

Zenith: program repair and the value of retained tests

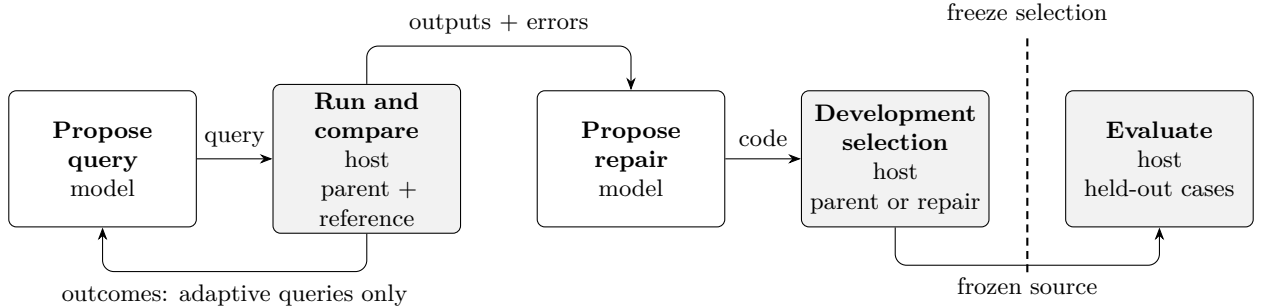
Will Cook • 21 October 2026

Abstract. We investigate whether retaining programs, observations and tests improves later work. Separate studies test model-driven repair, generated testers and a continuation learner. Across twelve episodes from three contracts, repair raises correct later answers from 259/576 to 365/576; added investigation gives no aggregate improvement. Two generated testers each find three of four authored SQLite faults; fixed tests find all four. Retained continuations save extension queries with cached answers held identical, but joint acquisition remains costlier than separate ordinary modules. We propose testing whether experience helps select questions on new software at lower total cost.

1. Programs and measured repair. A fresh SQLite AUTOINCREMENT table and one with row 7 inserted then deleted are both empty. The next insert without an explicit identifier returns 1 or 8, respectively. A rows-only program loses a distinction needed by this continuation: an observed failure can identify information that a future computation must preserve.

Zenith retains programs, tests and source-bound observations; we compare these with ordinary code reuse and cached answers. Language-model weights stay fixed; continuation learning uses a separate authored algorithm.

Program repair. A model generates a starting program from a contract and eight public examples. Fixed tests, a generated selector or direct model queries add evidence; repair-only adds none. The host compares the starting program with native-library answers, stopping at a discrepancy or eight query slots. Invalid and duplicate queries consume slots. A separate model call proposes replacement code. The host replays examples and valid probes, rejecting inadmissible code, lost prior successes, fewer correct answers, or more errors when scores tie. Rejection retains the starting program; unchanged code may pass. Both later panels remain withheld until selection freezes.



Native repair with model-proposed queries; repair-only omits new queries. Development selection preserves every previously correct measured case and nondecreasing aggregate scores; otherwise it retains the parent. Held-out outcomes are withheld from repair and selection.

Twelve episodes cover SQLite, least-recently-used caches (LRU) and UTF-8 four times each, with 48 confirmation inputs per episode. Repair-only raises agreement from 259/576 to 365/576; two initially invalid programs contribute 66 of the 106 added successes. Fixed testing scores 364, compiled investigation 354, direct batch 339, feedback-aware requests 343 and outcome-masked requests 331. The twelve-answer paired difference comes from separate repairs with identical prompts on one episode that stopped before a second query decision. The other eleven pairs tie, including all five with second decisions: this comparison does not identify a feedback benefit. Eleven starting programs are distinct; only three mechanisms were authored. The diagnostic used 130 Ling3.1Flash calls for \$0.12982944; two separate qualifications bring these stages to 200 calls and \$0.19002522, counting the resumed prefix once.

* **Authorship and AI use.** Will Cook built and directed the research infrastructure and reviewed the claims when he could. AI agents did most of the research and drafting. Cook did not independently verify every claim.

2. Test reuse and continuation learning. Reusable SQLite testers. A model generates two Python testers from five authored batch clients’ history and the transfer contract: a diagnostic procedure and ordinary reusable code. Both freeze before testing five opaque clients. Generated and fresh-query arms receive the contract, grammar, benign example and feedback; cold omits history. Each arm stops at a reference/client discrepancy or six slots per client. Invalid and duplicate queries consume slots without native execution. Fixed tests use an authored three-query schedule. A separate 400-case panel opens after all 25 runs and identifies four faulty clients; it evaluates clients, not testers. Table entries cover all five targets; model calls count deployment only.

Method	Faults / 4	Slots	Probes	Duplicates	Model calls
Retained tester	3	15	5	10	0
Ordinary code reuse	3	15	5	10	0
Same-history model	3	19	15	4	19
Cold model	3	15	15	0	15
Fixed tester	4	7	7	0	0

Each generated tester repeats a constant query and ignores feedback. Its failing transfers stop before a debit. Fixed tests also try a missing destination after debit, exposing the fourth fault. DeepSeek V4.1 Flash on Morph uses ten shared development calls, one preparation call per generated tester and 34 target calls: 46 calls, \$0.007117764. Both v1 generated testers stopped; v2’s procedure was invalid and its ordinary tester stopped. All three versions total 133 calls and \$0.019295552. The family was inspected during engineering; independent transfer remains untested. Charges exclude engineering and host costs.

Continuation learning. A continuation is a further input sequence. In a queue retaining at most four items, each input appends 1 and f returns the sum. A history h of three inputs followed by a test t of two gives 4. Histories index observation-table rows; tests index columns: $H_{h,t} = f(ht)$ records their combined outcome. The initial two-coordinate model predicts 5; retaining this paired history and test raises the table rank to three.

With exact field-valued observations and a square full-rank table containing the empty history and test, the weighted-automaton construction turns a counterexample to its hypothesis into a paired row/column extension increasing rank [2, Proposition 1, pp. 2–3]. Our repair restores that established requirement. Finite challenges replace the global equivalence oracle, so a completed fit establishes only agreement on those challenges. The five-coordinate queue model fits in ten acquisition executions and agrees on lengths 0–32; one nonlinear integer counter suffices.

Predictive state representations instead describe stochastic state through future-test probabilities. Their linear predictions use normalised conditional updates; the cited construction starts from a known finite partially observable Markov decision process [1, §1, Eq. (3); Theorem 1 and §2]. Zenith fits deterministic outputs with rational-linear transitions and readouts.

Across eight native configurations, the repaired learner fits from default seeds or generic one-symbol questions. Generic seeding costs 11,379 acquisition calls versus 11,395; both conditions refine counterexamples. Later finite checks find errors: UTF-8 scores 4,903/4,913; SQLite rowid 190/192 and AUTOINCREMENT 191/192. These are diagnostic fixtures, not eight independent applications.

In six leave-one-out conditions, paired arms share cached answers for five programs, then fit a joint model of all six. Retaining questions reduces this joint extension’s source queries by 11.2–63.1%. Initial plus extension acquisition costs 5,427–5,698 executions, against 3,071 for separate ordinary modules. Importing every learned suffix into the ordinary learner raises summed target acquisition from 3,071 to 3,541 calls, increasing work on five targets. With prior module acquisition, each condition costs 3,036–3,212 executions versus 3,071 without transfer. Acquisition excludes later validation. Exact certificates apply to supplied rational-linear references; native checks establish finite agreement.

3. Related work and next experiments. The comparisons establish measured repairs, executable tester reuse and query savings inside an authored learner. The next question is whether retained distinctions help a procedure choose informative tests on changed mechanisms. The continuation learner and generated testers remain separate implementations.

Related work. ILF trains a neural fuzzing policy from symbolic-execution demonstrations and evaluates it on withheld Ethereum contracts [3, §§4–6]. CoverUp uses a fixed language model and coverage/error feedback to retain passing regression tests that add coverage, treating the implementation as correct [4, §§1,3–4]. Voyager retains and composes executable Minecraft skills within an authored acquisition loop [5, §§2.2–2.3]. These methods retain a trained policy, tests or action programs. STOP recursively rewrites the program that organises code improvement, with the model and utility fixed [6, §§3–5]. Hyperagents edits task and meta-agent code, transferring both to maths grading. It reports improved graders with the transferred meta-agent held fixed; after 200 steps of continued self-modification, the transferred-versus-fresh advantage is statistically unresolved [7, §§5.2–5.3].

The next comparison. The present client family offers no detection headroom over fixed testing. Choose independently sourced clients before observing tester outcomes. Compare ordinary reusable code and fixed tests with a frozen procedure that selects retained questions, matching source access, development history and budgets. Hold selector and solver code fixed; vary access to retained questions while preserving the observation cache. Charge prior acquisition, selection and failures; report new-target queries separately from revisits to old sources.

Evaluating an acquired improver. If this procedure helps fresh tasks, compare initial and successor improvers on identical untouched starting solvers. Freeze each improver, select its proposed repairs on development cases, then evaluate once on held-out inputs. Keep the evaluator outside the edits and count acquisition, execution, evaluation, failed proposals and human intervention. Only then compare both procedures editing the same starting improver at matched total cost. Any gain remains bounded by the tested tasks and budgets.

Mathematical evaluation tasks. The [Plectis mathematical corpus](#) contains Lean proofs and open obligations; its argument-continuation exporter matches required hypotheses to candidate theorems and records remaining obligations. [Plectis Atlas](#) presents OpenAI’s mathematics release and attributed outside assessments. Neither is integrated with Zenith as a demonstrated learner. Evaluation would need action, outcome, cost and target-relevance records; finished proofs omit alternative attempts. Lean checks supplied proofs; failed search does not establish falsehood. Mathematics remains a possible harder transfer domain after cheap software tests.

Collaboration. Zenith is an exploratory project developed with AI coding agents. I welcome corrections and collaboration on prior work, evaluations and useful tasks through the [project’s issues](#). [Results and replay links](#) retain failures and exact costs; the [programme](#) expands the proposed comparisons.

References

- [1] M. L. Littman, R. S. Sutton and S. Singh. [Predictive Representations of State](#). NIPS, vol. 14, 2001.
- [2] J. Worrell. [Exactly Learning Weighted Automata over a Field](#). Oxford teaching notes, undated.
- [3] J. He et al. [Learning to Fuzz from Symbolic Execution with Application to Smart Contracts](#). CCS, 2019.
- [4] J. Altmayer Pizzorno and E. D. Berger. [CoverUp: Effective High Coverage Test Generation for Python](#). PACMSE 2 (FSE), 2025.
- [5] G. Wang et al. [Voyager: An Open-Ended Embodied Agent with Large Language Models](#). arXiv, 2023.
- [6] E. Zelikman et al. [Self-Taught Optimizer \(STOP\): Recursively Self-Improving Code Generation](#). COLM, 2024.
- [7] J. Zhang et al. [Hyperagents](#). arXiv, 2026.