ECDYSIS: RETHINKING FAILURE-DRIVEN EVOLUTION OF LLM AGENT HARNESSES

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-evolving runtime harnesses can substantially improve the capabilities of large language model (LLM) agents and provide a promising paradigm for optimizing agent execution. Existing failure-driven approaches often treat observed agent failures as direct evidence for harness modification. A key challenge in failure-driven harness evolution is that observed failures can reflect either limitations of the underlying model or systematic deficiencies of the harness. Directly optimizing against individual failures can therefore induce model-specific accommodation and impair generalization across tasks and models. We study whether failure evidence accumulated across task instances can provide a more reliable signal for harness training. Our key insight is that failures recurring across distinct tasks provide stronger inductive evidence for systematic harness deficiencies than isolated failures. Based on this insight, we propose ECDYSIS, which aggregates failure evidence across task instances before promoting recurring failure patterns into persistent harness evolution, biasing evolution toward repairs that are more likely to generalize beyond individual model behaviors. ECDYSIS further employs collaborative failure analysis to refine modification specifications, trading additional evolution-time reasoning for improved modification quality. Across multiple LLMs and benchmarks, ECDYSIS improves the reasoning accuracy of evolved harnesses by 18.56% over existing harness evolution while achieving up to 1.84× faster harness training. ECDYSIS also enables more data-efficient training. Fine-grained analysis shows that ECDYSIS reduces model-specific accommodation during evolution, while the resulting harnesses exhibit stronger cross-LLM generalization and lower inference-time token consumption. These results suggest that effective harness training depends not only on how failures are repaired, but also on which failures provide reliable evidence for persistent harness changes.

1 INTRODUCTION

Large language models (LLMs) have demonstrated increasingly strong capabilities in complex reasoning, yet practical LLM agents often rely critically on runtime harnesses that govern task planning, tool interaction, and context management (Chen et al., 2025b; Jiang et al., 2026a; Yao et al., 2023; Wan et al., 2026). Recent harness self-evolution methods (Lee et al., 2026; Chen et al., 2026a; Wang et al., 2026b; Shao et al., 2026b) enable these runtime mechanisms to improve automatically from execution feedback, without modifying the underlying model. By iteratively identifying failures and revising the harness, this paradigm provides a promising approach to optimizing agent behavior. However, failure-driven evolution also introduces a fundamental challenge: execution failures must be correctly interpreted before being translated into persistent harness modifications. Otherwise, repeated adaptation to observed failures may lead the harness to become specialized to particular tasks or model behaviors, increasing evolution cost and potentially impairing generalization (Shao et al., 2026a; Jiang et al., 2026b; Wang et al., 2026c).

More fundamentally, *an observed agent failure does not necessarily indicate a harness deficiency*. A failure may instead reflect a limitation of the underlying model. Modifying the harness to accommodate such behavior may improve performance on the current model while unnecessarily coupling the harness to it, potentially harming generalization to unseen tasks and other LLMs. We distinguish between *model-specific accommodation* (MSA), which compensates for model-level limitations, and *harness-level repair* (HLR), which addresses deficiencies in the runtime mechanism (Zhang et al.,

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.

2026c). Individual failures provide limited evidence for distinguishing between these two cases. In contrast, failures that recur across distinct task instances provide stronger inductive evidence that the underlying mechanism, rather than an isolated model behavior, may be responsible. This observation suggests that persistent harness evolution should be driven not simply by whether a failure occurs, but by whether the failure constitutes sufficiently general evidence for a harness-level deficiency.

This motivates our research question: **Can cross-task failure evidence guide harness training toward systematic repair while reducing unnecessary MSA?** We propose ECDYSIS, a framework that shifts failure-driven harness training from instance-level correction toward cross-task evidence aggregation. ECDYSIS jointly analyzes failures across a batch of task instances, identifies recurring failure patterns, and uses them to prioritize candidate harness modifications. ECDYSIS biases training toward changes supported by repeated failure mechanisms rather than isolated model behaviors. It further employs collaborative failure (Yu et al., 2025; Chen et al., 2025a; Zhang et al., 2025b) analysis to refine modification specifications. This refinement is complementary to the central aggregation mechanism: ECDYSIS first determines *which failure evidence should drive evolution*, and then *improves how that evidence is translated into harness modifications*.

We systematically evaluate ECDYSIS across multiple LLMs and benchmarks. Experimental results show that ECDYSIS achieves up to a 1.84× speedup in harness training compared with existing harness evolution methods while improving the reasoning accuracy of the resulting harnesses by 18.56%. Further experiments demonstrate that harnesses trained with ECDYSIS exhibit stronger cross-LLM generalization, while also reducing inference-time token consumption. The cross-task perspective also enables failure-aware training data curation. This indicates that the informativeness and recurrence structure of failure evidence can matter more than the amount of failure data alone. Notably, the ablation study reveals a clear trade-off between efficient evidence aggregation and reliable failure attribution. With collaborative refinement, MSA decreases from 75.61% to 45.45%, leading to stronger cross-LLM generalization of the evolved harnesses, at the cost of additional evolution-time computation. These results indicate that recurrence provides an efficient mechanism for selecting failure evidence, but recurrence alone is insufficient for reliably distinguishing recurring model limitations from harness deficiencies. Explicit failure attribution is therefore complementary to cross-task aggregation. We have the following contributions:

- **Cross-Task Failure Recurrence for Efficient Evidence Selection**. We propose cross-task failure recurrence as an efficient mechanism for selecting failure evidence that warrants persistent harness evolution. By aggregating failures across task instances, ECDYSIS substantially reduces the evolution cost and training time required to identify recurring failure patterns. However, recurrence alone may over-amplify recurring model-specific failures, revealing the need for explicit failure attribution before translating recurring evidence into persistent harness modifications.
- **Collaborative Refinement for Failure Attribution**. We introduce a collaborative refinement that analyzes recurring failure evidence before implementation, providing an explicit stage for distinguishing how failures should be interpreted and translated into harness modifications. Our experiments show that this additional reasoning incurs computational overhead but improves final harness accuracy and cross-LLM generalization while substantially reducing MSA.
- **Failure-Aware Data Curation**. We show that recurrence-aware failure analysis can guide training-data selection, allowing effective harness evolution with substantially fewer instances.

2 RELATED WORK

Inference-time mechanisms such as tool interaction, deliberate planning, and feedback-driven adaptation have substantially enhanced LLM agents (Yao et al., 2023; Shinn et al., 2023). Building on these advances, automatic optimization has progressed from prompt optimization (Khattab et al., 2024; Yang et al., 2024) to agentic workflows and self-modifying agent programs (Hu et al., 2025; Zhang et al., 2025a; 2026b). More recently, several works have focused specifically on evolving runtime harnesses, using execution feedback and iterative evaluation to propose and validate harness modifications (Zhang et al., 2026a; Lee et al., 2026; Chen et al., 2026b; Wang et al., 2026a; Huang et al., 2026; Xu et al., 2026b). Although harnesses are ultimately optimized to improve LLM task performance, harness evolution should not simply adapt to every observed failure, as failures may reflect model limitations rather than harness deficiencies, leading to MSA. ECDYSIS therefore treats recurrent failure patterns across independent task instances as stronger evidence of system-

atic harness deficiencies, guiding evolution toward reusable behavioral improvements rather than instance-specific adaptation. Self-Harness (Zhang et al., 2026a) treats model-specific weaknesses as actionable targets for harness adaptation, whereas ECDYSIS seeks to distinguish such accommodation from persistent harness deficiencies. ECDYSIS biases the evolution process toward systematic harness deficiencies by aggregating failure evidence across distinct task instances and retaining recurring failure patterns as actionable signals. Consequently, MSA is mitigated rather than prohibited: ECDYSIS reduces the influence of idiosyncratic model-specific failures on persistent harness evolution while preserving potentially useful adaptations when supported by recurring evidence.

3 ECDYSIS

3.1 PRELIMINARY ANALYSIS

A fundamental challenge lies in failure attribution: an observed failure may arise from the current model's behavior or from a systematic harness deficiency. The former leads to *Model-Specific Accommodation* (MSA), whereas the latter calls for *Harness-Level Repair* (HLR). Individual failures provide limited evidence for distinguishing these cases, while treating failures from a common harness issue as independent modification signals can also incur redundant modification and validation costs. This distinction motivates a conceptual decomposition of harness adaptation. Consider the aggregate harness update ΔH produced over an evolution round, composed of a set of independent modification decisions issued by the coding agent. Each decision either accommodates a limitation of the current task model or repairs a systematic harness deficiency, contributing to ΔH_{MSA} and ΔH_{HLR}, respectively. Letting $t \in [0, 1]$ denote the proportion of these decisions that are MSA, the aggregate update decomposes as

$$\Delta H = t\,\Delta H_{\mathrm{MSA}} + (1 - t)\,\Delta H_{\mathrm{HLR}}, \qquad t \in [0, 1],$$

so that t is at once the fraction of model-accommodating decisions and the weight of the model-accommodation component in the overall update. A larger t may resolve observed failures efficiently but can overfit to the current model and training tasks. We hypothesize that failure patterns recurring across distinct task instances provide stronger evidence for systematic harness deficiencies than isolated failures, and therefore can shift adaptation away from unnecessary MSA. Our goal is not to develop a sophisticated evolution algorithm, but to use a simple and interpretable mechanism to validate our analysis that recurring failures provide a useful signal for harness-level evolution.

3.2 METHODOLOGY

Given a task model θ, a runtime environment $\mathcal{E}$, a training task set $\mathcal{D}_{\mathrm{train}}$, and an initial runtime harness H_{base}, ECDYSIS optimizes the harness using execution trajectories collected from the training tasks. Throughout training, the task model parameters and runtime environment remain fixed. We initialize the harness as $H_0 = H_{\mathrm{base}}$ and denote by H_i the validated harness retained after evolution round i. For each evolution round $i \in \{1, 2, \ldots, R\}$, ECDYSIS proceeds in three stages. First, it executes the training tasks with the current harness H_{i-1} and aggregates the resulting failure evidence. Second, it transforms the aggregated evidence into a structured modification specification, which is provided to a coding agent to modify H_{i-1} and produce a candidate harness H_i^c. Third, it validates H_i^c on the training set and retains it only if it improves the overall training score; otherwise, the previous harness is preserved. After R rounds, training terminates and ECDYSIS freezes the most recently validated harness, denoted by $H_F = H_R$. The complete procedure is summarized in Algorithm 1. Let $J_{\mathrm{train}}(H)$ denote the overall score of harness H on the training set under the fixed evaluation framework. In evolution round i, ECDYSIS accepts the candidate harness if and only if $J_{\mathrm{train}}(H_i^c) > J_{\mathrm{train}}(H_{i-1})$. Accordingly,

$$H_i = \begin{cases} H_i^c, & \text{if } J_{\mathrm{train}}(H_i^c) > J_{\mathrm{train}}(H_{i-1}), \\ H_{i-1}, & \text{otherwise.} \end{cases}$$

This criterion constrains only the overall training-set score and does not require every individual training task to improve. We next describe the two core components of ECDYSIS: Batch-Level Failure Aggregation and Failure-Driven Collaborative Refinement (FDCR).

Algorithm 1: ECDYSIS Training for Harness Self-Evolution

Input: Fixed task model θ, runtime environment $\mathcal{E}$, training task set $\mathcal{D}_{\text{train}}$, initial harness H_{base}, failure threshold λ, number of evolution rounds R, and number of refinement passes K

Output: Frozen final harness H_F

$H_0 \leftarrow H_{\text{base}}$;
for $i = 1$ **to** R **do**
  $\mathcal{T}_i \leftarrow \text{Collect}(\theta, H_{i-1}, \mathcal{E}, \mathcal{D}_{\text{train}})$ # execution trajectories;
  $f_\lambda(\tau) \leftarrow \mathbb{I}[S(\tau) < \lambda]$;
  $\mathcal{B}_i \leftarrow \text{Aggregate}(\{\tau \in \mathcal{T}_i : f_\lambda(\tau) = 1\})$ # structured failure evidence;
  if $\mathcal{B}_i = \varnothing$ **then**
    $H_i \leftarrow H_{i-1}$;
    continue;
  $\mathcal{G}_i \leftarrow \text{Group}(\mathcal{B}_i)$ # failure patterns;
  $\mathcal{A} \leftarrow \{\text{Analyst}, \text{Critic}, \text{Engineer}\}$;
  $\mathcal{M}_i \leftarrow \varnothing$ # shared role transcript;
  for $k = 1$ **to** K **do**
    foreach $A \in \mathcal{A}$ **do**
      $m_i^{k,A} \leftarrow A(\mathcal{G}_i, \mathcal{M}_i)$;
      $\mathcal{M}_i \leftarrow \mathcal{M}_i \circ m_i^{k,A}$;
  $q_i \leftarrow \text{Moderator}(\mathcal{G}_i, \mathcal{M}_i)$ # structured modification specification;
  $H_i^c \leftarrow \text{Edit}(H_{i-1}, q_i)$ # coding agent;
  if $J_{\text{train}}(H_i^c) > J_{\text{train}}(H_{i-1})$ **then**
    $H_i \leftarrow H_i^c$;
  else
    $H_i \leftarrow H_{i-1}$;
$H_F \leftarrow H_R$;
return H_F;

3.3 BATCH-LEVEL FAILURE AGGREGATION

At evolution round i, ECDYSIS executes the training tasks using the current harness H_{i-1} and collects the resulting execution trajectories, denoted by $\mathcal{T}_i$. A task specifies the target objective, whereas its execution trajectory records the concrete process leading to the observed outcome. The trajectory therefore provides execution-level evidence for failure analysis beyond the final task-level signal.

For each trajectory $\tau \in \mathcal{T}_i$, the fixed evaluation framework produces a task score $S(\tau)$. We define the binary failure signal as $f_\lambda(\tau) = \mathbb{I}[S(\tau) < \lambda]$, where λ is a predefined threshold and $f_\lambda(\tau) = 1$ indicates failure. Trajectories satisfying this condition are converted into structured records to form the failure evidence set $\mathcal{B}_i$. Each record retains the task identifier, failure decision, termination reason, tool-call history, and necessary execution context. This procedure uses the fixed evaluation framework as the training signal and does not modify or replace its scoring mechanism. The failure records are organized into failure groups to facilitate the analysis of recurring patterns. ECDYSIS prioritizes groups that cover at least two distinct tasks, since repeated failures from a single task alone do not establish that the underlying failure mechanism generalizes across tasks. Groups containing only one task identifier are nevertheless retained as auxiliary evidence for subsequent analysis. Importantly, failure groups serve as diagnostic evidence rather than mandatory repair targets. They help determine whether an observed pattern indicates a harness deficiency and guide decisions regarding the modification level, trigger conditions, modification scope, and safety constraints. Thus, a candidate harness need not address every observed failure group. Batch-level aggregation also reduces repeated coding-agent calls for candidate harness modification. Let $n_i = |\mathcal{B}_i|$ denote the number of failure records collected in round i. Processing failures independently requires $N_{\text{serial}} = \sum_{i=1}^{r} n_i$ coding-agent calls, whereas ECDYSIS requires only $N_{\text{round}} = \sum_{i=1}^{r} \mathbb{I}(n_i > 0)$ calls. Therefore, $N_{\text{round}} \leq N_{\text{serial}}$, with strict inequality whenever any round contains multiple failure records.

3.4 FAILURE-DRIVEN COLLABORATIVE REFINEMENT

For each evolution round containing failure evidence, ECDYSIS applies FDCR to transform the aggregated failure patterns into a structured harness modification specification. Inspired by collaborative criticism and iterative refinement in multi-agent systems (Yu et al., 2025; Chen et al., 2025a), FDCR uses observed failure patterns as the driving evidence for role-based diagnosis and iterative refinement, rather than treating individual failures as independent modification requests. FDCR separates failure analysis and modification planning from the actual harness implementation. Given the failure groups $\mathcal{G}_i$, the Analyst, Critic, and Engineer iteratively refine modification proposals through a shared transcript. The Analyst identifies potential harness deficiencies and proposes minimal, targeted updates. The Critic evaluates these proposals against the observed failure evidence and examines potential risks, including overly broad triggers, unintended blocking of legitimate behavior, violations of the runtime contract, and regressions on previously successful tasks. The Engineer tracks agreements and unresolved disagreements and identifies issues requiring clarification in subsequent refinement. The roles are executed sequentially, with each role receiving the accumulated transcript so that subsequent analysis can build on preceding discussion. The refinement proceeds for a fixed number of rounds. After the role-based refinement, the Moderator reads the failure evidence and the complete transcript and produces a structured modification specification. The Moderator serves as a conservative arbitration stage that consolidates the refined proposals, resolves remaining disagreements, and prioritizes modifications that address recurring failure patterns while avoiding unnecessary changes. The resulting specification describes the identified failure patterns, proposed harness changes, and relevant implementation guidance, but does not directly modify the harness. Instead, the coding agent uses the specification together with the relevant execution evidence and the source code of the current harness to modify H_{i-1} and produce the candidate harness H_i^c. This separation keeps failure analysis structured and failure-driven while delegating implementation to the coding agent. By prioritizing failure patterns recurring across distinct task instances, FDCR biases the modification process toward systematic HLR and away from unnecessary MSA.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Models. Our evaluation follows prior work (Xu et al., 2026c) and considers the compatibility between LLM reasoning capabilities and benchmark difficulty. We evaluate five task LLMs: Qwen3-8B, Qwen3-14B, Qwen3-32B (Yang et al., 2025), MiniMax-M2.7 (230B), and Llama-3.1-8B. For reproducibility, all LLMs are accessed through APIs. We use the same inference configuration for all task models in both training and held-out evaluation. For evaluation parameters, we refer to the baseline methods. The sampling temperature is set to 0.0. We use OpenCode[1] with DeepSeek-V4-Pro as its underlying model as the coding agent. In ECDYSIS (w/ FDCR), the Analyst, Critic, Engineer, and Moderator also use DeepSeek-V4-Pro (Xu et al., 2026a).

Datasets. To ensure comprehensive coverage of diverse task types, we use AgentBench (Liu et al., 2024) for relatively simple tasks and τ^2-Bench for more complex and challenging tasks. Following prior work, we use the Airline and Retail subsets of τ^2-Bench, which we refer to as τ^2-Airline and τ^2-Retail, respectively. Both subsets require agents to perform tool interactions that continuously modify the environment state (Barres et al., 2026).

Baselines and Ablation Study. For the runtime harness, we adopt *Life-Harness*, a mature and well-structured harness framework (Xu et al., 2026c). To enable fair and controlled comparisons, we construct five runtime harness configurations, covering a direct baseline, a human-optimized baseline, and three harness evolution strategies.

- **Direct**: retains the base agent loop, message handling, and tool interface required by τ^2-Bench, while disabling the harness layers adopted from Life-Harness. This configuration serves as the minimal baseline without any additional harness mechanisms.
- **Human-Augmented Harness (Human-Aug.)**: uses a fixed harness without harness evolution. This harness is obtained through human-involved optimization in prior work and serves as a strong

[1] https://opencode.ai

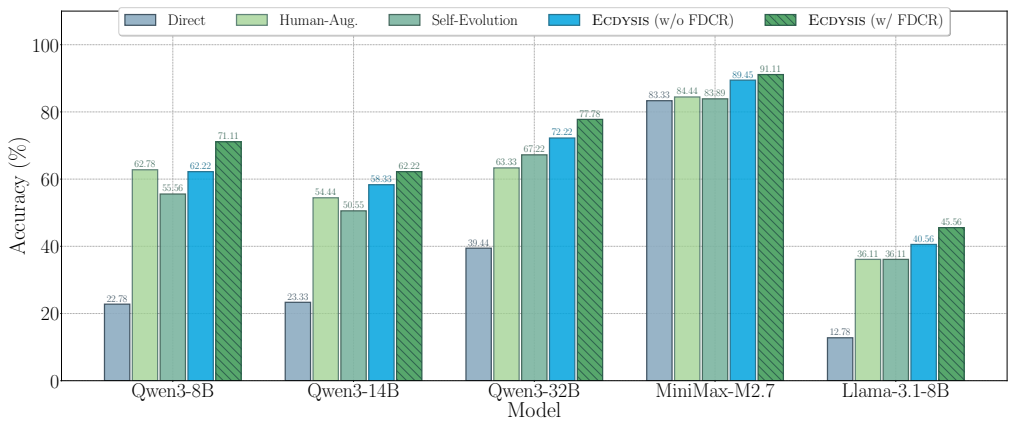


Figure 1: Overall accuracy of five harness self-evolution methods across task models and datasets.

baseline representing the performance of a manually optimized runtime harness. Moreover, the human-optimized harness is used as the common base harness for harness post-training. That is, all three evolution-based methods described below are initialized from the same Human-Aug. harness, thereby keeping the initial harness configuration fixed across methods.

- **Self-Evolution (SE)**: performs instance-level serial updates. Each failure record independently invokes the coding agent, and the modifications produced for individual failure instances are applied sequentially and accumulated into a single round-level candidate. The accumulated candidate is then evaluated in the next complete training evaluation (Xu et al., 2026c).
- **ECDYSIS (w/o FDCR)**: performs mixed training by aggregating failure evidence from multiple training tasks and evolution rounds. For each evolution round with non-empty failure evidence, it constructs a single round-level modification plan from the aggregated evidence and invokes the coding agent once to implement the proposed changes.
- **ECDYSIS (w/ FDCR)**: operates on the same mixed training input as ECDYSIS (w/o FDCR). The Analyst, Critic, and Engineer agents jointly analyze the aggregated failure evidence through two rounds of FDCR. The Moderator then synthesizes the results and formulates the final modification plan, which is subsequently implemented by the coding agent.

Across all methods, only the runtime harness is updated during training, while the model parameters remain fixed. Thus, the comparison isolates the effect of different harness optimization strategies while controlling for the initial harness configuration and the underlying model. Due to the high complexity and cost of agent evaluation, we follow existing work (Qu et al., 2026) in designing the experimental setup and scale. Additional experimental setup details are provided in Appendix C. We select Life-Harness's SE as our representative baseline, as it covers major runtime interventions while retaining the core principles of Self-Harness (Zhang et al., 2026a) and HarnessEvolve (Jiang et al., 2026b), while avoiding additional designs that could introduce confounding factors into our analysis. Moreover, our study isolates harness evolution under a frozen model, unlike approaches such as HarnessForge (Chen et al., 2026b) that jointly evolve the model and harness.

5 RESULTS

Overall Results. Across the fifteen combinations of five task models and three datasets, ECDYSIS (w/ FDCR) achieves the highest average accuracy (see Table 1 and Figure 1). Its relative improvements over SE and Human-Aug. are 18.56% and 15.51%, respectively. It also improves Pass^3 by 28.56% relative to SE, showing more consistent success across repeated trials. Beyond task accuracy, ECDYSIS also improves both training and inference efficiency relative to SE. We analyze the sources of these gains in Section E.

Ablation Results. All three task-performance metrics improve from SE to ECDYSIS (w/o FDCR). The accuracy increases from 58.67% to 64.56%, an improvement of 5.89 percentage points. Meanwhile, Pass@3 and Pass^3 reach 72.67% and 56.00%, improving by 2.67 and 9.33 percentage points, respectively. Adding FDCR further increases the average task success rate to 69.56%, Pass@3 to 77.67%, and Pass^3 to 60.00%, corresponding to additional gains of 5.00, 5.00, and 4.00 percentage

Table 1: Overall task performance and inference efficiency across five LLMs and three datasets. Inference efficiency is measured by token consumption (M) and execution time (s).

Method	Accuracy (%)			Efficiency	
	AVG	Pass@3	Pass^3	Token Cost	Time
Direct	36.33	48.00	23.67	7.301 ± 4.189	119.39 ± 81.65
Human-Aug.	60.22	70.67	49.33	8.215 ± 5.754	111.30 ± 62.74
Self-Evolution	58.67	70.00	46.67	8.301 ± 6.324	111.12 ± 58.43
ECDYSIS (w/o FDCR)	64.56	72.67	56.00	7.378 ± 5.490	108.40 ± 76.52
ECDYSIS (w/ FDCR)	**69.56**	**77.67**	**60.00**	7.240 ± 4.963	98.41 ± 55.28

points over ECDYSIS (w/o FDCR). These results show that round-level aggregation of failure evidence across task instances provides performance gains on its own, with FDCR further improving the aggregate results.

Training Efficiency. The training-efficiency results show that round-level failure aggregation reduces repeated modification calls during harness evolution. ECDYSIS (w/o FDCR) completes end-to-end training in 1,292.4 seconds on τ^2-Retail and 2,510.8 seconds on τ^2-Airline, corresponding to speedups of 1.42× and 3.23× over SE. ECDYSIS (w/ FDCR) takes 1,405.9 and 4,403.0 seconds, corresponding to speedups of 1.30× and 1.84× (Table 4). API cost follows the same pattern. On τ^2-Retail and τ^2-Airline, SE costs \$8.484 and \$6.382, whereas ECDYSIS (w/o FDCR) reduces these costs to \$2.485 and \$2.136, and ECDYSIS (w/ FDCR) costs \$5.763 and \$2.609, respectively (see Table 9). Overall, ECDYSIS decouples efficient failure-driven evolution from costly collaborative refinement. Failure aggregation provides the primary efficiency gains, while FDCR serves as an accuracy-oriented refinement module that trades additional evolution-time cost for improved harness quality. Across the complete training process, SE, Ecdysis (w/o FDCR), and Ecdysis (w/ FDCR) made 881, 267, and 482 LLM calls, respectively. These results suggest that the gains from our schemes stem from our algorithmic design rather than increased LLM usage.

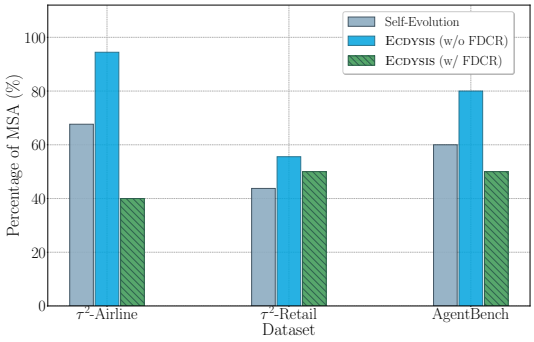


Figure 2: Model-accommodation ratios across datasets and harness evolution methods.

Inference Efficiency. We report token consumption and mean per-trajectory runtime for the complete held-out matrix in Table 2. Final evaluation tokens include only the input and output tokens consumed by the task model during the formal held-out trajectories. Runtime is measured per trajectory rather than as the wall-clock duration of the concurrent evaluation. Averaged over the 15 model and dataset combinations, final evaluation tokens are 8.301M for SE, 7.378M for ECDYSIS (w/o FDCR), and 7.240M for ECDYSIS (w/ FDCR), corresponding to relative reductions of 11.12% and 12.78%. This indicates that the efficiency acquired during evolution persists on held-out trajectories without any test-time modification. Runtime follows the same trend, with ECDYSIS (w/ FDCR) reducing the mean from 111.12 to 98.41 seconds per trajectory. The largest gain is on Qwen3-14B over τ^2-Airline, where the mean runtime drops from 197.61 to 57.16 seconds, corresponding to a 3.46× speedup. Because runtime also reflects model service latency and trajectory length, we report the full means and standard deviations in Table 2 rather than only the aggregate.

6 DATA STUDY

Motivation. Recent work on on-policy distillation shows that training efficiency depends not only on the amount of training data, but also on the information contained in individual training instances. Hou et al. (2026) find that a small set of carefully selected hard examples can nearly match training on a much larger dataset, with the gains attributed primarily to the longer reasoning trajectories induced by challenging problems. This raises a related question for harness evolution: *which training instances provide the most informative signal for harness training?* Unlike model training, where

Table 2: Inference cost comparison across five LLMs, three datasets, and five methods. Inference cost is measured by token consumption (M) and execution time (s).

Model	Method	τ^2-Airline		τ^2-Retail		AgentBench	
		Tokens	Time	Tokens	Time	Tokens	Time
Qwen3-8B	Direct	9.544 ± 0.352	67.55 ± 3.07	8.470 ± 0.416	108.92 ± 15.41	5.615 ± 0.073	96.09 ± 3.63
	Human-Aug.	13.961 ± 0.666	108.58 ± 15.42	9.536 ± 0.182	113.88 ± 2.25	1.562 ± 0.058	46.67 ± 0.73
	Self-Evolution	12.321 ± 1.205	175.23 ± 27.82	9.483 ± 0.495	171.32 ± 18.96	1.418 ± 0.094	48.21 ± 1.70
	ECDYSIS (w/o FDCR)	12.516 ± 0.428	102.43 ± 1.59	8.205 ± 0.118	180.13 ± 20.16	1.413 ± 0.001	47.24 ± 0.70
	ECDYSIS (w/ FDCR)	12.070 ± 1.458	98.87 ± 17.85	9.790 ± 0.326	142.30 ± 20.28	1.413 ± 0.028	45.68 ± 0.74
Qwen3-14B	Direct	13.685 ± 0.858	103.27 ± 1.99	19.239 ± 2.460	134.06 ± 20.80	4.853 ± 0.089	198.25 ± 1.10
	Human-Aug.	15.718 ± 1.539	101.83 ± 19.15	20.099 ± 3.226	158.45 ± 42.92	1.645 ± 0.000	52.01 ± 0.06
	Self-Evolution	24.308 ± 0.420	197.61 ± 11.94	14.392 ± 3.268	107.25 ± 31.44	1.618 ± 0.001	56.07 ± 0.34
	ECDYSIS (w/o FDCR)	16.089 ± 1.954	157.02 ± 31.85	18.424 ± 2.998	128.51 ± 49.22	1.392 ± 0.002	51.77 ± 1.12
	ECDYSIS (w/ FDCR)	16.417 ± 1.462	57.16 ± 11.15	12.746 ± 1.282	90.82 ± 18.23	1.514 ± 0.003	54.32 ± 2.20
Qwen3-32B	Direct	6.469 ± 0.349	75.51 ± 9.04	7.182 ± 0.066	77.40 ± 4.93	4.330 ± 0.183	98.18 ± 4.22
	Human-Aug.	11.047 ± 0.287	105.05 ± 8.83	12.297 ± 1.813	88.55 ± 11.70	2.139 ± 0.007	61.74 ± 0.26
	Self-Evolution	13.206 ± 0.483	117.99 ± 3.33	9.023 ± 0.353	79.43 ± 14.13	1.789 ± 0.026	60.79 ± 1.39
	ECDYSIS (w/o FDCR)	9.748 ± 0.578	91.34 ± 9.55	7.895 ± 0.220	73.45 ± 1.11	1.436 ± 0.001	54.40 ± 1.12
	ECDYSIS (w/ FDCR)	11.670 ± 1.009	100.36 ± 13.24	9.015 ± 0.299	102.02 ± 5.28	1.330 ± 0.006	51.54 ± 0.83
MiniMax-M2.7	Direct	5.252 ± 0.171	135.52 ± 12.59	5.173 ± 0.048	115.32 ± 4.57	2.937 ± 0.444	150.48 ± 18.61
	Human-Aug.	6.554 ± 0.074	154.62 ± 3.26	6.054 ± 0.077	270.38 ± 40.65	2.310 ± 0.119	162.26 ± 10.01
	Self-Evolution	8.161 ± 0.327	170.69 ± 15.05	5.728 ± 0.052	164.34 ± 3.70	2.066 ± 0.065	169.65 ± 6.06
	ECDYSIS (w/o FDCR)	6.127 ± 0.083	139.25 ± 3.80	5.779 ± 0.119	335.37 ± 43.90	1.664 ± 0.118	114.02 ± 4.39
	ECDYSIS (w/ FDCR)	6.526 ± 0.218	202.03 ± 9.94	5.804 ± 0.051	201.45 ± 4.04	1.474 ± 0.025	90.57 ± 3.43
Llama-3.1-8B	Direct	5.818 ± 0.360	32.23 ± 5.01	6.181 ± 0.266	29.31 ± 4.14	4.771 ± 0.170	368.80 ± 13.20
	Human-Aug.	9.640 ± 0.144	51.68 ± 3.22	8.586 ± 0.450	32.32 ± 5.33	2.076 ± 0.135	161.42 ± 7.92
	Self-Evolution	10.319 ± 0.299	43.38 ± 1.74	8.724 ± 0.276	33.53 ± 5.73	1.952 ± 0.197	71.34 ± 3.51
	ECDYSIS (w/o FDCR)	10.321 ± 0.434	51.97 ± 4.52	8.438 ± 0.254	54.72 ± 1.40	1.225 ± 0.000	44.43 ± 0.48
	ECDYSIS (w/ FDCR)	9.015 ± 0.178	159.60 ± 15.38	8.517 ± 0.228	32.30 ± 4.29	1.293 ± 0.004	47.15 ± 0.83

Table 3: Pooled input-cache utilization during harness evolution across three datasets.

Method	Cached Tokens	Total Input Tokens	Cache Hit Rate (%)
Self-Evolution	33,527,168	37,326,451	89.82
ECDYSIS (w/o FDCR)	15,881,728	16,731,412	**94.92**
ECDYSIS (w/ FDCR)	30,305,792	31,935,263	94.90

longer reasoning trajectories can be particularly valuable, harness training requires sufficient task complexity and behavioral breadth to exercise diverse interaction structures and expose distinct failure mechanisms. We view training tasks not merely as execution instances, but as diagnostic probes that reveal both the coverage of runtime behaviors and the deficiencies of the harness.

6.1 INSIGHTS FOR TRAINING DATA CURATION

The patterns across tasks motivate a curation strategy based on failure information. Training tasks differ in diagnostic value. Some expose new execution paths or interaction structures. Some reveal harness deficiencies missed in earlier runs. Others repeat known failure mechanisms. Selecting training data only by quantity or surface diversity can be costly and may provide little new information for harness evolution. We therefore select training failures using interaction structures and failure mechanisms together. We compare the evidence across task instances. A failure enters the reduced training set when it adds a new path, interaction structure, or failure mechanism. A failure provides little new information when it repeats a signal already present in the selected set. This process also supports failure diagnosis. Failures from different tasks can be different forms of the same underlying harness deficiency. Aggregating these failures helps ECDYSIS identify shared deficiencies and enable more general harness changes. To evaluate this strategy, we train ECDYSIS with a reduced training set containing only five training tasks. We compare its performance with training on the full training set and with training on a randomly selected set of five tasks. All settings use the same evolution procedure and evaluation protocol. The reduced training set achieves performance comparable to the full training set while substantially reducing the training cost of harness evolution. These results suggest that the benefit comes from the information in the selected failures rather than from reducing the training set alone. Results are summarized in Table 5.

6.2 BEYOND LOCAL FAILURE-DRIVEN EVOLUTION

Table 4: Training time (s) for the three harness self-evolution methods.

Dataset	Self-Evolution	ECDYSIS (w/o FDCR)	ECDYSIS (w/ FDCR)	Speedup
τ^2-Airline	8,120.6	2,510.8	4,403.0	**3.23**× / 1.84×
τ^2-Retail	1,831.4	1,292.4	1,405.9	1.42× / 1.30×
AgentBench	1,955.3	1,136.6	1,890.5	**1.72**× / 1.03×

Table 5: Data efficiency of harness evolution under different training configurations.

Method	AVG (%)	Pass@3	Pass^3	Training Cost
Self-Evolution	65.00 ± 5.00	75.00	55.00	\$8.484
ECDYSIS (Full)	$\mathbf{75.00 \pm 0.00}$	85.00	**60.00**	\$5.763
ECDYSIS (Reduced)	71.67 ± 7.64	**90.00**	45.00	**\$0.291**
ECDYSIS (Random)	65.00 ± 8.66	80.00	45.00	\$0.505

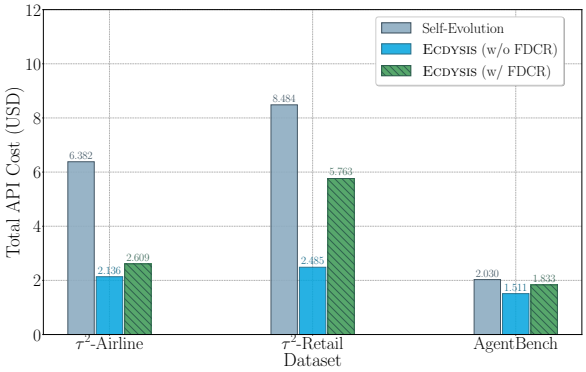


Figure 3: API cost of harness training.

To better understand how failure-driven evolution modifies the runtime harness, we conducted a fine-grained manual analysis of the training-time data and quantified t introduced in Section 3.1. Here, t denotes the proportion of MSA. Local failure-driven evolution can overfit to task-specific outcomes by promoting a particular training-task answer into a general runtime constraint. In some cases, the evolution agent removes a legitimate action option to avoid a specific model error, replacing a conditional decision that should be made by the model with a global harness-level prohibition. Although such modifications may improve the triggering evaluation case, they shrink the valid action space of the harness and may impair its generalization across models. We manually audited substantive harness modifications recorded in the saved round artifacts, including candidates later rolled back, while excluding temporary, unpersisted attempts. An independent modification decision is classified as either MSA or HLR. We compute t by pooling independent modification decisions across the three datasets. We observe $t = 60.0\%$ for SE, compared with $t = 45.5\%$ for ECDYSIS. Cross-instance aggregation alone does not automatically reduce MSA: ECDYSIS without FDCR has a pooled ratio of 75.61%, whereas FDCR reduces it to 45.45% (see Figure 2). This indicates that the reduction is primarily associated with the combination of cross-instance failure analysis and FDCR. These results suggest that ECDYSIS mitigates rather than eliminates MSA, directing persistent modifications toward evidence-supported runtime deficiencies while preserving a broader valid action space and reducing dependence on the model used during evolution. Detailed audit criteria, dataset-level results, and qualitative cases are provided in Appendix A.

7 CONCLUSION

In this paper, we present ECDYSIS, a framework for failure-driven evolution of runtime harnesses for LLM agents. Our central insight is that execution failures may reflect model limitations rather than harness deficiencies, making individual failures unreliable modification signals. ECDYSIS therefore separates failure-driven evolution into two complementary problems: cross-task failure recurrence for efficiently selecting evidence worth acting on, and FDCR for attributing recurring failures before translating them into harness modifications. Recurrence improves evolution efficiency and data efficiency, but may over-amplify recurring model-specific failures. FDCR incurs additional computation while reducing such accommodation and improving cross-LLM generalization. Across multiple LLMs and benchmarks, ECDYSIS improves harness performance and reduces inference-time token consumption. Overall, effective harness evolution requires both efficient evidence selection and reliable failure attribution to produce modifications that generalize beyond the models and tasks used during evolution.

AI USE STATEMENT

In this work, we used generative AI tools to assist with code development for implementing our methods. We did not use generative AI tools for other required research tasks, and the remaining required disclosure tasks are not applicable to this work. Additionally, AI assistants were used to polish the writing. We have reviewed all AI-assisted work, and the AI-assisted code was verified and tested for correctness. We take responsibility for the final content of this work.

ETHICS STATEMENT

All human experts involved in the manual annotation experiments are authors of this paper. No external human participants are involved in the annotation process.

REPRODUCIBILITY STATEMENT

To facilitate the reproducibility of our work, we provide the complete source code in the supplementary materials. The datasets used in our experiments are publicly available and open-source, enabling researchers to independently access the necessary resources and reproduce our experimental setup. In addition, we provide specific examples in the appendix to further facilitate readers' understanding and analysis of the data.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik R Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=OC2z7iSQKa.

Justin Chen, Archiki Prasad, Swarnadeep Saha, Elias Stengel-Eskin, and Mohit Bansal. MAgICoRe: Multi-agent, iterative, coarse-to-fine refinement for reasoning. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 32663–32686, Suzhou, China, November 2025a. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1660. URL https://aclanthology.org/2025.emnlp-main.1660/.

Mengzhuo Chen, Junjie Wang, Zhe Liu, Yawen Wang, Haiming Zheng, and Qing Wang. From failed trajectories to reliable llm agents: Diagnosing and repairing harness flaws. *arXiv preprint arXiv:2606.06324*, 2026a.

Mingju Chen, Can Lv, Guibin Zhang, Heng Chang, and Shiji Zhou. Harnessforge: Joint harness and policy evolution for adaptive agent systems. *arXiv preprint arXiv:2606.01779*, 2026b.

Zhaoling Chen, Robert Tang, Gangda Deng, Fang Wu, Jialong Wu, Zhiwei Jiang, Viktor Prasanna, Arman Cohan, and Xingyao Wang. LocAgent: Graph-guided LLM agents for code localization. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8697–8727, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.426. URL https://aclanthology.org/2025.acl-long.426/.

Zhinan Hou, Jiaqi Zhang, Xunliang Cai, and Keyou You. What matters in on-policy distillation? a perspective on data efficiency and data selection, 2026. URL https://arxiv.org/abs/2609.05198.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/36b7acf6f6010652b3f2a433774a66fe-Paper-Conference.pdf.

Yue Huang, Wenjie Wang, Han Bao, Yuchen Ma, Xiaonan Luo, Yi Nian, Haomin Zhuang, Zheyuan Liu, Yue Zhao, and Xiangliang Zhang. Memoharness: Agent harnesses that learn from experience, 2026. URL https://arxiv.org/abs/2607.14159.

Che Jiang, Jincheng Zhong, Yu Fu, Kai Tian, Junlin Yang, Kaikai Zhao, Yuchong Wang, Tianwei Luo, Weizhi Wang, Yuxin Zuo, et al. Self-improving agents in the era of experience: A survey of self-to meta-evolution. 2026a.

Wen Jiang, Mingmin Chu, Yimeng Tian, Qianxin Zhang, Haofei Yang, Rui Yang, Yang Liu, Tao Lv, and Fangming Li. Harnessevolve: Learning from reference trajectories for reliable agent self-evolution. *arXiv preprint arXiv:2609.00829*, 2026b.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=sY5N0zY5Od.

Hyunin Lee, Jinglue Xu, Jeffrey Seely, Donghyun Lee, Matei Zaharia, and Yujin Tang. Recursive harness self-improvement, 2026. URL https://arxiv.org/abs/2607.15524.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=zAdUB0aCTQ.

Wanying Qu, Qinghua Mao, Yu Li, Jiyao Liu, Xin Zhang, Dadi Guo, Yanxu Zhu, Qingyu Liu, Leitao Yuan, Xi Lin, Shanfeng Zhu, Yanwei Fu, Jing Shao, Xia Hu, and Dongrui Liu. She: Trajectory-driven safety harness evolution for llm agents, 2026. URL https://arxiv.org/abs/2608.09885.

Shuai Shao, Qihan Ren, Dongrui Liu, Chen Qian, Boyi Wei, Dadi Guo, Yang JingYi, Xinhao Song, Linfeng Zhang, Weinan Zhang, and Jing Shao. Your agent may misevolve: Emergent risks in self-evolving LLM agents. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=Fd1jgQQW28.

Shuai Shao, Kangning Zhang, Qingyao Li, Shijian Wang, Hao Wang, Wenxiang Jiao, Yuan Lu, Yi Guo, Weiwen Liu, and Weinan Zhang. Harness-r1: Learning to edit executable runtime harnesses from agent failure trajectories. *arXiv preprint arXiv:2608.02276*, 2026b.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

Guangya Wan, Mingyang Ling, Xiaoqi Ren, Rujun Han, Sheng Li, and Zizhao Zhang. COMPASS: Enhancing agent long-horizon reasoning with evolving context. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3360–3380, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.152. URL https://aclanthology.org/2026.acl-long.152/.

Ruhan Wang, Yucheng Shi, Zongxia Li, Zhongzhi Li, Yue Yu, Junyao Yang, Kishan Panaganti, Haitao Mi, Dongruo Zhou, and Leoweiliang. Harness handbook: Making evolving agent harnesses readable,navigable, and editable, 2026a. URL https://arxiv.org/abs/2607.13285.

Wenyi Wang, Piotr Piękos, Li Nanbo, Firas Laakom, Yimeng Chen, Mateusz Ostaszewski, Mingchen Zhuge, and Jürgen Schmidhuber. Huxley-gödel machine: Human-level coding agent development by an approximation of the optimal self-improving machine. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=T0EiEuhOOL.

Yike Wang, Huaisheng Zhu, Zhengyu Hu, Yige Yuan, Zhengyu Chen, Shakti Senthil, Hannaneh Hajishirzi, Yulia Tsvetkov, Pradeep Dasigi, and Teng Xiao. Rethinking the evaluation of harness evolution for agents. In *COLM 2026 The 2nd Workshop on Lifelong Agents: Learning, Aligning, and Evolving*, 2026c. URL https://openreview.net/forum?id=WVAeeSlVim.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J Yang, JQ Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu, Jiping Yu, Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang, Liyu Cai, Liyue Zhang, Longhao Chen, MS Di, MY Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qiwei Jiang, Rui Tian, Ruifan Xu, Ruijie Lu, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqian Chen, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, Ruyi Chen, SH Liu, Shanghao Lu, Shangmian Sun, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W Zhang, WL Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, YC Yan, YQ Wang, YW Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yuduan Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, ZF Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026a. URL https://arxiv.org/abs/2606.19348.

Jinghan Xu, Yikai Zhang, Aili Chen, Weiyuan Li, Jiaqing Liang, and Deqing Yang. Verify smarter, evolve further: Efficient harness evolution through behavior-aware verification. *arXiv preprint arXiv:2608.27311*, 2026b.

Tianshi Xu, Huifeng Wen, and Meng Li. Adapting the interface, not the model: Runtime harness adaptation for deterministic llm agents. *arXiv preprint arXiv:2605.22166*, 2026c.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 12028–12068, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/3339f19c5fcee3ad74502947a32be9e6-Paper-Conference.pdf.

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

Peiying Yu, Guoxin Chen, and Jingjing Wang. Table-critic: A multi-agent framework for collaborative criticism and refinement in table reasoning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17432–17451, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.853. URL https://aclanthology.org/2025.acl-long.853/.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-harness: Harnesses that improve themselves, 2026a. URL https://arxiv.org/abs/2606.09498.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Tjarko Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=pUpzQZTvGY.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, XiongHui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. Aflow: Automating agentic workflow generation. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/5492ecbce4439401798dcd2c90be94cd-Paper-Conference.pdf.

Jingsen Zhang, Zihang Tian, Xueyang Feng, Xu Chen, and Chong Chen. Enhancing recommendation explanations through user-centric refinement. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 8177–8191, Suzhou, China, November 2025b. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.434. URL https://aclanthology.org/2025.findings-emnlp.434/.

Luan Zhang, Ruochen Zhou, Dandan Song, Zhengyu Chen, Yuhang Tian, Jun Yang, Huipeng Ma, Chenhao Li, Guangyuan Feng, Xudong Li, Yizhou Jin, and Yan Xu. Harnesscompass: Guiding automatic harness evolution toward generalizable and effective agent harnesses, 2026c. URL https://arxiv.org/abs/2608.01918.

A MANUAL ANALYSIS

We conducted a fine-grained manual audit of the training-time harness modifications to characterize how observed failures are ultimately translated into modification decisions. The audit covers substantive modifications recorded in the saved round artifacts, including candidates that were later rolled back, while excluding temporary attempts that were never persisted. The unit of analysis is an *independent modification decision*, defined by its target problem and direct behavioral change. Multiple edits serving the same purpose are merged into a single decision, even when they span multiple functions, registration points, or supporting skills. Conversely, distinct issues within the same function are counted separately. For each decision, we examine the failure trajectory, the information available to the model and its actual actions, the task requirements and environment rules, the coding agent's modification record, and the behavior before and after modification. We then inspect which triggering conditions and available actions are changed. Thus, classification cannot be inferred from the failure score or implementation form alone: a prompt modification is not necessarily MSA, and a code-level rule is not necessarily an HLR. Each independent modification decision is assigned to one of two categories:

- **MSA**: the modification primarily compensates for limitations of the current task model in understanding, planning, action selection, execution, or stopping decisions through prompting, recovery steps, additional constraints, action gating, or related mechanisms, rather than correcting a confirmed harness deficiency. Such support can be effective and reusable across tasks. Its classification does not imply inevitable overfitting.
- **HLR**: environmental rules or execution evidence indicate an error or omission in parsing, state maintenance, interface semantics, execution conditions, or feedback presentation, and the modification directly repairs the runtime mechanism rather than deciding which task-level action the model should take.

We define the MSA ratio as the number of MSA decisions divided by all independent modification decisions. We further validate the MSA/HLR classification with five human experts, who independently evaluate each instance. The final labels are determined by majority voting, providing additional evidence for the reliability of the reported MSA ratios. The Overall result is computed by pooling decisions across τ^2-Retail, τ^2-Airline, and AgentBench rather than taking an arithmetic mean of dataset-level ratios. The ratio varies across datasets, indicating that modification tendencies depend on task structure and failure mechanisms rather than on a fixed harness layer or method label. For SE, the pooled ratio is 60.00%, compared with 45.45% for ECDYSIS (w/ FDCR). This corresponds to a reduction of 14.55 percentage points. On τ^2-Airline and AgentBench, the reductions are 27.65 and 10.00 percentage points, respectively, whereas τ^2-Retail increases slightly from 43.75% to 50.00%. Nevertheless, ECDYSIS with FDCR is lower than its w/o FDCR counterpart on all three datasets.

The ablation isolates the contributions of cross-instance aggregation and FDCR. ECDYSIS without FDCR has higher MSA ratios than SE on all three datasets, showing that aggregation alone does not automatically reduce MSA. Adding FDCR reduces the ratio on every dataset: from 55.56% to 50.00% on τ^2-Retail, from 94.44% to 40.00% on τ^2-Airline, and from 80.00% to 50.00% on AgentBench. When all independent modification decisions are pooled, the ratio decreases from 75.61% to 45.45%, a reduction of 30.16 percentage points. This pattern is consistent with the roles of the two components: cross-instance aggregation organizes related failure evidence, while FDCR further scrutinizes the failure evidence, modification basis, triggering conditions, and potential collateral effects before a harness change is committed. The results therefore indicate that the reduction in MSA is primarily associated with their combination rather than aggregation alone.

The qualitative cases illustrate why the distinction requires evidence-based analysis. In a τ^2-Retail trajectory, the model completed a return but selected a gift card when the user did not specify a refund destination, whereas the evaluation target was the original payment method. The environment policy nevertheless permits a gift-card refund when explicitly requested. The coding agent globally restricted refunds to the original payment method, preventing the observed error but eliminating a policy-permitted option. This is classified as *MSA*. In another Retail modification, the prompt stated that a modified order could still be canceled, while the cancellation tool rejected such requests. The modification aligned the prompt with the actual execution semantics and is therefore an HLR.

A second HLR occurred in τ^2-Airline. The task model used a write operation to query reservation information and submitted a baggage count identical to the current value. Although the baggage state remained unchanged, the operation created an unnecessary payment-history record that could affect subsequent refund calculations. The coding agent therefore added a safeguard that detects and rejects such no-op writes while directing the model to use a read operation instead. This modification prevents unintended state changes rather than prescribing a task-level action and is therefore classified as HLR.

These cases show that the same failure-driven process can yield different modification categories. The Retail refund case converts a local model error into a global restriction, whereas the other Retail and Airline cases correct erroneous runtime feedback or prevent harmful execution side effects. Consequently, neither failure outcomes nor prompt/code implementation forms can substitute for examining the underlying evidence, environment contract, and behavioral scope. The analysis thus provides the classification basis for the model-accommodation ratio and explains why ECDYSIS combines cross-task failure analysis with FDCR: to direct persistent modifications toward evidence-supported runtime deficiencies while avoiding unnecessary conversion of local model errors into global constraints.

B SUPPORTING EVIDENCE FOR THE QUALITATIVE CASES

We provide the supporting policy and code excerpts for the three qualitative cases discussed in the preceding section: the Retail refund restriction, the Retail cancellation-feedback correction, and the Airline no-op baggage-write safeguard. The excerpts show the relevant rules and actual modifications underlying their classification as MSA or HLR.

Refund restriction: Self-Evolution on τ^2-Retail

Existing policy.

> The user needs to provide a payment method to receive the refund.
>
> The refund must either go to the original payment method, or an existing gift card.

Harness diff.

```
-        if not isinstance(pm, GiftCard) and payment_method_id != original_pm_id:
+        if payment_method_id != original_pm_id:
             raise ValueError(
                 ...
             )
```

The removed condition allowed an existing gift card. The added condition rejects every refund destination other than the original payment method. This is the policy-permitted option excluded by the refund restriction described in the preceding section.

Cancellation feedback: Self-Evolution on τ^2-Retail

Existing cancellation check.

```
        if st == "pending (item modified)":
            raise ValueError(
                f"Order {order_id} has status 'pending (item modified)'. "
                "After items have been modified once, this order can no longer be cancelled
    "
                "with cancel_pending_order."
            )
```

Harness diff.

```
                 f"Note: Order {oid} items have already been modified once "
                 "(status: 'pending (item modified)'). Items cannot be modified "
-                "a second time. Address or payment method changes and full-order "
-                "cancellation are still available."
+                "a second time. Address or payment method changes are still "
+                "available, but this order CANNOT be cancelled with "
+                "cancel_pending_order."
```

The existing check already rejected cancellation after an item modification. The diff changes the feedback to match that check rather than adding a new cancellation restriction. This is a separate modification decision from the Retail refund restriction.

No-op baggage-write protection: Self-Evolution on τ^2-Airline

Failure described in the coding record.

The coding record reports that the model used `update_reservation_baggages` to inspect a reservation and submitted unchanged baggage counts. It identifies an unintended payment-history entry from this no-op write as a source of errors in later cancellation refunds.

Harness diff.

```
+class NoOpBaggageUpdateRule:
      ...
+     tool_name = "update_reservation_baggages"
      ...
+         r = db.reservations.get(reservation_id)
+         if r is None:
+             return
+         if (total_baggages == r.total_baggages
+                 and nonfree_baggages == r.nonfree_baggages):
+             raise ValueError(
                      ...
+                     "This update would create a no-op payment entry that alters the "
+                     "reservation's payment history. "
                      ...
              )
```

The added check blocks the write only when both requested baggage counts equal their current values. This is the no-op write safeguard described in the preceding section, addressing the payment-history side effect identified in the coding record.

C Additional Experimental Setup

Models. We evaluate five task LLMs: Qwen3-8B, Qwen3-14B, Qwen3-32B (Yang et al., 2025), MiniMax-M2.7 (230B)[2], and Llama-3.1-8B[3].

Evaluation Protocol. The three methods share the same training-time evaluation and candidate acceptance protocol, differing only in how they organize failure evidence and construct candidate modifications. We use Qwen3-8B as the task model and allow at most three candidate-generation rounds. Starting from the same initial harness, each method generates candidates from the resulting evidence, which are evaluated only in the subsequent complete evaluation and never on the trajectories used for their generation. A candidate is retained only if it strictly improves over the previously retained harness; otherwise, it is discarded. Accepted candidates are used for subsequent generation. After the final candidate is generated, a final evaluation over all training tasks is performed solely to determine whether it is accepted, with no further candidate generation. All evolution decisions are based exclusively on the training split. After evolution, the resulting harnesses are frozen and evaluated on held-out data under two independent protocols: first, Qwen3-8B is evaluated on all held-out tasks in each subset with three trials per task; second, we evaluate the full matrix of five task models, three datasets, and five methods, directly reusing the harnesses evolved with Qwen3-8B for the three evolution-based methods without further evolution.

Evaluation Metrics. We evaluate ECDYSIS from three perspectives: inference performance, training efficiency, and cost.

- **Inference Performance**. On the held-out split, we report *average task accuracy*, *Pass@3*, and *Pass^3*. Pass@3 denotes the proportion of tasks solved in at least one of the three trials, whereas Pass^3 denotes the proportion solved in all three trials.

[2]https://huggingface.co/MiniMaxAI/MiniMax-M2.7
[3]https://huggingface.co/meta-llama/Llama-3.1-8B

- **Training Efficiency**. *Training Time* measures the end-to-end wall-clock time from the initial training evaluation until the final harness is frozen. For held-out evaluation, we report *Final Evaluation Tokens*, defined as the total input and output tokens consumed by the task model.
- **Cost**. We report *Input Cache Hit Rate* and *API Cost*. Input Cache Hit Rate is the ratio of cached input tokens to total input tokens. API Cost is reported separately for training task evaluation and harness modification.

D Detailed Experimental Results

This appendix provides detailed results underlying the aggregate comparisons in the main text. We report per-dataset held-out task performance (Tables 6 and 7), dataset-level evolution-efficiency statistics (Tables 8 and 9), and MSA ratios (Table 10). The main text focuses on pooled comparisons, whereas the tables below show dataset-level variation and detailed cost breakdowns.

D.1 Task Performance

Averaged over the five task models and the three datasets (τ^2-Airline, τ^2-Retail, and AgentBench), the average accuracy increases from 58.67% under SE to 69.56% with ECDYSIS (w/ FDCR), a relative gain of 18.56%. Results on τ^2-Airline and τ^2-Retail are detailed in Table 6, and those on AgentBench are reported in Table 7. The improvement is especially pronounced for Qwen3-8B on the τ^2-Airline subset, where accuracy rises from 35.00% to 60.00%, Pass@3 from 50.00% to 80.00%, and Pass^3 from 20.00% to 40.00%. Notably, this improvement extends to models not used during evolution. On the same subset, Qwen3-32B improves from 51.67% under SE to 68.33% with ECDYSIS. These results demonstrate that a harness evolved with Qwen3-8B transfers to other task models without further evolution.

D.2 Held-Out Task Performance

Table 6 reports the detailed held-out results on the two τ^2-Bench subsets, while Table 7 reports the corresponding results on AgentBench. These tables provide the detailed values underlying the aggregate performance reported in Table 1.

D.3 Evolution Efficiency

Table 8 provides dataset-level cache-utilization statistics, while Table 9 reports the API-cost breakdown between training evaluation and evolution. These tables complement the pooled cache-utilization table (Table 3) and API-cost figure (Fig. 3) in the main text.

D.4 Model-Specific Accommodation (MSA)

Table 10 summarizes the dataset-level and pooled MSA ratios.

E Analysis of Evolution Process

This section analyzes the actual execution of different harness evolution methods from four aspects: candidate generation, API cost, token usage, and end-to-end training time. All statistics are collected from the complete evolution runs described in Section 4.1. The three methods use the same training evaluation, candidate validation, rollback, and early stopping protocols. We report the model calls and resource usage observed during actual execution. Therefore, these results reflect the cost of different candidate generation mechanisms along their actual execution paths rather than theoretical budgets normalized by a fixed number of evolution rounds.

E.1 Candidate Generation

Different evolution methods exhibit different candidate generation costs. Compared with SE, ECDYSIS requires fewer coding-agent calls, mainly because the two methods use different update granularities. SE generates and accumulates modifications for individual failures, while ECDYSIS aggregates

Table 6: Held-out task performance across five LLMs and two datasets.

Model	Dataset	Method	Accuracy (%)		
			AVG	Pass@3	Pass^3
Qwen3-8B	τ^2-Airline	Direct	16.67 ± 7.64	35.00	0.00
		Human-Aug.	55.00 ± 5.00	65.00	40.00
		Self-Evolution	35.00 ± 10.00	50.00	20.00
		ECDYSIS (w/o FDCR)	48.33 ± 5.77	55.00	**45.00**
		ECDYSIS (w/ FDCR)	**60.00 ± 5.00**	**80.00**	40.00
	τ^2-Retail	Direct	41.67 ± 5.77	70.00	15.00
		Human-Aug.	48.33 ± 2.89	80.00	20.00
		Self-Evolution	41.67 ± 15.28	75.00	15.00
		ECDYSIS (w/o FDCR)	53.33 ± 2.89	75.00	25.00
		ECDYSIS (w/ FDCR)	**63.33 ± 7.64**	**85.00**	**45.00**
Qwen3-14B	τ^2-Airline	Direct	16.67 ± 7.64	30.00	0.00
		Human-Aug.	38.33 ± 10.41	55.00	25.00
		Self-Evolution	23.33 ± 7.64	40.00	10.00
		ECDYSIS (w/o FDCR)	41.67 ± 7.64	60.00	**25.00**
		ECDYSIS (w/ FDCR)	**43.33 ± 7.64**	**65.00**	**25.00**
	τ^2-Retail	Direct	25.00 ± 5.00	50.00	5.00
		Human-Aug.	45.00 ± 13.23	65.00	20.00
		Self-Evolution	43.33 ± 12.58	65.00	20.00
		ECDYSIS (w/o FDCR)	48.33 ± 5.77	**70.00**	**35.00**
		ECDYSIS (w/ FDCR)	**53.33 ± 2.89**	**70.00**	30.00
Qwen3-32B	τ^2-Airline	Direct	20.00 ± 5.00	35.00	10.00
		Human-Aug.	56.67 ± 14.43	70.00	40.00
		Self-Evolution	51.67 ± 11.55	**80.00**	20.00
		ECDYSIS (w/o FDCR)	61.67 ± 5.77	**80.00**	40.00
		ECDYSIS (w/ FDCR)	**68.33 ± 2.89**	**80.00**	**50.00**
	τ^2-Retail	Direct	58.33 ± 2.89	75.00	35.00
		Human-Aug.	58.33 ± 7.64	80.00	45.00
		Self-Evolution	65.00 ± 5.00	75.00	55.00
		ECDYSIS (w/o FDCR)	70.00 ± 8.66	**85.00**	45.00
		ECDYSIS (w/ FDCR)	**75.00 ± 0.00**	**85.00**	**60.00**
MiniMax-M2.7	τ^2-Airline	Direct	80.00 ± 5.00	**95.00**	60.00
		Human-Aug.	81.67 ± 5.77	90.00	70.00
		Self-Evolution	73.33 ± 2.89	90.00	45.00
		ECDYSIS (w/o FDCR)	81.67 ± 2.89	85.00	**80.00**
		ECDYSIS (w/ FDCR)	**86.67 ± 7.64**	**95.00**	70.00
	τ^2-Retail	Direct	90.00 ± 5.00	**100.00**	80.00
		Human-Aug.	93.33 ± 5.77	**100.00**	85.00
		Self-Evolution	95.00 ± 5.00	**100.00**	90.00
		ECDYSIS (w/o FDCR)	**100.00 ± 0.00**	**100.00**	**100.00**
		ECDYSIS (w/ FDCR)	96.67 ± 2.89	**100.00**	90.00
Llama-3.1-8B	τ^2-Airline	Direct	25.00 ± 8.66	35.00	15.00
		Human-Aug.	31.67 ± 7.64	40.00	20.00
		Self-Evolution	28.33 ± 10.41	**45.00**	10.00
		ECDYSIS (w/o FDCR)	31.67 ± 5.77	40.00	20.00
		ECDYSIS (w/ FDCR)	**35.00 ± 5.00**	40.00	**30.00**
	τ^2-Retail	Direct	8.33 ± 2.89	10.00	5.00
		Human-Aug.	8.33 ± 2.89	**15.00**	5.00
		Self-Evolution	10.00 ± 0.00	**15.00**	5.00
		ECDYSIS (w/o FDCR)	10.00 ± 5.00	**15.00**	5.00
		ECDYSIS (w/ FDCR)	**11.67 ± 2.89**	**15.00**	**10.00**

the failure evidence within each evolution round and generates candidate modifications from the aggregated evidence. As the number of failures within a round increases, the candidate generation cost of SE grows accordingly, while ECDYSIS continues to generate candidates at the round level. The runtime logs further reveal differences between the two methods during actual candidate generation. For τ^2-Retail, approximately 67% of the coding-agent calls from SE completed normally, compared with approximately 97% in τ^2-Airline. In contrast, all candidate generation calls from ECDYSIS completed normally in both domains. These results show that accumulating modifications for in-

Table 7: Accuracy results across five models on AgentBench.

Model	Method	Accuracy (%)		
		AVG	Pass@3	Pass^3
Qwen3-8B	Direct	10.00	15.00	5.00
	Human-Aug.	85.00	**90.00**	80.00
	Self-Evolution	**90.00**	**90.00**	**90.00**
	ECDYSIS (w/o FDCR)	85.00	85.00	85.00
	ECDYSIS (w/ FDCR)	**90.00**	**90.00**	**90.00**
Qwen3-14B	Direct	28.33	30.00	25.00
	Human-Aug.	80.00	80.00	80.00
	Self-Evolution	85.00	85.00	85.00
	ECDYSIS (w/o FDCR)	85.00	85.00	85.00
	ECDYSIS (w/ FDCR)	**90.00**	**90.00**	**90.00**
Qwen3-32B	Direct	40.00	45.00	30.00
	Human-Aug.	75.00	75.00	75.00
	Self-Evolution	85.00	85.00	85.00
	ECDYSIS (w/o FDCR)	85.00	85.00	85.00
	ECDYSIS (w/ FDCR)	**90.00**	**90.00**	**90.00**
MiniMax-M2.7	Direct	80.00	**90.00**	65.00
	Human-Aug.	78.33	85.00	70.00
	Self-Evolution	83.33	85.00	80.00
	ECDYSIS (w/o FDCR)	86.67	**90.00**	85.00
	ECDYSIS (w/ FDCR)	**90.00**	**90.00**	**90.00**
Llama-3.1-8B	Direct	5.00	5.00	5.00
	Human-Aug.	68.33	70.00	65.00
	Self-Evolution	70.00	70.00	70.00
	ECDYSIS (w/o FDCR)	80.00	80.00	80.00
	ECDYSIS (w/ FDCR)	**90.00**	**90.00**	**90.00**

Table 8: Input-cache utilization during harness evolution across three datasets.

Dataset	Method	Cached Tokens	Total Input Tokens	Cache Hit Rate (%)
τ^2-**Airline**	Self-Evolution	12,368,896	13,824,385	89.47
	ECDYSIS (w/o FDCR)	4,384,768	4,601,409	**95.29**
	ECDYSIS (w/ FDCR)	5,103,104	5,470,490	93.28
τ^2-**Retail**	Self-Evolution	16,375,808	18,327,058	89.35
	ECDYSIS (w/o FDCR)	7,570,944	7,885,487	**96.01**
	ECDYSIS (w/ FDCR)	19,547,136	20,485,995	95.42
AgentBench	Self-Evolution	4,782,464	5,175,008	92.41
	ECDYSIS (w/o FDCR)	3,926,016	4,244,516	92.50
	ECDYSIS (w/ FDCR)	5,655,552	5,978,778	**94.59**

dividual failures not only increases candidate generation cost but is also associated with a higher proportion of incomplete calls in some runs. By aggregating failure evidence, ECDYSIS reduces repeated candidate generation and exhibits more stable execution behavior in our experiments.

E.2 API COST AND TOKEN USAGE

We divide the API cost of the complete training process into training evaluation and evolution (Fig. 3). Training evaluation covers model calls made during training task execution. Evolution covers the cost of candidate analysis and implementation after failure evidence is generated, including calls to the coding agent, FDCR when enabled, and other model calls during the evolution process. Each cost is calculated from the input tokens, output tokens, and API prices applicable at execution time, as recorded in the call logs. The training evaluation cost remains below $1 for all three methods, with most cost differences arising during evolution. SE incurs total costs of $8.484 and $6.382 on τ^2-Retail and τ^2-Airline, respectively. ECDYSIS (w/o FDCR) reduces these costs to $2.485 and $2.136, corresponding to reductions of 70.71% and 66.53%. The total costs of ECDYSIS (w/ FDCR) are $5.763 and $2.609, representing reductions of 32.07% and 59.12% relative to SE.

Table 9: Comparison of API costs for harness self-evolution across three datasets.

Dataset	Method	Training Evaluation	Evolution	Total
τ^2-**Airline**	Self-Evolution	\$0.837	\$5.545	**\$6.382**
	ECDYSIS (w/o FDCR)	\$0.967	\$1.169	**\$2.136**
	ECDYSIS (w/ FDCR)	\$0.870	\$1.739	**\$2.609**
τ^2-**Retail**	Self-Evolution	\$0.609	\$7.875	**\$8.484**
	ECDYSIS (w/o FDCR)	\$0.555	\$1.930	**\$2.485**
	ECDYSIS (w/ FDCR)	\$0.506	\$5.257	**\$5.763**
AgentBench	Self-Evolution	\$0.290	\$1.740	**\$2.030**
	ECDYSIS (w/o FDCR)	\$0.272	\$1.239	**\$1.511**
	ECDYSIS (w/ FDCR)	\$0.320	\$1.513	**\$1.833**

Table 10: Model-accommodation ratios across datasets. The pooled ratio is calculated after merging all independent modification decisions, rather than by averaging the three dataset-level ratios.

Dataset	Self-Evolution	ECDYSIS (w/o FDCR)	ECDYSIS (w/ FDCR)
τ^2-Airline	67.65%	94.44%	40.00%
τ^2-Retail	43.75%	55.56%	50.00%
AgentBench	60.00%	80.00%	50.00%
Overall	**60.00%**	**75.61%**	**45.45%**

Overall, both ECDYSIS configurations incur lower total API costs than SE. This result is consistent with the reduction in coding-agent calls discussed above and suggests that generating candidates from failure evidence aggregated over an entire round can effectively reduce the cost of repeated implementation. SE achieves a pooled cache hit rate of 89.82% across the three datasets (see Table 3). In comparison, the pooled cache hit rate of ECDYSIS (w/o FDCR) increases to 94.92%, while ECDYSIS (w/ FDCR) reaches 94.90%. Notably, on τ^2-Retail, ECDYSIS (w/ FDCR) incurs substantially lower total API cost than SE despite using more input tokens. These results further demonstrate the cost efficiency of ECDYSIS.

E.3 END-TO-END EVOLUTION TIME

We report the end-to-end training time from the initial training evaluation to the point when the final harness passes validation and is frozen in Table 4. For τ^2-Retail, ECDYSIS (w/o FDCR) and ECDYSIS (w/ FDCR) achieve end-to-end training speedups of 1.42× and 1.30× over SE, respectively. For τ^2-Airline, the corresponding speedups reach 3.23× and 1.84×. These results indicate that generating candidates from failure evidence aggregated over an entire round reduces redundant work during evolution and consistently shortens end-to-end training time across both domains.