REPAIRING FAILED CODING-AGENT TRAJECTORIES FOR COST-EFFICIENT TRAINING DATA EXPANSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Coding-agent trajectories provide supervision for model training and evidence for failure analysis, yet collecting them through repeated rollouts is costly and often produces unsuccessful runs. Existing curation pipelines largely retain solved trajectories or regenerate complete runs, leaving action-level reuse of failed trajectories insufficiently explored. We introduce ReComp, a framework for augmenting coding-agent trajectory datasets by recomposing failed runs. ReComp localizes anomalous actions, replaces them with compatible spans from successful trajectories, and verifies the resulting repository states. For the first research question (RQ1), the strongest checker reaches 93.72 anomaly F1, while a compact 3B variant retains 87.77 F1 and powers the remaining pipeline. In RQ2, 18.23% of failed trajectories pass build and executable functional tests, and 11.78% yield verified alternative implementations, outperforming the evaluated local-replacement baselines. In RQ3, three-seed training on a 16,000-record source-verified, state-aligned corpus reduces held-out action negative log-likelihood by 22.05% relative to the donor-context baseline. A separate resource-normalized evaluation finds that the marginal cost of acquiring an accepted output is 3.9–210$\times$ lower than complete rollout with the evaluated agents. These results show that trajectory recomposition recovers useful training supervision from failed coding-agent runs without generating complete replacement rollouts.

1 INTRODUCTION

Solving a repository-level coding task involves more than producing a final patch. An agent must inspect unfamiliar files, locate relevant functionality, modify code, and respond to feedback from tools and execution (Yang et al., 2024; Wang et al., 2025c). Trajectories record code changes, action contexts, and execution feedback, revealing how a solution develops over successive decisions. Prior work has established trajectory data as valuable for training coding agents and diagnosing failures in LLM agents (Pan et al., 2025; Yang et al., 2025; Wang et al., 2025a). Yet collecting trajectories requires repeated model calls and repository execution, making large-scale rollout collection costly, particularly with frontier models. These costs also accrue to unsuccessful runs, which successful-only data curation leaves unused for training.

Recent studies show that failed trajectories can also provide useful training and diagnostic signals through preference construction, step-level correction, trajectory grafting, and guided continuation (Song et al., 2024; Wang et al., 2025a; Li et al., 2026b; Lauffer, 2026). Compared with generating another complete rollout, reusing compatible recorded actions to repair failed coding trajectories could reduce collection costs and diversify training contexts.

At the same time, training only on successful trajectories provides limited coverage of the states an agent encounters. Prior work ties the value of trajectory data to its coverage of decision states and corrective signals, beyond the outcome of the complete rollout (Wang et al., 2025b; Chen et al., 2025b). Pairing difficult states exposed by failed runs with corrected actions can provide broader and more diverse positive supervision than successful trajectories alone (Wang et al., 2025a; Xia et al., 2025b). Therefore, trajectory analysis has drawn growing attention in both academic research and industrial agent development.

Despite this promise, scalable action-level trajectory reuse remains underexplored due to three key challenges. **First, trajectory availability does not imply repair-ready supervision.** Executable

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.

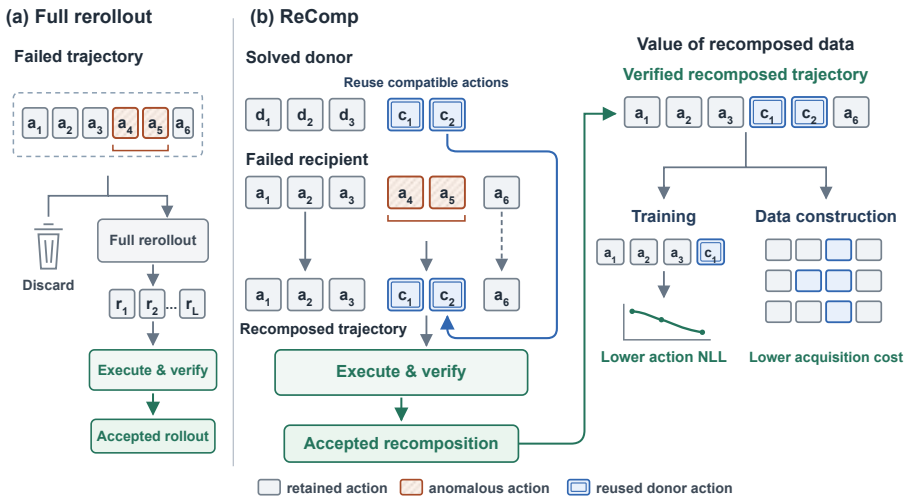

Figure 1: ReComp overview. (a) Conventional curation discards failed work or regenerates a complete rollout. (b) ReComp preserves compatible recipient actions, replaces only the suspicious span, and verifies the recomposed execution. The resulting trajectories provide additional training supervision while reducing acquisition cost through action reuse.

task datasets support agent training, but action-level reuse requires normalized tool records, aligned failed and successful actions, and compatibility labels that run-level outcomes do not provide (Pan et al., 2025; Yang et al., 2025). **Second, correctness is state-dependent and costly to verify.** A repair may depend on state absent from the recipient run or invalidate later steps, so verification requires repository reconstruction, execution, and build or task-specific tests (Yang et al., 2025). **Third, repair quality must be balanced against acquisition cost.** Repair is worthwhile only if diagnosing, constructing, and verifying a corrected trajectory costs less than generating a new complete rollout. A practical approach must therefore be lightweight, inexpensive, and fast without sacrificing repair quality. These challenges motivate an integrated pipeline that turns mixed successful and failed traces into action-level supervision and couples compact decision models with execution verification.

To address these challenges, we introduce ReComp, an integrated framework with two learned stages for recomposing failed coding-agent trajectories into training supervision (Figure 1). A checker first localizes suspicious actions, and a planner then selects a compatible donor span and its replacement boundaries. ReComp replaces only the localized span with recorded donor actions, preserves compatible existing work, and avoids generating a complete rerun (Figure 1b). In the example in Figure 1b, donor actions c_1, c_2 replace suspicious actions a_4, a_5, with the prefix and suffix retained subject to state compatibility. The stitched intervention is executed and its repository state checked by the verifier, completing the select–stitch–verify flow. Accepted recompositions provide additional context–action supervision. The right side of Figure 1b summarizes the two resulting benefits evaluated later: downstream training utility and lower acquisition cost per accepted trajectory.

We evaluate ReComp through three research questions (RQs) that cover complementary dimensions of its design. RQ1 evaluates anomaly localization by asking whether a compact checker can identify suspicious actions in failed trajectories. These locations define the recipient windows used by RQ2, which evaluates whether a planner can select compatible donor spans that produce verified endpoints after execution. RQ3 applies the same recomposition principle to a source-verified, state-aligned corpus and evaluates both downstream training value and acquisition cost through matched-checkpoint utility, measured throughput, and cost per accepted output. Together, the three questions connect diagnosis, executable recomposition, and data utility without requiring the evaluations to reuse the same sample population.

In our experiments, the Qwen2.5-Coder-7B checker is strongest for RQ1, reaching 93.72 F1, while Qwen2.5-Coder-3B reaches 87.77 F1, offers a favorable accuracy–efficiency trade-off, and is therefore used throughout the remaining pipeline. For RQ2, the planner reaches 98.57 selection F1

and surpasses all local-replacement baselines in strict acceptance, which requires the final endpoint to pass build and its executable functional test. It improves over the strongest baseline by 1.06 percentage points (18.23% versus 17.18%, paired exact $p < 10^{-4}$) and achieves the highest alternative-implementation yield (11.78%). For RQ3, relative to the matched baselines, ReComp reduces held-out action negative log-likelihood by 22.05% across three training seeds. In a separate resource-normalized acquisition study, its measured marginal full-pipeline cost per accepted output is 3.9–210$\times$ lower than complete rollout with the evaluated agents.

Our contributions are:

- We formulate offline coding-agent trajectory recomposition, which transfers recorded actions from solved donors into aligned failed-recipient states to construct corrective supervision without generating new complete rollouts.
- We develop a compact checker–planner framework with task-specific adapters on a shared 3B backbone, keeping anomaly localization and compatible-span selection lightweight before repository execution and endpoint verification.
- We show through matched target supervision that failure-state contexts improve downstream action modeling, and that reusing an existing trajectory library lowers the marginal acquisition cost per accepted trajectory relative to complete rollout with the evaluated agents.

2 Related Work

Coding agents and trajectory datasets. SWE-agent and OpenHands establish repository-level agents that interact with files, shells, and execution environments (Yang et al., 2024; Wang et al., 2025c). SWE-Gym, SWE-smith, and SWE-RL turn executable software tasks and collected trajectories into supervision for training coding agents (Pan et al., 2025; Yang et al., 2025; Wei et al., 2025). AgentTrek instead synthesizes agent trajectories through guided replay (Xu et al., 2025). However, these methods primarily collect, synthesize, or learn from fresh trajectories. ReComp recovers additional supervision from failed and solved runs that have already been collected.

Failed-trajectory reuse. ETO and NAT show that failed trajectories can support preference learning and negative-aware tuning (Song et al., 2024; Wang et al., 2025b). ATLAS selects critical expert steps, while STeCa identifies deviating actions and constructs calibrated trajectories (Chen et al., 2025b; Wang et al., 2025a). SAND explores alternatives at expert states, P-BRIDGE distills repaired continuations, and T-STAR grafts corrective reasoning at divergence points (Xia et al., 2025b; Guo et al., 2026; Li et al., 2026b). CSO identifies critical actions in failed trajectories, generates expert alternatives, and uses branch rollouts to retain outcome-flipping replacements for step-level preference learning (Li et al., 2026a). Concurrent work on TrajAD trains trajectory-level anomaly detectors that identify and localize abnormal agent behavior (Liu et al., 2026). ReComp differs from these methods by reusing actions already recorded in solved coding trajectories and validating their transfer through repository execution. It requires neither newly generated corrective actions nor branch rollouts during recomposition. Our central question is whether recorded actions can provide useful supervision after transfer into difficult failure states.

Trajectory augmentation. Self-Instruct generates new instructions, while DEITA, LESS, and AlpaGasus select high-value examples for instruction tuning (Wang et al., 2023; Liu et al., 2024b; Xia et al., 2024; Chen et al., 2024). Magicoder constructs code instruction data from open-source code, and SemCoder uses executable semantic supervision for code models (Wei et al., 2024; Ding et al., 2024). Concurrent work on SERA constructs low-cost repository-agent data through soft verification (Shen et al., 2026). Repeated-data studies further show that data composition matters under constrained budgets (Muennighoff et al., 2023). However, these approaches generate, filter, or repeat independent examples and do not repair stateful coding-agent traces. ReComp augments trajectory data by pairing a failed context with compatible recorded actions and then validating the recomposed repository state.

3 METHODOLOGY

ReComp converts a failed coding-agent trajectory into recomposed action supervision through four stages: anomaly localization, donor retrieval, compatibility planning, and execution-grounded recomposition. Figure 2 maps these stages to the modules defined below. The framework separates two decisions that are often conflated. The checker identifies which part of a failed run should be reconsidered, while the planner determines whether a recorded action span can be transferred from a solved run. An executor then applies the proposed intervention to the receiving repository state and returns evidence for strict endpoint acceptance. We evaluate the resulting endpoints through repository-level verification. A separate downstream study constructs source-verified, state-aligned context–action examples at scale to measure their training value.

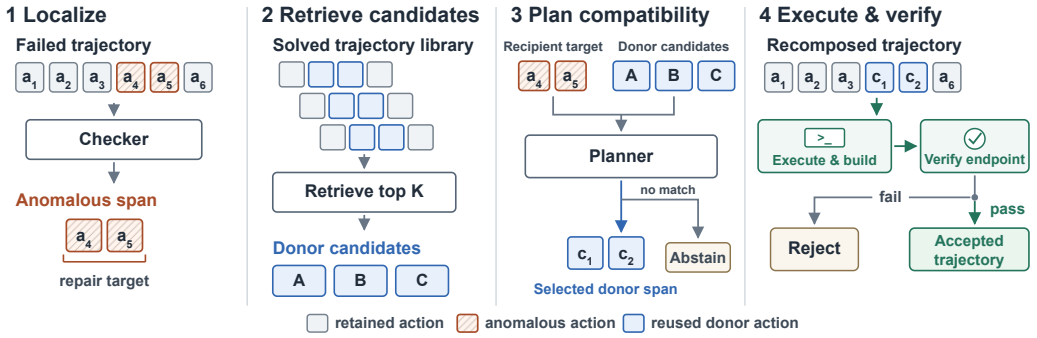

Figure 2: ReComp methodology. The checker localizes an anomalous recipient span. Retrieval collects plausible donor spans, and the planner selects a compatible span and its repair boundaries or abstains. The executor applies the intervention and checks the resulting build before endpoint verification.

3.1 PROBLEM FORMULATION

We represent an agent trajectory as

$$\tau = (s_0, a_1, o_1, s_1, \ldots, a_T, o_T, s_T), \tag{1}$$

where s_t is the repository state, a_t is an action, o_t is the resulting observation, and T is the number of actions. We write τ^- for a failed recipient trajectory, τ^+ for a solved donor trajectory, and $\mathcal{D}^+$ for the solved-donor library. Our goal is to construct an additional training example from these recorded executions without generating another complete rollout.

Let $\mathbf{a}^-$ and $\mathbf{a}^+$ be the action sequences of a failed recipient trajectory and a solved donor trajectory. Given a recipient span $[i, j]$ and a donor span $[u, v]$, recomposition proposes

$$\widetilde{\mathbf{a}} = \mathbf{a}^-_{<i} \parallel \mathbf{a}^+_{u:v} \parallel \mathbf{a}^-_{>j}. \tag{2}$$

Here $\parallel$ denotes sequence concatenation, so the donor actions from u through v replace the recipient actions from i through j. The recipient span may be empty: setting $j = i - 1$ inserts the donor span immediately before a^-_i without removing any recipient action. Equation 2 defines a candidate intervention, not a valid trajectory. An action that succeeded in one run may be incompatible with the files, dependencies, or execution history of another. ReComp must therefore localize $[i, j]$, retrieve plausible donor windows, select a compatible replacement, and validate the resulting execution. Selection precedes execution, and a candidate enters the curated output only after repository-level verification.

3.2 ANOMALY LOCALIZATION

Step 1 in Figure 2 converts a run-level failure into an action-level repair target. For each action a^-_t, the checker uses its pre-state, resulting observation, and post-state to estimate

$$p_t = \Pr(z_t = 1 \mid s^-_{t-1}, a^-_t, o^-_t, s^-_t), \tag{3}$$

where $z_t = 1$ denotes evidence of an anomalous action. Adjacent flagged actions are grouped into a candidate recipient span $[i, j]$. Training maximizes the likelihood of action-level anomaly labels. When the evidence does not support localization, the checker stops the recomposition path without forcing a repair.

3.3 Donor Retrieval and Compatibility Planning

Steps 2 and 3 in Figure 2 retrieve plausible donor spans and determine whether they transfer to the recipient state. A lightweight retriever ranks spans from $\mathcal{D}^+$ using observable action signatures such as affected paths, tool roles, and state effects. It returns the K highest-ranked candidates as $\mathcal{C}_K = \{d_1, \ldots, d_K\}$, where K is the number of retained candidates, d denotes a donor span, and w^- is the recipient window. The retrieval step is

$$\mathcal{C}_K = \mathrm{Retrieve}_K(w^-, \mathcal{D}^+). \tag{4}$$

The planner first retains candidates whose learned compatibility probability is at least θ, forming the subset $\mathcal{C}_K^{\mathrm{comp}}$. It then applies the retrieval ranking within this subset:

$$\widehat{d} = \arg \max_{d \in \mathcal{C}_K^{\mathrm{comp}}} r(w^-, d), \tag{5}$$

where $r(w^-, d)$ is the retrieval score. ReComp abstains when no candidate passes the filter. The planner also finalizes the four boundaries i, j, u, v used by Equation 2. Retrieval therefore proposes plausible spans, while the learned planner determines whether and where they transfer. The checker and planner use a shared backbone with separate task-specific adapters. Planner training uses compatible alignments as positive supervision and mismatched spans as negative supervision.

3.4 Execution-Grounded Recomposition

Step 4 in Figure 2 materializes the selected intervention from the recipient pre-state. The executor applies each proposed action to the current repository state and records the resulting observation and next state, producing the recomposed trajectory $\widetilde{\tau}$. Recorded donor observations are never copied into the recomposed run because they describe a different execution state. Suffix actions are retained only as part of the proposed composition and remain subject to execution and state verification.

We distinguish benchmark-facing build success from a stricter endpoint decision:

$$\mathrm{Accept}(\widetilde{\tau}) = \mathrm{Build}(\widetilde{\tau}) \wedge \mathrm{Verify}(\widetilde{\tau}). \tag{6}$$

The verification step executes a validated task-specific functional test. We use dataset-provided tests when available and construct executable tests with LLM assistance otherwise. Each constructed test is validated to fail on the original failed state and pass on a known-correct state before it is frozen for endpoint evaluation. Appendix A.5 specifies the protocol. The acceptance decision does not require every intermediate command to exit successfully. Buildability and strict endpoint acceptance answer different questions. The framework retains a recomposed artifact only under the acceptance policy selected for its downstream use.

3.5 Recomposed Training Supervision

An accepted output of Step 4 can become a recomposed training record. For scalable offline construction, we also admit actions from donor trajectories marked verified or resolved by their source dataset when the donor and recipient share the same pre-action source state. Let c^- be the recipient context before the repaired action, c^+ the donor's original context, and a^+ the compatible donor action. ReComp constructs two supervision sets from the same N aligned records:

$$\begin{aligned} \mathcal{D}_{\mathrm{ReComp}} &= \{(c_n^-, a_n^+)\}_{n=1}^N, \\ \mathcal{D}_{\mathrm{Donor}} &= \{(c_n^+, a_n^+)\}_{n=1}^N. \end{aligned} \tag{7}$$

For each aligned record, both supervision relations use the same target action a_n^+ but condition it on different contexts. ReComp uses the compatible failed-recipient context c_n^-, whereas Donor retains the original solved context c_n^+. Training with standard conditional likelihood under matched targets therefore isolates the value of recomposed contexts.

4 EXPERIMENTAL SETUP

Shared setting. We use coding-agent trajectories from NanoSWE, a heterogeneous rollout collection, and SWE-smith, an executable repository-level task and trajectory resource (ricdomolm, 2026; Yang et al., 2025; SWE-bench, 2025). We use task-hashed splits throughout and additionally hold out entire repositories for downstream evaluation.

4.1 RQ1: ANOMALY DETECTION

Data and models. RQ1 uses a 1,615-record main test and a separate 847-record model-comparison split. We evaluate checkers from two model families and multiple scales: Qwen2.5-Coder at 3B, 7B, 14B, and 32B (Hui et al., 2024), and DeepSeek-Coder at 1.3B and 6.7B (Guo et al., 2024). Fine-tuned checkers use an operation-evidence warm start followed by semantic SFT.

Baselines. For each architecture and scale, the corresponding untuned checkpoint is the matched baseline for measuring the effect of checker fine-tuning.

Metrics. We report anomaly precision, recall, F1, and false alarms.

4.2 RQ2: RECOMPOSITION AND VERIFICATION

Data and model. RQ2 uses failed-recipient and solved-donor actions from the two source datasets. Its task-disjoint selection test contains 6,107 candidate pairs, and its execution cohort contains 2,649 failed trajectories with restorable repository states. We fine-tune Qwen2.5-Coder-3B as the compatibility planner.

Baselines. All local methods receive identical candidates. *Random span* tests arbitrary donor reuse, *Path* ranks candidates by affected-path overlap, and *Retrieval* ranks them by embedding similarity (Liu et al., 2024a). *Planner + retrieval* first filters candidates by learned compatibility and then applies the same retrieval ranking. *Full donor* replays the complete successful mutation sequence as a higher-cost control.

Metrics. Selection is evaluated by compatibility F1 and exact-span recall. To test robustness, we add $D \in \{0, 2, 5, 10, 20\}$ unmarked incompatible spans to each candidate pool and measure compatible-hit yield. At execution time, build pass provides benchmark-facing screening (Jimenez et al., 2024; Yang et al., 2025). Strict acceptance requires a build-passing endpoint and passage of its task-specific executable functional test. Appendix A.5 details test construction and validation. Alternative-implementation yield counts strictly accepted outputs whose production-code delta differs from the reference.

4.3 RQ3: TRAINING UTILITY AND COST

Data and models. RQ3 uses 16,000 source-verified, state-aligned relations from 619 repositories and evaluates transfer on 1,769 records from 80 held-out repositories. We fine-tune Qwen2.5-Coder-3B under Donor and ReComp conditions for three seeds. We also train a shuffled-context ablation that preserves the target actions and corpus size while permuting their recipient contexts. The cost study uses 2,649 failed trajectories and 20 executable tasks constructed from the same source project pools.

Baselines. Donor learns each target action from its original solved context, whereas ReComp uses an aligned failed-recipient context. The comparison matches target actions, 3,641,879 retained target tokens, backbone, and training recipe. The shuffled-context condition isolates whether ReComp's gain depends on aligning each corrective action with its recipient failure state.

Metrics. Mean action negative log-likelihood (NLL) measures held-out target-action loss, where lower is better (Bengio et al., 2003). Verified endpoint success requires a build-passing endpoint and passage of its task-specific executable functional test. Acquisition efficiency is the measured marginal cost per verifier-accepted trajectory given an existing trajectory library, charging unsuccessful attempts for both ReComp and complete-rollout agents.

Appendix A provides complete data-construction, training, replay, acceptance, and resource-accounting details.

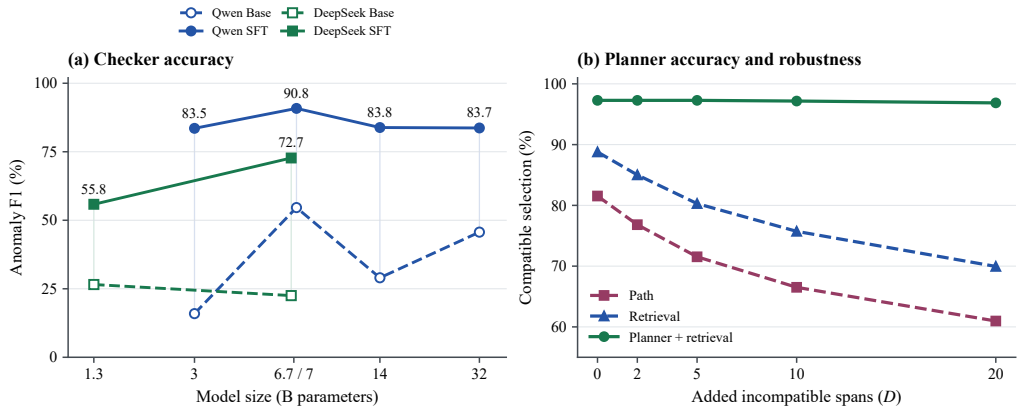

Figure 3: Checker and planner effectiveness. (a) Checker F1 on the same 847-record full-sampled split. Dashed curves denote base models and solid curves denote SFT models. (b) Compatible-hit yield after adding D construction-labeled incompatible distractors per failed-recipient query. $D = 0$ uses only the original candidate pools.

5 Results

5.1 RQ1: Compact Trajectory Anomaly Detection

RQ1 examines whether trajectory anomaly detection can be delegated to a compact model that serves as the first-stage gate in ReComp.

Figure 3(a) shows consistent gains from SFT on the shared 847-record split. Qwen2.5-Coder at 3B, 7B, 14B, and 32B improves from 15.93, 54.63, 29.01, and 45.67 to 83.55, 90.82, 83.84, and 83.68 F1, respectively. DeepSeek-Coder at 1.3B and 6.7B improves from 26.55 and 22.51 to 55.83 and 72.73 F1. On the 1,615-record main test, Qwen7B is the strongest checker with 98.15% precision, 89.68% recall, and 93.72 F1, while Qwen3B reaches 92.18%, 83.76%, and 87.77. Across comparable configurations, SFT adds 29.28–67.62 F1 points, exceeding the effect of model scaling alone. Because 3B retains 93.7% of the best F1 with less than half the nominal parameter count, we select it as the accuracy–efficiency configuration for the remaining pipeline.

Case study. On a held-out SunPy trajectory, the untuned 3B model incorrectly flagged six corrected edit-and-test spans as anomalous, whereas the SFT checker correctly recognized the trajectory as completed. This shows that SFT distinguishes unresolved failures from errors corrected later in the trajectory. Further case details are provided in Appendix A.1.

Together, these results show that SFT on our constructed anomaly data enables compact models to identify anomalous trajectory actions effectively, validating both the training paradigm and the checker-data construction. This compact anomaly gate provides the localized failure evidence used by the planner in RQ2.

5.2 RQ2: Can Donor Actions Produce Verified Alternatives?

RQ2 evaluates two stages of trajectory repair. First, we test whether the planner selects a compatible donor span and identifies the actions to replace. Second, we execute the selected actions from the failed trajectory's repository state and verify the resulting endpoint.

Robustness to incompatible candidates. With the original pools ($D = 0$, 2.3 candidates per query on average), Planner + retrieval achieves 97.28% compatible-hit yield, outperforming Retrieval by 8.49 percentage points and Path by 15.74 points. At the pair level, the planner also achieves 98.57 F1 and recovers both replacement boundaries for 3,050/3,094 positive pairs (98.58%).

As D increases, Figure 3(b) shows that Planner + retrieval ranks first at every noise level and loses only 0.41 points, compared with 18.84 and 20.57 for Retrieval and Path. At $D = 20$, it retains 96.87% compatible-hit yield versus 69.95% and 60.97%, respectively.

Case study. On a Bottle HTTP-date task, Retrieval incorrectly chose an unrelated Unicode-decoding span because it edited the same file with a similar action pattern. The planner correctly selected an HTTP-date donor that changed the endpoint from build failure to build pass. Further case details are provided in Appendix A.1.

Executed recomposition. Table 1 reports the execution results on the 2,649 failed trajectories.

Table 1: Verified recomposition on 2,649 failed trajectories. Counts are followed by fixed-cohort percentages, and columns form a cumulative acceptance funnel.

Method	Build pass	Strict accepted	Alternative impl.
Random span	860 (32.47%)	284 (10.72%)	243 (9.17%)
Path heuristic	810 (30.58%)	426 (16.08%)	280 (10.57%)
Retrieval top-1	850 (32.09%)	455 (17.18%)	296 (11.17%)
Planner + retrieval	**861 (32.50%)**	**483 (18.23%)**	**312 (11.78%)**
Full donor sequence	1,015 (38.32%)	799 (30.16%)	101 (3.81%)

Table 1 makes the verification funnel explicit. For Planner + retrieval, 861 outputs pass the repository build check, 483 also pass strict endpoint acceptance, and 312 of those are alternative implementations. These are the highest observed yields among local-replacement methods. The paired comparison with Retrieval contains 449 joint accepts, 34 Planner-only accepts, and six Retrieval-only accepts. The resulting 1.06-point gain has a paired bootstrap 95% interval of [0.60, 1.51] points and an exact McNemar $p < 10^{-4}$. Full donor replay has higher build and strict yields, but only 101 outputs (3.81%) differ from the reference, limiting its diversity to implementations already present in the donor library.

RQ2 therefore shows that learned compatibility filtering is robust to candidate noise and provides the best strict and alternative-implementation yields among local replacement methods. RQ3 next tests whether recomposed trajectories provide effective supervision at lower acquisition cost.

5.3 RQ3: Training Utility and Acquisition Efficiency

RQ3 tests whether recomposed trajectories provide useful downstream supervision and whether that supervision can be acquired efficiently. We compare matched Donor and ReComp checkpoints through action modeling and execution replay, then measure acquisition cost.

Downstream training utility. Table 2 summarizes the matched three-seed comparison. ReComp reduces mean held-out action NLL from 0.3527 to 0.2750, a 22.05% reduction, and improves in every seed by 21.79–22.46%. Independent replay of the same checkpoints provides a secondary execution check: mean verified endpoint success rises from 1.39% to 2.12% on the 769 eligible boundaries, with a positive difference in every seed (Appendix A.8). When recipient contexts are shuffled while target actions and corpus size remain fixed, mean NLL instead rises to 0.3806 across three seeds, above both Donor and ReComp. Thus, additional action supervision alone does not explain the improvement. Each corrective action must be aligned with the recipient failure state.

Table 2: Matched downstream utility (mean $\pm$ SD over three seeds). NLL uses all 1,769 held-out records, and verified endpoint success uses 769 eligible boundaries. The shuffled-context ablation is evaluated on NLL only.

Condition	Action NLL $\downarrow$	Verified endpoint success $\uparrow$
Donor	0.3527 ± 0.0020	$1.39 \pm 0.20\%$
Shuffled context	0.3806 ± 0.0008	–
ReComp	$\mathbf{0.2750 \pm 0.0002}$	$\mathbf{2.12 \pm 0.08\%}$

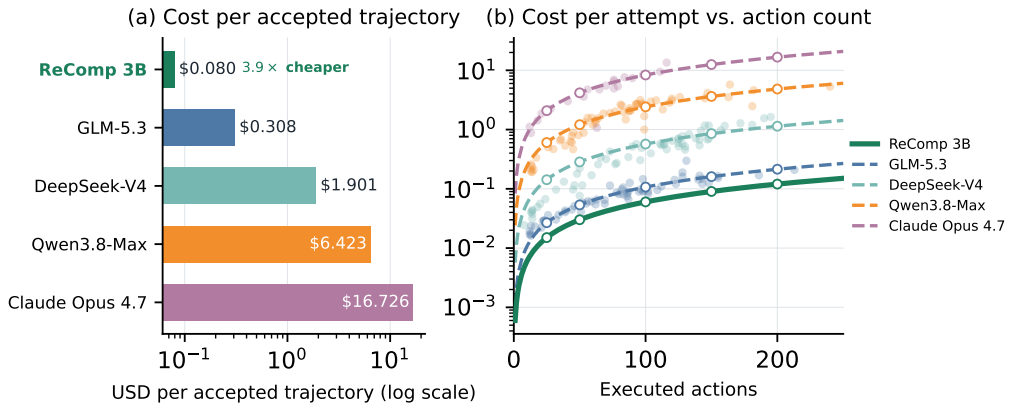

Figure 4: Acquisition efficiency. (a) Cost per verifier-accepted trajectory. (b) Task-level observations and fitted action-cost curves. Both panels use **logarithmic cost axes**, and lower is better.

Case study. On a held-out TreeSet task, all three Donor checkpoints fail to repair `Empty()`: they predict either the wrong edit direction or an invalid edit, and none passes execution. In contrast, all three ReComp checkpoints make the required correction and pass endpoint replay. This difference reflects the supervision provided by verified recomposition. Golden trajectories demonstrate actions only in already successful states, whereas recomposed trajectories pair unfamiliar failure states with corrective actions that have been execution-verified. ReComp therefore teaches the model how to recover from states absent from golden-path supervision, which is central to its downstream gain. Further case details are provided in Appendix A.1.

Acquisition efficiency. Figure 4 compares ReComp with complete-rollout baselines using GLM-5.3, DeepSeek-V4, Qwen3.8-Max, and Claude Opus 4.7 (Z.ai, 2026; DeepSeek-AI, 2026; Alibaba Cloud Model Studio, 2026; Anthropic, 2026), charging all failed attempts. ReComp's measured marginal acquisition cost is \$0.080 versus \$0.308–\$16.726, making repair 3.9–210× cheaper than these baselines. ReComp uses 4.74 donor actions per attempt on average, while complete rollouts execute 62.3–95.7 actions per attempt. Panel (b) shows that the absolute cost gap widens with longer trajectories. Appendix Table 9 reports the attempts, accepted outputs, and total measured charges underlying these ratios.

RQ3 shows that recomposed contexts improve matched action learning and endpoint replay across three seeds, while costing less per accepted trajectory than complete rerollout.

6 Discussion and Future Work

ReComp has two main limitations. First, repair depends on finding a donor span compatible with the failed recipient state, so the planner must abstain when the solved-trajectory library lacks one. Second, the current method assumes a localized intervention whose retained actions remain compatible, whereas distributed errors may require coordinated multi-span repair. Future work could combine action reuse with a generative fallback for unmatched states and iterative replanning guided by verification feedback.

7 Conclusion

In this paper, we introduced ReComp to localize anomalous actions, transfer compatible recorded donor spans, and verify recomposed executions. It yields 483 accepted endpoints from 2,649 failed trajectories. Across three matched seeds, recomposed contexts reduce action NLL by 22.05%, improve endpoint replay, and cost less than complete rollout. ReComp thereby recovers training value from failed trajectories without generating complete replacement rollouts.

Reproducibility Statement

Sections 3–5 document the method, data, training, metrics, and cost accounting. Appendix A provides implementation, verification, and provenance details. Our anonymous artifact at https://anonymous.4open.science/r/trajectory_repair_code-DC42 releases the experiment code, sanitized configurations, aggregate evidence, tests, data-reconstruction instructions, and the final planner LoRA adapter with checksum-verified reconstruction.

AI Use Statement

Generative AI tools assisted experiment implementation and analysis, literature search, and polishing the manuscript and figures. The authors reviewed all AI-assisted outputs and take responsibility for the final content.

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*, 2025. URL https://aclanthology.org/2025.acl-long.973/.

Alibaba Cloud Model Studio. Qwen3.8-Max model documentation. Online model documentation, 2026. URL https://help.aliyun.com/en/model-studio/qwen3-8-max. Accessed September 25, 2026.

Anthropic. Claude Opus 4.7 system card. System card, 2026. URL https://www-cdn.anthropic.com/037f06850df7fbe871e206dad004c3db5fd50340/Claude%20Opus%204.7%20System%20Card.pdf. Accessed September 25, 2026.

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*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a1e6783e4d739196cad3336f12d402bf-Abstract-Conference.html.

AutoDL. GPU Rental Pricing. Online pricing page, 2026. URL https://www.autodl.com/home. Accessed September 22, 2026.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003. URL https://www.jmlr.org/papers/v3/bengio03a.html.

Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, and Hongxia Jin. AlpaGasus: Training a better alpaca with fewer data. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=FdVXgSJhvz.

Xiancai Chen, Zhengwei Tao, Kechi Zhang, Changzhi Zhou, Xinyu Zhang, Wanli Gu, Yuanpeng He, Mengdi Zhang, Xunliang Cai, Haiyan Zhao, and Zhi Jin. Revisit self-debugging with self-generated tests for code generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pp. 18003–18023, 2025a.

Zhixun Chen, Ming Li, Yuxuan Huang, Yali Du, Meng Fang, and Tianyi Zhou. ATLAS: Agent tuning via learning critical steps. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025b. URL https://aclanthology.org/2025.findings-acl.1299/.

CloudGPU. GPU Rental Pricing per Hour. Online pricing page, 2026. URL https://cloudgpu.app/pricing. Accessed September 22, 2026.

DeepSeek-AI. DeepSeek-V4: Entering the era of affordable million-token context. Official model announcement, 2026. URL https://deepseek.com/en/news/v4-preview/. Accessed September 25, 2026.

Yangruibo Ding, Jinjun Peng, Marcus J. Min, Gail Kaiser, Junfeng Yang, and Baishakhi Ray. SemCoder: Training code language models with comprehensive semantics reasoning. In *Advances in Neural Information Processing Systems*, 2024.

Shihan Dou, Yan Liu, Haoxiang Jia, Enyu Zhou, Limao Xiong, Junjie Shan, Caishuang Huang, Xiao Wang, Xiaoran Fan, Zhiheng Xi, Yuhao Zhou, Tao Ji, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. StepCoder: Improving code generation with reinforcement learning from compiler feedback. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4571–4585, 2024. URL https://aclanthology.org/2024.acl-long.251/.

Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Taco Cohen, and Gabriel Synnaeve. RLEF: Grounding code LLMs in execution feedback with reinforcement learning. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

Daya Guo, Qihao Zhu, Dejian Yang, et al. DeepSeek-Coder: When the large language model meets programming—the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024. URL https://arxiv.org/abs/2401.14196.

Jiaxin Guo, Hao Sun, Wenhao Zhang, Chunyu Yang, and Yan Zhang. Failures are treasures: Constructing a pedagogical bridge for agentic strategy distillation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 18808–18823, 2026. URL https://aclanthology.org/2026.findings-acl.938/.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with APPS. In *Advances in Neural Information Processing Systems Datasets and Benchmarks*, 2021.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Binyuan Hui, Jian Yang, Zeyu Cui, et al. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*, 2024. URL https://arxiv.org/abs/2409.12186.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida I. Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/94074dd5a072d28ff75a76dabed43767-Abstract-Conference.html.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, JD Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M Zhang, Kay McKinney, Daya Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2409.12917.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-Tau Yih, Daniel Fried, Sida Wang, and Tao Yu. DS-1000: A natural and reliable benchmark for data science code generation. In *Proceedings of the 40th International Conference on Machine Learning*, 2023. URL https://proceedings.mlr.press/v202/lai23b.html.

Niklas Lauffer. Training robust multi-turn LM agents with on-policy expert corrections. Scale Labs research blog, January 2026. URL https://labs.scale.com/blog/training-robust-multi-turn-lm-agents-with-on-policy-expert-corrections.

Mukai Li, Qingcheng Zeng, Tianqing Fang, Zhenwen Liang, Linfeng Song, Qi Liu, Haitao Mi, and Dong Yu. Verified critical step optimization for LLM agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026a. URL https://aclanthology.org/2026.findings-acl.1974/.

Yu Li, Sizhe Tang, and Tian Lan. Reason in chains, learn in trees: Self-rectification and grafting for multi-turn agent policy optimization. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026b. URL https://aclanthology.org/2026.findings-acl.229/.

Tianyang Liu, Canwen Xu, and Julian McAuley. RepoBench: Benchmarking repository-level code auto-completion systems. In *International Conference on Learning Representations*, 2024a. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/d191ba4c8923ed8fd8935b7c98658b5f-Abstract-Conference.html.

Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. What makes good data for alignment? a comprehensive study of automatic data selection in instruction tuning. In *International Conference on Learning Representations*, 2024b. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/6091f2bb355e960600f62566ac0e2862-Abstract-Conference.html.

Yibing Liu, Chong Zhang, Zhongyi Han, Hansong Liu, Yong Wang, Yang Yu, Xiaoyan Wang, and Yilong Yin. TrajAD: Trajectory anomaly detection for trustworthy LLM agents. *arXiv preprint arXiv:2602.06443*, 2026. URL https://arxiv.org/abs/2602.06443.

Niklas Muennighoff, Alexander Rush, Boaz Barak, Teven Le Scao, Nouamane Tazi, Aleksandra Piktus, Sampo Pyysalo, Thomas Wolf, and Colin Raffel. Scaling data-constrained language models. In *Advances in Neural Information Processing Systems*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/9d89448b63ce1e2e8dc7af72c984c196-Abstract-Conference.html.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 47717–47737, 2025. URL https://proceedings.mlr.press/v267/pan25g.html.

ricdomolm. nanoswe SWE-agent trajectories (v0). Hugging Face dataset, 2026. URL https://huggingface.co/datasets/ricdomolm/nanoswe-trajs-v0. Revision 0b2d3676341314861268b1dba3d49537cae9168b.

Ethan Shen, Daniel Tormoen, Saurabh Shah, Ali Farhadi, and Tim Dettmers. SERA: Soft-verified efficient repository agents. *arXiv preprint arXiv:2601.20789*, 2026. URL https://arxiv.org/abs/2601.20789.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7584–7600, 2024. URL https://aclanthology.org/2024.acl-long.409/.

SWE-bench. SWE-smith trajectories. Hugging Face dataset, 2025. URL https://huggingface.co/datasets/SWE-bench/SWE-smith-trajectories. Revision 08e109b4a59eaeebf80e4675cd125d42e7ac99a4.

Hanlin Wang, Jian Wang, Chak Tou Leong, and Wenjie Li. STeCa: Step-level trajectory calibration for LLM agent learning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 11597–11614, 2025a. URL https://aclanthology.org/2025.findings-acl.604/.

Renxi Wang, Xudong Han, Yixuan Zhang, Timothy Baldwin, and Haonan Li. NAT: Enhancing agent tuning with negative samples. 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)*, 2025b. URL https://aclanthology.org/2025.naacl-long.378/.

Xingyao Wang et al. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations*, 2025c. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a4b6ad6b48850c0c331d1259fc66a69c-Abstract-Conference.html.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pp. 13484–13508, 2023. URL https://aclanthology.org/2023.acl-long.754/.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with OSS-instruct. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 52632–52657, 2024. URL https://proceedings.mlr.press/v235/wei24h.html.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. SWE-RL: Advancing LLM reasoning via reinforcement learning on open software evolution. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2502.18449.

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), 2025a. URL https://conf.researchr.org/details/fse-2025/fse-2025-research-papers/85/Demystifying-LLM-based-Software-Engineering-Agents.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. LESS: Selecting influential data for targeted instruction tuning. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 54104–54132, 2024. URL https://proceedings.mlr.press/v235/xia24c.html.

Yu Xia, Yiran Shen, Junda Wu, Tong Yu, Sungchul Kim, Ryan A. Rossi, Lina Yao, and Julian McAuley. SAND: Boosting LLM agents with self-taught action deliberation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025b. URL https://aclanthology.org/2025.emnlp-main.152/.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. AgentTrek: Agent trajectory synthesis via guiding replay with web tutorials. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/c681fb2bf1d785fbc766f3ea14758aab-Abstract-Conference.html.

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*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/8b86cf5ace600c48fd188efbb8dedec8-Abstract-Datasets_and_Benchmarks_Track.html.

Z.ai. GLM-5.3 model documentation. Online model documentation, 2026. URL https://autoclaw.z.ai/models/. Accessed September 25, 2026.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, et al. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2406.15877.

A Additional Results and Reproducibility

Table 3: Evaluation populations and experimental units.

Experiment	Size	Unit
RQ1 main / model comparison	1,615 / 847	Evidence record
RQ2 planner test	6,107	Candidate pair
RQ2 execution cohort	2,649	Failed trajectory
RQ2 strictly accepted	483	Executed endpoint
RQ3 train / test	16,000 / 1,769	Context–action record
RQ3 matched training runs	3	Seed

A.1 Qualitative Case Studies

RQ1: distinguishing resolved debugging from failure. In a held-out NanoSWE trajectory for SunPy, the untuned 3B model labeled a completed run as failed and marked six spans as suspicious. They comprised two multi-step edit-and-correction intervals in the same function and four validation actions: a task-specific script, two repository `pytest` commands, and an inline smoke test. The trajectory did contain an intermediate indentation error and a mistargeted test path, but subsequent actions corrected the edit, recovered the three expected files, and passed 16 repository tests. The SFT checker returned the gold completed-control label with no suspect span. This contrast suggests that SFT teaches the checker to distinguish unresolved repetition from iterative debugging in which later evidence resolves intermediate failures, a distinction absent from the untuned model.

RQ2: semantic compatibility beyond path overlap. In a Bottle HTTP-date task, the failed trajectory treated a `datetime` as a plain `date`, which discarded its time component. Retrieval selected a same-repository span because it followed a similar edit-and-test pattern and also modified `bottle.py`. However, that span repaired `FormsDict` Unicode decoding and repeatedly rewrote its `_fix` method, which is unrelated to HTTP-date conversion. The untuned 3B model likewise selected this incompatible span and rejected the compatible donor. After SFT, the planner reversed both decisions. It selected a donor whose tests distinguish naive and timezone-aware datetimes, dates, and timestamps, and whose production edit excludes `datetime` from the plain-date branch. When replayed from the failed repository state, the planner-selected repair changed the endpoint from build failure to build pass and achieved full reference core and path coverage. The retrieval-selected repair remained build-failing and covered only 3.1% of the reference core. This contrast shows that the planner learned task- and state-level compatibility beyond repository and path overlap.

RQ3: recipient-conditioned action learning. In a held-out `gods` TreeSet task, failure evidence shows that `Empty()` remains true after an element is inserted, while the implementation returns `set.tree.Size() > 0`. The target action changes this condition to `set.tree.Size() == 0`. All three ReComp checkpoints generate the exact repository-relative correction. Two Donor checkpoints instead propose the reverse substitution, and the third targets an absent `if` form. All three Donor commands also retain an unresolved `${WORKSPACE}` path and fail replay with return code 2, whereas the ReComp commands execute cleanly in every seed. This case illustrates how training on the failed recipient context teaches both the corrective direction and an executable action form tied to the state in which it is needed.

A.2 Scope and Limitations

Donor-library dependence. ReComp avoids fresh action generation by transferring spans from solved trajectories, which makes its repair coverage depend on the contents of the donor library. Retrieval and compatibility planning can reject unsuitable candidates, but they cannot construct a correction when no compatible donor action has been recorded. Abstention limits unsafe transfer at the cost of leaving such failures unrepaired. A hybrid system could preserve reuse for matched cases and invoke generation only when retrieval and planning find no compatible span.

Local repair scope. The current recomposition operator inserts or replaces one localized action span and then verifies the resulting repository state. This design is well suited to failures with a transferable local correction, but multiple interacting errors or repairs that change the preconditions of later actions may require coordinated edits across the trajectory. Execution verification rejects incompatible compositions but does not itself produce a revised plan. Iterative localization, multi-span recomposition, or selective continuation generation could extend the method to these cases.

A.3 Checker Comparisons

Table 4: Completed SFT checker results on the main test. Full-sampled F1 uses a separate 847-record split. Scores are percentages. Each row is one checkpoint.

Model	Precision	Recall	F1	False alarms	Full F1
Qwen2.5-Coder-3B	92.18	83.76	87.77	42	83.55
Qwen2.5-Coder-7B	98.15	89.68	93.72	10	90.82
Qwen2.5-Coder-14B	86.96	89.17	88.05	79	83.84
Qwen2.5-Coder-32B	83.44	89.51	86.37	105	83.68
DeepSeek-Coder-1.3B	77.36	48.56	59.67	84	55.83
DeepSeek-Coder-6.7B	72.01	75.30	73.61	173	72.73

A.4 Planner Diagnostics

Main-test denominators. The 6,107 decisions contain 3,067 true SELECTs, 62 false SELECTs, 2,949 correct abstentions, 14 positive abstentions, and 15 invalid outputs. Thirteen invalid outputs belong to positives. Fixed-denominator recall is therefore $3067/3094 = 99.13\%$. Exact failed and donor spans are recovered for $3050/3094 = 98.58\%$ of positives.

Candidate-noise protocol. Each failed-recipient query q has an original pool $C_0(q)$, comprising 6,107 recipient–donor pairs across 2,649 queries. We progressively add $D \in \{0, 2, 5, 10, 20\}$ unmarked distractors sampled from 91,609 spans and incompatible with the recipient by construction. All methods receive the same nested pools. A compatible hit requires selecting an original compatible span. Abstentions and distractor selections count as failures. Distractors are added to the 2,645 queries represented in the candidate-pair table. The remaining four queries have no retained pair, remain unchanged, and still count as failures in the fixed 2,649-query denominator.

Table 5: Candidate-pool stress test: compatible hits over all 2,649 assigned queries (%). D incompatible spans are added to each represented query. These are selection outcomes, not execution results.

D	Added share (%)	Path	Retrieval	Planner + retrieval
0	0.00	81.54	88.79	97.28
2	46.42	76.82	85.01	97.28
5	68.41	71.54	80.29	97.28
10	81.24	66.52	75.73	97.17
20	89.65	60.97	69.95	96.87

A.5 Execution Acceptance and Scope

The manual endpoint audit found build-passing outputs with empty, auxiliary-only, or task-irrelevant changes. This motivated the strict endpoint criterion used in Table 1: the final repository must pass both the build check and its task-specific executable functional test. The canonical comparison uses duplicate-safe case joins and retains all 2,649 assigned failures in its denominator.

Functional endpoint verification. Every evaluated endpoint is covered by an executable task-specific functional test. We use dataset-provided tests when available. Otherwise, we construct

tests with LLM assistance and manually review them against the task requirement. Before endpoint evaluation, each constructed test must fail on the original failed repository state and pass on a known-correct state. The validated test is then frozen and applied unchanged to outputs from every compared method. This fail-to-pass protocol evaluates functional behavior without requiring textual agreement with the reference patch, thereby permitting alternative implementations.

Strict acceptance is an endpoint-level decision. Intermediate command outcomes remain in the execution trace, while acceptance depends on the final build and verification evidence under the same policy for every compared method.

Paired uncertainty and pipeline outcomes. Planner + retrieval and Retrieval top-1 are evaluated on the same 2,649 case identifiers. Their strict outcomes contain 449 joint accepts, 34 Planner-only accepts, six Retrieval-only accepts, and 2,160 joint rejections. The paired difference is 1.06 percentage points, with a case-level paired bootstrap 95% interval of [0.60, 1.51] points and a two-sided exact McNemar $p = 8.36 \times 10^{-6}$. This inference treats cases as paired units and does not additionally cluster them by repository.

Table 6: Mutually exclusive outcomes for Planner + retrieval on the fixed 2,649-case RQ2 cohort.

Outcome	Count (%)
Abstention or invalid span	64 (2.42)
Execution state unavailable or preparation failed	441 (16.65)
Failed-recipient replay incomplete or state-mismatched	796 (30.05)
Donor execution not verified	6 (0.23)
Build failed	481 (18.16)
Functional verification failed after build	378 (14.27)
Strictly accepted	483 (18.23)

The failed recipient is reproducible only when its calibration replay completes and the normalized production state agrees with the recorded endpoint. The dominant remaining losses occur after executing a plan at the build or functional-verification gates. The stored outcomes do not support a causal separation of boundary errors from retained-suffix incompatibility.

Before donor selection, one solved peer per failed target is frozen as the reference implementation. The Full-donor control may replay a different solved peer, so its accepted output can differ from the reference without creating a new strategy during replay.

A.6 Implemented Build Contract

The executor dispatches the first supported repository-level check:

1. For Python repositories, run `python -m compileall`, excluding test, documentation, example, and benchmark directories.
2. For Go repositories, run `go test ./... -run '^$' -count=1`, which compiles packages without running the test suite.
3. For npm repositories, run `npm run build --if-present`.
4. Otherwise return unsupported.

These build checks provide the intermediate screening signal reported in RQ2. Strict acceptance separately requires passage of the functional endpoint verification defined above.

A.7 Trajectory Data Construction and Integrity

Source datasets and shared training configuration. NanoSWE contains 1,582,701 content-deduplicated coding-agent rollouts with conversations, patches, and verification metadata (ricdomolm, 2026). SWE-smith contains more than 50,000 executable software-engineering tasks from 128 Python repositories, and its public trajectory release provides a 24,100-rollout tool-using split with resolution metadata (Yang et al., 2025; SWE-bench, 2025). Except for the RQ1 model-scale

comparison, all learned models initialize from Qwen2.5-Coder-3B-Instruct. Fine-tuning uses rank-16 LoRA adapters with $\alpha = 32$, dropout 0.05, and BF16 computation. The planner trains its adapter over a 4-bit QLoRA base (Hu et al., 2022). Training uses completion-only loss, a per-device batch size of one, eight gradient-accumulation steps, a learning rate of 2×10^{-4}, and the trainer's AdamW optimizer with linear decay and no warmup. The checker uses 17,792 training and 1,586 development records for 500 updates, while the planner uses 111,762 training and 6,087 development records for 2,500 updates. Both use a 12,288-token limit and their fixed final checkpoints, so no test result is used for checkpoint selection. Checker and planner predictions use deterministic decoding with temperature zero. RQ3 instead trains for one epoch with seeds 20260917, 20260918, and 20260919.

Source processing. Source traces include NanoSWE and SWE-smith-derived records. We normalize heterogeneous traces into ordered action records containing action index, tool, affected path, semantic role, observable effect, and a clipped command representation. Repository states, observations, and recorded deltas are retained for alignment and execution checks.

Checker and planner supervision. Checker records pair local action windows with action-level anomaly labels. The main test contains 591 positive and 1,024 negative records, and the model-comparison split contains 847 records with 335 positives. Planner positives pair failed and solved trajectories for the same task at their first mutation divergence after an identical ordered mutation prefix. Both pre-divergence repository states must be recoverable, and the positive donor span is obtained by aligning mutation signatures, affected paths, tool roles, and observable effects. Hard negatives retain the recipient window but sample a span from a different task in the same repository. They are construction-labeled incompatible, preventing repository or path identity alone from determining the label. Task-hashed partitions are assigned before pairing, and the planner test contains 3,094 positive and 3,013 negative pairs.

Candidate-origin labels, verification-policy fields, trajectory identifiers, source-state equality, and shared-prefix length are removed from model inputs. Supplied action windows therefore test conditional compatibility independently of end-to-end retrieval.

Downstream corpus. The corpus constructor starts from donor trajectories marked `verified` in NanoSWE or `resolved` in SWE-smith and failed recipients that share the same pre-action source state. Operationally, this requires an identical ordered prefix of source-modifying actions and no ambiguous pre-pivot command effect. Recipient and donor contexts can still differ in their non-mutating interaction histories and observations leading to this shared state. The source labels provide offline admission. We do not natively replay every transferred training relation. We retain one relation per task in deterministic hash order. After donor joining, 16,910 relations are available and the main experiment uses 16,000. Evaluation contains 1,769 records from 80 repositories that do not occur in training, and held-out endpoint replay separately tests whether source-verified actions remain useful after transfer. Donor and ReComp use identical target actions relation by relation.

Training and metric controls. The final matched preprocessing retains all 16,000 records and exactly 3,641,879 target tokens in each condition, with no partial targets. Donor and ReComp retain 89,036,854 and 76,423,981 input tokens, respectively, so the comparison matches target supervision while allowing total FLOPs to differ. Both conditions use completion-only loss, one epoch, no packing, and an 8,192-token limit. The shuffled-context ablation keeps the ReComp target actions and corpus size fixed but permutes recipient contexts across records, breaking their state–action alignment.

For N evaluation records and target length L_i, mean action negative log-likelihood is

$$-\frac{1}{N}\sum_{i=1}^{N}\frac{1}{L_i}\sum_{t=1}^{L_i}\log p_\theta(y_{it} \mid c_i, y_{i,<t}). \tag{8}$$

We average per-record token-normalized losses over all 1,769 held-out records. Verified endpoint replay uses the 769 boundaries at which the reference action executes successfully and its endpoint passes the build and frozen task-specific executable functional test.

Table 7: Per-seed action NLL on the same 1,769 records. Shuffled context preserves ReComp's target actions and corpus size while breaking recipient-state alignment. Reduction is relative to Donor, and means are computed from unrounded values.

Training seed	Donor	Shuffled	ReComp	Reduction (%)
20260917	0.3519	0.3802	0.2748	21.89
20260918	0.3514	0.3816	0.2748	21.79
20260919	0.3550	0.3801	0.2752	22.46
Mean	0.3527	0.3806	0.2750	22.05

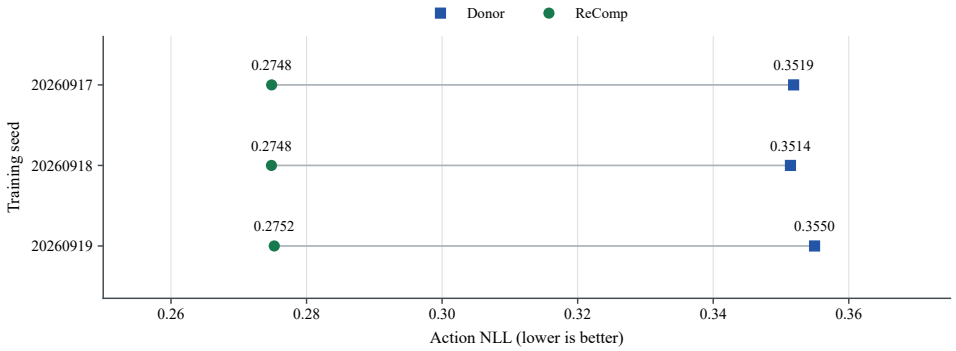

Figure 5: Target-supervision-matched action modeling. Each paired set of points is a training seed from Table 7, evaluated on the same 1,769 records.

A.8 Matched-Checkpoint Endpoint Utility

We replay generations from the same three Donor and ReComp checkpoints used for the matched NLL comparison. Endpoint eligibility is fixed from the reference action before checkpoint outputs are compared. A generated action succeeds only if it executes successfully, reaches a build-passing endpoint, and passes the frozen task-specific executable functional test. Parsing, execution, build, and functional-test failures count as unsuccessful within the 769-case denominator. We compute paired bootstrap intervals over fixed boundary identifiers and report three-seed means and standard deviations in the main comparison.

Table 8: Per-seed verified endpoint success on 769 reference-eligible boundaries (%).

Seed	Donor	ReComp
20260917	1.17	2.08
20260918	1.56	2.08
20260919	1.43	2.21
Mean $\pm$ SD	1.39 ± 0.20	2.12 ± 0.08

A.9 Acquisition Resource Accounting

The 20-task rollout cohort is the repair-eligible subset of a frozen 30-task source manifest. Eligibility requires a parseable historical failed recipient with at least one structured mutation action and a solved donor from the same repository. These criteria are applied before running ReComp, and one recipient per eligible task is selected by a fixed hash rule. The remaining ten source tasks lack a usable failed recipient under this construction. They are not excluded according to repair or rollout outcomes.

For rollout agents, we parse each agent's native JSONL or database session records to recover the executed action sequence and stored USD cost. The stored cost reflects the agent harness's provider-

specific accounting for input, output, cache-creation, and cache-read tokens when available, plus any tool charge already recorded by the harness. We do not independently re-price tokens or tool calls. We aggregate recorded cost and action counts across all attempts, so Cost/action is total cost divided by total executed actions and Cost/accepted is total cost divided by accepted outputs.

Table 9: Measured acquisition accounting used in Figure 4. Costs include unsuccessful attempts. GLM, DeepSeek, and Qwen each use three runs over 20 tasks, while Claude uses one run.

Method	Trials (accepted)	Actions/attempt	Cost/action	Total cost	Cost/accepted
ReComp 3B	2,649 (95)	4.74	$0.00060	$7.561	$0.080
GLM-5.3	60 (17)	80.73	$0.00108	$5.239	$0.308
DeepSeek-V4	60 (16)	95.73	$0.00529	$30.409	$1.901
Qwen3.8-Max	60 (18)	80.95	$0.02380	$115.616	$6.423
Claude Opus 4.7	20 (6)	62.30	$0.08054	$100.354	$16.726

Figure 4 compares the rollout cohort with ReComp's measured resource profile on 2,649 failed trajectories from the same source project pools. Both count an output as accepted only when its endpoint passes the build and executable functional test. We normalize measured H800 allocations at $2 per GPU-hour, within the range of public on-demand listings (CloudGPU, 2026; AutoDL, 2026), and charge CPU allocation at $0.03 per core-hour. Cost per accepted output charges failed and rejected attempts. The accounting excludes downstream training. Collection of the source trajectory library precedes the incremental acquisition stage.

Table 10: CPU allocation by stage. Core-hours equal allocated CPU cores times job wall time.

Stage	Job	CPU cores	Wall (s)	Core-hours
Preparation	135968	2	20	0.0111
Execution/build	135969	96	8,102	216.0533
Runtime retry	136008	48	324	4.3200
Endpoint verification	136742	8	129	0.2867
Total				220.6711

On the 2,649-target accounting cohort, checker and planner charges are $0.7344 and $0.2061 at $2/GPU-hour. At CPU rate r, the conservatively charged cohort cost is

$$C(r) = 0.940524 + 220.6711r.$$

At the $0.03 per CPU-core-hour rate used in Figure 4, the measured cohort costs $7.56. Unlike the candidate-fed RQ2 comparison, this accounting run includes checker gating before planner execution. Its 95 strictly accepted outputs therefore give $0.080 per accepted trajectory. This 95-output denominator is used only to normalize the checker-gated resource accounting. This calculation charges all 2,649 attempts, including rejected repairs.

Artifact provenance. The canonical RQ2 execution comparison uses corrected candidate-fed run 137669, direct-donor controls from run 137346, and random-span run 138299. Candidate-noise stress results use runs 137528–137530. Matched RQ3 NLL uses run 137670 and run 137816, and matched execution replay uses the corresponding three saved adapters. Figures and tables are generated from stored row-level summaries using fixed case identifiers.

B Additional Related Work

Executable code evaluation. SWE-bench evaluates repository-level issue resolution, while RepoBench studies repository-level completion (Jimenez et al., 2024; Liu et al., 2024a). APPS, DS-1000, BigCodeBench, and LiveCodeBench broaden functional evaluation across programming problems, data-science code, complex instructions, and contamination-aware settings (Hendrycks et al., 2021; Lai et al., 2023; Zhuo et al., 2025; Jain et al., 2025). These benchmarks establish the importance of executable evaluation, but they focus on final programs and do not address the production of repaired trajectory supervision. ReComp instead evaluates whether an action-level trajectory intervention remains executable and yields valid supervision after repository-state verification.

Code repair and search. Agentless decomposes fault localization and patch generation, SWE-Search searches over candidate solutions, and DARS resamples coding-agent branch points (Xia et al., 2025a; Antoniades et al., 2025; Aggarwal et al., 2025). StepCoder and RLEF optimize code generation with compiler or execution feedback, while SCoRe and self-debugging methods study learned correction across repeated attempts (Dou et al., 2024; Gehring et al., 2025; Kumar et al., 2025; Chen et al., 2025a). These methods primarily seek better task solutions by generating code patches or exploring new execution branches. ReComp instead targets the trajectory data itself by repairing localized action spans in recorded failed runs with compatible donor actions and validating the recomposed states as additional training supervision.