VETRA: COMPOSABLE EVALUATOR CONTROL FOR REPOSITORY REPAIR

Anonymous authors
Paper under double-blind review

ABSTRACT

We treat a repository agent's last two evaluator calls as a transferable control problem: choosing a targeted test or a regression suite determines both what the agent can learn and whether it can validate the patch it returns. Verification and Evaluator-Control Transfer for Repository Agents (VETRA) makes that choice reusable: it stores Observe, Diagnose, Repair, Verify, and Budget as executable fields, routes on the latest failure, and deterministically composes compatible test order, coverage, call reserves, and stopping conditions. EvalRepair-320 supplies 320 tasks from 160 unseen repositories and a complete $2 \times 2 \times 2$ intervention over transfer object, router evidence, and offline refinement while fixing the backend, online actions, bank capacity, and evaluator opportunities. At $B = 8$, VETRA achieves 84.7% success with 2 median evaluator calls, compared with 72.4% and 3 for the initialized typed router and 46.0% and 6 for bundled Meta-Policy Reflexion. Across the grid, typed-field system cells average 17.3 success points above bundled cells, failure-conditioned routing adds 9.9, and refinement adds 9.7. Verify+Budget leads all four prespecified equal-cardinality field pairs at both tested budgets. Hidden tests retain 79.5% success, blinded issue-satisfaction review yields 74.1% adjusted success, and frozen-bank gains persist on SWE-bench Lite and Verified-240. The result identifies evaluator policy—especially verification order and call reservation—as transferable state for repository agents operating with scarce execution feedback.

1 INTRODUCTION

A repository agent with two evaluator calls remaining faces a control decision before it faces another coding decision. A targeted test may distinguish competing diagnoses; an affected-module suite may expose collateral damage; reserving the final call may be the only way to certify a full regression. Spending those calls in the wrong order can leave a plausible patch either under-informed or unvalidated. This allocation problem is operationally important: in our worker, one build or test invocation is 22.8 times slower than a read-only action, and evaluator execution occupies 71.6% of measured end-to-end time.

Repository agents already combine capable search, editing, and execution loops (Jimenez et al., 2024; Yang et al., 2024b; Wang et al., 2025; Zhang et al., 2024; Xia et al., 2025). Experience-reuse systems retrieve trajectories, demonstrations, or reflections (Shinn et al., 2023; Wu et al., 2025; Lindenbauer et al., 2025; Mu et al., 2026); DARS branches from execution failures (Aggarwal et al., 2025); and PatchPilot engineers a cost-aware validation-and-refinement workflow (Li et al., 2025). These advances improve complete repair processes, but their reusable unit does not expose the evaluator schedule as a separately exchangeable object. A useful test order can consequently remain entangled with a repository-specific localization or edit rule. The unresolved technical problem is to transfer *which evidence to acquire next*—target, coverage breadth, reserve, and stopping condition—without replacing the rest of the repair program.

VETRA instantiates that transfer boundary. Each bank record contains five executable fields—Observe, Diagnose, Repair, Verify, and Budget—plus compatibility and admissibility metadata. A failure-conditioned router retrieves candidate records, and a deterministic composer resolves them in Budget–Verify–Diagnose/Repair–Observe precedence, rejecting combinations that violate reserves, diagnostic preconditions, or integrity rules. Offline refinement exchanges or rewrites complete same-

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.

type fields on repository-disjoint tasks; the bank is then frozen. Verify or Budget can therefore change while the patch interface, legal evaluator actions, and remaining policy fields stay fixed, creating both an agent mechanism and an intervention surface.

EvalRepair-320 makes the resulting claims testable on 320 tasks from 160 held-out repositories. A complete $2 \times 2 \times 2$ grid crosses bundled versus typed-field control, task-only versus task-plus-failure routing, and initialized versus refined banks under matched backend, prompt allocation, tools, verifier, bank capacity, retrieval depth, and online action space. Typed-field system cells average 17.3 success points above bundled cells, failure evidence adds 9.9, and refinement adds 9.7. The full VETRA system reaches 84.7% success with 2 median evaluator calls at $B = 8$, versus 72.4% and 3 for the initialized typed router and 46.0% and 6 for bundled MPR. Under equal-cardinality reuse, Verify+Budget reaches 73.4% at $B = 4$, 9.5 points above the strongest of the three prespecified alternative pairs. Hidden tests, blinded issue-satisfaction review, a shifted repository condition, frozen-bank SWE-bench transfer, and a second code-model family preserve the ordering.

Contributions. The paper introduces (i) a typed evaluator-policy interface whose executable fields can be retrieved, composed, and intervened on independently; (ii) a matched diagnostic design that separates the deployed typed-field package, observed-failure routing, offline refinement, and field identity; and (iii) EvalRepair-320, an evaluator-budgeted repository-repair protocol with paired task and repository inference, total-tool accounting, hidden tests, semantic adjudication, distribution shift, and frozen-policy external transfer. Together, these contributions identify verification order and evaluator reservation as high-value cross-task state rather than incidental trace content.

2 Related Work

Repository repair joins cross-file context, executable feedback, and issue-level correctness. RepoCoder studies repository-level completion, whereas SWE-bench and its extensions provide issue-resolution workloads (Zhang et al., 2023a; Jimenez et al., 2024; Yang et al., 2025; Deng et al., 2025). EvalRepair-320 is a reverted-commit repair collection rather than an instance of the RepoBench completion benchmark. SWE-agent, OpenHands, AutoCodeRover, Agentless, and InspectCoder develop interfaces, search procedures, and patch generators for repository repair (Yang et al., 2024b; Wang et al., 2025; Zhang et al., 2024; Xia et al., 2025; Wang et al., 2026). VETRA uses a common online interface and isolates the policy that selects execution feedback.

Execution feedback supports reinforcement, candidate selection, iterative debugging, language feedback, and fine-tuning (Le et al., 2022; Chen et al., 2023; Madaan et al., 2023; Chen et al., 2024b;a; Zhang et al., 2023b; Olausson et al., 2024; Silva et al., 2025); program-repair systems also learn operators or representations across bugs (Le Goues et al., 2012; Long & Rinard, 2016; Chen et al., 2021; Lutellier et al., 2020; Berabi et al., 2021; Xia & Zhang, 2022). DARS uses failure feedback to branch and resample actions during inference (Aggarwal et al., 2025). PatchPilot couples reproduction, localization, generation, validation, and refinement in a cost-efficient rule-based workflow (Li et al., 2025). Both make evaluator use central, but they optimize a current-task search or workflow. VETRA instead asks which evaluator-allocation decisions can be carried across tasks and exchanged without changing the legal online actions.

The nearest cross-task systems reuse larger objects. ExpeRepair retrieves episodic demonstrations and semantic repair insights for test and patch agents (Mu et al., 2026); Reflexion and MPR retain traces or bundled policies (Shinn et al., 2023; Wu et al., 2025); CTIM-Rover studies the benefits and distracting context of repository memories (Lindenbauer et al., 2025). Voyager retrieves complete skills, while ReAct, LATS, and other tree-search agents allocate reasoning through trajectories (Wang et al., 2024; Yao et al., 2023; Zhou et al., 2024; Koh et al., 2025). APE, LLM-based optimization, and PromptBreeder optimize complete prompts or prompt programs (Zhou et al., 2023; Yang et al., 2024a; Fernando et al., 2024). Appendix Table 9 separates these reused objects from VETRA's intervention: exchanging Verify or Budget while preserving Observe, Diagnose, Repair, the patch generator, and the evaluator action set.

3 PROBLEM SETTING

A fixed code state, a change request expressed in natural language, and an evaluator providing build or test feedback constitute the starting point for each task τ. Actions are distinguished by whether they only read information, as in code search or file inspection, or call the evaluator to execute the harness. In the measured worker configuration, evaluator execution dominates wall-clock cost. The main budget B counts feedback-bearing evaluator opportunities: a targeted test, affected-module suite, full regression suite, or build each consumes one call. This unit models how many execution outcomes the agent may observe; Budget-All separately counts read-only tool use.

For attempt t, the agent's maintained debugging policy is g_t, and its proposed patch is p_t. Its history $h_t = (\tau, p_{0:t}, f_{0:t}, b_t)$ records the task, patch sequence, observed feedback, and remaining evaluator allowance. A feedback-allocation policy $\pi^{\text{fb}}(h_t)$ selects the next test or build target, its breadth, the calls reserved for later coverage, and the stopping condition. The selected evaluator action returns f_{t+1}, such as failing tests, stack traces, or build diagnostics, and updates the next repair decision. The operating objective is

$$\max_{\pi} \ \mathbb{E}_{\tau}[\mathbf{1}\{\text{valid repair of } \tau\}] \quad \text{subject to} \quad N_{\text{eval}}(\tau) \leq B. \tag{1}$$

We define *success* as returning a patch that passes the task evaluator after at most B agent-requested evaluator actions, and *integrity-aware success* as success with every admissibility rule satisfied. The initial reproduction supplies f_0 before the budgeted loop, and the final score-only run returns no feedback to the agent.

Three estimands keep cross-task reuse distinct from bank learning. For a method with persistent prior-task state, *empty-state transfer gain* is the held-out difference from its standard empty initialization while preserving the within-task loop, prompt skeleton, and evaluator opportunities. The *refinement effect* compares a refined bank with its matched initialized bank. The factorial *typed-field effect* compares the operational typed package—schema-constrained records, field-wise composition, and type-aligned refinement operators—with capacity-matched bundled control at the other factor levels. Static Prompting, Self-Repair, and LATS have no cross-task state in our implementations, so they receive no empty-state transfer gain and remain end-to-end performance controls.

Bundled-memory baselines bind π^{fb} to a complete reflection or repair policy. VETRA factorizes evidence gathering, diagnosis, repair, verification, and budget allocation, making π^{fb} transportable across tasks and directly intervenable at field level.

4 VETRA: COMPOSABLE EVALUATOR CONTROL

Verification and Evaluator-Control Transfer for Repository Agents (VETRA) operationalizes the feedback-allocation policy π^{fb} from Section 3. Its central object is an executable cross-task record whose evidence gathering, diagnosis, repair, verification, and evaluator allocation fields can be retrieved, composed, and intervened on independently.

4.1 TYPED FEEDBACK-ALLOCATION BANK

Each bank record is

$$g = (F_O, F_D, F_R, F_V, F_B; m),$$

where $F_O, \ldots, F_B$ are Observe, Diagnose, Repair, Verify, and Budget fields and m stores `policy_id`, failure family, source split, refinement generation, fitness, and admissibility tags. Fields are serialized key–value records rather than unconstrained prose. Table 1 gives the required keys and legal outputs; omitted keys or values outside these vocabularies make a record inadmissible. A complete serialized record and composition trace appear in Appendix Table 10. Appendix Figure 2 sketches the resulting workflow.

Table 1 summarizes the slots.

Canonical serialization fixes the field order to Observe, Diagnose, Repair, Verify, and Budget, sorts keys, preserves declared list order, and normalizes evaluator targets. Metadata remains separate and cannot become an emitted action. This grammar makes selective reuse executable. For example, the

Table 1: Executable field grammar. Braces denote finite choices; lists are ordered.

Field	Required keys	Emitted control
Observe	`read_scope`, `parser`, `evidence_keys`	Read-only targets and structured evidence extracted from the request, trace, and top frame.
Diagnose	`family`, `hypotheses`, `require`	Ordered failure hypotheses and evidence prerequisites.
Repair	`operators`, `edit_scope`, `reject_if`	Ordered {local edit, propagation, refactor, replacement} operators and prohibited patch patterns.
Verify	`targets`, `coverage_order`, `pass_if`	Ordered {failing target, affected module, full regression} actions and pass predicates.
Budget	`reserve`, `broaden_if`, `stop_if`	Integer final-regression reserve, broadening guard, and evidence-sufficiency predicate.

composer can preserve a targeted-to-regression Verify schedule and a one-call Budget reserve while replacing an API-migration diagnosis with a build-contract diagnosis. The intervention changes the acquired feedback while leaving the patch interface and legal evaluator actions unchanged.

4.2 Failure-conditioned routing

Given task features x_τ and the current failure signature σ_t, VETRA scores each bank policy g by

$$S(g \mid \tau, f_t) = \alpha \cdot \cos\big(\phi(x_\tau), \phi(g)\big) + (1-\alpha) \cdot \cos\big(\psi(\sigma_t), \psi(g)\big), \tag{2}$$

where $\phi(\cdot)$ and $\psi(\cdot)$ embed task context and failure signatures, respectively. The task query x_τ canonically joins the issue text and failing target. The failure signature σ_t joins the build/test phase, coarse diagnostic family, and top stack frame extracted from the latest evaluator output. On the policy side, $\phi(g)$ embeds the canonical five-field record and $\psi(g)$ embeds its failure-family metadata. We use frozen all-mpnet-base-v2 embeddings for ϕ, frozen jina-embeddings-v2-base-code embeddings for ψ, L_2-normalize both vectors, and choose $\alpha = 0.4$ on the repository-disjoint validation partition. Retrieval returns eight records; for each field, the three highest-scoring parent records supply candidates. Candidates inherit the parent score and metadata. The composer first applies task-specific safety constraints, then resolves conflicts in the order Budget, Verify, Diagnose/Repair, and Observe. Within a field it chooses the highest-scoring candidate compatible with all higher-precedence choices; exact score ties are broken by `policy_id`. A Verify schedule is incompatible if it spends a reserved regression call, a Repair operator is incompatible when its precondition contradicts Diagnose, and any field carrying a prohibited admissibility tag is removed. The next compatible candidate is tried after each rejection. If no five-field tuple survives, the highest-scoring admissible whole record is used.

Training and inference both enforce admissibility. A patch is rejected by the checker if it deletes tests or skips them, weakens assertions, circumvents build or CI steps, removes checks critical to safety, or introduces mocks solely for failure suppression. The same deterministic checker is applied to every comparator before a patch can receive evaluator feedback.

Operationally, each evaluator step retrieves records from the frozen bank, composes an admissible typed policy, generates and filters one patch with the shared backend, and executes the selected evaluator action. The new failure signature updates the next retrieval query, while the Budget field preserves at least one full regression action whenever the remaining budget permits. If the loop exhausts its budget without a full-regression pass, candidate patches are ranked lexicographically by full-, module-, and target-pass indicators, then by fewer unresolved failures, parent-policy score, and fewer calls; `policy_id` breaks the final tie. Appendix Algorithm 1 gives pseudocode and Appendix Table 8 records method-specific transfer and update semantics.

4.3 Offline bank refinement

Initialization starts from 96 typed policies parsed from integrity-passing trajectories and failed trajectories that either reduce the failing-target set or change the diagnosed failure family. Parsing

must populate every required key in Table 1; records are canonicalized by sorted keys and normalized target names. Type-aligned recombination within each failure family creates a larger pool, exact canonical duplicates are removed, and fitness ranking subject to failure-family coverage selects a capacity-128 initialized bank. Within that pool, 256 tasks from 128 repositories drive refinement and 64 tasks from 32 different repositories select retrieval and evolutionary settings. Fitness is 0.55 success $+0.25$ integrity-aware success $+0.20(1 - \text{calls}/B)$; inadmissible candidates receive zero fitness. Mutation rewrites exactly one or two complete fields under the same grammar while preserving all other fields and metadata lineage; crossover exchanges complete same-type fields between parents from the same failure family and never splices field text. Schema-invalid, duplicate, or inadmissible proposals are discarded before evaluation, and the 64 highest parent-score unique proposals form the generation batch. Each generation protects 32 elite incumbents. The next capacity-128 bank is selected by fitness, then failure-family coverage, then `policy_id`, from the 128 incumbents and 64 evaluated proposals, with protected elites guaranteed to survive. Refinement stops after three generations with less than 0.25 percentage-point validation improvement or after 12 generations.

At held-out inference, MPR, typed-router-only, and VETRA share the backend, prompt skeleton and context budget, tool descriptions, patch interface, evaluator, B, verifier, admissibility filter, bank capacity, and retrieval depth; each method inserts its native reusable-control payload into the same reserved prompt block. Typed-router-only uses the same capacity-128 initialized bank and freezes it before the evolutionary schedule; only offline selection, mutation, and crossover distinguish it from VETRA. The complete mechanism grid crosses two representations (whole bundles or typed fields), two router observations (task context or task plus failure signature), and two bank states (initialized or refined). The initialized bundled/task cell is the faithful MPR adaptation: native MPR updates construct a capacity-matched bank on the refinement split, which is then frozen for held-out inference. The refined bundled cell starts from that bank and applies the same offline selection schedule used for the typed bank. Task-only routing sets the signature weight to zero; failure-conditioned bundled routing uses Equation 2 to select one whole memory; typed routing applies the same candidate scores before field-wise composition. For every representation–router pair, refinement either freezes the capacity-matched initialization or applies the selection, mutation, and crossover schedule above. The representation factor is therefore an operational system contrast: bundled construction, retrieval, composition, and refinement versus schema-constrained typed construction, field-wise composition, and type-aligned operators. The eight interventions preserve the online action space and expose the effects and interactions of that deployed typed-field package, observed failure evidence, and offline learning.

5 Experiments

5.1 Protocol

EvalRepair-320 is our repository-repair collection built from reverted bug-fix commits; the name distinguishes it from the RepoBench code-completion benchmark (Zhang et al., 2023a). A commit is eligible when the pre-fix snapshot reproduces the issue with a canonical executable command, the post-fix snapshot passes that command and its affected-module regression suite, the change request can be stated without the fix, and execution requires no network access. Exact patch hashes and duplicate issue text are removed before repository- and commit-disjoint splitting. The held-out split contains 320 tasks from 160 repositories. The other 320 tasks from 160 repositories are partitioned by repository into 256 refinement tasks from 128 repositories and 64 validation tasks from 32 repositories.

Tasks cover unit-test failures, build and compile errors, integration mismatches, performance regressions, and flaky failures. For flaky tasks, the pre-fix snapshot must fail at least two of three setup reruns and the fixed snapshot must pass three; each evaluated flaky target is run three times inside one evaluator action and passes only if all repetitions pass. Targeted, module, and full-regression commands are fixed per task before agent evaluation.

We compare Static Prompting, Self-Repair, Reflexion, LATS, Meta-Policy Reflexion (MPR), typed-router-only, and VETRA. All methods use Qwen2.5-Coder-32B-Instruct (Hui et al., 2024) with a 32,768-token context, temperature 0.2, and top-p 0.95; they also share the prompt skeleton and token allocation, read-only tools, patch and evaluator interfaces, admissibility checks, verifier, and task

Table 2: EvalRepair-320 evaluation contract.

Field	Value
Held-out test split	320 tasks from 160 repositories.
Bank construction	256 refinement and 64 validation tasks; all repositories disjoint from held-out test.
Agent-visible input	Issue text, repository snapshot, canonical failing target, and its initial failure output.
Budgeted action	One agent-requested invocation of a targeted test, affected-module suite, full regression suite, or build; the initial reproduction and final score-only run provide no agent feedback and do not consume B.
Headline success	The returned patch passes the canonical target and required regression suite after at most B budgeted actions.
Independent checks	Hidden tests are isolated from prompts, routing, refinement, and public feedback; semantic review is performed only after trajectories are frozen.

snapshots. Each method receives $B = 8$ evaluator calls unless a budget sweep states otherwise. Runs use three seeds on workers with four NVIDIA H100 80GB GPUs, dual AMD EPYC 9654 CPUs, and 512GB RAM; the model is served in bfloat16 with tensor parallelism four. Appendix Table 8 gives each method's transferred state, reset, online update, and training source. Static Prompting uses no persistent state; Self-Repair updates only within the current task; Reflexion appends verbal feedback to prior-task trace memory; LATS spends the shared opportunities on current-task tree search; and MPR retrieves one bundled policy from its capacity-matched frozen bank. Typed-router-only and VETRA share the typed interface and failure router, with the former freezing the initialized bank and the latter freezing the refined bank.

The mechanism study crosses bundled versus typed transfer, task-only versus task-plus-failure routing, and initialized versus refined banks. Typed-router-only is the typed, failure-conditioned, initialized cell; the frozen faithful MPR adaptation is the bundled, task-routed, initialized cell; VETRA is the typed, failure-conditioned, refined cell. The prompt skeleton, control-payload token block, candidate actions, and evaluation opportunities remain fixed across all eight cells.

Success is final-patch evaluator pass within budget. Integrity-aware success additionally requires all admissibility rules; hidden-test pass uses the secondary verifier. Budget-All counts every tool invocation, and Budget-All@50 reports success within 50 total calls. Headline values are three-seed means. Three-seed paired estimates use all 960 task–seed executions while resampling task identities or repository blocks with all seeds retained; canonical-seed tables are labeled explicitly and serve as outcome diagnostics. The factorial intervals use the same paired resamples across all eight cells before averaging the four within-resample contrasts for each factor. Evaluator-call medians are descriptive pooled task–seed summaries; inferential claims use paired success outcomes.

Research questions. The experiments test whether transferable evaluator control improves repair across budgets (RQ1), whether stricter validators and crossed interventions preserve and attribute the gain (RQ2), which field values carry it (RQ3), and whether the resulting ordering survives semantic review and frozen-policy transfer (RQ4).

5.2 RQ1: Does evaluator-policy transfer improve repair under tight budgets?

Table 3 establishes the matched $B = 8$ result.

The 38.7-point VETRA–MPR difference and four-call median reduction are full-system effects: VETRA changes the transferred object, routing observation, and refinement procedure, and its fields jointly influence patch and evaluator selection. The factorial below attributes effects to those three design factors; the equal-cardinality study separately tests whether Verify+Budget carries more reusable value than the other evaluated pairs.

Figure 1 shows the response across the complete evaluator-budget range. At $B = 2$, VETRA reaches 61.8%, compared with 46.3% for typed-router-only and 25.8% for MPR; at $B = 12$, the rates are

Table 3: EvalRepair-320 results at $B = 8$. Empty-state gain is reported only for methods with persistent prior-task state and uses the method-specific resets in Appendix Table 8.

Method	Success	Median eval. calls	Empty-state gain (pp)
Static	27.5	8	n/a
Self-Repair	33.6	8	n/a
Reflexion	36.9	7	+5.5
LATS	44.2	7	n/a
MPR	46.0	6	+12.0
Typed-router-only	72.4	3	+24.6
VETRA (ours)	84.7	2	+35.4

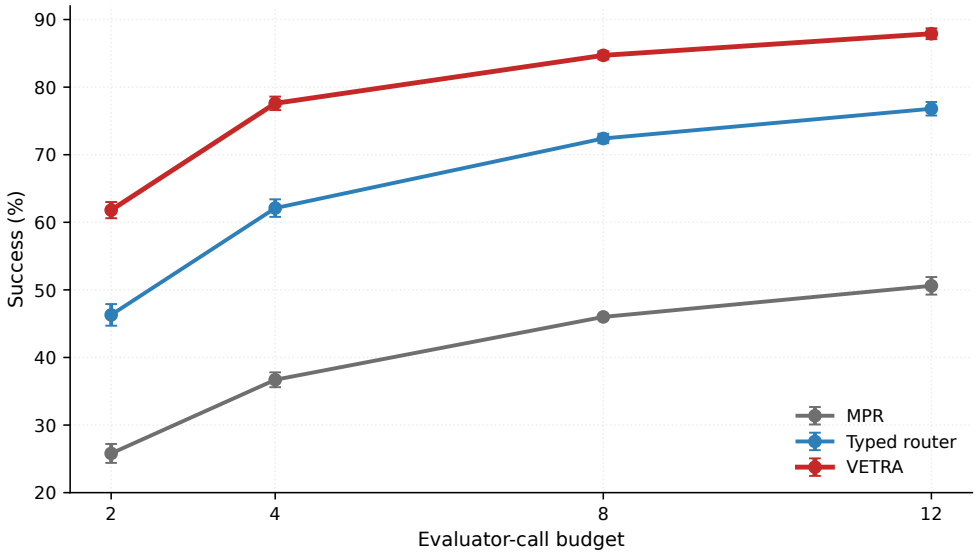


Figure 1: Success across evaluator budgets on the same 320 EvalRepair-320 tasks (three-seed mean; bands show one standard deviation). The absolute VETRA–MPR gap peaks at $B = 4$, and the ordering persists through $B = 12$. Exact values and three-seed paired intervals appear in Appendix Tables 20 and 21.

87.9%, 76.8%, and 50.6%. On the stated worker, evaluator calls account for 71.6% of end-to-end time, with 18.7 s median latency (IQR 11.4–31.9 s) versus 0.82 s for a read-only action.

5.3 RQ2: Does the gain survive stricter scoring, and where does it enter?

Table 4 applies stricter correctness and accounting to the closest controls.

Table 4: EvalRepair-320 credibility checks under stricter validators and accounting.

Method	Int.-aware	Hidden-test	B-All@50	Median total calls
MPR	45.6	41.5	43.2	55
Typed-router-only	70.8	67.1	69.4	31
VETRA (ours)	83.9	79.5	82.1	28

The full-system ordering survives every validator. VETRA retains a 38.0-point hidden-test advantage over MPR, while Budget-All preserves the ordering when read-only actions are counted. Hidden-test targets are fixed during benchmark construction and remain isolated from prompts, routing, refinement, and every feedback-bearing evaluator action.

Table 5: Complete $2 \times 2 \times 2$ intervention on EvalRepair-320 at $B = 8$. Every cell shares the backend, online action space, bank capacity, retrieval depth, verifier, and three seeds.

Transfer object	Router observation	Refined	Success (%)
Bundled	Task	No	46.0
Bundled	Task	Yes	56.8
Bundled	Task + failure	No	50.7
Bundled	Task + failure	Yes	61.3
Typed fields	Task	No	60.8
Typed fields	Task	Yes	65.9
Typed fields	Task + failure	No	72.4
Typed fields	Task + failure	Yes	84.7

Table 5 crosses every design factor and supports factor-wise attribution. Averaged over the four matched contrasts per factor and all three seeds, the operational typed-field package adds 17.3 points (task/repository 95% CI [14.2,20.4]/[13.1,21.6]), observed failure signatures add 9.9 ([7.4,12.5]/[6.2,13.7]), and bank refinement adds 9.7 ([7.1,12.2]/[5.9,13.5]). The printed cell means expose how these choices reinforce one another. Difference-in-differences averaged over the third factor are $+10.6$ points for typed package $\times$ failure routing, -2.0 for typed package $\times$ refinement, and $+3.5$ for failure routing $\times$ refinement; the three-way contrast is $+7.4$ points. Thus failure-conditioned routing has its largest matched difference in the refined typed cell, and refinement has its largest difference in the typed, failure-conditioned cell. As a canonical-seed diagnostic, typed-router-only beats MPR on 108 tasks and loses on 25 (+25.9 points over 320), while VETRA beats typed-router-only on 66 and loses on 25 (+12.8 points). The primary three-seed VETRA–MPR paired estimate is +38.6 points over 960 task–seed executions, with task/repository intervals [32.2,45.1]/[30.8,46.5]; Appendix Tables 21, 12, and 15 separate the three-seed and canonical-seed views.

5.4 RQ3: Where does transferable value concentrate?

We define the standard held-out condition as the 320-task, 160-repository EvalRepair-320 test split. The shifted held-out condition contains 160 tasks from 80 additional repositories, is repository- and commit-disjoint from both training and standard test sets, uses later snapshots, and increases the share of build, integration, and performance failures. Table 6 reports three-seed means; the gap is standard minus shifted success.

Removing Verify or Budget enlarges the distribution-shift penalty by 3.9 or 2.9 points. Equal-cardinality controls show that field semantics drive this localization.

Table 6: Within-VETRA slot reuse on the standard and shifted held-out EvalRepair-320 conditions. The EvalRepair-refined bank is frozen and receives no shifted-set tuning or update.

Setting	Standard (%)	Shifted (%)	Gap (pp)
VETRA (full reuse)	84.7	73.2	11.5
No Verify reuse	81.2	65.8	15.4
No Budget reuse	80.5	66.1	14.4
No field reuse	72.3	54.1	18.2

Table 7: Matched two-field transfer under $B \in \{4, 8\}$. These four prespecified pairs use the same bank, router, backend, tools, split, and three seeds.

Reusable fields	$B = 4$	$B = 8$	Int. $B = 8$	Calls	Shifted
Full VETRA	77.6	84.7	83.9	2	73.2
Verify + Budget	73.4	80.9	79.6	3	70.4
Observe + Diagnose	55.8	66.7	64.2	4	55.1
Diagnose + Repair	59.6	69.8	67.5	4	58.7
Observe + Verify	63.9	72.6	70.9	3	61.8

At $B = 4$, Verify+Budget exceeds the strongest of the three evaluated alternative pairs by 9.5 points and also has the highest shifted-condition result among these pairs. Canonical pairwise outcomes, reported in Appendix Table 19, include nonzero losses for every comparison and task- and repository-level intervals above zero. The nested component ladder in Appendix Table 17 complements the equal-cardinality comparison: Verify alone recovers 69% of the measured full-system gain, and adding Budget raises recovery to 86%. Together, the two interventions localize high-value transfer to verification scheduling and its call reserve without treating all ten possible field pairs as observed.

5.5 RQ4: Does the ordering survive semantic review and frozen-policy transfer?

The blinded issue-satisfaction audit yields 87.5% semantic precision and 74.1% adjusted success for VETRA, versus 30.7% [26.6,34.3] for MPR. The EvalRepair-refined bank and router are then frozen before every external run. The VETRA/typed-router-only/MPR ordering is 56.9%/52.4%/37.8% on SWE-bench Lite, 46.2%/41.8%/29.6% on Verified-240, and 77.6%/66.1%/43.8% with DeepSeek-Coder-V2-Instruct. Appendix Tables 22, 23, and 24 give the audit rubric, raw denominators, paired outcomes, uncertainty, and evaluator-use accounting.

6 Discussion

At $B = 8$, VETRA raises EvalRepair-320 success from 46.0% for bundled MPR to 84.7% while reducing descriptive median evaluator use from 6 to 2 calls. The crossed grid attributes the full-system difference to the deployed typed-field package, observed-failure routing, and refinement rather than to a single endpoint.

Verify alone recovers 69% of the nested sweep's measured gain, and Verify+Budget recovers 86% and leads all three prespecified equal-cardinality alternatives. Stricter validators, the shifted split, frozen-bank SWE-bench transfer, and a second backbone preserve this bounded operational result.

7 Conclusion

VETRA identifies verification order and call reservation as transferable control that improves repository repair under scarce execution feedback.

AI USE STATEMENT

Generative AI tools assisted the manuscript-preparation workflow, including drafting and revising prose, formatting the manuscript, checking citations, and checking the compiled layout. The experimental system uses Qwen2.5-Coder-32B-Instruct for policy-conditioned patch generation, as described in Sections 4 and 5; this use is part of the reported methodology.

REFERENCES

Vaibhav Aggarwal, Ojasv Kamal, Abhinav Japesh, Zhijing Jin, and Bernhard Schölkopf. DARS: Dynamic action re-sampling to enhance coding agent performance by adaptive tree traversal. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19808–19855, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.973. URL https://aclanthology.org/2025.acl-long.973/.

Berkay Berabi, Jingxuan He, Veselin Raychev, and Martin Vechev. TFix: Learning to fix coding errors with a text-to-text transformer. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 780–791, 2021. URL https://proceedings.mlr.press/v139/berabi21a.html.

Angelica Chen, Jérémy Scheurer, Jon Ander Campos, Tomasz Korbak, Jun Shern Chan, Samuel R. Bowman, Kyunghyun Cho, and Ethan Perez. Learning from natural language feedback. *Transactions on Machine Learning Research*, 2024a. ISSN 2835-8856. URL https://openreview.net/forum?id=xo3hI5MwvU.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. CodeT: Code generation with generated tests. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ktrw68Cmu9c.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=KuPixIqPiq.

Zimin Chen, Steve James Kommrusch, Michele Tufano, Louis-Noël Pouchet, Denys Poshyvanyk, and Martin Monperrus. SEQUENCER: Sequence-to-sequence learning for end-to-end program repair. *IEEE Transactions on Software Engineering*, 47(9):1943–1959, 2021. doi: 10.1109/TSE.2019.2940179. URL https://doi.org/10.1109/TSE.2019.2940179.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. SWE-Bench Pro: Can AI agents solve long-horizon software engineering tasks?, 2025. URL https://openreview.net/forum?id=9R2iUHhVfr. Submitted to ICLR 2026.

Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 13481–13544, 2024. URL https://proceedings.mlr.press/v235/fernando24a.html.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024. doi: 10.48550/arXiv.2409.12186. URL https://arxiv.org/abs/2409.12186.

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 *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Jing Yu Koh, Stephen Marcus McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents. *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=kpL66Mvd2a.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. CodeRL: Mastering code generation through pretrained models and deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 21314–21328, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/8636419dea1aa9fbd25fc4248e702da4-Abstract-Conference.html.

Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. GenProg: A generic method for automatic software repair. *IEEE Transactions on Software Engineering*, 38(1):54–72, 2012. doi: 10.1109/TSE.2011.104. URL https://doi.org/10.1109/TSE.2011.104.

Hongwei Li, Yuheng Tang, Shiqi Wang, and Wenbo Guo. PatchPilot: A cost-efficient software engineering agent with early attempts on formal verification. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 35922–35941. PMLR, 2025. URL https://proceedings.mlr.press/v267/li25cf.html.

Tobias Lindenbauer, Georg Groh, and Hinrich Schütze. From knowledge to noise: CTIM-Rover and the pitfalls of episodic memory in software engineering agents. In *Proceedings of the 1st Workshop for Research on Agent Language Models (REALM 2025)*, pp. 411–427, 2025. doi: 10.18653/v1/2025.realm-1.30. URL https://aclanthology.org/2025.realm-1.30/.

Fan Long and Martin Rinard. Automatic patch generation by learning correct code. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 298–312, 2016. doi: 10.1145/2837614.2837617. URL https://doi.org/10.1145/2837614.2837617.

Thibaud Lutellier, Hung Viet Pham, Lawrence Pang, Yitong Li, Moshi Wei, and Lin Tan. CoCoNuT: Combining context-aware neural translation models using ensemble for program repair. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 101–114, 2020. doi: 10.1145/3395363.3397369. URL https://doi.org/10.1145/3395363.3397369.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Fangwen Mu, Junjie Wang, Lin Shi, Song Wang, Shoubin Li, and Qing Wang. ExpeRepair: Dual-memory enhanced LLM-based repository-level program repair. *Proceedings of the ACM on Software Engineering*, 3(FSE):3954–3976, 2026. doi: 10.1145/3808181. URL https://doi.org/10.1145/3808181.

Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. Is self-repair a silver bullet for code generation? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/9ddc141bdbf9d1db510cefff56c586ad-Abstract-Conference.html.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

André Silva, Sen Fang, and Martin Monperrus. Repairllama: Efficient representations and fine-tuned adapters for program repair. *IEEE Transactions on Software Engineering*, 51(8):2366–2380, 2025. doi: 10.1109/TSE.2025.3581062. URL https://doi.org/10.1109/TSE.2025.3581062.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. URL https://mlanthology.org/tmlr/2024/wang2024tmlr-voyager/.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=OJd3ayDDoF.

Yunkun Wang, Yue Zhang, Guochang Li, Chen Zhi, Binhua Li, Fei Huang, Yongbin Li, and Shuiguang Deng. Inspectcoder: Dynamic analysis-driven self repair through interactive LLM-debugger collaboration. *Proceedings of the ACM on Programming Languages*, 10(OOPSLA1): 1041–1069, 2026. doi: 10.1145/3798238. URL https://doi.org/10.1145/3798238.

Chunlong Wu, Ye Luo, Zhibo Qu, and Min Wang. Meta-policy reflexion: Reusable reflective memory and rule admissibility for resource-efficient llm agent. *arXiv preprint arXiv:2509.03990*, 2025. doi: 10.48550/arXiv.2509.03990. URL https://arxiv.org/abs/2509.03990.

Chunqiu Steven Xia and Lingming Zhang. Less training, more repairing please: Revisiting automated program repair via zero-shot learning. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 959–971, 2022. doi: 10.1145/3540250.3549101. URL https://doi.org/10.1145/3540250.3549101.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying LLM-based software engineering agents. *Proceedings of the ACM on Software Engineering*, 2(FSE):801–824, 2025. doi: 10.1145/3715754. URL https://doi.org/10.1145/3715754.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=Bb4VGOWELI.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652, 2024b. doi: 10.52202/079017-1601. URL https://papers.nips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

John Yang, Carlos E. Jimenez, Alex L. Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik R. Narasimhan, Diyi Yang, Sida I. Wang, and Ofir Press. SWE-bench multimodal: Do AI systems generalize to visual software domains? In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=riTiq3i21b.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. Repocoder: Repository-level code completion through iterative retrieval and generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2471–2484, 2023a. doi: 10.18653/v1/2023.emnlp-main.151. URL https://aclanthology.org/2023.emnlp-main.151/.

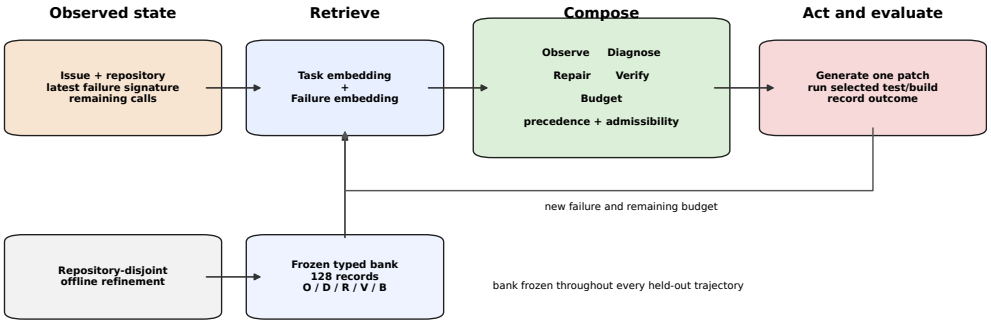


Figure 2: Repository-disjoint refinement produces a typed policy bank and freezes it before evaluation. At inference, task and failure embeddings retrieve records; deterministic composition selects compatible fields; Verify and Budget jointly choose the next evaluator action and reserve.

Kechi Zhang, Zhuo Li, Jia Li, Ge Li, and Zhi Jin. Self-edit: Fault-aware code editor for code generation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 769–787, 2023b. doi: 10.18653/v1/2023.acl-long.45. URL https://aclanthology.org/2023.acl-long.45/.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. AutoCodeRover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 1592–1604, 2024. doi: 10.1145/3650212.3680384. URL https://doi.org/10.1145/3650212.3680384.

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. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62138–62160, 2024. URL https://proceedings.mlr.press/v235/zhou24r.html.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=92gvk82DE-.

Reproducibility Statement

Sections 4 and 5 specify the typed policy representation, fixed online loop, repository-disjoint splits, evaluator budgets, comparators, metrics, and uncertainty procedures. The appendix documents the algorithms, hyperparameters, benchmark composition, paired outcomes, and robustness analyses supporting the reported comparisons.

A Algorithms and Implementation Details

The online and offline algorithms expose the two boundaries used by every intervention. Algorithm 1 consumes a frozen bank and updates only the current task state; Algorithm 2 constructs that bank on repository-disjoint tasks and freezes it before held-out evaluation.

Algorithm 1 Frozen-bank online repair loop

Require: Task τ, frozen bank $\mathcal{G}$, input failure f_0, evaluator budget B

Set $b \leftarrow B$, derive σ_0, and initialize candidate set $\mathcal{P} \leftarrow \varnothing$
while $b > 0$ **do**
  Retrieve eight records using $S(g \mid \tau, f_t)$; retain the top three parents per field
  Apply task constraints; select compatible Budget, Verify, Diagnose/Repair, and Observe candidates in precedence order
  Break score ties by `policy_id`; fall back to the highest-scoring admissible whole record if composition fails
  Generate one patch with the shared backend; reject prohibited diffs and commands
  Select the evaluator action prescribed by Verify subject to Budget
  Execute it, add the patch and outcomes to $\mathcal{P}$, update f_{t+1}, σ_{t+1}, and decrement b
  if the admissible patch passes the prescribed regression action **then**
    RETURN patch
  end if
end while
Rank $\mathcal{P}$ by full/module/target pass, fewer failures, policy score, fewer calls, then `policy_id`
RETURN the highest-ranked admissible patch

The online loop makes evaluator allocation explicit at every attempt: retrieval precedes composition, Verify selects the action, and Budget constrains both reserves and stopping. The offline loop below changes complete policy fields while preserving the online action set used by Algorithm 1.

Algorithm 2 Repository-disjoint offline bank refinement

Require: Initial typed bank $\mathcal{G}_0$, refinement tasks, validation tasks

Parse 96 schema-complete trajectory policies and create type-aligned recombinations within each failure family
Canonicalize keys and target names; remove exact duplicates and schema-invalid records
Retain a fitness-ranked, failure-family-covered capacity-128 initialized bank
for generation $j = 1, \ldots, 12$ **do**
  Protect the 32 highest-fitness admissible incumbents
  Rewrite one or two whole fields for mutation; exchange whole same-type fields between same-family parents for crossover
  Remove inadmissible and duplicate proposals; evaluate the 64 highest parent-score unique candidates
  Evaluate candidates with the fixed outer loop and compute solve, integrity, and evaluator-efficiency fitness
  Select 128 entries by fitness, family coverage, and `policy_id`, preserving protected elites
  if validation improvement is below 0.25 points for three generations **then**
    stop
  end if
end for
Freeze the bank, retrieval settings, and admissibility metadata before opening held-out repositories

Task embeddings use all-mpnet-base-v2; failure signatures use jina-embeddings-v2-base-code. Both are frozen, L_2-normalized, and cached. Validation selects α from $\{0, 0.2, 0.4, 0.6, 0.8, 1\}$, retrieval depth from $\{4, 8, 12\}$, and bank capacity from $\{64, 128, 192\}$; the selected configuration is $\alpha = 0.4$, depth eight, and capacity 128. Mutations rewrite one or two fields while preserving the field schema. Crossover swaps aligned fields only. Task-specific safety constraints precede Budget reserves and stopping rules; Budget precedes Verify, and Verify precedes Diagnose/Repair.

Table 8: Method-by-method transfer protocol. Every row shares the backend, prompt allocation, tools, patch/evaluator interface, budget, verifier, and admissibility rules. "None" in the second column means that empty-state transfer gain is undefined.

Method	Cross-task state	Matched reset	Held-out update	Construction source
Static	None	n/a	None	Fixed prompt
Self-Repair	None	n/a	Current-task reflection	Current task only
Reflexion	Prior-task traces	Empty trace memory	Append current-task reflection	Refinement split
LATS	None	n/a	Current-task tree search	Current task only
MPR	Bundled policy bank	Native empty initialization	None; bank frozen	Native updates on refinement split
Bundled, refined	Bundled policy bank	Same initialized bank	None; bank frozen	MPR initialization + offline refinement
Bundled + failure	Bundled policy bank	Same initialized bank	None; bank frozen	Refinement split; failure routing varied factorially
Typed-router-only	128 typed records	Schema-valid empty initialization	None; bank frozen	Typed trajectories from refinement split
Typed + task, refined	128 typed records	Same initialized bank	None; bank frozen	Offline refinement; signature weight zero
VETRA	128 typed records	Schema-valid empty initialization	None; bank frozen	Offline refinement on refinement split

The reset column defines empty-state transfer gain; it removes persistent prior-task contents while preserving the method's within-task loop. By contrast, a factorial refinement effect compares a refined cell with the corresponding initialized bank, such as $84.7 - 72.4 = 12.3$ points for the typed, failure-conditioned cell. These estimands therefore answer distinct questions and use distinct denominators.

Table 9: Operational contrast with the closest evaluator-control and experience-reuse families. "Selective V/B" means that Verify or Budget can be exchanged while the remaining repair policy and online action set are preserved.

Family	Control or reused object	Adaptation point	Selective V/B
DARS	Current-task trajectory	Failure-triggered branching	No
PatchPilot	Five-stage repair workflow	Validation and refinement	No
ExpeRepair	Episodic and semantic memory	Dynamic test/patch prompts	No
Reflexion / MPR	Trace or bundled policy	Whole-memory reuse	Indirect
VETRA	Typed O/D/R/V/B fields	Constrained field composition	Yes

Table 10: Schema-complete illustrative record and deterministic composition step. Scores demonstrate the rule and are not empirical measurements.

Component	Serialized content
Metadata	`policy_id`: `api-signature-v3`; `family`: `api_migration`; `source`: `refinement`; `generation`: 0; `fitness`: 0.78; `tags`: `no-test-edit`, `no-ci-bypass`.
Observe	`read_scope`: `top-frame`, `affected-module`; `parser`: `stack-trace`; `evidence_keys`: `callee`, `keyword`.
Diagnose	`family`: `api_migration`; `hypotheses`: `renamed-keyword`, `stale-wrapper`; `require`: `definition`, `call-sites`.
Repair	`operators`: `local-edit`, `propagation`; `edit_scope`: `affected-callers`; `reject_if`: `test-edit`, `assertion-weakening`.
Verify	`targets`: `failing-target`, `affected-module`, `full-regression`; `coverage_order`: `target-to-full`; `pass_if`: `all-selected-pass`.
Budget	`reserve`: 1; `broaden_if`: `target-pass`; `stop_if`: `full-regression-pass`.
Composition	With two calls left, the top Verify candidate (score 0.84) ends after the target and conflicts with `reserve=1`; it is rejected. The next candidate (0.79) schedules affected-module then full regression and is selected. A Repair candidate for `algorithmic-bug` is rejected against the selected Diagnose family; the 0.76 propagation candidate is used.
Emitted action	Run the affected-module suite now, retain one call for full regression, and stop only after that regression passes.

A.1 Representative repair trace

A held-out Python integration task first exposed a keyword-signature mismatch through a targeted failing test. The retrieved Observe field parsed the top stack frame and affected module; Diagnose identified an API migration and requested evidence at the migrated call boundary; Repair propagated the renamed argument across two call sites; Verify selected the targeted test followed by the affected module suite; and Budget reserved the second evaluator action for that regression pass. VETRA completed the task in 2 evaluator calls, while the matched MPR trajectory used 6: two broad reruns, two patch checks, one targeted test, and one final suite. The 4-call reduction is the concrete behavior measured by the aggregate evaluator-use columns.

B Budget-All Analysis

Budget-All counts every tool invocation, preventing evaluator savings from being offset by unreported search or inspection actions.

Table 11: Budget-All analysis on EvalRepair-320.

Method	Succ.	B-All@50	B-All@100	Median total calls
Static	27.5	25.8	27.2	48
Reflexion	36.9	34.5	36.1	52
LATS	44.2	40.1	43.5	61
MPR	46.0	43.2	45.8	55
Typed-router-only	72.4	69.4	71.8	31
VETRA	84.7	82.1	84.5	28

Counting every tool invocation preserves the main ordering: VETRA reaches 82.1 Budget-All@50 with 28 median total calls, compared with 43.2 and 55 for MPR. The evaluator-call gain is therefore not displaced into a larger volume of repository reads or searches.

C Paired Task-Level Evidence

Every row in Table 12 uses the same 320 aligned seed-0 tasks. The four outcome counts are VETRA-only solve, comparator-only solve, shared solve, and shared failure; they sum to 320, and the paired delta is $(n_{\mathrm{V\ only}} - n_{\mathrm{C\ only}})/320$.

Table 12: Canonical-seed paired evidence for VETRA on 320 EvalRepair-320 tasks. CIs are task-bootstrap and repository-block 95% intervals from 10,000 resamples.

Comparison	Delta	Task / repo CI	Only wins/losses; shared S/F	McNemar p
VETRA vs Static	+57.2	[50.9,63.5] / [48.8,64.1]	197/14; 74/35	$< 10^{-20}$
VETRA vs Reflexion	+47.8	[41.2,54.5] / [39.8,56.0]	172/19; 99/30	$< 10^{-20}$
VETRA vs LATS	+40.6	[34.1,47.2] / [32.4,49.1]	150/20; 121/29	$< 10^{-20}$
VETRA vs MPR	+38.8	[32.0,45.5] / [30.5,46.9]	147/23; 124/26	$< 10^{-20}$

The MPR row, for example, contains 147+23+124+26=320 tasks and yields (147-23)/320=38.8 percentage points after one-decimal rounding.

D Fully Crossed Mechanism and Paired Controls

Table 13: Complete $2 \times 2 \times 2$ mechanism grid at $B = 8$. Solves give seeds 0/1/2 on 320 tasks each; calls count all tool invocations.

Object	Router	Ref.	Solves	Succ.	Int.	Hidden	B-All@50	Calls
Bundled	Task	No	147/147/148	46.0	45.6	41.5	43.2	55
Bundled	Task	Yes	182/178/185	56.8	55.5	51.2	53.7	47
Bundled	Task + failure	No	161/164/162	50.7	49.8	45.9	47.6	50
Bundled	Task + failure	Yes	195/198/195	61.3	60.0	56.4	58.1	43
Typed	Task	No	194/192/198	60.8	59.4	55.7	57.6	40
Typed	Task	Yes	211/210/212	65.9	64.4	60.6	62.8	36
Typed	Task + failure	No	230/231/234	72.4	70.8	67.1	69.4	31
Typed	Task + failure	Yes	271/273/269	84.7	83.9	79.5	82.1	28

The average matched main effects on success are +17.3 points for the operational typed-field package, +9.9 for task-plus-failure routing, and +9.7 for refinement. Their task/repository 95% CIs are [14.2,20.4]/[13.1,21.6], [7.4,12.5]/[6.2,13.7], and [7.1,12.2]/[5.9,13.5], respectively. Using the printed cell means, the averaged two-way difference-in-differences are $+10.6$, -2.0, and $+3.5$ points for typed package $\times$ failure routing, typed package $\times$ refinement, and failure routing $\times$ refinement; the three-way contrast is $+7.4$ points.

Table 14: Canonical-seed paired factorial controls on 320 tasks. Counts are first-only/second-only; shared success/shared failure.

Comparison	Delta	Counts; shared S/F	Task / repo 95% CI	McNemar p
VETRA vs refined bundled	+27.8	105/16; 166/33	[21.7,33.8] / [20.1,35.7]	$< 10^{-12}$
VETRA vs task-context typed	+18.8	83/23; 188/26	[13.0,24.5] / [11.4,26.2]	$< 10^{-8}$

The two canonical contrasts retain the treatment changes represented in the averaged factorial effects: representation differs against the refined bundled bank, while router observation differs against task-context typed routing.

Table 15: Canonical-seed paired evidence for typed-router-only on 320 EvalRepair-320 tasks. The printed delta is computed from the same seed-0 outcome counts.

Comparison	Delta	Only W/L; shared S/F	Task / repo 95% CI	McNemar p
Typed-router-only vs MPR	+25.9	108/25; 122/65	[19.4,32.5] / [18.1,34.0]	$< 10^{-12}$
VETRA vs typed-router-only	+12.8	66/25; 205/24	[7.2,18.5] / [5.8,20.1]	2.0×10^{-5}

E Field Localization

Three similarly named controls answer different questions. The switch matrix below is part of the intervention definition; values from different rows are not interchangeable baselines.

Table 16: Control semantics across the nested-component and transfer-ablation studies.

Control	Cross-task field values	Router/bank machinery	Estimand
Frozen control	Selective replacement disabled	Prerefinement checkpoint fixed for the nested sweep	Gain from successively enabling named components
Matched no-transfer	Reset to each variant's standard empty initialization	Within-task loop preserved	Gain from all prior-task state in that variant
No field reuse	O/D/R/V/B regenerated on the target task	Learned router and refined-bank machinery preserved	Loss from removing transferred field contents only

This matrix separates a nested component sweep from the method-specific prior-state reset and from the field-content ablation used below.

Table 17: Minimal necessary set analysis on EvalRepair-320. The rightmost column reports recovery of the full measured gain.

Enabled components	Success	Gain over frozen	Full-gain recovered
Frozen control	57.5	–	0%
Verify	76.2	+18.7	69%
Verify + Budget	80.9	+23.4	86%
Verify + Budget + Retrieval	82.8	+25.3	93%
Full VETRA	84.7	+27.2	100%

Verify recovers 69% of this nested sweep's measured gain, and adding Budget raises recovery to 86% before retrieval and the remaining fields are enabled.

Table 18: Full within-system ablation matrix at $B = 8$. Reset is the matched no-transfer success; gain is success minus reset.

Variant	Success	Reset	Median eval. calls	Gain
VETRA	84.7	49.3	2	+35.4
No crossover	80.1	51.5	3	+28.6
No mutation	75.6	53.2	3	+22.4
No retrieval	68.9	53.7	4	+15.2
No field reuse	72.3	59.5	4	+12.8
Retrieval only	70.5	54.0	4	+16.5
Verify + Budget only	80.9	50.8	3	+30.1

The full ablation varies one mechanism at a time, whereas the following equal-cardinality comparison changes which two field values are reusable while preserving the shared machinery.

Table 19: Equal-cardinality alternatives paired against Verify+Budget on 320 aligned tasks. Counts are Verify+Budget-only/alternative-only; shared success/shared failure.

B	Alternative	Only W/L	Shared S/F	Delta	Task / repo 95% CI
4	Observe + Diagnose	74/18	161/67	+17.5	[12.0,23.0] / [10.4,24.8]
4	Diagnose + Repair	68/24	167/61	+13.8	[8.4,19.1] / [6.8,20.9]
4	Observe + Verify	57/27	178/58	+9.4	[4.3,14.5] / [2.6,16.3]
8	Observe + Diagnose	61/16	198/45	+14.1	[9.2,19.0] / [7.5,20.8]
8	Diagnose + Repair	53/17	206/44	+11.3	[6.4,16.1] / [4.9,17.7]
8	Observe + Verify	46/19	213/42	+8.4	[3.7,13.1] / [2.0,14.9]

At $B = 8$, matched no-transfer success is 50.8, 49.2, 50.0, and 50.5 for Verify+Budget, Observe+Diagnose, Diagnose+Repair, and Observe+Verify. The corresponding gains are +30.1, +17.5, +19.8, and +22.1 points.

F BUDGET FRONTIER AND EVALUATOR COST

Table 20: Budget frontier on 320 tasks and three seeds. Success is mean ± sample standard deviation; solves give seeds 0/1/2.

Method	B	Success	Raw solves	Int.-aware	Median eval. calls
MPR	2	25.8±1.4	78/83/87	24.4	2
MPR	4	36.7±1.1	114/117/121	35.1	4
MPR	8	46.0±0.2	147/147/148	45.6	6
MPR	12	50.6±1.3	158/162/166	49.8	8
Typed-router-only	2	46.3±1.6	143/148/153	44.5	2
Typed-router-only	4	62.1±1.3	195/198/203	60.2	3
Typed-router-only	8	72.4±0.7	230/231/234	70.8	3
Typed-router-only	12	76.8±1.0	243/245/249	75.5	4
VETRA	2	61.8±1.2	195/196/202	59.7	2
VETRA	4	77.6±1.0	245/249/251	75.3	2
VETRA	8	84.7±0.6	271/273/269	83.9	2
VETRA	12	87.9±0.8	279/281/284	86.8	3

The raw seed counts above define the means used in the main budget curve. The paired table below instead treats the task–seed execution as the outcome while retaining all three seeds whenever a task or repository block is resampled.

Table 21: Three-seed paired VETRA–MPR budget effects across 960 aligned task-seed runs. Counts are VETRA-only/MPR-only; shared success/failure. CIs resample task identities or repository blocks with all seeds retained.

B	Counts; shared S/F	Delta	Task / repo 95% CI
2	393/48; 200/319	+35.9	[29.7,42.2] / [28.0,44.0]
4	445/52; 300/163	+40.9	[34.4,47.3] / [32.8,49.1]
8	440/69; 373/78	+38.6	[32.2,45.1] / [30.8,46.5]
12	394/36; 450/80	+37.3	[30.8,43.7] / [29.0,45.4]

Success-versus-budget AUC is 42.1 for MPR, 67.6 for typed-router-only, and 80.9 for VETRA. Table 21 shows broad paired gains at every budget; the $B = 8$ canonical-seed analysis in Table 12 independently gives +38.8 points.

On the four-H100 worker described in Section 5, evaluator execution consumes 71.6% of end-to-end time. Evaluator latency has median 18.7 s and IQR 11.4–31.9 s; read-only actions have median 0.82 s, for a 22.8× latency ratio.

G REPOSITORY-LEVEL EFFECT DISTRIBUTION

The canonical-seed repository aggregation is reported separately from the three-seed task estimator to avoid mixing analysis bases. Of the 160 held-out repositories, 119 have positive VETRA–MPR differences, 18 are tied, and 23 are negative. Repository-level quartiles are +16.7/+33.3/+50.0 points and the worst decile is -16.7. The 119 positive repositories establish a broad aggregate advantage, while the negative tail supplies the heterogeneity reflected by the repository-block interval in Table 21.

H EXPANDED SEMANTIC AUDIT

We proportionally stratify 120 claimed solves per method by failure type, repository size, and patch type. Two research-team raters who did not run the evaluated trajectories independently label each solve correct, partial, wrong, or inconclusive while blinded to method identity; disagreements are adjudicated under the fixed issue-satisfaction rubric. No external participants are recruited or compensated, and the review concerns generated patches rather than human-subject data. Correct requires the requested behavior and absence of an issue-relevant regression; partial denotes an incomplete request; wrong denotes an evaluator pass that fails the request; and inconclusive denotes insufficient repository evidence for a decision. For stratum h, population count N_h, sample count n_h, and correct count c_h, semantic precision is $\hat{q}_m = \sum_h (N_h/N)(c_h/n_h)$. Proportional allocation yields the printed aggregate proportions. Adjusted success is $\mathrm{Success}_m \hat{q}_m$; its displayed audit-component interval multiplies the fixed headline success by the Wilson endpoints for $\hat{q}_m$ and does not add run-to-run uncertainty.

Table 22: Expanded semantic audit. Counts are correct/partial/wrong/inconclusive; adjusted-success intervals propagate the Wilson interval for the audited correct share.

Method	Counts	Correct share (95% Wilson)	Adj. success (audit CI)	κ	Disagr.
MPR	80/15/17/8	66.7% [57.8,74.5]	30.7 [26.6,34.3]	0.82	15
Typed-router-only	95/11/9/5	79.2% [71.1,85.5]	57.3 [51.5,61.9]	0.85	12
VETRA	105/7/5/3	87.5% [80.4,92.3]	74.1 [68.1,78.2]	0.88	9

I EXTERNAL AND CROSS-BACKBONE VALIDATION

The EvalRepair-refined policy banks, retrieval models, α, depth, composition rules, and stopping thresholds are frozen before every external run. SWE-bench Lite and Verified-240 contribute no task to initialization, refinement, validation, or hyperparameter selection. The DeepSeek replication retains the same text policies and router and changes only the patch-generation backbone. SWE-bench Lite uses all 300 tasks under the same $B = 8$, tools, patch interface, integrity rules, and total-call accounting as EvalRepair-320. For both SWE-bench settings, official FAIL_TO_PASS tests define the canonical target and initial failure, affected-package tests define the module action, and the official harness defines full regression and final scoring. Initial reproduction is supplied without debiting B; every subsequent feedback-bearing evaluator invocation consumes one call.

Table 23: Frozen-bank external evaluation on 300 SWE-bench Lite tasks. Success columns are three-seed means; paired counts use the predeclared canonical seed.

Method	Success	Int.-aware	B-All@50	Median total calls	Seed-0 W/L; shared S/F vs MPR
MPR	37.8	36.1	34.6	58	–
Typed-router-only	52.4	50.3	49.1	36	78/34; 79/109
VETRA	56.9	54.8	54.0	33	91/34; 79/96

The paired deltas are +14.7 [8.0,21.3]/[6.1,23.2] points for typed-router-only versus MPR and +19.0 [12.0,26.0]/[10.3,27.9] for VETRA versus MPR, where intervals are task/project-block 95% CIs.

Table 24: Frozen-bank SWE-bench Verified-240 and cross-backbone evaluation at $B = 8$. Displayed rates are three-seed means. Verified-240 denotes the fixed 240-task setting shared by every compared method and reports B-All@50 with median total calls; DeepSeek uses all 320 EvalRepair-320 tasks and reports hidden-test success with median evaluator calls.

Setting	Method	Success	Int.-aware	Robustness metric	Call metric
SWE-bench Verified-240	MPR	29.6	28.3	26.7	61
	Typed-router-only	41.8	40.0	38.5	41
	VETRA	46.2	44.6	43.1	38
DeepSeek / EvalRepair	MPR	43.8	42.9	39.4	6
	Typed-router-only	66.1	64.7	61.5	3
	VETRA	77.6	76.4	72.8	3

On the canonical Verified seed, typed-router-only versus MPR records 46/17; 54/123 paired wins/losses and ties, while VETRA versus MPR records 59/19; 52/110; the task CIs are [5.8,18.4] and [9.8,23.6]. VETRA versus typed-router-only records 18/7; 93/122 and a +4.6 [0.6,8.6]-point paired difference. On the canonical DeepSeek seed, VETRA versus MPR records 119/11; 129/61 (+33.8 [28.0,39.5]), and VETRA versus typed-router-only records 52/16; 196/56 (+11.3 [6.5,16.0]).

J EvalRepair-320 Composition and Shift Construction

Candidate commits are traversed chronologically and retained only when a clean pre-fix checkout reproduces the linked failure and the post-fix checkout passes both that target and the affected-module suite. The natural-language task is formed from the issue or commit request without exposing the fixing diff. Tasks requiring unavailable network services, credentials, nondeterministic external state, or an unbounded performance environment are excluded. For the flaky category, setup requires at least two failures in three pre-fix reruns and three post-fix passes; evaluation executes three repetitions as one action and requires all three to pass.

Table 25: Composition of the 320-task standard held-out split. Buckets are mutually exclusive within each block.

Block	Count	Definition
Python	128 (40.0%)	Primary patched language
JavaScript/TypeScript	70 (21.9%)	Primary patched language
Java/Kotlin	54 (16.9%)	Primary patched language
C/C++	42 (13.1%)	Primary patched language
Go/Rust	26 (8.1%)	Primary patched language
Small repository	102	Below 100k source lines
Medium repository	143	100k–500k source lines
Large repository	75	Above 500k source lines
Unit-test failure	122	Initial failure taxonomy
Build/compile	67	Initial failure taxonomy
Integration mismatch	64	Initial failure taxonomy
Performance regression	39	Initial failure taxonomy
Flaky failure	28	Initial failure taxonomy

Commits span 2019–2025; exact patch size has median 18 changed lines and IQR 7–46. Exact patch hashes, repository forks, and duplicate issue text are removed before splitting, eliminating 37 duplicate candidates. Train, validation, standard test, and shifted test sets share no repository or commit. The shifted set contains 160 tasks from 80 later-snapshot repositories and raises the combined build/integration/performance share from 53.1% to 68.8%. The task input contains the canonical initial failure and therefore does not debit B. Every subsequent agent-requested target, module, regression, or build invocation debits one call. Final scoring reruns the canonical target and required regression suite without returning output to the agent; hidden tests run in a separate process after the trajectory and bank are frozen.