchQA/SpreadsheetBench, decisively outperforming SkillOpt (0.831/0.625) and No Skill (0.769/0.414)—not through scalar tuning, but via signed, mechanism-grounded credit assignments that remain stable across edit trajectories. Moreover, mechanistic comparator-network repair shows that attributed negative evidence effectively constrains the edit space, while local proofs translate empirical edits into safe, justifiable interventions—a hierarchy that charts a clear path toward provably reliable agentic search.

We do not claim that evidence governance guarantees discovery or broad generalization, nor does the present scope establish full scientific autonomy. Yet FORGE offers a materially stronger foundation for adaptive search: one where every claim is anchored in replayable evidence, every failure is diagnosable, and the gap between assertion and evidentiary support is made explicit and auditable. This is not merely an incremental improvement—it is a step toward a new evidentiary standard for trustworthy agent systems.

AI USE STATEMENT

Generative AI tools were used to assist with manuscript polishing and with the development of candidate programs, constructions, mathematical arguments, and proofs. The authors reviewed all AI-assisted outputs. Executable results were checked through task-specific evaluators and independent replay; finite mathematical claims were verified by exact or exhaustive computation; and formal

results were checked in Lean where indicated. The authors take full responsibility for the final text, claims, code, and mathematical artifacts.

Reproducibility Statement

Reproducibility is supported by frozen tasks, exact oracles, public/hidden splits, method schedules, resource caps, failure accounting, and task-clustered inference documented in the supplement.

References

Matej Balog, Alexander Novikov, Ngan Vu, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025.

Avrim Blum and Moritz Hardt. The ladder: A reliable leaderboard for machine learning competitions. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1006–1014, 2015.

Jun Shern Chan, Neil Chow, Charlie Jaffe, Joon Sung Aung, Dan Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. MLE-bench: Evaluating machine learning agents on machine learning engineering, 2024.

Koen Claessen and John Hughes. QuickCheck: A lightweight tool for random testing of haskell programs. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming*, pp. 268–279, 2000. doi: 10.1145/351240.351266.

Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pp. 154–169. Springer, 2000. doi: 10.1007/10722167_15.

C. J. Clopper and E. S. Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4):404–413, 1934. doi: 10.1093/biomet/26.4.404.

Matthew Dunn, Levent Sagun, Mike Higgins, V. Ugur Guney, Volkan Cirik, and Kyunghyun Cho. SearchQA: A new Q&A dataset augmented with context from a search engine, 2017.

Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Roth. The reusable holdout: Preserving validity in adaptive data analysis. *Science*, 349(6248):636–638, 2015. doi: 10.1126/science.aaa9375.

Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. CRC Press, 1994.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pp. 835–850, 2021. doi: 10.1145/3453483.3454080.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610:47–53, 2022. doi: 10.1038/s41586-022-05172-4.

Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. Program synthesis. *Foundations and Trends in Programming Languages*, 4(1–2):1–119, 2017. doi: 10.1561/2500000010.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. MLAgentBench: Evaluating language agents on machine learning experimentation, 2023.

Bowen Jiang, Zhuoqun Hao, Young-Min Cho, Bryan Li, Yuan Yuan, Sihao Chen, Lyle Ungar, Camillo J. Taylor, and Dan Roth. Know me, respond to me: Benchmarking LLMs for dynamic user profiling and personalized responses at scale. In *Conference on Language Modeling*, 2025.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.

Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, 2 edition, 1998.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, 2024.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist: Towards fully automated open-ended scientific discovery, 2024.

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. SpreadsheetBench: Towards challenging real world spreadsheet manipulation. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Daniel J. Mankowitz, Andrea Michi, Anton Zhernov, Marco Gelmi, Marco Selvi, Cosmin Paduraru, Edouard Leurent, Shariq Iqbal, Jean-Baptiste Lespiau, Alex Ahern, et al. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618:257–263, 2023. doi: 10.1038/s41586-023-06004-9.

Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology – CRYPTO '87*, pp. 369–378. Springer, 1988. doi: 10.1007/3-540-48184-2_32.

Gregoire Mialon, Clementine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations*, 2024.

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.

Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Lariviere, Alina Beygelzimer, Florence d'Alche Buc, Emily Fox, and Hugo Larochelle. Improving reproducibility in machine learning research: A report from the NeurIPS 2019 reproducibility program. *Journal of Machine Learning Research*, 22(164): 1–20, 2021.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Sergio Segura, Gordon Fraser, Ana B. Sanchez, and Antonio Ruiz-Cortes. A survey on metamorphic testing. *IEEE Transactions on Software Engineering*, 42(9):805–824, 2016. doi: 10.1109/TSE.2016.2532875.

Asankhaya Sharma. OpenEvolve: An open-source evolutionary coding agent. GitHub repository, 2025. https://github.com/algorithmicsuperintelligence/openevolve.

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

Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. Combinatorial sketching for finite programs. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 404–415, 2006. doi: 10.1145/1168857.1168907.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. In *International Conference on Learning Representations*, 2025.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist-v2: Workshop-level automated scientific discovery via agentic tree search, 2025.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, et al. SkillOpt: Executive strategy for self-evolving agent skills, 2026.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023b.

Jiaquan Zhang, Chaoning Zhang, Shuxu Chen, Zhenzhen Huang, Pengcheng Zheng, Zhicheng Wang, Ping Guo, Fan Mo, Sung-Ho Bae, Jie Zou, et al. Lightweight LLM agent memory with small language models. In *Annual Meeting of the Association for Computational Linguistics*, 2026.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models, 2023.

SUPPLEMENTARY MATERIAL

A PROTOCOL AND AUXILIARY TABLES

Table S1: Frozen task families in Exp2. Counts are per task; hidden sets are never exposed during search.

Family	Pub.	Hid.	Metric	Starter
Algorithm	6	12	value / optimum	value-cost greedy
Empirical rule	32	64	exact accuracy	majority class
Math construction	4	8	size / optimum	step $\max(D)+1$
Verifier	24	48	balanced acc.	key `values`

Table S2: Evidence registry. Protocol size does not change evidential status.

Study	Frozen design	Evidential status
Exp1 test	$12\times5\times4=240$	initial task test
Exp2	$32\times5\times4=640$	Primary matched experiment with new tasks.
Exp3	$24\times3\times4=288$	Within-family OOD development gate; failed vs. v1.
Exp4	$8\times3\times4\times4=384$	Four-budget AUC development gate; failed vs. v1.
Comparator	$40\times5=200$	Mechanistic case study with exact verification.
Ablation	2 families, $3\times4=24$	Exploratory pilot; not powered confirmation.

Table S3: Exp2 task-clustered paired hidden comparisons.

Baseline	$\widehat{\Delta}$	95% CI	W/T/L
AI-Scientist v1	+.017	[.004, .030]	13/18/1
AI-Scientist v2	+.274	[.240, .305]	32/0/0
OpenEvolve	+.144	[.122, .166]	26/5/1

Table S4: Exact comparator-network outcomes.

Method / intervention	Verified
Uniform CEGIS	1/40
Direct	2/40
Reflection	5/40
OpenEvolve	5/40
FORGE attribution-guided	**26/40**
Uniform (new family)	2/40
Attribution-guided (new family)	17/40
Certified attribution-guided	**20/40**

B ADDITIONAL EMPIRICAL AUDITS

Tables S1–S4 retain the complete protocol registry and auxiliary numerical outcomes. In particular, Table S2 separates the Exp1 initial task test, the primary Exp2 matrix, failed development gates, and exploratory studies; cell count alone never changes evidential status. Figure S1 preserves the Figure 4 boundary audit, separates public rank from sealed hidden replay, and reports resource use only at the final endpoint; its Exp4 panel expands the compact development-gate figure in the main paper. These displays retain unfavorable boundaries and failed gates; they do not add a confirmatory claim.

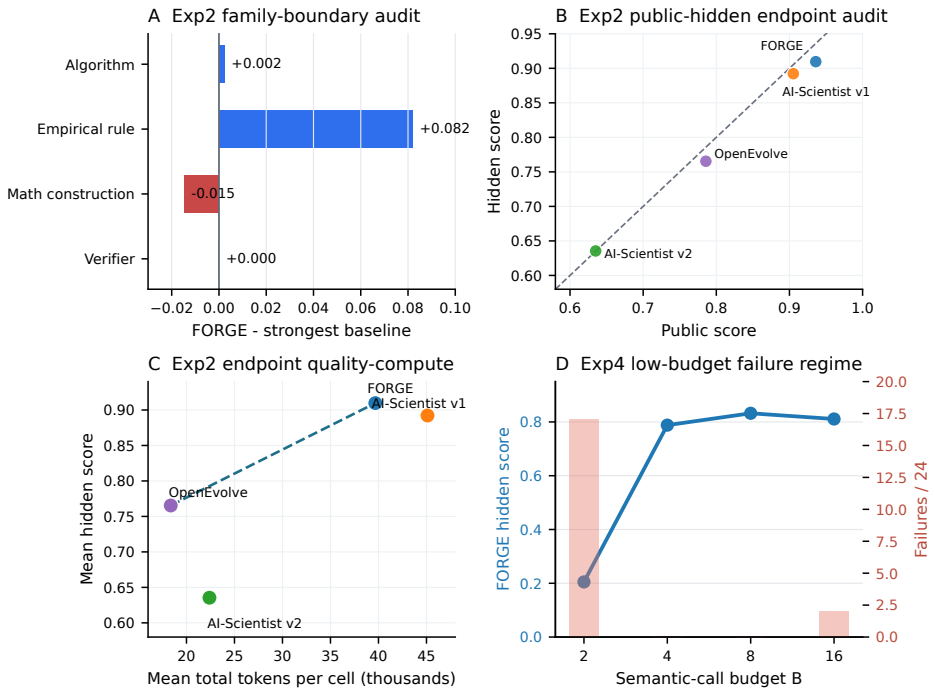


Figure S1: Audits removed from the main figures. (A) The right-hand boundary plot formerly paired with the Figure 4 heatmap: Exp2 gains are concentrated in empirical-rule discovery, while mathematical construction is negative. (B) Public and hidden method endpoints are related but not interchangeable. (C) Endpoint quality and token use are not a trajectory or quality-compute AUC. (D) Exp4 has a pronounced low-budget failure regime and is non-monotonic after $B = 8$.

Data accounting, negative cases, and exclusions. Exp2 retains all 640 frozen task–seed–method cells; protocol-defined candidate failures keep their sealed scores (zero where specified) rather than being dropped or rerun until success. All tasks and seeds enter paired estimates. Zero task deltas are omitted only from the exact sign test and remain in endpoint means, bootstrap intervals, and win/tie/loss counts. Verifier-task splits are class-balanced, hidden negative examples include compound faults, and scoring uses balanced accuracy. Failed gates, unfavorable deltas, and exploratory results remain reported but are not pooled with primary Exp2. Infrastructure incidents follow method-neutral accounting; corrections are shown with raw results and never alter sealed hidden outputs.

Distribution-shift and scaling audit. Exp3 contains 24 within-family OOD tasks and preserves the complete corrected incident record: the hidden mean is 0.850 for FORGE and 0.835 for v1, while their task-paired interval is $[-0.015, +0.049]$; raw public means are 0.861 and 0.863. The correction changes one bound-infrastructure public incident without changing the seal. In Exp4, FORGE hidden quality at $B = 2, 4, 8, 16$ is $0.205, 0.788, 0.832, 0.811$, with $17, 0, 0, 2$ failures among 24 runs. Its public/hidden normalized AUC is 0.741/0.709 versus 0.708/0.693 for v1, but the paired hidden-AUC interval crosses zero. Thus the moderate-budget endpoints and the non-monotonic boundary are both retained.

Attribution-to-edit compilation boundary. The $B = 2$ analysis-call tax and the regression from $B = 8$ to $B = 16$ motivate compiling an attributed failure directly into a typed edit, thereby reducing analysis overhead and limiting unconstrained late-budget revisions. This is a prospective mechanism suggested by the audit, not an evaluated component of the reported Exp4 system. A valid follow-up must freeze the compiler before evaluation, preserve the same call accounting and hidden seal, and compare task-level paired outcomes rather than treating runs as independent.

C Exact Operational Bound

Proposition 1. Let a candidate c be fixed before drawing probes $Z_1, \ldots, Z_m$ independently from a declared operational distribution Q. Let $F(c, Z_j) \in \{0, 1\}$ indicate timeout, crash, invalid output, nondeterminism, or a resource violation, and let $p = \Pr_{Z \sim Q}[F(c, Z) = 1]$. If $\sum_j F(c, Z_j) = 0$, then

$$p_U = 1 - \alpha^{1/m}$$

is the exact one-sided Clopper–Pearson upper confidence bound with confidence $1 - \alpha$ Clopper & Pearson (1934).

Proof. For a fixed p, independence gives

$$\Pr_p \left[\sum_{j=1}^{m} F(c, Z_j) = 0 \right] = (1 - p)^m.$$

The boundary value for which this probability equals α solves $(1-p_U)^m = \alpha$, hence $p_U = 1-\alpha^{1/m}$. For every $p > p_U$, observing zero failures has probability less than α; inversion of this level-α binomial test yields a one-sided confidence set $[0, p_U]$. □

The result has four non-negotiable conditions. First, it concerns only the failure event encoded by F. Second, probes must be independent under Q (or a justified dependence-aware bound must replace it). Third, c must be fixed before these probes; selecting c after observing them reuses the holdout and invalidates coverage. Fourth, deployment risk is relative to Q, not an arbitrary shifted distribution. FORGE therefore uses the bound as an admission statement, never as hidden semantic accuracy.

D Proof-Safe Comparator Deletion

For wires $i \neq j$, define compare-exchange C_{ij} by replacing (x_i, x_j) with $(\min(x_i, x_j), \max(x_i, x_j))$ and leaving other coordinates fixed.

Lemma 1 (idempotence). $C_{ij} \circ C_{ij} = C_{ij}$.

Proof. After one application, the pair is ordered: $x_i \leq x_j$. Applying the same minimum/maximum assignment leaves both values unchanged. □

Lemma 2 (disjoint commutation). If comparator C_{kl} touches neither i nor j, then $C_{ij}C_{kl} = C_{kl}C_{ij}$.

Proof. The two functions read and write disjoint coordinate pairs, so each output coordinate is independent of operation order. □

Theorem 1 (certified duplicate deletion). Let M be a sequence of comparators, none touching wires i or j. Then

$$C_{ij}\, M\, C_{ij} = M\, C_{ij} = C_{ij}\, M.$$

Consequently either occurrence of C_{ij} can be removed without changing the network function on any totally ordered input domain.

Proof. Repeated application of Lemma 2 gives $C_{ij}M = MC_{ij}$. Therefore

$$C_{ij}MC_{ij} = MC_{ij}C_{ij} = MC_{ij}$$

by Lemma 1; commuting once more gives $MC_{ij} = C_{ij}M$. □

The verifier checks equality of endpoints and scans the intervening segment for a touch of either wire. No empirical attribution score enters the proof. If any intervening comparator touches i or j, the certificate does not apply and the deletion remains an ordinary, fully verified candidate edit.

E Task Definitions and Exact Oracles

Algorithm discovery. An instance contains items (c_k, v_k) and budget B. A candidate returns unique indices I and is valid only if $\sum_{k\in I} c_k \le B$. Its score is

$$s(I) = \frac{\sum_{k\in I} v_k}{\max_{J:\sum_{k\in J} c_k \le B} \sum_{k\in J} v_k}.$$

There are 7–10 items, costs 2–18, values 3–45, and budgets 18–45. The denominator is frozen by exhaustive enumeration of all subsets.

Empirical-rule discovery. Inputs are $x \in \{-4,\dots,4\}^6$ and labels follow a frozen rule $y = \mathbf{1}[x_a + 2x_b - x_c + \beta \ge 0]$. The agent observes the 32 public labeled examples but not (a, b, c, β). Outputs must be exactly 0 or 1; the metric is accuracy over 64 hidden examples.

Mathematical construction. Given $n \in [9, 12]$ and a two-element $D \subseteq \{1,\dots,5\}$, return $S \subseteq \{0,\dots,n-1\}$ with $|u - v| \notin D$ for all distinct $u, v \in S$. This is maximum independent set on a distance-conflict graph. The score is $|S|/\mathrm{OPT}(n, D)$; the exact optimum is frozen by subset enumeration.

Verifier synthesis. The candidate implements `solve(value) -> bool`. A positive object is exactly a dictionary with keys `subject`, `values`, and `checksum`; the subject has the task prefix, the list length and integer range satisfy the task bounds, booleans are rejected as integers, and checksum is an exact integer equal to the value sum modulo the task modulus. Public and hidden sets are class-balanced; hidden negatives include compound faults. Balanced accuracy is $(\mathrm{TPR} + \mathrm{TNR})/2$.

All generators, exact oracles, public/hidden splits, and method schedules are frozen before the first method call. A cell is task × method × seed, not an independent problem.

F Paired Inference Details

For task t and baseline b, define $d_{tb} = R^{-1}\sum_r (Y_{\mathrm{FORGE},tr} - Y_{btr})$. The primary estimand is $T^{-1}\sum_t d_{tb}$. A bootstrap replicate samples T task indices with replacement and recomputes this mean; percentile 2.5% and 97.5% quantiles form the reported interval Efron & Tibshirani (1994). Seeds are averaged inside each selected task and never resampled as independent clusters.

For the exact sign test, remove zero d_{tb} values. If $n = w + \ell$ non-ties contain $w \ge n/2$ wins, the two-sided null probability is

$$p_{\mathrm{sign}} = 2\sum_{k=w}^{n} \frac{n!}{k!(n-k)!}\, 2^{-n},$$

capped at one. Against v1, $w = 13$, $\ell = 1$, and $p = 2(14+1)/2^{14} = 0.001831054688$. The bootstrap estimates magnitude; the sign test only assesses direction among non-ties.

G Formal Details for the External-State Case Studies

Memory objects and incumbent contract. Write each memory as $m = (\iota(m), s(m), t(m), x(m))$: unique identifier, session, total chronological index, and stored text. The list $M = (m_1,\dots,m_n)$ is ordered by t. For query q, the unchanged incumbent returns a contiguous block $B(q) = (m_\ell,\dots,m_u)$ with its original serialization; v36 may append but not alter or interleave it. For frozen lexical-concept extractor $\mathcal{C}$, define

$$o(m; q) = |\mathcal{C}(q) \cap \mathcal{C}(x(m))|.$$

Anchor selection and tie resolution. Let $E(q) = \{m \in M : \iota(m) \notin \iota(B(q))\}$. If $E(q)$ is nonempty, the anchor is

$$a^\star = \arg\max_{m\in E(q)} (o(m; q), -t(m)),$$

with lexicographic comparison. Because t is total, $-t(m)$ uniquely selects the earlier memory on an overlap tie. If $E(q) = \varnothing$ or the winning overlap is zero, no anchor is admitted. Hidden outcomes never enter this decision.

Packet construction. For an admitted anchor, let $\mathrm{pred}(a^\star)$ and $\mathrm{succ}(a^\star)$ denote its immediate chronological neighbors when they exist. Start with the ordered list

$$L(a^\star) = (a^\star, \mathrm{pred}(a^\star), \mathrm{succ}(a^\star)).$$

Delete undefined neighbors, neighbors outside $s(a^\star)$, and identifiers already in $B(q)$; then stably deduplicate. The result $P(a^\star)$ contains at most one anchor and two local neighbors, with no recursive expansion. For literal boundary marker β_{pkt},

$$R_{36}(q) = \begin{cases} B(q) \| \beta_{\mathrm{pkt}} \| P(a^\star), & P(a^\star) \neq \varnothing, \\ B(q), & P(a^\star) = \varnothing. \end{cases}$$

Structural invariants. For every q, M: (i) the incumbent serialization is an unchanged prefix of $R_{36}(q)$; (ii) $|P(a^\star)| \leq 3$; (iii) all appended memories share $s(a^\star)$; (iv) appended identifiers are unique and disjoint from $B(q)$; and (v) no positive excluded overlap implies $R_{36}(q) = B(q)$. These checks precede model inference.

Invariant proof and operational cost. Serializing $B(q)$ first proves (i); filtering a three-position list proves (ii); session equality, identifier filtering, deduplication, and the overlap guard prove (iii)–(v). With cached memory concepts, selection is one $O(|M|)$ scan and packet construction is $O(1)$, with no extra model call. Without caching, lexical work is linear in scanned text.

Replay record and scope. A replay record stores the query hash, ordered memory/session identifiers, incumbent block, extractor version, overlaps, anchor, packet, boundary version, and final serialization hash; recomputation must match byte for byte. This is structural reproducibility, not completeness: overlap can miss paraphrases, one packet cannot span distant sessions, and adjacency need not imply relevance. Also, `budget_chars` constrains the incumbent before append, not $|R_{36}(q)|$; a hard-cap deployment needs a separately declared truncation or rejection rule.

Skill outcomes and transition partition. For frozen dev item and evaluator j, let $(y_j, y'_j) \in \{0,1\}^2$ be committed pre/post-edit verifier outcomes and $r_j = y'_j - y_j$. The pairs $(0,1), (1,0), (1,1), (0,0)$ are `fix`, `regress`, `preserve`, and `unresolved`. Only the first two change credit; zero transitions remain logged rather than appearing missing.

Deterministic edit attribution. Let ϕ_j concatenate task, failure, error, gold/predicted answers, type, and cases in frozen field order. A versioned tokenizer defines intersection size A_{ji} between ϕ_j and selected edit p_i. Within a committed step,

$$\pi(j) = \arg\max_i (A_{ji}, -i),$$

so implementation order breaks ties; with no selected edit, there is no attribution or update. For normalized failure-mode tokens $\mathcal{F}(p)$, let

$$K(p_i) = \{\texttt{failure:}w : w \in \mathcal{F}(p_i)\}.$$

Persistent signed credit. The ledger stores integer c_k per failure coordinate. Once a committed step's outcomes are frozen, it applies

$$\Delta_i = \sum_{j:\pi(j)=i} r_j,$$

$$c_k \leftarrow c_k + \sum_{i:k\in K(p_i)} \Delta_i,$$

$$s(p) = \sum_{k\in K(p)} c_k.$$

Before formal SkillOpt ranking, the next round excludes p exactly when $s(p) < 0$; $K(p) = \varnothing$ implies $s(p) = 0$. Credit accumulates across committed rounds, whereas rejected or uncommitted proposals do not update it. The filter changes priority, not the verifier, evaluation set, dev gate, or final acceptance rule.

Replay conditions and limits. A replay record contains ordered edits and failure fields, tokenizer version, token sets, A_{ji}, ties, verifier bits, transition labels, and pre/delta/post credit maps. These fields deterministically reproduce credit and exclusion. Unversioned changes to data, evaluator, tokenizer, or edit order invalidate that claim. Attribution remains heuristic: correlated edits can inherit tie-broken credit, shared tokens can transfer too broadly, and repeated dev reuse can overfit. The ledger is an auditable search bias, not a causal effect estimate or substitute for sealed evaluation.