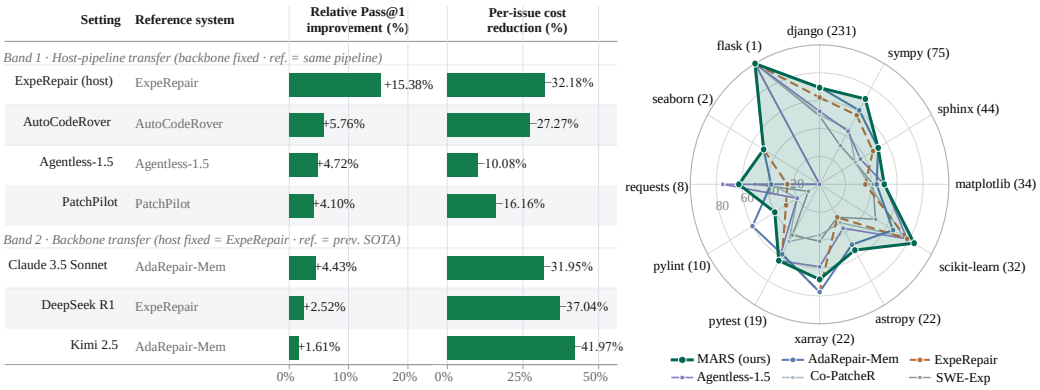
MARS: MULTI-EVIDENCE ADAPTIVE SEARCH WITH STATE-PRESERVING CONTROL FOR REPOSITORY-LEVEL PROGRAM REPAIR

Anonymous authors
Paper under double-blind review

ABSTRACT

Repository-level program repair aims to resolve real software issues under a bounded compute budget, requiring decisions about how much computation to allocate, which repair state to preserve, and what to commit. Existing staged repair pipelines largely follow fixed search schedules that repeatedly generate, execute, and refine candidate patches while underusing runtime evidence. We refer to this mismatch as Budget–Evidence Decoupling: computation is spent without conditioning on whether current evidence justifies further search, and later refinement may overwrite previously discovered useful repair states. We propose **MARS**, a training-free controller that formulates candidate search as a budget-aware sequential decision process using execution, verification, review, and resource signals. Specifically, Multi-Evidence Adaptive Search (AS) deterministically selects among verification, refinement, expansion, and termination according to the current repair state. MARS further introduces Incumbent–Challenger Control (IC), separating candidate admission from incumbent replacement, together with Candidate Commitment (CC), which freezes the accumulated candidate state for downstream final selection. These designs jointly condition compute allocation, state preservation, and commitment on runtime evidence while keeping valuable repair states recoverable. Experiments on SWE-bench Lite and SWE-bench Verified show that MARS improves pass@1 over the strongest same-backbone baseline by 2.0 and 2.8 percentage points, respectively, while reducing per-issue cost by 31% and 22%. We further demonstrate its transferability across three backbones and three additional host pipelines.

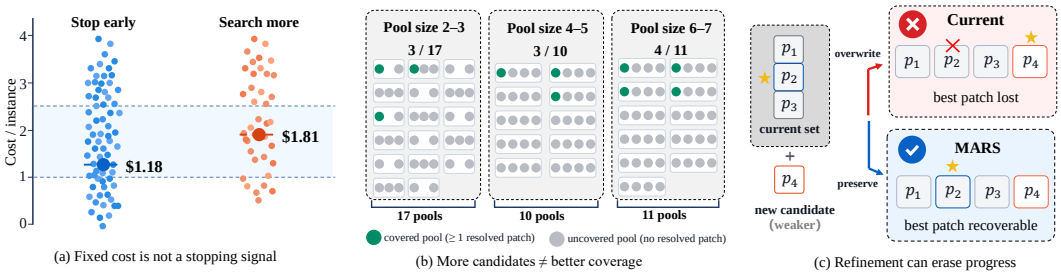


(a) Cross-pipeline and cross-backbone improvement

(b) Per-repository coverage on SWE-bench Verified

Figure 1: **Performance overview of MARS.** Bars report relative changes (%) against band-specific references: the same host's default controller in band 1 and the strongest available baseline under the fixed ExpeRepair host in band 2; main-text Pass@1 gains are percentage-point differences. Radar values are resolved-issue rates (%) by repository; parentheses give issue counts.

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.



(a) Fixed cost is not a stopping signal (b) More candidates ≠ better coverage (c) Refinement can erase progress

Figure 2: **Budget–Evidence Decoupling in repository-level repair. (a)** Similar observed cost can correspond to different search needs: runs that stop after the first plausible patch and runs that undergo repeated refinement have overlapping cost distributions. **(b)** Larger candidate pools do not guarantee resolved-patch coverage. **(c)** In-place refinement can erase the strongest state reached, whereas preserving the new patch as a challenger keeps that state recoverable.

1 Introduction

Resolving a repository-level issue requires localizing faults that may span multiple files, editing code under project-specific conventions, and validating the resulting patch against the repository's own tests (Jimenez et al., 2024). Existing systems broadly follow two paradigms: agentic systems that interleave reasoning with repository tools (Yao et al., 2022; Yang et al., 2024; Wang et al., 2025a), and procedural pipelines that execute a localization–generation–validation workflow (Xia et al., 2025; Zhang et al., 2024; Li et al., 2025). This work focuses on the latter class and related staged repair workflows, where candidate generation, execution, refinement, and validation form an explicit iterative search process (Brown et al., 2024; Madaan et al., 2023; Antoniades et al., 2025).

Yet the control surrounding this search is largely predetermined before the run reveals whether additional computation is useful. Search depth, candidate replacement, and final commitment are commonly governed by fixed schedules or stage-local rules, even though each action produces informative runtime signals: patch application and test outcomes, verification results, reviewer judgments, and accumulated cost. We refer to this mismatch as Budget–Evidence Decoupling. It matters because none of these signals alone is a reliable oracle. However, no single signal reliably identifies a correct patch. Tests may accept plausible but incorrect patches (Qi et al., 2015; Smith et al., 2015), particularly when repository test suites are incomplete (Aleithan et al., 2024); model-based judgments are inexpensive but fallible (Zheng et al., 2023); and a finite budget limits the additional evidence that can be collected (Kapoor et al., 2024). Increasing search does not resolve these limitations: resampling under an imperfect verifier can preserve false positives (Stroebl et al., 2026; Snell et al., 2025), larger candidate pools make selection more difficult (Ehrlich et al., 2025; Zhang et al., 2025), and refinement can degrade rather than improve its input (Huang et al., 2024).

We examine this decoupling in 150 SWE-bench Verified instances executed under the host pipeline's default search budget.The 150 instances are sampled using the same stratified procedure described in Appendix B.7—jointly by repository and by whether the default ExpeRepair trace stops at the first plausible patch or proceeds to refinement—and are reused unchanged throughout this analysis; no official resolved label is used in the selection. Figure 2 reveals three complementary effects. First, cost is only a weak indicator of how much search an issue requires. Among the 150 runs, 112 terminate after the first plausible patch, whereas 38 undergo repeated refinement. Their average costs under the host's default schedule are $1.18 and $1.81, although the two cost distributions overlap substantially. Second, larger candidate pools do not guarantee resolved-patch coverage. The 38 multi-patch pools are grouped by their observed pool sizes: 17 pools contain 2–3 candidates, 10 contain 4–5, and 11 contain 6–7. Only 10 of the 38 pools contain at least one officially resolved candidate; the remaining 28 pools contain 110 candidates in total but no resolved candidate. Among the 10 covered pools, 32 of their 45 candidates are officially resolved. Thus, the result does not indicate that larger pools reliably improve coverage. Instead, it suggests that the central difficulty lies not only in selecting from a mixed pool, but also in reaching and preserving a repair state that contains a resolved candidate. Third, refinement can destroy such a state: replacing a candidate in place with a weaker revision can discard the strongest patch already reached. These observations

suggest that the key question is not simply how much test-time computation to allocate, but **how runtime evidence should determine the next search action while preserving useful repair state under a finite budget**.

This perspective turns repair control into a metareasoning problem (Alomrani et al., 2025; Zhai et al., 2026): each additional proposal, verification, or refinement has a cost, competes for a shared budget (Zhai et al., 2026; McCleary & Ghawaly, 2026), and changes which candidates remain available for commitment. A useful controller must therefore couple three decisions that fixed search schedules treat separately: whether further search is warranted, whether a newly produced candidate should replace the current best state, and which candidates should remain eligible for final selection.

We address this problem with **MARS (Multi-Evidence Adaptive Search with State-Preserving Control)**. MARS formulates repository-level candidate search as a budget-aware sequential decision process (Aggarwal et al., 2025; Li & Peng, 2026), but realizes the controller as a deterministic, training-free policy: it uses no reward model, policy learning, or correctness label at inference time. At each search step, MARS observes the current candidate pool together with execution, verification, review, and resource evidence, and decides whether to verify, refine, expand, or terminate the search. Two state-preserving mechanisms complement this adaptive control. First, Incumbent–Challenger Control (IC) prevents a newly generated or refined patch from automatically overwriting the retained candidate. Second, Candidate Commitment (CC) separates search termination from final selection: once search stops, the accumulated candidate state is frozen and passed to the downstream selector without reopening proposal or refinement. The host pipeline still performs localization and patch generation; MARS controls only the search decisions around those operations.

On SWE-bench Lite and SWE-bench Verified (Jimenez et al., 2024), MARS improves pass@1 over its ExpeRepair host (Mu et al., 2026) by 4.7 and 8.8 percentage points, respectively, while reducing all-in average cost per issue by 39% and 32%. Relative to AdaRepair-Mem, the strongest baseline using the same backbone pair, the corresponding gains are 2.0 and 2.8 percentage points at 31% and 22% lower cost. The improvement is not tied to one proposal model or one repair pipeline: MARS transfers across three backbones and across AutoCodeRover, Agentless-1.5, and PatchPilot while leaving their underlying patch generators unchanged. Figure 1 summarizes these cross-pipeline and cross-backbone gains together with repository-level coverage on SWE-bench Verified.

Contributions.

- We identify Budget–Evidence Decoupling in repository-level repair and show empirically that fixed search expenditure, candidate-pool expansion, and in-place refinement can each fail to preserve useful repair progress.
- We formulate repair search control as a budget-aware sequential decision process over runtime execution, verification, review, and resource evidence, and introduce **MARS**, a deterministic, training-free controller combining Multi-Evidence Adaptive Search (AS), Incumbent–Challenger Control (IC), and Candidate Commitment (CC).
- We show that MARS improves both repair success and test-time efficiency, and that its controller transfers across three backbones and three additional host pipelines without modifying their patch generators.

2 Related Work

2.1 Repository-Level Repair Systems

Classical automated program repair generates candidate patches and filters them by tests (Le Goues et al., 2011; 2019). Tests are an incomplete oracle, so plausible patches need not be correct (Qi et al., 2015; Smith et al., 2015), a gap that persists for model-generated patches (Xia et al., 2023). SWE-bench moved the task to repository scale (Jimenez et al., 2024). At that scale, agentic systems apply the reason–act loop (Yao et al., 2022) to repository tools (Yang et al., 2024; Wang et al., 2024a; 2025a; Bouzenia et al., 2025; Ma et al., 2025; Örwall, 2024), while procedural pipelines follow a fixed localization–generation–validation decomposition (Xia et al., 2025; Zhang et al., 2024; Ruan et al., 2025; Li et al., 2025; Xie et al., 2025). Either family can draw on retrieved repair experience, workflows, or repository memory (Mu et al., 2026; Chen et al., 2025; Wang et al., 2024b; 2026a;

Luo et al., 2026). These systems determine how a patch is produced; MARS governs the subsequent search: which external operation is executed next, whether a challenger should replace the incumbent, and which accumulated candidates remain eligible for commitment. Its retained state is also narrower: MARS accumulates nothing across issues or runs, only within the issue it is solving.

2.2 Test-Time Search and Budgeted Control

Given a proposal mechanism, additional test-time computation enables repeated sampling (Wang et al., 2022; Brown et al., 2024), feedback-driven refinement (Madaan et al., 2023; Shinn et al., 2023; Chen et al., 2024), or tree search over trajectories (Kocsis & Szepesvári, 2006; Yao et al., 2023; Zhou et al., 2023; Antoniades et al., 2025; Aggarwal et al., 2025); scaling studies characterize the associated performance—cost trade-offs (Snell et al., 2025; Muennighoff et al., 2025). All three enlarge the candidate set and shift the burden to selection, handled by verifiers and process supervision (Cobbe et al., 2021; Lightman et al., 2024), model judges (Zheng et al., 2023), or a terminal ranking over the surviving pool (Ehrlich et al., 2025; Zhang et al., 2025). Overwriting a refinement's input is a property of that family, not of tree search, which retains the tree (Kocsis & Szepesvári, 2006; Antoniades et al., 2025; Aggarwal et al., 2025). MARS does not replace those procedures: it leaves the host generator in place, withholds replacement until a challenger satisfies the promotion condition, and makes pool composition part of the search.

The allocation of computational resources presents a complementary challenge. Cost-aware routing assigns queries across models under quality–cost trade-offs (Wang et al., 2025b), adaptive test-time scaling allocates reasoning effort according to query difficulty or confidence (Alomrani et al., 2025; Zhai et al., 2026), and context reduction shortens agent inputs (Wang et al., 2026b); all act upstream of the repair state. Inside it, the question becomes one of adaptive search control under a finite inference budget (Aggarwal et al., 2025). MARS brings that view to repository repair: execution, verification, and review evidence govern candidate transitions, while resource evidence determines whether further search remains admissible. The resulting trace is a deterministic state–action process over observable repair evidence, requiring no reward model, learned policy, or correctness label at inference.

3 Method

Overview. MARS is a training-free controller for staged repository-level repair. It leaves the host system's localization, patch generation, execution, refinement, and downstream selection modules unchanged, and controls the search decisions around these operations. MARS has two coupled layers. *Multi-Evidence Adaptive Search (AS)* converts heterogeneous runtime evidence into budget-aware search actions. *State-Preserving Control* combines *Incumbent–Challenger Control (IC)* with *Candidate Commitment (CC)*, separating candidate admission, active-state replacement, and the search-to-selection boundary. Figure 3 summarizes the framework; full evidence definitions and pseudocode are provided in Appendix A.

3.1 Multi-Evidence Repair State

For a repair instance x, comprising an issue report and its target repository, MARS maintains

$$s_t = \big(\mathcal{P}_t, i_t, \widetilde{p}_t, \mathcal{E}_t, \rho_t\big), \tag{1}$$

where $\mathcal{P}_t$ is the accumulated candidate pool, $i_t \in \mathcal{P}_t$ is the incumbent, $\widetilde{p}_t$ is the challenger under assessment (with $\widetilde{p}_t = \varnothing$ when none is pending), $\mathcal{E}_t$ is the accumulated evidence, and ρ_t records consumed resources. Both $\mathcal{P}_t$ and i_t are case-local: every patch is generated during the current execution of the same issue, and no prediction from another seed or run is used to initialize the state. For each candidate p, MARS records

$$\mathbf{z}_t(p) = (z_t^{\mathrm{exe}}(p), z_t^{\mathrm{ver}}(p), z_t^{\mathrm{rev}}(p)), \qquad \rho_t = \big(C_t, d_t, n_t^{\mathrm{gen}}, n_t^{\mathrm{ref}}, n_t^{\mathrm{ver}}\big). \tag{2}$$

Execution evidence records the observed reproduction outcome; verification evidence records the strong-verifier verdict; and review evidence records the reviewer's Bug Fixing Score. These are online signals used to choose search actions, not official correctness labels. The latter are obtained only from the SWE-bench evaluator after a run and are used to report pass@1 and the candidate-lifecycle diagnostics. Resource variables determine whether further computation is admissible; they do not indicate patch correctness. Appendix A.1 specifies the signal values and controller gates.

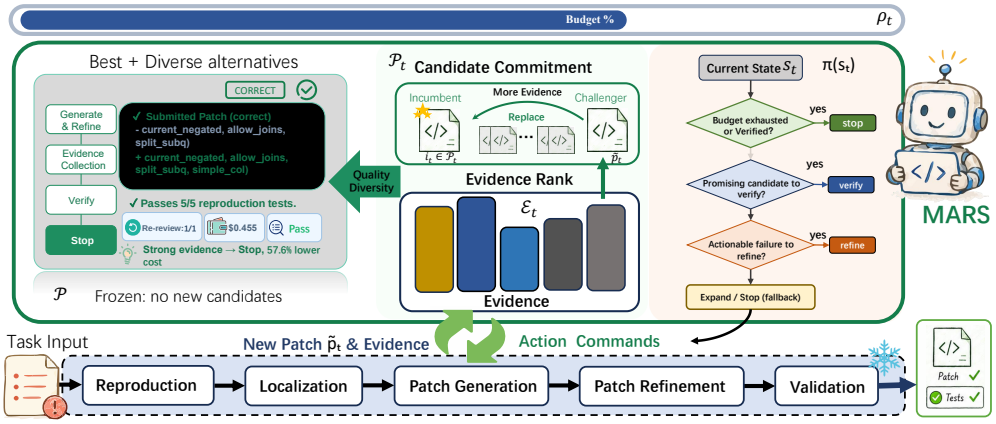


Figure 3: **Overview of MARS.** Multi-Evidence Adaptive Search controls which repair operation is executed, while Incumbent–Challenger Control preserves the active repair state and Candidate Commitment separates search termination from downstream selection.

3.2 Multi-Evidence Adaptive Search

MARS controls four external operations,

$$\mathcal{A} = \{\mathsf{verify}, \mathsf{refine}, \mathsf{expand}, \mathsf{stop}\}. \tag{3}$$

Verification evaluates the current incumbent once a review-supported candidate exists; the supported candidate acts as the trigger for this verification step, while the verifier is applied to i_t. Refinement addresses an evidenced failure; expansion opens a new repair direction on the host fallback path, where no usable reproduction test is available; and stopping freezes the search.

Let B_{cfg} denote the configured all-in cost checkpoint and D_{cfg} the maximum refinement depth. The cost cap is evaluated only at predefined controller checkpoints and is therefore a soft control limit rather than a strict per-call bound; the reported all-in average cost is defined separately in Section 4.1. At controller checkpoints, the soft cost gate together with the refinement-depth gate determines whether resource-consuming actions remain admissible. Given the semantic predicates Verified_t, Supported_t, Refinable_t, and Exhausted_t, MARS applies

$$\pi(s_t) = \begin{cases} \mathsf{stop}, & \text{Exhausted}_t \vee \text{Verified}_t, \\ \mathsf{verify}, & \text{Supported}_t \wedge \mathsf{verify} \in \mathcal{A}_t^{\text{adm}}, \\ \mathsf{refine}, & \text{Refinable}_t, \\ \mathsf{expand}, & \mathsf{expand} \in \mathcal{A}_t^{\text{adm}}, \\ \mathsf{stop}, & \text{otherwise}. \end{cases} \tag{4}$$

The branch order is intentional: budget exhaustion and verified success terminate search; a promising but unverified candidate receives stronger scrutiny; an actionable failure triggers targeted refinement; and uncertainty alone does not justify unlimited generation. The exact admissibility constraints and guarded-rule construction appear in Appendix A.2. Official SWE-bench correctness is never exposed to the controller. The online policy depends only on the observable repair state above.

3.3 State-Preserving Control

Case-local incumbent–challenger update. Iterative repair is path dependent: a later refinement need not improve the patch from which it was produced. Let $(p_1, \ldots, p_m)$ be the initial patches generated for the current case. MARS sets $i_0 = p_1$, treats the remaining patches as challengers, and applies the same rule to every later proposal or refinement $\widetilde{p}_t$:

$$\mathcal{P}_{t+1} = \text{dedup}\big(\mathcal{P}_t \cup \{\widetilde{p}_t\}\big),$$
$$i_{t+1} = \begin{cases} \widetilde{p}_t, & \gamma_t(\widetilde{p}_t, i_t) = 1, \\ i_t, & \text{otherwise}. \end{cases} \tag{5}$$

Here, $\gamma_t(\widetilde{p}_t, i_t) \in \{0, 1\}$ is the controller's evidence-gated promotion decision at an incumbent-replacement boundary, not a learned scalar score and not an order relation on patches.

Candidate commitment. Search termination and final candidate selection are separate stages. Let

$$\overline{\mathcal{L}}_T = \mathrm{dedup}(\mathcal{L}_T) = (q_1, \ldots, q_{M_T})$$

denote the accumulated candidate list at termination after exact-duplicate removal, with the host-side order preserved. MARS commits this accumulated state to downstream selection without reopening proposal or refinement:

$$\mathcal{P}_T^{\mathrm{com}} = \begin{cases} \mathrm{prefix}_{K_{\mathrm{cfg}}}\left(\overline{\mathcal{L}}_T\right), & \chi_T^{\mathrm{cap}} = 1, \\ \overline{\mathcal{L}}_T, & \chi_T^{\mathrm{cap}} = 0, \end{cases} \qquad K_{\mathrm{cfg}} = 8, \tag{6}$$

where χ_T^{cap} indicates whether the terminating host path uses the capped write-out routine. Thus, the configured capacity applies conditionally:

$$\chi_T^{\mathrm{cap}} = 1 \quad \Longrightarrow \quad |\mathcal{P}_T^{\mathrm{com}}| \le K_{\mathrm{cfg}} = 8. \tag{7}$$

No learned reranking or quality–diversity optimization is applied at this boundary. The downstream selector may validate and rank the committed candidates, but cannot request additional proposal or refinement actions. During search, incumbent protection determines whether a challenger replaces the active state; at termination, candidate commitment determines which accumulated candidates remain available for final selection. The host-specific write-out details are given in Appendix A.4. Thus, candidate admission does not imply incumbent replacement, and search termination does not imply that only the active incumbent remains available for final selection. The configured commitment cap constrains the capped write-out path rather than defining a universal cardinality guarantee.

4 Experiments

4.1 Experimental Protocol

We evaluate MARS on SWE-bench Lite and SWE-bench Verified (Jimenez et al., 2024). ExpeRepair (Mu et al., 2026) provides the host repair pipeline whose candidate search MARS controls, while AdaRepair-Mem (Luo et al., 2026) is the strongest memory-augmented baseline using the same backbone pair.

We keep the upstream retrieval and localization algorithms and configurations fixed across the ExpeRepair-based comparisons (Appendix B.2). The runs are executed independently, so their realized upstream outputs can differ because localization contains unseeded model calls. Ablations change the stated controller components; backbone transfer substitutes the proposal model; and pipeline transfer integrates the same control logic into AutoCodeRover, Agentless-1.5, and PatchPilot (Zhang et al., 2024; Xia et al., 2025; Li et al., 2025). The main Lite and Verified evaluations contain 300 and 500 issues, respectively. Figure 2 uses a stratified 150-issue Verified sample for motivation analysis. The fixed-budget comparison and the AS–IC factorial and policy-ladder analyses use the same separately drawn 200-issue Verified subset (Appendix B.7).

Our primary metrics are official pass@1 and average cost per issue. We additionally report execution success rate (ESR) and reproduction success rate (RSR), defined in Appendix B.1. ESR records successful termination of the final reproduction script, whereas RSR additionally requires non-empty output that passes the host reproduction check. Both are diagnostic evidence metrics rather than official correctness measures.

4.2 Main Results

Table 1 uses AdaRepair-Mem as the reference for the MARS badges because the two systems use the same backbone pair. This matched backbone comparison provides the closest reference for assessing the effect of search control separately from proposal-model choice. The two systems differ in how they obtain additional repair capability: AdaRepair-Mem introduces cross-issue repair memory, whereas MARS operates cold-start and modifies only the control of the within-issue candidate

Table 1: Main comparison on SWE-bench Lite and SWE-bench Verified. Cost is the all-in average per issue; badges report the **MARS** change relative to AdaRepair-Mem.

Method	Backbone	SWE-bench Lite		SWE-bench Verified	
		Pass@1 (%)	Cost ($)	Pass@1 (%)	Cost ($)
SWE-agent	Claude 3.5 Sonnet	23.0	1.62	33.6	1.59
Aider	GPT-4o + Claude 3 Opus	26.3	–	–	–
AutoCodeRover	GPT-4o	30.7	0.19	38.2	0.22
SpecRover	Claude 3.5 Sonnet + GPT-4o	31.0	0.65	46.2	–
Agentless-1.5	Claude 3.5 Sonnet	40.7	1.12	50.8	1.19
OpenHands	CodeAct v2.1	41.7	1.33	53.0	0.78
PatchPilot	Claude 3.5 Sonnet	45.3	0.97	53.6	0.99
DARS	Claude 3.5 Son. + DeepSeek R1	47.0	12.24	–	–
ExpeRepair	Claude 3.5 Sonnet + o4-mini	48.3	1.89	57.2	1.74
AdaRepair-Mem	Claude 3.5 Sonnet + o4-mini	51.0	1.67	63.2	1.52
MARS	Claude 3.5 Sonnet + o4-mini	**53.0** ▲ +2.0	**1.15** ▼ −31%	**66.0** ▲ +2.8	**1.18** ▼ −22%

Table 2: Mechanism ablation. Each row removes one component: adaptive search (AS), incumbent–challenger control (IC), or candidate commitment (CC). ESR and RSR characterize execution evidence and are diagnostic; their column maxima are not marked as best. Bold marks the full controller.

Configuration	SWE-bench Lite				SWE-bench Verified			
	Pass@1 (%)	ESR (%)	RSR (%)	Cost ($)	Pass@1 (%)	ESR (%)	RSR (%)	Cost ($)
MARS	**53.0**	**90.7**	**76.3**	**1.15**	**66.0**	**89.8**	**78.8**	**1.18**
w/o AS	51.7	92.0	78.0	1.42	65.0	90.4	81.6	1.62
w/o IC	51.0	89.6	75.6	1.11	63.6	89.8	76.8	1.13
w/o CC	50.3	89.3	76.3	1.08	61.4	88.2	74.4	1.02

search. Accordingly, the comparison evaluates whether runtime control can improve the accuracy–cost operating point without adding cross-issue state.

Absolute dollar costs in Table 1 should not be interpreted as a global efficiency ranking, because the table contains systems with different backbone configurations and model prices. The more relevant observation is the joint movement of repair accuracy and computation under the matched MARS–AdaRepair-Mem comparison. MARS changes which search transitions are executed and which intermediate candidates remain available for final selection, rather than obtaining its improvement by uniformly extending the search. This pattern is consistent with Budget–Evidence Decoupling: the value of an additional search action depends on the evidence available in the current repair state, not solely on the amount of computation already allocated. The following ablations therefore examine separately how MARS allocates search actions, preserves the incumbent state, and exposes accumulated candidates at the search-to-selection boundary.

4.3 Mechanism Ablation

Table 2 removes one control at a time. *w/o AS* restores the host's fixed search schedule; *w/o IC* removes incumbent–challenger protection and permits refinement to replace the active state in place; and *w/o CC* forwards only the controller-selected patch rather than the accumulated candidate state available to downstream final selection. These interventions target the three decisions separated by MARS: whether to continue search, whether a challenger should replace the incumbent, and what candidate state is exposed at termination.

The ablation profiles separate the roles of the three controls. Removing AS raises Verified cost from $1.18 to $1.62 while changing pass@1 by only one point; the same pattern holds on Lite. AS primarily changes the search trajectory: on the full benchmark its most visible effect is lower cost, while the controlled factorial analysis in Appendix B.8.2 further shows that it can increase useful candidate discovery under matched initial-candidate and refinement-round ceilings, and a policy ladder in the same appendix isolates which evidence channels drive the effect.

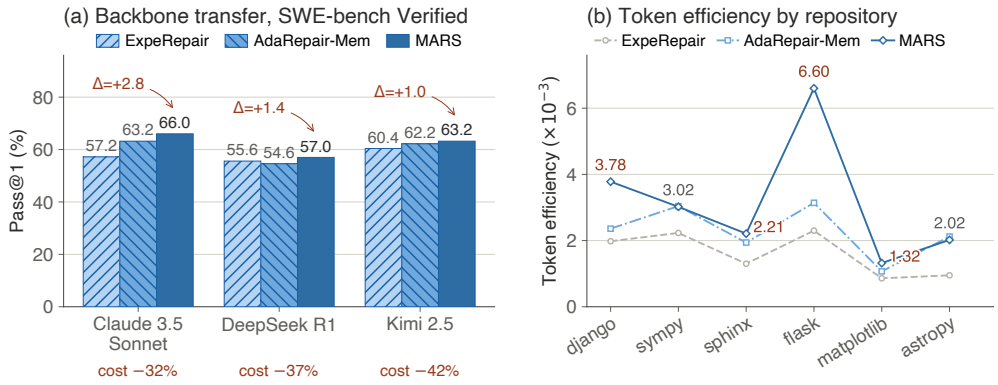


Figure 4: Transfer and efficiency on SWE-bench Verified. **(a)** Pass@1 differences are measured in percentage points against the stronger baseline for each backbone; cost reductions are measured relative to the corresponding ExpeRepair host and are reported as relative percentages. **(b)** Token efficiency is the number of officially resolved issues divided by the total number of input tokens, reported in units of 10^{-3} resolved issues per input token, separately by repository.

CC produces the largest commitment-side difference: forwarding only the controller-selected state saves additional cost but loses 2.7 points on Lite and 4.6 on Verified. This row changes both candidate availability and the amount of downstream validation, so the gap should be interpreted as the effect of the complete commitment interface rather than as evidence for a separate terminal-ranking objective; Appendix B.8.3 reports a control with matched final-pool size and validation budget, and Appendix B.8.1 traces where correct patches are lost along the candidate lifecycle. Notably, *w/o AS* attains the highest RSR while resolving fewer issues, confirming that reproduction success is diagnostic evidence rather than a surrogate for official correctness. The single-component interventions are not additive because the AS and IC implementations share one control path. To separate evidence-conditioned control from simple budget reduction, Appendix B.7 evaluates three fixed host budgets on the same 200 issues.

4.4 Backbone and Pipeline Transfer

Figure 4(a) evaluates backbone transfer by replacing the proposal backbone while keeping the MARS controller and the ExpeRepair host pipeline fixed. MARS improves pass@1 over the stronger backbone-specific baseline by 2.8, 1.4, and 1.0 percentage points for Claude 3.5 Sonnet, DeepSeek R1, and Kimi 2.5, respectively, while reducing the all-in cost of the ExpeRepair host by 32%, 37%, and 42%. The consistent improvement across these settings shows that MARS does not depend on a particular proposal backbone.

The transfer results also clarify how the controller contributes to this advantage. Multi-Evidence Adaptive Search uses execution, review, verification, and resource signals to suppress search actions that lack sufficient support, independently of which backbone generated the candidate. Incumbent–Challenger Control and Candidate Commitment preserve useful candidates once they have been produced, allowing backbone-specific repair strengths to remain available for final selection. MARS therefore combines a backbone-agnostic search-allocation mechanism with state preservation that converts useful proposals into retained repair opportunities.

Table 3 asks a different transfer question: whether the controller remains useful when the surrounding repair procedure changes. Within each host pair, the backbone configuration is held fixed, so the reported cost deltas isolate search control rather than differences in model pricing. Pass@1 gains remain comparatively narrow across the three hosts (2.2–3.4 points), whereas cost reductions range from 2% to 27%. This asymmetry is consistent with the controller removing host-dependent redundant search: pipelines expose different amounts of avoidable computation, while incumbent protection only acts at the smaller set of transitions where a useful candidate is at risk. These results support transferability across compatible host interfaces rather than independence from host-pipeline design. MARS can be integrated into pipelines that expose candidate generation, feedback, refinement, verification, and commitment boundaries.

Table 3: Controller transfer across host pipelines on SWE-bench Lite and Verified. MARS changes only search control; each badge compares with the corresponding host's default controller.

		SWE-bench Lite		SWE-bench Verified	
Host pipeline	**Controller**	**Pass@1 (%)**	**Cost ($)**	**Pass@1 (%)**	**Cost ($)**
AutoCodeRover	Default	30.7	0.19	38.2	0.22
	MARS	**33.0** ▲ +2.3	**0.16** ▼ −16%	**40.4** ▲ +2.2	**0.16** ▼ −27%
Agentless-1.5	Default	40.7	1.12	50.8	1.19
	MARS	**43.0** ▲ +2.3	**1.10** ▼ −2%	**53.2** ▲ +2.4	**1.07** ▼ −10%
PatchPilot	Default	45.3	0.97	53.6	0.99
	MARS	**48.7** ▲ +3.4	**0.88** ▼ −9%	**55.8** ▲ +2.2	**0.83** ▼ −16%

4.5 Analysis of Accuracy and Efficiency Gains

Aggregate pass@1 does not reveal whether MARS simply preserves the same easy issues more cheaply or changes which issues are solved. On Verified, MARS resolves 330 issues and AdaRepair-Mem 316, but the difference is not a 14-issue superset: MARS resolves 50 issues that AdaRepair-Mem misses, while AdaRepair-Mem resolves 36 that MARS misses. Across the seven systems included in the resolved-set analysis, 131 issues form a common core, while 12 resolutions are exclusive to MARS and 10 to AdaRepair-Mem; Appendix B.4 lists the compared systems and reports the full intersection structure. The controller therefore changes the repair trajectory on a non-trivial subset of issues rather than merely retaining the baseline's resolved set at lower cost.

Figure 4(b) localizes the efficiency effect. The largest host-relative gain occurs on django, where MARS achieves $1.91\times$ the ExpeRepair token efficiency, measured in resolved issues per input token, and where 231 of the 500 Verified issues reside. Against AdaRepair-Mem, however, MARS is already within 0.10×10^{-3} resolved issues per input token on sympy and astropy. These repositories mark a useful boundary condition: when the surrounding system already turns its search budget into successful repairs efficiently, less unsupported computation remains for the controller to remove. A complementary cumulative analysis shows that the aggregate advantage is not created by a late block of issues; the corresponding stability analysis is reported in Appendix B.5, and Appendix B.6 illustrates the two controller effects on individual issues.

5 Conclusion

We study repository-level program repair as evidence-guided candidate search under a finite compute budget. MARS combines Multi-Evidence Adaptive Search (AS), Incumbent–Challenger Control (IC), and Candidate Commitment (CC) to allocate computation adaptively, preserve useful repair states, and expose accumulated candidates for final selection. Across SWE-bench Lite and Verified, MARS improves the accuracy–cost trade-off and transfers across proposal backbones and host pipelines. Ablations show that AS improves candidate discovery and efficiency, whereas IC preserves discovered solutions; their effects are complementary rather than additive. MARS remains dependent on the evidence exposed by its host pipeline and offers less benefit when that pipeline already uses computation efficiently.

AI Use Statement

The evaluated repair systems use large language models as task components, whereas the authors use generative AI only as an auxiliary tool. The evaluated models operate within the specified repair pipelines, and their configurations are documented with the experiments. Generative AI tools assisted with language editing, literature searches, figure layout, LaTeX, plotting code, and consistency checks. We manually reviewed all AI-assisted outputs and checked reported results and citations against their underlying sources. The authors take full responsibility for the final content.

Ethics Statement

This work evaluates automated program repair using publicly available software benchmarks and repositories. It involves no human-subject data collection, does not deploy generated patches to production systems, and follows the benchmark evaluation protocol. Automated repair should be supplemented with independent testing, code review, and appropriate software-security controls.

Reproducibility Statement

We provide the source code, configuration files, prompts, evaluation protocol, and supporting documentation required to reproduce the reported experiments. The anonymized materials are available at: `https://anonymous.4open.science/r/mars-E86B`.

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, 2025.

Reem Aleithan, Haoran Xue, Mohammad Mahdi Mohajer, Elijah Nnorom, Gias Uddin, and Song Wang. Swe-bench+: Enhanced coding benchmark for llms. *arXiv preprint arXiv:2410.06992*, 2024.

Mohammad Ali Alomrani, Yingxue Zhang, Derek Li, Qianyi Sun, Soumyasundar Pal, Zhanguang Zhang, Yaochen Hu, Rohan Deepak Ajwani, Antonios Valkanas, Raika Karimi, et al. Reasoning on a budget: A survey of adaptive and controllable test-time compute in llms. *arXiv preprint arXiv:2507.02076*, 2025.

Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. Swe-search: Enhancing software agents with monte carlo tree search and iterative refinement. In *International Conference on Learning Representations*, volume 2025, pp. 64485–64515, 2025.

Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. Repairagent: An autonomous, llm-based agent for program repair. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 2188–2200. IEEE, 2025.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Silin Chen, Shaoxin Lin, Yuling Shi, Heng Lian, Xiaodong Gu, Longfei Yun, Dong Chen, Lin Cao, Jiyang Liu, Nu Xia, et al. Swe-exp: Experience-driven software issue resolution. *arXiv preprint arXiv:2507.23361*, 2025.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations*, volume 2024, pp. 8746–8825, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Ryan Ehrlich, Bradley Brown, Jordan Juravsky, Ronald Clark, Christopher Ré, and Azalia Mirhoseini. Codemonkeys: Scaling test-time compute for software engineering. *arXiv preprint arXiv:2501.14723*, 2025.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International conference on learning representations*, volume 2024, pp. 32808–32824, 2024.

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*, volume 2024, pp. 54107–54157, 2024.

Sayash Kapoor, Benedikt Stroebl, Zachary S Siegel, Nitya Nadgir, and Arvind Narayanan. Ai agents that matter. *arXiv preprint arXiv:2407.01502*, 2024.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.

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, 2011.

Claire Le Goues, Michael Pradel, and Abhik Roychoudhury. Automated program repair. *Communications of the ACM*, 62(12):56–65, 2019.

Hongwei Li, Yuheng Tang, Shiqi Wang, and Wenbo Guo. Patchpilot: A cost-efficient software engineering agent with early attempts on formal verification. *arXiv preprint arXiv:2502.02747*, 2025.

Zixu Li and Zhiqi Peng. Reason-code: Reliable code generation via test-driven monte carlo tree search. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL 2026)*, pp. 447–458, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

ZC Luo, JC Guo, WJ He, SY Wang, JC Yu, FM Zhao, Yuan Chen, Te Cao, LQ Liu, Ning Zheng, et al. Adarepair-mem: Adaptive experience orchestration for repository-level program repair. *arXiv preprint arXiv:2609.20130*, 2026.

Yingwei Ma, Qingping Yang, Rongyu Cao, Binhua Li, Fei Huang, and Yongbin Li. Alibaba lingmaagent: Improving automated issue resolution via comprehensive repository exploration. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, pp. 238–249, 2025.

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. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Kyle McCleary and James Ghawaly. Quantifying the accuracy and cost impact of design decisions in budget-constrained agentic llm search. *arXiv preprint arXiv:2603.08877*, 2026.

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.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20286–20332, 2025.

Albert Örwall. Moatless tools. *Zenodo*, 2024.

Zichao Qi, Fan Long, Sara Achour, and Martin Rinard. An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. In *Proceedings of the 2015 international symposium on software testing and analysis*, pp. 24–36, 2015.

Haifeng Ruan, Yuntong Zhang, and Abhik Roychoudhury. Specrover: Code intent extraction via llms. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 963–974. IEEE, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Edward K Smith, Earl T Barr, Claire Le Goues, and Yuriy Brun. Is the cure worse than the disease? overfitting in automated program repair. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, pp. 532–543, 2015.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, volume 2025, pp. 10131–10165, 2025.

Benedikt Stroebl, Sayash Kapoor, and Arvind Narayanan. The limits of inference scaling through resampling. In *International Conference on Learning Representations*, volume 2026, pp. 92601–92633, 2026.

Boshi Wang, Weijian Xu, Yunsheng Li, Xuemei Gao, Yujia Xie, Huan Sun, and Dongdong Chen. Improving code localization with repository memory. In *International Conference on Learning Representations*, volume 2026, pp. 111266–111285, 2026a.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030*, 2024a.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. In *International Conference on Learning Representations*, volume 2025, pp. 65882–65919, 2025a.

Xinyuan Wang, Yanchi Liu, Wei Cheng, Xujiang Zhao, Zhengzhang Chen, Wenchao Yu, Yanjie Fu, and Haifeng Chen. Mixllm: Dynamic routing in mixed large language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 10912–10922, 2025b.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Yuhang Wang, Yuling Shi, Mo Yang, Rongrui Zhang, Shilin He, Heng Lian, Yuting Chen, Siyu Ye, Kai Cai, and Xiaodong Gu. Swe-pruner: Self-adaptive context pruning for coding agents. *arXiv preprint arXiv:2601.16746*, 2026b.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024b.

Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. Automated program repair in the era of large pre-trained language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 1482–1494. IEEE, 2023.

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.

Chengxing Xie, Bowen Li, Chang Gao, He Du, Wai Lam, Difan Zou, and Kai Chen. Swe-fixer: Training open-source llms for effective and efficient github issue resolution. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 1123–1139, 2025.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Zhiyuan Zhai, Bingcong Li, Bingnan Xiao, Ming Li, and Xin Wang. Adaptive test-time compute allocation for reasoning llms via constrained policy optimization. *arXiv preprint arXiv:2604.14853*, 2026.

Kexun Zhang, Weiran Yao, Zuxin Liu, Yihao Feng, Zhiwei Liu, Rithesh Ramapura Narasimha Murthy, Tian Lan, Lei Li, Renze Lou, Jiacheng Xu, et al. Diversity empowers intelligence: Integrating expertise of software engineering agents. In *International Conference on Learning Representations*, volume 2025, pp. 86691–86714, 2025.

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.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

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. *arXiv preprint arXiv:2310.04406*, 2023.

A Additional Method Details

A.1 Evidence Representation and Gate Semantics

MARS does not infer an online correctness probability. Instead, the controller operates on three discrete evidence channels whose semantics follow the host implementation.

Execution evidence. Let $\mathrm{rc}_t(p)$ denote the execution return code associated with candidate p.

$$z_t^{\mathrm{exe}}(p) = \Psi_{\mathrm{exe}}(\mathrm{rc}_t(p)) := \begin{cases} \mathsf{pass}, & \mathrm{rc}_t(p) = 0, \\ \mathsf{fail}, & \mathrm{rc}_t(p) \notin \{0, -1\} \text{ and the failure is candidate-attributable}, \\ \bot, & \mathrm{rc}_t(p) = -1 \text{ or the observation is infrastructure-related}. \end{cases} \tag{8}$$

The implementation contains no independent regression state. Timeout and infrastructure failure are therefore represented as unknown evidence rather than as proof that the candidate is incorrect.

Verification evidence. Before strong verification is invoked, $z_t^{\mathrm{ver}}(p) = \bot$. Once invoked, the verifier produces

$$z_t^{\mathrm{ver}}(p) \in \{\mathsf{pass}, \mathsf{refine}\}. \tag{9}$$

An unparsable verifier output is conservatively projected onto refine. Equivalently,

$$\Psi_{\mathrm{ver}}(y) = \begin{cases} \mathsf{pass}, & y \text{ is parsed as a verification pass}, \\ \mathsf{refine}, & y \text{ is parsed as refinement-required}, \\ \mathsf{refine}, & y \text{ is unparsable}. \end{cases} \tag{10}$$

Review evidence. The reviewer assigns a Bug Fixing Score

$$b_t(p) \in \{0, 1, 2\}. \tag{11}$$

A candidate is review-supported exactly when its score reaches the implementation's highest support level:

$$\begin{aligned} \mathrm{Supported}_t(p) &:= \mathbf{1}[b_t(p) = 2]\,, \\ \mathcal{S}_t &:= \{p \in \mathcal{P}_t : \mathrm{Supported}_t(p) = 1\}\,. \end{aligned} \tag{12}$$

Thus, the implementation-level support threshold is $\tau_{\mathrm{sup}} = 2$ when expressed on the reviewer score scale.

Evidence state. Collecting the three channels gives

$$\mathbf{z}_t(p) = (z_t^{\mathrm{exe}}(p), z_t^{\mathrm{ver}}(p), z_t^{\mathrm{rev}}(p)), \qquad z_t^{\mathrm{rev}}(p) := b_t(p), \tag{13}$$

while resource consumption is represented separately by

$$\rho_t = \big(C_t,\, d_t,\, n_t^{\mathrm{gen}},\, n_t^{\mathrm{ref}},\, n_t^{\mathrm{ver}}\big)\,. \tag{14}$$

This separation is deliberate: execution, verification, and review determine the semantic branch taken by the controller, whereas ρ_t determines whether another resource-consuming action is admissible.

A.2 Admissible Actions and Guarded Rules

The reported configuration uses three initial candidates, at most two refinement rounds, and at most two refinements within each round. Let

$$m_0 = 3, \qquad D_{\mathrm{cfg}} = 2, \qquad R_{\mathrm{cfg}} = 2 \tag{15}$$

denote these three implementation constants. Here D_{cfg} bounds the refinement depth entering the gate g_t^{depth}, whereas R_{cfg} bounds the number of refinements the host issues within a single round and is enforced by the host rather than by a controller gate.

The controller also uses the configured all-in cost checkpoint

$$B_{\mathrm{cfg}} = \$2.20. \tag{16}$$

Importantly, B_{cfg} is not a strict bound on the all-in cost reported in the experiments. The ledger is queried only at predefined controller checkpoints. Let $\chi_t^{\mathrm{chk}} \in \{0,1\}$ indicate that step t is such a checkpoint. The corresponding budget gate is

$$g_t^{\mathrm{bud}} := \mathbf{1}\big[\chi_t^{\mathrm{chk}} = 1 \wedge C_t \geq B_{\mathrm{cfg}}\big]\,. \tag{17}$$

Hence a trajectory may exceed B_{cfg} between two consecutive ledger checks.

The implementation-level refinement condition is represented by

$$\begin{aligned} g_t^{\mathrm{ref}} &:= \mathbf{1}[\mathcal{S}_t = \varnothing \;\vee\; z_t^{\mathrm{ver}}(i_t) = \mathsf{refine}]\,, \\ g_t^{\mathrm{depth}} &:= \mathbf{1}[d_t < D_{\mathrm{cfg}}]\,. \end{aligned} \tag{18}$$

Strong verification is applied to the current incumbent only after the review stage identifies at least one supported candidate and the incumbent has not yet received a strong-verification verdict:

$$g_t^{\mathrm{ver}} := \mathbf{1}[\mathcal{S}_t \neq \varnothing \wedge z_t^{\mathrm{ver}}(i_t) = \bot]\,. \tag{19}$$

Let $\xi_t \in \{0,1\}$ denote the host fallback condition triggered when no usable reproduction test is available. The admissible external-action set is then written as

$$\begin{aligned} \mathcal{A}_t^{\mathrm{adm}} = \{\mathsf{stop}\} &\cup \{\mathsf{verify} : g_t^{\mathrm{ver}} = 1 \wedge n_t^{\mathrm{ver}} = 0\} \\ &\cup \Big\{\mathsf{refine} : g_t^{\mathrm{ref}} = 1 \wedge g_t^{\mathrm{depth}} = 1 \wedge g_t^{\mathrm{bud}} = 0\Big\} \\ &\cup \big\{\mathsf{expand} : \xi_t = 1 \wedge g_t^{\mathrm{bud}} = 0\big\}\,. \end{aligned} \tag{20}$$

For correspondence with the compact policy in Equation 4, we define the following semantic predicates from these implementation-level gates:

$$\mathrm{Verified}_t := \mathbf{1}[z_t^{\mathrm{ver}}(i_t) = \mathsf{pass}]\,, \tag{21}$$

$$\mathrm{Supported}_t := \mathbf{1}[\mathcal{S}_t \neq \varnothing]\,, \tag{22}$$

$$\mathrm{Refinable}_t := \mathbf{1}\big[\mathsf{refine} \in \mathcal{A}_t^{\mathrm{adm}}\big]\,, \tag{23}$$

$$\mathrm{Exhausted}_t := \mathbf{1}\big[\mathcal{A}_t^{\mathrm{adm}} = \{\mathsf{stop}\} \;\vee\; g_t^{\mathrm{bud}} = 1\big]\,. \tag{24}$$

The parenthesized $\mathrm{Supported}_t(p)$ of Equation 12 is per-candidate, whereas the unparenthesized $\mathrm{Supported}_t$ is the state-level predicate obtained through $\mathcal{S}_t$. Here, $\mathrm{Supported}_t$ is a state-level trigger for making incumbent verification admissible; it does not by itself change the incumbent or

expose an official correctness label. The budget disjunct in $\mathrm{Exhausted}_t$ makes the compact policy of Equation 4 agree with the ordered rule family below, in which r_{bud} has highest precedence: a reached budget checkpoint terminates the search even when a not-yet-verified supported candidate is present. These predicates provide the semantic view used in the main text, whereas Equations 17–20 specify the corresponding code-level conditions. In particular, a review-supported candidate does not trigger repeated verification after a verifier verdict has already been obtained, because verify is then absent from $\mathcal{A}_t^{\mathrm{adm}}$. The ordered rule family is

$$\mathcal{R} = (r_{\mathrm{bud}}, r_{\mathrm{pass}}, r_{\mathrm{verify}}, r_{\mathrm{refine}}, r_{\mathrm{expand}}, r_{\mathrm{term}}) , \tag{25}$$

and the executed action is

$$\begin{aligned} r_t^\star &= \min_{\prec} \left\{ r \in \mathcal{R} : g_r(s_t) = 1 \wedge \alpha(r) \in \mathcal{A}_t^{\mathrm{adm}} \right\} , \\ a_t &= \alpha(r_t^\star). \end{aligned} \tag{26}$$

Here $\prec$ is the precedence order in which $\mathcal{R}$ is listed in Equation 25, $g_r(\cdot)$ is the guard of rule r, and $\alpha(r)$ is the external action rule r issues. The precedence makes termination after a reached budget checkpoint or a verified pass take precedence over any action that would consume additional model calls.

A.3 Incumbent Acceptance and Preservation

Let $\widetilde{p}_t$ denote a newly generated or refined challenger and let i_t be the current incumbent. Strong verification, when admissible, is applied to i_t as specified in Equation 19. The challenger is considered separately at the incumbent-replacement boundary.

The implementation uses a confidence-controlled guard

$$\gamma_t(\widetilde{p}_t, i_t) \in \{0, 1\}, \tag{27}$$

to decide whether the challenger promotes to incumbent. The guard is a discrete implementation decision, not a learned scalar score or an order relation over patches. In the full configuration, a verifier-declined challenger, a timeout, or infrastructure-corrupted evidence does not automatically replace an already retained incumbent.

Candidate registration occurs before incumbent replacement:

$$\mathcal{P}_{t+1} = \mathrm{dedup}\left(\mathcal{P}_t \cup \{\widetilde{p}_t\}\right). \tag{28}$$

The active-state transition is then

$$i_{t+1} = \begin{cases} \widetilde{p}_t, & \gamma_t(\widetilde{p}_t, i_t) = 1, \\ i_t, & \gamma_t(\widetilde{p}_t, i_t) = 0. \end{cases} \tag{29}$$

This formulation makes explicit the distinction used throughout MARS:

$$\widetilde{p}_t \in \mathcal{P}_{t+1} \;\not\Rightarrow\; i_{t+1} = \widetilde{p}_t. \tag{30}$$

Timeout and infrastructure-corrupted observations are treated as unknown evidence and do not by themselves license incumbent replacement.

A.4 Candidate Commitment with a Configured Write-Out Cap

This subsection specifies the implementation-level write-out rule for the commitment state defined in Section 3.3. The rule preserves the order in which the host pipeline generated candidates and applies the configured capacity only on host paths that use the capped write-out routine.

Let

$$\mathcal{L}_T = (\ell_1, \ldots, \ell_{N_T}) \tag{31}$$

denote the ordered candidate sequence accumulated when the controller terminates. Exact duplicate removal gives the order-preserving list

$$\begin{aligned} \overline{\mathcal{L}}_T &= \mathrm{dedup}(\mathcal{L}_T) = (q_1, \ldots, q_{M_T}), \\ M_T &= \left|\mathrm{dedup}(\mathcal{L}_T)\right|. \end{aligned} \tag{32}$$

The implementation defines the configured write-out capacity

$$K_{\text{cfg}} := \texttt{PATCH_BUDGET_MAX_FINAL_PATCHES} = 8. \tag{33}$$

Let $\chi_T^{\text{cap}} \in \{0, 1\}$ indicate whether the terminating host path uses the capped write-out routine. The number of emitted candidates is

$$M_T^{\text{com}} = \chi_T^{\text{cap}} \min(M_T, K_{\text{cfg}}) + \big(1 - \chi_T^{\text{cap}}\big) M_T. \tag{34}$$

The committed state is the order-preserving prefix of the deduplicated list:

$$\mathcal{P}_T^{\text{com}} := \text{prefix}_{M_T^{\text{com}}}\big(\overline{\mathcal{L}}_T\big) = (q_1, \ldots, q_{M_T^{\text{com}}}). \tag{35}$$

Consequently,

$$\begin{aligned} \chi_T^{\text{cap}} = 1 &\Longrightarrow |\mathcal{P}_T^{\text{com}}| = \min(M_T, K_{\text{cfg}}) \le 8, \\ \chi_T^{\text{cap}} = 0 &\Longrightarrow |\mathcal{P}_T^{\text{com}}| = M_T. \end{aligned} \tag{36}$$

The capacity bound is therefore conditional rather than universal, because historical host-side paths did not uniformly enforce the configured truncation. MARS applies no learned terminal scorer, quality–diversity objective, or additional controller-side reranking at this boundary. After the commitment state is emitted, downstream validation and selection may operate on the emitted candidates, but cannot reopen proposal or refinement.

B Additional Results

B.1 Protocol Details

This subsection records implementation and evaluation details omitted from Section 4.1.

Execution and reproduction metrics. For an evaluation set $\mathcal{D} = \{x_i\}_{i=1}^N$, let $\text{rc}_i^{\text{final}}$ denote the return code of the final reproduction script for issue x_i, and let o_i^{final} denote its output. We define the execution success rate and reproduction success rate as

$$\text{ESR} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\big[\text{rc}_i^{\text{final}} = 0\big], \tag{37}$$

and

$$\text{RSR} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\big[o_i^{\text{final}} \neq \varnothing \ \wedge \ \text{ReproCheck}\big(o_i^{\text{final}}\big) = 1\big]. \tag{38}$$

Here, $\text{ReproCheck}(\cdot)$ denotes the host pipeline's reproduction-check predicate. ESR measures whether the final reproduction script exits successfully, whereas RSR additionally requires non-empty output that passes the host check. Both metrics summarize host-side execution behavior and are reported as diagnostic evaluation quantities.

Controller configuration. The reported MARS runs use the following code-level configuration: The table records the controller settings needed to reproduce the reported runs. On non-budget early-exit paths, MARS requires a strong-verification pass for the incumbent. Review support only makes verification admissible; it does not terminate the search. A reached budget checkpoint is a separate termination condition. The $B_{\text{cfg}} = \$2.20$ value is a soft all-in cost checkpoint, and the reported cost uses the same all-in accounting. Because the ledger is queried only at fixed checkpoints, an individual run may exceed this value before the next check. The configured $K_{\text{cfg}} = 8$ capacity applies only on capped write-out paths and is therefore not a universal bound on all historical host-side outputs.

Ablation scope and overlap. The three ablations of Table 2 are defined in Section 4.3. They are not fully disjoint: the AS and IC interventions both act on the confidence-controlled execution path, and *w/o CC* removes downstream validation on the omitted candidates as well as their availability. We therefore read the three rows as individual mechanism probes rather than as an additive decomposition of the full gain, and analyze the AS–IC interaction separately in the 2×2 factorial of Appendix B.8.2 and the commitment interface in Appendix B.8.3.

Algorithm 1 MARS: Multi-Evidence Adaptive Search with State-Preserving Control. Symbols follow Section 3.1 and Appendix A: $\mathcal{P}$ is the candidate pool, i the incumbent, $\widetilde{p}$ the pending challenger, $\mathcal{E}$ the accumulated evidence, and ρ the resource vector of Equation 14. $\mathcal{L}$ is the ordered host-side candidate list of Equation 31, $\|$ denotes appending to it, and $\oplus$ denotes the evidence update. The operator prefix_M returns the first M elements while preserving their host-side order. $\mathrm{FrozenSelect}$ is the host's downstream validation-and-selection procedure applied to the frozen commitment state; it issues no proposal or refinement action.

Require: Repair instance x, host system $\mathcal{H}$, soft cost checkpoint B_{cfg}, refinement limit D_{cfg}, per-round refinement limit R_{cfg}, configured write-out cap K_{cfg}

Generate $m_0 = 3$ case-local candidates $(p_1, \ldots, p_{m_0})$ using $\mathcal{H}$
$\mathcal{P} \leftarrow \mathrm{dedup}(\{p_1, \ldots, p_{m_0}\})$; $\mathcal{L} \leftarrow (p_1, \ldots, p_{m_0})$; $i \leftarrow p_1$; $\widetilde{p} \leftarrow \varnothing$
$\mathcal{E} \leftarrow \varnothing$; $\rho \leftarrow (C_0, 0, m_0, 0, 0)$ {cost ledger, refinement depth, and call counters}
Execute and review i; update $\mathcal{E}$ and ρ
for $p \in \mathcal{P} \setminus \{i\}$ **do**
  Execute and review p; update $\mathcal{E}$ and ρ
  Evaluate the incumbent-promotion gate $\gamma(p, i)$
  if $\gamma(p, i) = 1$ **then**
    $i \leftarrow p$
  end if
end for
while true **do**
  Construct $s = (\mathcal{P}, i, \widetilde{p}, \mathcal{E}, \rho)$ and controller gates $g^{\mathrm{bud}}, g^{\mathrm{ver}}, g^{\mathrm{ref}}, g^{\mathrm{depth}}$
  Construct $\mathcal{A}^{\mathrm{adm}}$ using Eq. 20
  The host-side action handler enforces at most R_{cfg} refinement actions within the current refinement round
  $a \leftarrow \pi(s)$ according to the ordered rule family
  if $a = \mathsf{stop}$ **then**
    break
  end if
  $(\widetilde{p}, o) \leftarrow \mathcal{H}(a, s)$
  $\mathcal{E} \leftarrow \mathcal{E} \oplus o$; update ρ
  if $\widetilde{p} \neq \varnothing$ **then**
    $\mathcal{L} \leftarrow \mathcal{L} \,\|\, (\widetilde{p})$; $\mathcal{P} \leftarrow \mathrm{dedup}(\mathcal{P} \cup \{\widetilde{p}\})$
    Evaluate $\gamma(\widetilde{p}, i)$
    if $\gamma(\widetilde{p}, i) = 1$ **then**
      $i \leftarrow \widetilde{p}$
    end if
  end if
end while
$\overline{\mathcal{L}} \leftarrow \mathrm{dedup}(\mathcal{L})$
Observe host write-out path $\chi^{\mathrm{cap}} \in \{0, 1\}$
$M^{\mathrm{com}} \leftarrow \chi^{\mathrm{cap}} \min(|\overline{\mathcal{L}}|, K_{\mathrm{cfg}}) + (1 - \chi^{\mathrm{cap}})|\overline{\mathcal{L}}|$
$\mathcal{P}^{\mathrm{com}} \leftarrow \mathrm{prefix}_{M^{\mathrm{com}}}(\overline{\mathcal{L}})$
$\widehat{p} \leftarrow \mathrm{FrozenSelect}(\mathcal{P}^{\mathrm{com}})$
return $\widehat{p}$

Budget checkpoints. The budget gate g_t^{bud} of Equation 17 is evaluated at five controller checkpoints, that is, at the five steps for which $\chi_t^{\mathrm{chk}} = 1$: after review, after multi-candidate review, before refinement following a declined verification, after each refinement, and after each refinement round. The ledger is not consulted between these checkpoints. Once $g_t^{\mathrm{bud}} = 1$, MARS freezes the candidates accumulated so far and issues no further model calls.

Strong verification. Strong verification is invoked at most once per issue and produces a discrete verdict. An unparsable output is conservatively mapped to refine and logged as an error; parsing failure therefore withholds commitment rather than granting it.

Table 4: Controller configuration used in the reported MARS runs, with the corresponding symbol or gate in Appendix A.

Setting	Value	Controller semantics
Initial candidates	3	m_0 in Equation 15
Maximum refinement rounds	2	D_{cfg}; enters g_t^{depth}
Maximum refinements per round	2	R_{cfg}; enforced by the host
Verification-gated early stopping	enabled	On non-budget early-exit paths, termination requires $z_t^{\text{ver}}(i_t) = \mathsf{pass}$; a reached budget checkpoint may terminate the search independently
Confidence gate	enabled	Review support alone does not terminate search: $\text{Supported}_t = 1$ admits verify through g_t^{ver} rather than stop
Incumbent guard	enabled	$\gamma_t(\widetilde{p}_t, i_t) = 0$ for a verifier-declined challenger or missing evidence
Verify incumbent once	enabled	$n_t^{\text{ver}} = 0$ in Equation 20
Validation safety net	enabled	On guarded early-exit paths, retains the selected patch and distinct, already-generated candidates for downstream validation; their ordering follows the terminating host write-out path
Configured final-candidate cap	$K_{\text{cfg}} = 8$	Applied only on capped write-out paths, $\chi_T^{\text{cap}} = 1$ (Equations 33–36)
all-in cost checkpoint	$2.20	B_{cfg} in g_t^{bud} (Equation 17); soft, checkpoint-evaluated
Reviewer support threshold	Bug Fixing Score $= 2$	τ_{sup} in Equation 12

Fallback schedule. Issues with no usable reproduction test follow a separate host fallback on which expand and context compression are available. On the main path, candidate growth is driven by evidence-conditioned refinement rather than unconditional expansion.

Transfer reporting. Cross-pipeline experiments do not assume that all hosts expose identical intermediate evidence. ESR and RSR are therefore omitted from Table 3: AutoCodeRover, Agentless-1.5, and PatchPilot do not expose the frozen reproduction artifacts required by those definitions.

B.2 Upstream Localization

MARS operates after file retrieval and localization and does not modify either procedure. Table 5 reports the upstream outputs from the independently executed Lite runs.

Table 5: Upstream retrieval and localization on SWE-bench Lite. AL denotes Agentless Lite, RR denotes RepoRepair, and Ada-Mem denotes AdaRepair-Mem.

Repository	Inst.	File retrieval (%)				File localization (%)			
		AL	RR	Ada-Mem	MARS	AL	RR	Ada-Mem	MARS
astropy/astropy	6	100	100	100	83.33	83.33	100	100	100
django/django	114	95.61	96.49	96.49	99.12	85.09	87.72	92.98	95.61
matplotlib/matplotlib	23	91.30	95.65	95.65	95.65	43.48	69.57	95.65	91.30
mwaskom/seaborn	4	100	100	100	75.00	50.00	75.00	100	100
pallets/flask	3	100	100	100	100	100	100	100	100
psf/requests	6	100	100	83.33	100	83.33	100	83.33	100
pydata/xarray	5	100	100	100	100	80.00	100	100	100
pylint-dev/pylint	6	100	66.67	83.33	100	100	50.00	83.33	100
pytest-dev/pytest	17	100	88.24	94.12	88.24	94.12	5.88	100	100
scikit-learn/scikit-learn	23	100	95.65	95.65	91.30	65.22	73.91	95.65	100
sphinx-doc/sphinx	16	100	100	100	100	75.00	75.00	100	100
sympy/sympy	77	94.81	89.61	88.31	84.42	63.64	84.42	87.01	83.12
Total	**300**	**96.33**	94.00	94.00	93.33	74.67	79.00	92.67	**93.33**

B.3 Backbone Transfer Diagnostics

Table 6 gives the detailed results behind Figure 4(a). Pass@1 badges compare MARS with the stronger baseline within each backbone block, whereas cost badges compare MARS with the ExpeRepair host. The 100% ESR values for DeepSeek R1 and Kimi 2.5 also show that execution success alone does not determine official repair correctness.

Table 6: Backbone transfer on the SWE-bench Verified. Pass@1 badges compare with the stronger baseline in each block; cost badges compare with the ExpeRepair host.

Backbone	Method	Pass@1 (%)	ESR (%)	RSR (%)	Cost ($)
Claude 3.5 Sonnet	ExpeRepair	57.2	84.6	**82.2**	1.740
	AdaRepair-Mem	63.2	78.0	81.8	1.520
	MARS	**66.0** ▲ +2.8	**89.8**	78.8	**1.184** ▼ −32%
DeepSeek R1	ExpeRepair	55.6	100.0	63.0	0.988
	AdaRepair-Mem	54.6	100.0	**65.6**	0.986
	MARS	**57.0** ▲ +1.4	100.0	62.4	**0.622** ▼ −37%
Kimi 2.5	ExpeRepair	60.4	100.0	65.4	0.529
	AdaRepair-Mem	62.2	100.0	64.8	0.495
	MARS	**63.2** ▲ +1.0	100.0	**69.6**	**0.307** ▼ −42%

B.4 Resolved-Set Structure and Cost

The resolved-set analysis compares systems with available per-issue verdicts and describes set overlap rather than providing a controlled ranking. MARS resolves more issues than ExpeRepair and AdaRepair-Mem while using lower average cost than its ExpeRepair host. The result is consistent with the controller changing which search transitions are executed rather than simply extending the search.

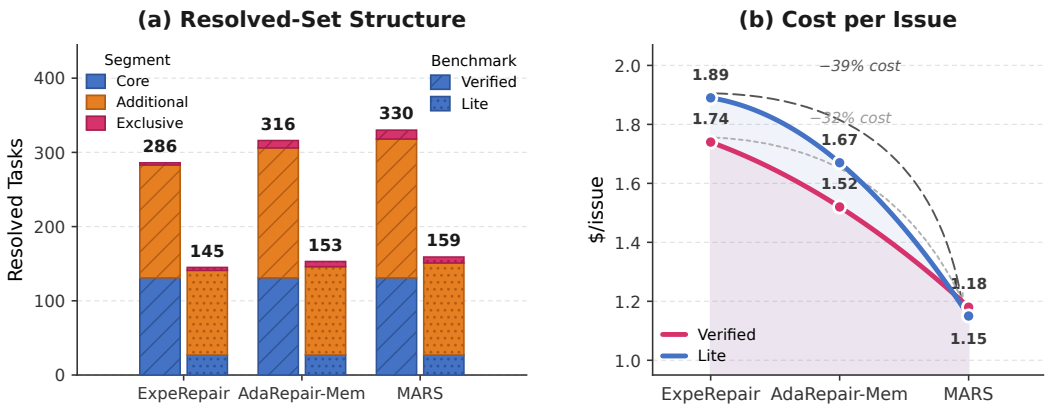


Figure 5: Resolved-set structure and all-in cost on SWE-bench Lite and SWE-bench Verified. **(a)** Resolved issues for ExpeRepair, AdaRepair-Mem, and MARS, partitioned into shared and method-specific components. **(b)** All-in average cost per issue; annotations show the reduction of MARS relative to its ExpeRepair host.

B.5 Trajectory and Repository Stability

We examine cumulative pass@1 over the fixed benchmark order and running pass@1 within each repository. MARS remains competitive across prefixes and repositories rather than obtaining its aggregate result from a narrow late block of instances. Small repositories produce larger rate fluctuations, so these plots are interpreted as evidence of repository breadth rather than as estimates of cross-repository variance.

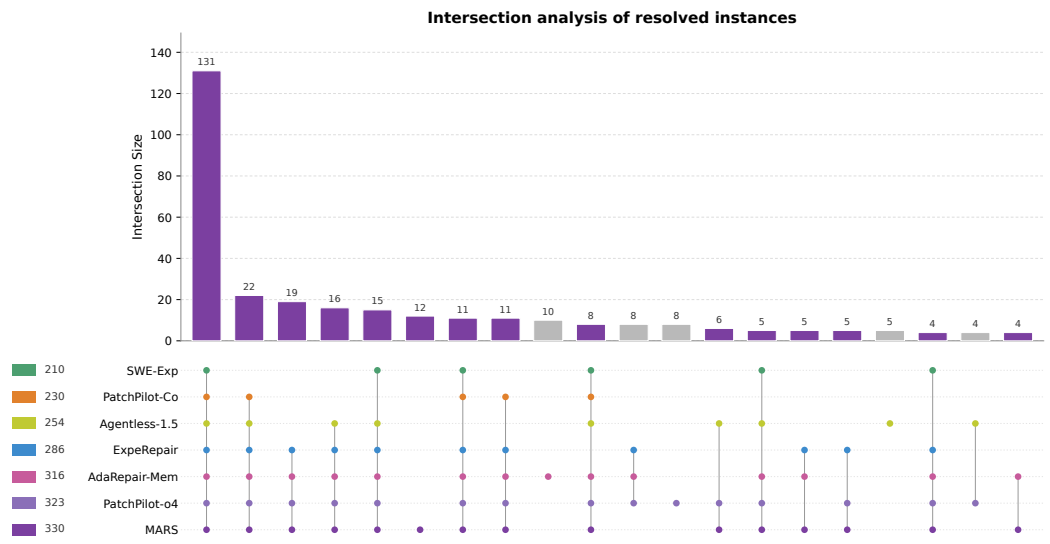


Figure 6: Exact intersections among resolved issue sets on SWE-bench Verified. Purple bars denote intersections containing MARS, and bar heights give issue counts.

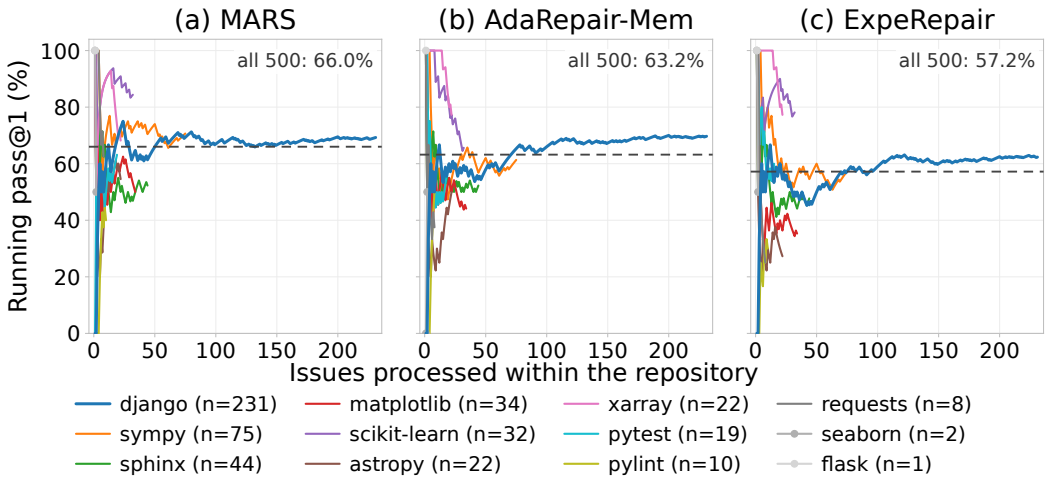


Figure 7: Running pass@1 within each repository of SWE-bench Verified. Panels show MARS, AdaRepair-Mem, and ExpeRepair. Each curve follows the repository-specific issue order, and the dashed line marks the corresponding aggregate pass@1.

B.6 Case Studies

Figure 8 illustrates the two controller effects isolated by the ablations. The first case shows AS terminating a fixed refinement path once the available evidence supports stopping. The second shows IC admitting a weaker challenger without allowing it to replace a better-supported incumbent. Together, the cases illustrate the distinction between adaptive search allocation and state preservation.

B.7 Adaptive Control versus Fixed Budget Reduction

A lower search budget can reduce computation without introducing adaptive control. We therefore test whether the operating point of MARS can be reproduced by simply truncating the host pipeline's fixed search schedule. This experiment complements the mechanism ablations in Table 2: those

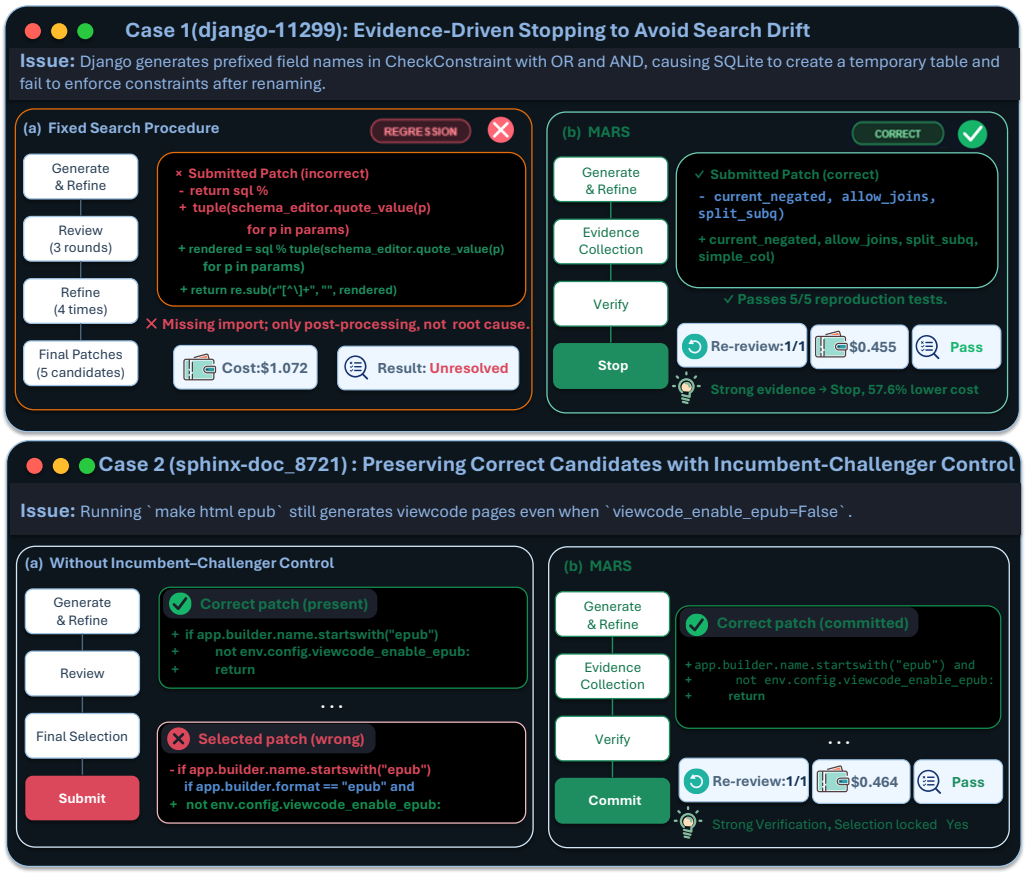


Figure 8: Case studies on SWE-bench Verified. **Left:** AS terminates an unproductive fixed-schedule refinement path. **Right:** IC preserves a stronger incumbent when a later challenger is weaker.

Table 7: Fixed-budget comparison on the same 200-issue SWE-bench Verified subset. MARS badges are relative to Fixed-H.

Configuration	Init. (k)	Refine. (r)	Control	Pass@1 (%)	Cost ($)
Fixed-L	1	0	Fixed	54.5	0.85
Fixed-M	2	1	Fixed	59.0	1.14
Fixed-H	3	2	Fixed	61.5	1.42
ExpeRepair	–	–	Shipped	57.5	1.82
MARS	3	2	**Evidence-gated**	**67.0** ▲ +5.5	**1.31** ▼ −8%

ablations identify the roles of AS, IC, and CC, whereas the comparison here isolates a simpler alternative—whether reducing the amount of fixed search is sufficient to explain the joint accuracy–cost improvement.

Setup. We construct three fixed-budget variants of the ExpeRepair host. Let k denote the maximum number of initial candidate patches and r the maximum number of refinement rounds. The three schedules are Fixed-L $(k{=}1, r{=}0)$, Fixed-M $(k{=}2, r{=}1)$, and Fixed-H $(k{=}3, r{=}2)$. All evidence-conditioned control is disabled in these variants, so each issue follows its prescribed schedule up to the corresponding search ceiling. We additionally report ExpeRepair in its shipped configuration.

None of these four baselines uses MARS's Multi-Evidence Adaptive Search (AS), Incumbent–Challenger Control (IC), or Candidate Commitment (CC). In the notation of Appendix A.2, k cor-

Table 8: Correct-patch lifecycle on the SWE-bench Verified. Losses are measured relative to issues for which a correct patch was generated at least once. *Resolved* is the issue count N_{final}, not a percentage; E2E loss is $N_{\text{ever}} - N_{\text{final}}$ and therefore subsumes pool loss.

Configuration	Ever	Final pool	Pool loss	Resolved	E2E loss	Retention
ExpeRepair	300	292	8	286	14	95.3%
MARS	**342**	**338**	**4**	**330**	**12**	**96.5%**
w/o IC	338	326	12	318	20	94.1%
w/o CC	329	307	22	307	22	93.3%

responds to the initial-candidate count m_0 and r to the refinement-depth limit D_{cfg}. MARS is configured with the same two ceilings as Fixed-H, $(k{=}3, r{=}2)$, but decides online which admissible actions to execute and which repair states to preserve.

Evaluation subset. All configurations are evaluated on the same 200-issue subset of SWE-bench Verified. The subset is drawn once with a fixed seed and reused unchanged across all runs. Sampling is stratified jointly by repository and by the search behavior observed in the default ExpeRepair trace: whether an issue stops at the first plausible patch or proceeds to refinement. The second factor preserves the population mixture of issues on which a reduced search ceiling can and cannot become active. Neither official resolved labels nor outcomes from MARS or the fixed-budget variants are used for sampling. The selected instance identifiers are fixed before evaluation and released with the experimental manifest.

Analysis. The fixed-budget variants establish the host pipeline's response to predetermined search ceilings. Increasing the ceiling from Fixed-L to Fixed-H raises both cost and pass@1, but the accuracy gain diminishes. MARS uses the same ceilings as Fixed-H while choosing actions from runtime evidence.

On the shared subset, MARS reaches 67.0% pass@1 at an average cost of $1.31, whereas Fixed-H reaches 61.5% at $1.42. Matching the search ceilings therefore does not reproduce MARS's operating point: the difference concerns both which actions are executed and which intermediate states remain available. These results show that the tested fixed schedules do not explain the joint accuracy–cost result.

B.8 Mechanism Diagnostics

Aggregate pass@1 does not reveal where a repair trajectory loses a useful candidate, nor does a one-at-a-time ablation fully separate controls that share part of the same execution path. We therefore perform three complementary diagnostics on candidate preservation, the interaction between adaptive search and incumbent protection, and the commitment interface.

B.8.1 Correct-Patch Lifecycle

We first trace official correctness through the candidate lifecycle on SWE-bench Verified. For each issue, *Ever-correct* indicates that at least one officially correct patch was generated at any point during search, while *Final-pool correct* indicates that a correct patch remains among the candidates exposed to final validation and selection. We define

$$L_{\text{pool}} = N_{\text{ever}} - N_{\text{pool}}, \tag{39}$$

$$L_{\text{e2e}} = N_{\text{ever}} - N_{\text{final}}, \tag{40}$$

where N_{ever} is the number of ever-correct issues, N_{pool} the number whose final pool still contains a correct patch, and N_{final} the number whose submitted patch is officially resolved. Since $N_{\text{final}} \le N_{\text{pool}} \le N_{\text{ever}}$, the end-to-end loss L_{e2e} subsumes L_{pool} and additionally absorbs issues for which a correct patch reaches the final pool but is not the patch ultimately submitted. The conditional retention rate is $N_{\text{final}}/N_{\text{ever}}$.

The lifecycle separates candidate discovery from candidate preservation. MARS retains a correct patch in 338 of 342 ever-correct cases. Removing IC increases pool loss from 4 to 12 while changing ever-correct coverage only slightly, which is consistent with later challengers displacing useful

Table 9: 2×2 mechanism-factorial analysis of Adaptive Search (AS) and Incumbent–Challenger Control (IC) on the fixed 200-issue Verified subset. Candidate Commitment is enabled in all four cells. E2E loss is $N_{\text{ever}} - N_{\text{final}}$ and therefore subsumes pool loss. The first row is the full controller.

AS	IC	Pass@1	Cost ($)	Ever	Final pool	Pool loss	E2E loss
✓	✓	**67.0%**	**1.31**	**140**	**138**	**2**	**6**
✓	×	64.0%	1.27	137	130	7	9
×	✓	61.5%	1.53	129	127	2	6
×	×	59.0%	1.39	127	121	6	9

Table 10: Adaptive-search policy ladder on the 200-issue SWE-bench Verified subset. Each row adds one adaptive signal while keeping Incumbent–Challenger Control and Candidate Commitment enabled. Exec., Rev., Ver., and Res. denote execution, review, verification, and resource signals, respectively; the reference disables adaptive search. Costs are all-in averages per issue.

	Signal used adaptively				Operating point		
AS policy	**Exec.**	**Rev.**	**Ver.**	**Res.**	**Pass@1**	**Cost**	**Δ vs. reference**
AS disabled (reference)	–	–	–	–	61.5	1.53	reference
Budget-only	–	–	–	✓	60.5	1.23	−1.0 pp, −20%
Verifier-only	–	–	✓	✓	63.0	1.34	+1.5 pp, −12%
Review + verifier	fixed	✓	✓	✓	64.5	1.37	+3.0 pp, −10%
MARS AS (full)	✓	✓	✓	✓	**67.0**	**1.31**	+5.5 pp, −14%

states. Removing CC creates a larger gap between ever-correct and final-pool coverage because fewer accumulated candidates are exposed at termination.

B.8.2 Factorial Analysis of AS and IC

The standard ablations modify AS and IC individually, but the two mechanisms share part of the confidence-controlled execution path. We therefore perform a 2×2 mechanism-factorial analysis on the same fixed 200-issue Verified subset used in Appendix B.7. All four cells share the same initial-candidate and refinement-round ceilings (k=3, r=2), the same backbone, the same upstream retrieval and localization configuration, and the same evaluation protocol. Candidate Commitment and all remaining controller settings are held fixed across the four cells; only AS and IC are toggled.

The four cells are MARS variants with individual controller components disabled, and are therefore distinct from the host-side fixed schedules of Appendix B.7, which remove the MARS controller entirely.

B.8.3 Commitment under Matched Validation

The factorial results separate the roles of AS and IC. AS changes candidate discovery and cost: enabling AS improves pass@1 and reduces cost under both IC settings. IC changes candidate retention more than ever-correct coverage: removing IC increases pool loss and end-to-end loss while changing discovery only slightly. The two mechanisms therefore act at different stages and should not be interpreted as an additive decomposition of the full gain.

Decomposing the adaptive-search axis. Table 10 decomposes AS by adding execution, review, verification, and resource signals as adaptive inputs while keeping IC and CC enabled. The resource signal alone reduces cost but does not improve accuracy, whereas review and verification recover part of the accuracy gain. The full policy additionally uses execution feedback to terminate or redirect refinement paths, yielding the strongest joint accuracy–cost operating point.

The original *w/o-CC* ablation forwards only one patch and therefore also reduces downstream validation. It cannot by itself distinguish candidate-state preservation from validation volume. We therefore compare MARS with a matched control that uses the same final-pool size and the same number of downstream validation calls per issue.

The matched control runs the same AS and IC controller as MARS. It differs only in how candidates are exposed at termination: MARS forwards the order-preserving, deduplicated list accu-

Table 11: Commitment control on SWE-bench Verified. The matched control uses the same final-pool size and downstream validation budget as MARS.

Configuration	Pool	Validation	Pass@1	Ever	Final pool	Pool loss
MARS	3.0	matched	**66.0%**	**342**	**338**	**4**
Matched, *w/o CC*	3.0	matched	64.8%	340	328	12
Original *w/o CC*	1.0	lower	61.4%	329	307	22

mulated during the search, subject to the applicable write-out path, whereas the control forwards the controller-selected patch together with the most recent host-side candidates until the emitted pool reaches the same size. Neither arm uses controller-side reranking or official resolved labels to construct the pool.

Because the matched control is independently executed, its ever-correct coverage differs from MARS by two issues. MARS retains a correct patch in 338 of 342 ever-correct cases, whereas the matched control retains one in 328 of 340 cases. This difference is consistent with a benefit from preserving accumulated candidate state.