# ADIAS: AUTOMATED DESIGN OF INTERACTIVE AGENTIC SYSTEMS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Self-evolving agents can improve without updating model weights by iteratively optimizing the harness based on evaluation feedback. Existing methods are largely candidate-centric: they record *evolution outcomes* (which agent variants have been produced and how they perform), rather than *evolution progress* (which issues, i.e., recurring failure patterns, have been fixed and which remain). Such methods pose two difficulties for cumulative harness evolution. First, repair context is scattered across candidate records and must be reconstructed every round, which leads to inefficient, redundant, or misdirected revisions. Second, because candidates are evaluated as a whole, partial fixes across different agents are difficult to combine, while ineffective changes persist alongside beneficial ones. To address these problems, we formulate **issue-centric agent optimization**, which explicitly carries forward evolutionary progress through a persistent issue state. We instantiate this paradigm in **ADIAS**, a harness-evolution framework with two key components. An *Issue Manager* maintains the issue state with stable issue identities, lifecycle status, supporting evidence, and intervention-outcome histories. An *Issue-Guided Optimizer* proposes repair targets from the issue state and performs focused harness revision. On five interactive benchmarks, ADIAS outperforms the strongest baseline by **25.2%** on average and achieves consistent gains across four backbone models. Controlled ablations show that removing the Issue Manager or the Issue-Guided Optimizer reduces performance by up to **40.7%**.

## 1 INTRODUCTION

Large language models (LLMs) are increasingly deployed as *agentic systems*, whose capabilities depend on both backbone models and the surrounding *agent harness* that governs planning, memory, tool use, and other mechanisms (Huang et al., 2024; Wang et al., 2024a). As model weights often remain fixed after deployment, improving the agent harness offers a practical and efficient way to adapt agents in target environments (Weng, 2026; Wang et al., 2026a). However, manually designing effective harnesses is costly. This motivates growing interest in automated agent design or harness self-evolution[1], where an optimizer repeatedly evaluates and revises the agent system itself (Madžar & Mekterović, 2026; ang Gao et al., 2026; Yue et al., 2026; Ning et al., 2026).

Agent harness self-evolution generally proceeds through iterative *revise–evaluate–summarize* loops. At each round, the system revises the current agent harness, evaluates the resulting candidate to obtain scores and trajectories, and summarizes these outcomes into a cross-round history for subsequent revision (Hu et al., 2025; Zhang et al., 2026a; Lee et al., 2026; Lin et al., 2026a; Chen et al., 2026; Zhang et al., 2026b). Existing methods can be characterized as **candidate-centric agent optimization**: cross-round experience is organized around candidate agents, and the objective of each round is to propose a better next candidate (Lee et al., 2026; Chen et al., 2026). Such methods record *evolution outcomes*: which agents have been produced and how they perform; rather than explicitly tracking *evolution progress*: how issues are addressed and which issues remain unresolved.

The candidate-centric methods introduce two difficulties for cumulative harness evolution. First, *inefficient optimization targeting*: evidence about whether an issue remains unresolved, which aspect requires modification, and which interventions have already been attempted is scattered across

[1]Throughout this paper, *agent design*, *agent optimization*, and *harness evolution* are used interchangeably to refer to modifying the agent harness while keeping the backbone model fixed.



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.

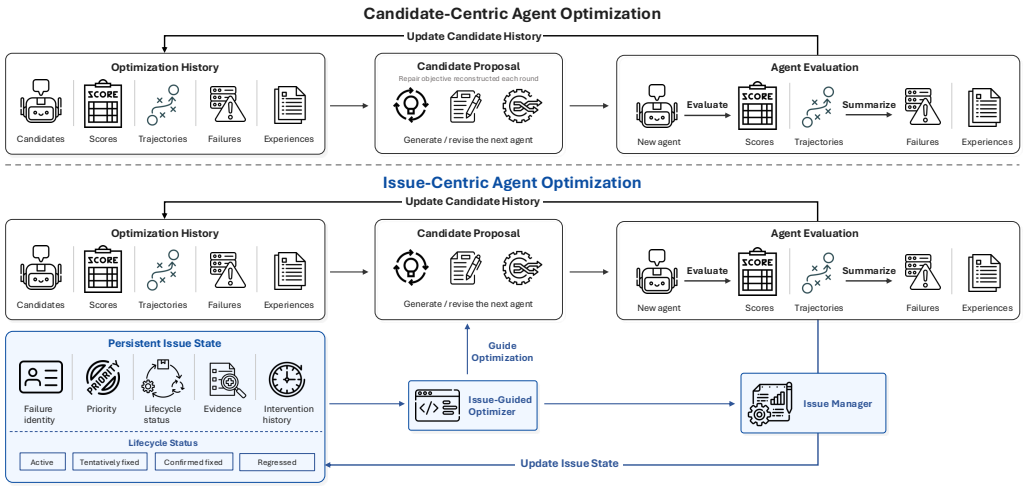



Figure 1: Comparison between candidate-centric agent optimization and issue-centric optimization.

candidate records. The optimizer must reconstruct this repair context before each revision, often resulting in broad, redundant, or misdirected changes. Second, *difficult progress consolidation*: candidate-centric methods treat complete candidates as the units of accumulated experience. Consequently, complementary partial repairs to the same issue remain distributed across different candidate branches and are difficult to combine, while ineffective interventions may entangle with beneficial changes and be carried into subsequent rounds.

To fill these gaps, we introduce **issue-centric agent optimization**, which organizes cross-round experience around persistent issues rather than candidate agents. Evaluation outcomes and diagnostic feedback are accumulated into a persistent issue state that explicitly records each issue's current status, supporting evidence, and intervention history. This state directly guides subsequent repair targets and revision directions, which enables the optimizer to continue, consolidate, or reconsider earlier attempts rather than repeatedly reconstructing the progress from candidate histories. Figure 1 contrasts this paradigm with previous candidate-centric evolution.

Realizing issue-centric agent optimization raises two key challenges. First, *reliable issue-state maintenance* requires mapping heterogeneous execution trajectories and sparse outcomes to stable, evidence-grounded issue identities, while reconciling new observations with historical records to track their repair progress across rounds. The same issue may appear differently across candidates, while a single revision may improve some behaviors but regress others, which complicates both identity association and outcome attribution. Second, *actionable and attributable revision* requires translating a dynamic multi-issue state into a concrete code modification. Repair targets and revision directions must be jointly determined from current evidence and intervention history, while modifications must remain focused enough to attribute subsequent outcomes to the intended repair.

To address these challenges, we propose **ADIAS** (**A**utomated **D**esign of **I**nteractive **A**gentic **S**ystems), an issue-centric framework for harness evolution with two key components. First, an *Issue Manager* associates newly diagnosed failures with stable issue identities and tracks their supporting evidence, lifecycle transitions, and intervention–outcome histories. This structure consolidates heterogeneous observations across agent generations while distinguishing the outcome of a targeted repair from simultaneous improvements or regressions in other issues. Second, an *Issue-Guided Optimizer* jointly selects target issues and revision directions from the current state and realizes this plan through focused full-code modification. Joint planning makes the descriptive issue state actionable for agent improvement, while focused revision improves the attribution of subsequent behavioral outcomes without restricting the agent design space. Our main contributions are:

- We propose **issue-centric agent optimization**, a novel agent harness self-evolution paradigm that organizes cross-round experience around persistent issues and explicitly tracks evolution progress.
- We instantiate this paradigm in **ADIAS**, a full-code harness-evolution framework with an Issue Manager for tracking evolution progress and an Issue-Guided Optimizer for focused code revision.

• We show that ADIAS outperforms the strongest baseline by **25.2%** on average across five interactive benchmarks, with consistent improvements across four backbone models. Controlled ablations illustrate the importance of issue-centric evolution: removing the persistent issue state or replacing issue-guided revision with candidate-centric policies results in performance drops of up to **40.7%**.

## 2 Related Work

**Self-Evolving Agent Harness.** Prior work spans increasing levels of harness expressiveness. Prompt-level methods optimize instructions, demonstrations, or other textual modules while keeping the surrounding agent structure fixed (Zhou et al., 2023; Pryzant et al., 2023; Yang et al., 2024; Agrawal et al., 2026; Khattab et al., 2024). Architecture-level methods search over predefined workflows, module compositions, or structured harness primitives (Li et al., 2024; Zhang et al., 2025b; Zhuge et al., 2024; Shang et al., 2025; Chen et al., 2026). Full-code methods expose executable agent implementations themselves to optimization, which enables revisions to prompts, memory, planning, tool use, control flow, and other components through iterative search or self-modification (Hu et al., 2025; Zhang et al., 2026a; Wang et al., 2026b; Lee et al., 2026; Lou et al., 2026; Cai et al., 2026; Nie et al., 2026). ADIAS operates at the full-code level but differs in its search-state control.

**Experience and Reusable Knowledge.** A line of work improves agents by accumulating reusable experience. Reflection and experiential-learning methods retain lessons or heuristics derived from previous trajectories, while systems such as ACE and SkillOpt maintain editable playbooks or skill documents (Shinn et al., 2023; Zhao et al., 2024; Zhang et al., 2026d; Yang et al., 2026). Other approaches persist and evolve structured knowledge, workflow, tool, or validation artifacts across iterations (Lin et al., 2026b; Huang et al., 2026; Lin et al., 2026a; Wang et al., 2026a). These methods demonstrate the value of carrying information across rounds. However, the persistent object is primarily reusable knowledge or an editable agent artifact. ADIAS instead maintains the evolving state of the failures that motivate those edits, including their status, evidence, and intervention history.

## 3 Method

### 3.1 Task Definition and Candidate-Centric Optimization

**Task Definition.** We study the automated design of an interactive agent's harness for a fixed task environment, evaluator, and backbone model, which are isolated from the agent-design workspace. Let $A \in \mathcal{A}$ be an executable agent implementation in the admissible full-code design space $\mathcal{A}$. Executing $A$ on a task instance $x$ yields a trajectory $\mathcal{T}(A, x) = (z_1, \ldots, z_L)$, where $z_h = (o_h, a_h, r_h, d_h)$ records the observation $o_h$ available to the agent, action $a_h$, task score $r_h$, and termination indicator $d_h$ at interaction step $h$. We partition task instances into training, validation, and held-out test splits (Wang et al., 2026c), and denote aggregate performance on split $q \in \{\text{train}, \text{val}, \text{test}\}$ by $M_q(A)$. Starting from an initial implementation $A_0$ and an optimization budget of $T$ rounds, the design system generates candidate $A_t$ at each round $t = 1, \ldots, T$, which is evaluated on task instances to obtain trajectories $\mathcal{T}_t$ and optimization-time performance measurements $M_t$. After $T$ rounds, the candidate in $\{A_0, \ldots, A_T\}$ with the highest validation performance $M_{\text{val}}(A_t)$ is selected and evaluated on the held-out test set. The objective is to achieve high test performance $M_{\text{test}}$.

**Candidate-Centric Optimization.** Most existing automated harness-design methods organize cross-round evolution histories around candidate agents. We formalize such history as $H_t = (h_0, \ldots, h_t), h_k = (A_k, \mathcal{T}_k, M_k, \xi_k)$, where each record $h_k$ is anchored to candidate $A_k$, and $\xi_k$ denotes any method-specific auxiliary experience associated with that candidate, such as reflections, diagnostic summaries, or edit histories. These candidate-centric methods can be abstracted as generating the next candidate from candidate-organized history:

$$A_{t+1} = \text{Optimizer}(H_t). \tag{1}$$

Here, Optimizer abstracts over the internal harness evolution procedure rather than assuming a particular revision pipeline. It may reason over previous candidate records and modify an archived candidate agent. Although $H_t$ may retain rich behavioral evidence, including explicit failure analysis in $\xi_k$, evidence and interventions concerning the same underlying issue remain distributed across separate candidate records. To determine the next revision, the optimizer must therefore recon-

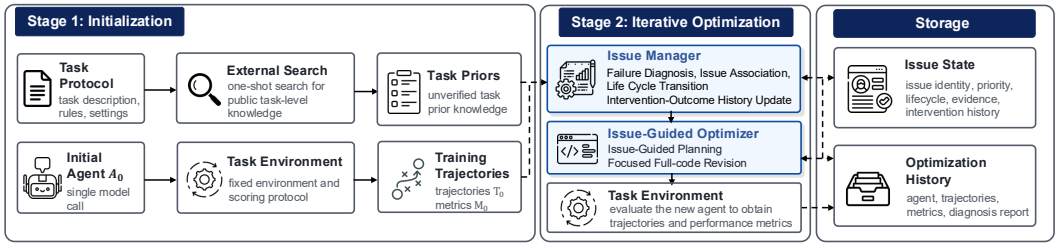



Figure 2: Overview of ADIAS that consists of two stages: initialization and iterative optimization.

struct the latent issue-level state: which issues remain unresolved, which interventions have been attempted, and what outcomes those interventions produced.

### 3.2 Issue-Centric Agent Optimization

Candidate-centric optimization represents cumulative cross-round experience through $H_t$, which leaves the evolution progress implicit and distributed across candidate records. Issue-centric optimization retains $H_t$ but introduces a complementary, persistent issue state $E_t = \{e_i^t\}_{i \in \mathcal{I}_{\leq t}}$, where $\mathcal{I}_{\leq t}$ denotes the set of issues identified up to round $t$, and $e_i^t$ records the accumulated evolution progress associated with issue $i$. Whereas each record $h_k \in H_t$ is indexed by a candidate at a particular round, each record $e_i^t \in E_t$ is indexed by an underlying issue and persists across candidate generations. Evidence, attempted interventions, and their outcomes can therefore be accumulated around the same issue rather than remaining attached to separate candidates. At each round, issue-centric optimization updates the issue state and conditions the next revision on both representations:

$$E_t = \mathrm{IssueUpdate}(E_{t-1}, H_t), \tag{2}$$

$$A_{t+1} = \mathrm{Optimizer}(H_t, E_t). \tag{3}$$

Here, $\mathrm{IssueUpdate}$ abstracts any procedure that identifies new issues, associates new evidence with existing issues, and consolidates outcomes of attempted interventions. The optimizer then uses both $H_t$ and $E_t$ to determine subsequent revisions. Therefore, while candidate-centric optimization needs to repeatedly reconstruct issue-level progress from $H_t$, issue-centric optimization explicitly maintains and carries this progress across rounds.

### 3.3 ADIAS

ADIAS instantiates the issue-centric formulation in Eq.(2) and Eq.(3) with two concrete components: an *Issue Manager* and an *Issue-Guided Optimizer*. The Issue Manager maintains a persistent issue state, while the Issue-Guided Optimizer achieves issue-conditioned harness code modification. Figure 2 provides an overview workflow of ADIAS, which consists of two stages: initialization and iterative optimization. The **initialization stage** searches for task-level priors and behavioral evidence collected from the initial agent. ADIAS begins with an initial agent $A_0$ with a single model call. To reduce cold-start uncertainty, a one-shot search over publicly available task-level information produces provisional priors $P$ about task requirements, likely failure modes, and potentially useful agent mechanisms. $P$ is incorporated into the initial issue state as unverified task prior knowledge. Details are in Appendix C.1. The environment evaluates $A_0$ to obtain initial trajectories $\mathcal{T}_0$ and performance metrics $M_0$. The **optimization stage** then proceeds as a three-step loop: Issue Manager updates the issue state, Issue-Guided Optimizer modifies the agent code, and environment evaluates the new agent to get trajectories and metrics to update the optimization history.

#### 3.3.1 Issue Manager

The issue record $e_i^t$ is designed as $e_i^t = (id_i, q_i^t, s_i^t, \mathcal{B}_i^t, \mathcal{U}_i^t)$, where $id_i$ is a stable identity, $q_i^t$ is priority, $s_i^t$ is lifecycle status, $\mathcal{B}_i^t$ is supporting evidence, and $\mathcal{U}_i^t$ is intervention-outcome history. The Issue Manager maintains this state by failure diagnosis, issue association, lifecycle transition, and intervention-outcome history update.

**Failure Diagnosis.** At each round $t \geq 1$, a dedicated diagnostic agent analyzes trajectories $\mathcal{T}_{t-1}$ and performance metrics $M_{t-1}$ of agent $A_{t-1}$, and produces a diagnostic report $D_t$ containing observed failure categories, supporting evidence (trajectories that support the failure), and inferred causes. Details are in Appendix C.2. This process converts candidate-level execution evidence into observations that can be reconciled with the persistent issue state.

**Issue Association.** For each issue in $D_t$, the Issue Manager either associates it with an existing issue in $E_{t-1}$ or creates a new persistent identity $id_i$ based on the failure category. Repeated observations and supporting evidence $\mathcal{B}_i^t$ are merged into the corresponding issue record, which preserves issue identity across rounds and allows evolution progress to accumulate without rediscovering the same underlying failure. Each issue is also assigned a priority $q_i^t$ based on issue severity, as determined by LLMs. Details of issue control are in Appendix C.3.

**Lifecycle Transition.** The issue status $s_i^t$ transitions among ACTIVE, TENTATIVELY-FIXED, CONFIRMED-FIXED, and REGRESSED. New and currently observed issues are marked ACTIVE, while a previously fixed issue that reappears is marked REGRESSED. An ACTIVE or REGRESSED issue becomes TENTATIVELY-FIXED once absent, and becomes CONFIRMED-FIXED after remaining absent for $\alpha_{\min}$ consecutive evaluations ($\alpha_{\min} = 2$ used in our experiments). These transitions distinguish temporary disappearance from reliable resolution while making recurring failures explicit.

**Intervention-Outcome History Update.** For each targeted issue, the Issue Manager records the round-$t$ revision and its outcome in $\mathcal{U}_i^t$. The targeted intervention outcome is stored separately from lifecycle changes observed for other issues, making progress on the intended repair distinguishable from concurrent improvements or regressions elsewhere. Supported intervention outcomes are distilled into reusable repair lessons. This issue-specific history supports outcome attribution and helps subsequent rounds reuse effective interventions while avoiding previously unsuccessful directions.

### 3.3.2 ISSUE-GUIDED OPTIMIZER

Issue-Guided Optimizer comprises two steps: issue-guided planning and focused full-code revision.

**Issue-Guided Planning.** At round $t$, an LLM optimizer jointly determines a parent candidate and revision plan as $(A_{p_t}, R_t) = \mathrm{Plan}(H_t, E_t)$, specifying *what to repair*, *where to resume optimization*, and *how to revise*. It first selects a small set of target issues from $E_t$, prioritizing issues that are severe, repeatedly observed, currently active, or recently regressed. It then selects $A_{p_t}$ from $H_t$ based on its repair history with respect to the target issues, which favors candidates that preserve useful progress on these issues to provide a suitable starting point for further evolution. Finally, the optimizer analyzes previous intervention outcomes to specify corresponding revision plan in $R_t$.

**Focused Full-Code Revision.** The optimizer executes the plan as $A_t = \mathrm{Modify}(A_{p_t}, R_t)$ under two fixed constraints. First, the entire full-code space $\mathcal{A}$ remains editable, which allows revisions to any agent component, including prompts, observation processing, memory, planning, tool-use policies, control flow, verification, and recovery mechanisms. Second, each revision modifies at most $K$ harness components ($K = 3$ in our experiments), which keeps the intervention focused for more reliable outcome attribution while preserving full-code flexibility.

The resulting agent $A_t$ is evaluated to obtain $\mathcal{T}_t$ and $M_t$. In the following round, its execution behavior is diagnosed to produce the diagnostic report $D_{t+1}$, after which the optimization history and issue state are updated to $H_{t+1}$ and $E_{t+1}$, respectively. The updated pair $(H_{t+1}, E_{t+1})$ then guides the next optimization round.

## 4 EXPERIMENTS

### 4.1 EXPERIMENTAL SETUP

**Benchmarks.** We evaluate ADIAS on five interactive agent benchmarks spanning different scenarios, including $\tau$-Bench for tool-use in customer service (Yao et al., 2025), ALFWorld for embodied AI planning (Shridhar et al., 2021), TextCraft for compositional crafting games (Prasad et al., 2024), WebShop for web navigation (Yao et al., 2022), and ScienceWorld for scientific experimentation (Wang et al., 2022). These environments require agents to make a sequence of state-dependent de-

Table 1: Task performance and efficiency across five interactive benchmarks using DeepSeek-V4-Flash. Best and second-best results are highlighted in green and blue, respectively.

| Method | $\tau$-Bench-Return | | ALFWorld | | TextCraft | | WebShop | | ScienceWorld | | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Score ↑ | Eff. ↑ | Score ↑ | Eff. ↑ | Score ↑ | Eff. ↑ | Score ↑ | Eff. ↑ | Score ↑ | Eff. ↑ | Score ↑ |
| Handcrafted | 75.0 | 5.73 | 70.9 | 3.60 | 38.0 | 1.52 | 39.1 | 2.13 | 38.1 | 3.05 | 52.2 |
| SkillOpt | 12.5 | 0.45 | 0.0 | 0.00 | 37.0 | 1.70 | 22.3 | 1.17 | 30.1 | 1.89 | 20.4 |
| Meta-Harness | 43.8 | 2.39 | 59.7 | 3.32 | 24.0 | 0.95 | 46.0 | 3.71 | 26.5 | 1.65 | 40.0 |
| AHE | 56.2 | 3.05 | 78.4 | 4.48 | 35.0 | 1.39 | 57.3 | 5.46 | 42.6 | 3.44 | 53.9 |
| DGM-H | 59.4 | 3.93 | 69.4 | 3.42 | 76.0 | 5.98 | 57.5 | 5.42 | 50.9 | 3.98 | 62.6 |
| ADIAS (Ours) | 81.3 | 6.35 | 94.0 | 8.95 | 91.0 | 9.01 | 69.4 | 5.55 | 56.3 | 3.99 | 78.4 |

cisions rather than produce a single static response. To control computational costs, we construct representative task subsets for the main experiments. We report benchmark details in Appendix D.1.

**Baselines.** We compare ADIAS with five representative baselines on different editable scopes and experience organization. We use a manually designed ReAct-style agent (Yao et al., 2023) with task-level memory as the fixed, non-automated baseline. SkillOpt (Yang et al., 2026) performs prompt-level agent optimization by maintaining a skill document. Agentic Harness Engineering (AHE) (Lin et al., 2026a) performs architecture-level optimization over pre-defined components of agent harnesses. Meta-Harness (Lee et al., 2026) performs end-to-end optimization over the entire agent harness. DGM-Hyperagents (DGM-H) (Zhang et al., 2026b) integrates the task agent and meta agent into a single editable program and allows the resulting hyperagent to modify both its task-solving logic and the mechanism used to generate future improvements. All methods use the same benchmark wrappers, task splits, action interfaces, and scoring scripts. Details are in Appendix D.2.

**Models.** We use DeepSeek-V4-Flash (DeepSeek, 2026) as the default backbone model. To evaluate cross-model robustness, we additionally conduct experiments with GLM-5.2 (Z.ai, 2026), Hy3-Preview (Tencent Hy, 2026), and GPT-5.4 (OpenAI, 2026). For each experimental setting, the same backbone model is used for both the design agent and the task agent. We apply this matched-model protocol to all automated baselines and ADIAS, which ensures that performance differences are attributable to the agent-design method rather than to a stronger optimizer model. For $\tau$-Bench, the user simulator is held fixed to DeepSeek-V4-Flash across all settings.

**Evaluation.** For each method, we select the agent with the highest validation performance and evaluate it on the held-out test set. All methods use the same task splits, environment interfaces, optimization budgets, and benchmark-specific evaluators. We report the native metric of each benchmark (%): success rate for $\tau$-Bench, ALFWorld and TextCraft, and the normalized native task score for WebShop and ScienceWorld. No test tasks or validation trajectory feedback are exposed during optimization. Moreover, we report interaction efficiency, which measures task performance per environment interaction. Since the semantics of an interaction step differ across environments, efficiency is compared only within the same benchmark. More details are in Appendix D.3.

**Experimental Settings.** We run each automated method for a fixed budget of 10 optimization iterations. At each iteration, 15 training episodes are sampled for agent optimization. The same iteration and rollout budgets are applied to all methods within each benchmark. Each task-agent episode is limited to 30 environment interaction steps or dialogue turns. We also keep decoding and inference configurations fixed across methods using the same backbone model. We separately report meta-level optimization token consumption and controlled cost comparisons in Appendix F.3.

## 4.2 Results

### 4.2.1 Main Results

We report task performance, cross-model robustness, and optimization efficiency in the main text. Additional analyses of execution robustness, optimization variability, test-set uncertainty, and broader task distributions are provided in Appendix E.

**Strong Task Performance and Efficiency.** Table 1 compares ADIAS with fixed handcrafted strategies and recent automated design baselines across five interactive benchmarks. ADIAS achieves the best task performance and interaction efficiency on all five benchmarks, which shows robust

Table 2: Cross-model task performance and efficiency on $\tau$-Bench-Return. Best and second-best results are highlighted in green and blue, respectively.

| Method | DeepSeek-V4-Flash | | GLM-5.2 | | Hy3-Preview | | GPT-5.4 | | Avg. Score ↑ |
|---|---|---|---|---|---|---|---|---|---|
| | Score ↑ | Eff. ↑ | Score ↑ | Eff. ↑ | Score ↑ | Eff. ↑ | Score ↑ | Eff. ↑ | |
| Handcrafted | 75.0 | 5.73 | 84.4 | 6.49 | 15.6 | 0.86 | 71.9 | 5.89 | 61.73 |
| SkillOpt | 12.5 | 0.45 | 12.5 | 0.67 | 12.5 | 0.59 | 15.6 | 0.74 | 13.28 |
| Meta-Harness | 43.8 | 2.39 | 78.1 | 6.51 | 68.8 | 5.38 | 65.6 | 5.56 | 64.08 |
| AHE | 56.2 | 3.05 | 62.5 | 4.08 | 78.1 | 6.20 | 75.0 | 5.81 | 67.95 |
| DGM-H | 59.4 | 3.93 | 65.6 | 4.10 | 75.0 | 5.86 | 71.9 | 5.33 | 67.98 |
| ADIAS (Ours) | 81.3 | 6.35 | 90.6 | 7.13 | 84.4 | 6.54 | 87.5 | 6.63 | 85.95 |

generalization across environments with diverse interaction patterns and task objectives. Averaged across benchmarks, ADIAS reaches a score of 78.4 versus 62.6 for the strongest baseline (DGM-H), a relative improvement of 25.2%. Specifically, scores of ADIAS on $\tau$-Bench, ALFWorld, TextCraft, WebShop, and ScienceWorld are 81.3, 94.0, 91.0, 69.4, and 56.3, respectively, which consistently surpass all baselines. These gains are not obtained through longer interactions. ADIAS also achieves the highest interaction efficiency on every benchmark, which indicates that its learned strategies translate environment interactions into task progress more effectively. The advantage is particularly pronounced on ALFWorld and TextCraft, where ADIAS reaches efficiencies of 8.95 and 9.01, and substantially exceeds the strongest baseline results of 4.48 (AHE) and 5.98 (DGM-H), respectively.

**Cross-Model Robustness.** We conduct a cross-model evaluation on $\tau$-Bench-Return using four different models: DeepSeek-V4-Flash, GLM-5.2, Hy3-Preview, and GPT-5.4. Table 2 shows that ADIAS consistently achieves the highest performance across all backbone models. The corresponding interaction efficiencies are also consistently the highest among all compared methods. These results show that ADIAS provides robust harness evolution across different backbone models.

**High Optimization Efficiency and Stability.** Figure 3 reports the mean best-so-far validation performance across three independent ALFWorld runs, with shaded regions indicating $\pm 1$ standard deviation. ADIAS rapidly discovers strong agent designs, reaching nearly 90% validation performance by the second round and maintaining the highest mean best-so-far score throughout the remaining optimization process. Its consistently strong performance across independent runs indicates that the gains are not specific to a single favorable optimization trajectory. In comparison, DGM-H and Meta-Harness require more rounds to reach competitive performance. SkillOpt shows limited progress as it modifies only a persistent textual skill document and cannot introduce additional agent components, such as explicit memory mechanisms, needed for effective interaction in ALFWorld.

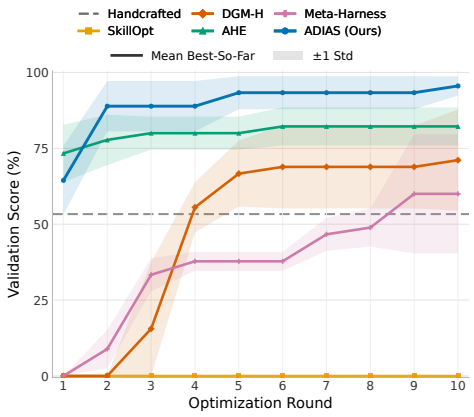



Figure 3: Results on ALFWorld.

### 4.2.2 Ablation Study

As shown in Table 3, we ablate three coupled aspects of our proposed Issue Manager and the Issue-Guided Optimizer: the evidence used to update issue states, the issue-level representation for cross-round persistence, and whether this state actively guides optimization.

**External priors and round-level diagnosis provide necessary evidence.** Removing either source reduces both performance and interaction efficiency across all three benchmarks. Without the external prior, the largest drop occurs on TextCraft, from 91.0 to 61.0. Without round-level diagnosis, ALFWorld decreases from 94.0 to 63.4. These results indicate that the two mechanisms play important roles: the external prior reduces cold-start uncertainty, while trajectory diagnosis provides the behavioral evidence required to ground and update the issue state.

Table 3: Ablation study on representative benchmarks using DeepSeek-V4-Flash. Average Score is the unweighted average across the three benchmarks, and $\Delta$ denotes its change relative to full ADIAS. Best and second-best results are highlighted in green and blue, respectively.

| Setting | $\tau$-Bench-Return | | ALFWorld | | TextCraft | | Avg. Score ↑ | $\Delta$ |
|---|---|---|---|---|---|---|---|---|
| | Score ↑ | Eff. ↑ | Score ↑ | Eff. ↑ | Score ↑ | Eff. ↑ | | |
| ADIAS (Full) | 81.3 | 6.35 | 94.0 | 8.95 | 91.0 | 9.01 | 88.8 | – |
| *Ablating evidence for issue-state construction* | | | | | | | | |
| w/o External Prior | 75.0 | 5.64 | 82.8 | 5.27 | 61.0 | 3.10 | 72.9 | -15.9 (17.9%) |
| w/o Round-Level Diagnosis | 78.1 | 5.66 | 63.4 | 3.41 | 76.0 | 5.80 | 72.5 | -16.3 (18.4%) |
| *Replacing persistent issue state with a raw archive* | | | | | | | | |
| w/ Archive-Wide Synthesis | 65.6 | 4.43 | 60.4 | 3.25 | 32.0 | 1.27 | 52.7 | -36.1 (40.7%) |
| *Decoupling issue state from optimization control* | | | | | | | | |
| w/ Best-Candidate Revision | 71.9 | 5.10 | 75.4 | 4.54 | 74.0 | 5.03 | 73.8 | -15.0 (16.9%) |
| w/ Latest-Candidate Continuation | 62.5 | 4.37 | 41.0 | 1.81 | 60.0 | 3.53 | 54.5 | -34.3 (38.6%) |

**A raw candidate archive cannot replace persistent issue state.** *Archive-Wide Synthesis* removes the durable issue state and follows a Meta-Harness-style mechanism (Lee et al., 2026). The designer receives a raw archive containing candidate agents, evaluation results, trajectory histories, and diagnostic reports, and synthesizes the next optimization step directly from this archive. Its scores decrease to 65.6, 60.4, and 32.0 on $\tau$-Bench-Return, ALFWorld, and TextCraft, respectively. Although the archive preserves rich historical evidence, identifying an effective revision direction remains difficult in interactive environments with long trajectories and sparse feedback, where useful signals must be recovered from heterogeneous candidate-level records.

**Maintaining issue state is insufficient unless it controls optimization.** *Best-Candidate Revision* and *Latest-Candidate Continuation* retain the same candidate history and persistent issue states as ADIAS, but the issue state is treated only as contextual information: the actual parent is selected by validation score or generation order, rather than the Issue-Guided Optimizer. *Best-Candidate Revision*, the strongest ablation, still underperforms ADIAS by 9.4, 18.6, and 17.0 points across the three benchmarks. As each round returns to the best-score candidate, locally useful interventions discovered in different branches remain isolated and are difficult to consolidate into a single improved lineage. *Latest-Candidate Continuation* exhibits another limitation. As every round continues from the most recent candidate, the next revision is more likely to inherit ineffective or regressive modifications. The parents selected by these two methods are not necessarily the most suitable starting point for the current evolution. The persistent issue state should therefore serve as an operational control signal to guide optimization.

### 4.3 Qualitative Analysis

We inspect optimization traces to understand how issue state shapes the harness evolution process. The behaviors described below recur across all three benchmarks in Table 3. Figure 4 illustrates a representative example of the optimization process on the TextCraft benchmark.

**A raw candidate archive provides weaker control signals than persistent issue state in long-horizon, sparse-feedback environments.** We observe an anchoring mechanism similar to that described in Lee et al. (2026): optimization often starts from the highest-scoring candidate or the latest accepted one, and then attempts to summarize a new improvement direction from the surrounding archive. This strategy is less reliable when useful evidence is distributed across long interaction trajectories and sparse outcome signals. For example, in TextCraft (Figure 4), later rounds repeatedly return to previous high-score generations 2 and 6 and explore prompt restructuring, admissible-command matching, and command-selection heuristics, but the underlying missing-quantity error is never localized (get quartz vs. get 1 quartz). Reconstructing issues from candidate records each round therefore obscures effective repairs, whereas persistent issue state preserves explicit issue–intervention–outcome histories to guide revision.

**Best-Candidate Revision makes partial repairs slower to consolidate across branches.** When a revision partially addresses an issue but lowers the aggregate score, its code-level changes are not inherited by the next candidate. The optimizer has to recover the useful insight from history and attempt a related repair again on a higher-scoring parent. In TextCraft, early branches introduce repairs

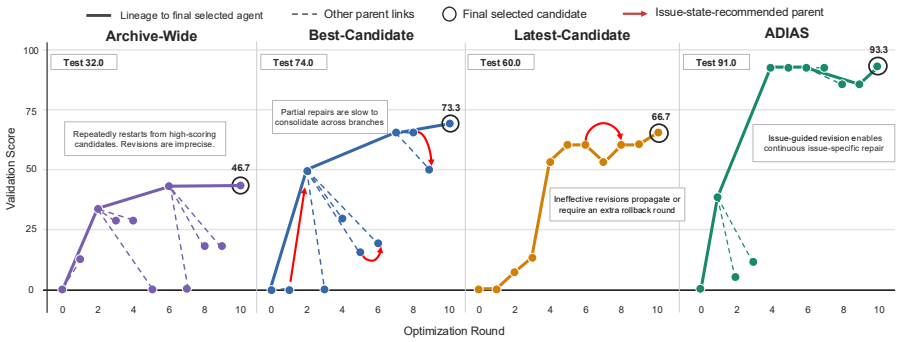



Figure 4: Comparison of optimization processes on the TextCraft benchmark.

of command recovery and crafting guidance, but their validation scores remain below generation 2, causing subsequent rounds to repeatedly return to that candidate and abandon parts of the previous implementations. Only after several alternative revisions, generation 7 combines effective revisions and improves validation from 50.0 to 66.7. The pattern is stronger in ALFWorld, where generations 4-10 repeatedly branch from generation 3 while separately targeting object identity, wrong-object recovery, action validation, and planner behavior. Although these interventions remain in history, their useful components must be repeatedly attributed and reimplemented across branches, making consolidation slower and less direct than with issue-specific intervention outcomes.

**Latest-Candidate Continuation makes ineffective revisions more likely to propagate, either accumulating errors or consuming additional rounds to recover.** For example, in TextCraft, generation 7 introduces fallback exploration and reduces validation success from 60.0 to 53.3. Although the issue state recommends returning to generation 6, the fixed policy makes generation 8 inherit generation 7. The next round removes the ineffective modification merely to recover the previous validation score, which spends one additional round on rollback rather than further improvement. In contrast, ADIAS would return to generation 6 and resume optimization from a more suitable parent. The same problem is more pronounced in ALFWorld. After generation 8 is identified as unsuccessful, the issue state recommends generation 4 as the next parent, yet generation 9 is still derived from generation 8. In this case, the ineffective revision remains instead of being bypassed, which makes subsequent candidates more likely to inherit its errors.

**ADIAS enables continuous issue-specific repair by selecting a more appropriate starting point and revision direction in each round.** In TextCraft, ADIAS identifies the recurring command failure and repairs it by normalization. The intervention and its outcome are then written back to the persistent issue state, which allows subsequent rounds to retain the successful repair and continue addressing the remaining grounding and recovery issues. When modifications at generation 9 fail to improve task success rate but contribute to other issues, such as interaction efficiency, ADIAS still treats round 9 as a more suitable parent to accumulate partial revision. A similar process appears in ALFWorld, where optimization progresses from broad planning failures to more specific issues in action correction, planning heuristics, and target-object extraction. Across all benchmarks, ADIAS uses the active issue, previous interventions, and their observed outcomes to determine both where to restart and what to modify, which supports a continuous repair process.

## 5 Conclusion

We introduced issue-centric agent optimization, which reframes agent harness self-evolution as a cumulative optimization process rather than repeated candidate generation. We instantiated this formulation in ADIAS, a framework for automated full-code harness evolution that combines persistent issue state maintenance and issue-guided optimization. Across five interactive benchmarks, ADIAS outperforms the strongest baseline by 25.2% on average and generalizes consistently across four backbone models. Controlled ablations and qualitative analyses further show that both persistent issue-level organization and its role as an optimization control state are necessary. These results suggest that future self-evolving agentic systems should treat optimization progress as an explicit, persistent, and operational state.

## AI USE STATEMENT

In this work, we used generative AI tools to assist with coding and to improve the grammar, clarity, and readability of the manuscript. AI-assisted code was reviewed, verified, and tested by the authors before being incorporated into the experiments. AI-assisted text editing was limited to language refinement and did not alter the scientific content, claims, or conclusions of the work. We reviewed all AI-assisted outputs and take full responsibility for the final content of this work, including any text or code produced with the aid of generative AI.

## REPRODUCIBILITY STATEMENT

The main paper describes the task definition, proposed method, benchmarks, baselines, backbone models, evaluation protocols, and overall experimental setup. Appendix C and D provide implementation and experimental details. The supplementary material includes the source code and configuration files needed to reproduce the reported results.

## REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RQm2KQTM5r.

Huan ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Qihan Ren, Yiran Wu, Hongru WANG, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=CTr3bovS5F. Survey Certification.

Qianshu Cai, Yonggang Zhang, Xianzhang Jia, Huajiang Zheng, Wei Xue, Jun Song, Xinmei Tian, and Yike Guo. Moss: Self-evolution through source-level rewriting in autonomous agent systems. *arXiv preprint arXiv:2605.22794*, 2026.

Tingyang Chen, Shuo Lu, Kang Zhao, Weicheng Meng, Hanlin Teng, Tianhao Li, Chao Li, Xule Liu, Jian Liang, Zhizhong Zhang, et al. Harnessx: A composable, adaptive, and evolvable agent harness foundry. *arXiv preprint arXiv:2606.14249*, 2026.

DeepSeek. DeepSeek V4 Preview Release. https://api-docs.deepseek.com/news/news260424, 2026. Accessed: 2026-07.

Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=9ZxnPZGmPU.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ZG3RaNIsO8.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=t9U3LW7JVX.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. Understanding the planning of llm agents: A survey. *arXiv preprint arXiv:2402.02716*, 2024.

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. *arXiv preprint arXiv:2607.14159*, 2026.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 54928–54958, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/f1cf02ce09757f57c3b93c0db83181e0-Paper-Conference.pdf.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Yu Li, Lehui Li, Zhihao Wu, Qingmin Liao, Jianye Hao, Kun Shao, and Fengli Xu. Agentswift: Efficient llm agent design via value-guided hierarchical search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 31843–31851, 2026.

Zelong Li, Shuyuan Xu, Kai Mei, Wenyue Hua, Balaji Rama, Om Raheja, Hao Wang, He Zhu, and Yongfeng Zhang. Autoflow: Automated workflow generation for large language model agents. *arXiv preprint arXiv:2407.12821*, 2024.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, et al. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026a.

Minhua Lin, Hanqing Lu, Zhan Shi, Bing He, Rui Mao, Zhiwei Zhang, Zongyu Wu, Xianfeng Tang, Hui Liu, Zhenwei Dai, et al. Position: Agentic evolution is the path to evolving llms. *arXiv preprint arXiv:2602.00359*, 2026b.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024.

Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin P Murphy. Autoharness: improving llm agents by automatically synthesizing a code harness. *arXiv preprint arXiv:2603.03329*, 2026.

Maksim Madžar and Igor Mekterović. Automated design of agentic systems: A survey of algorithms for searching, optimizing, and evolving llm agents, workflows, and prompts. *Preprints*, June 2026. doi: 10.20944/preprints202606.0238.v1. URL https://doi.org/10.20944/preprints202606.0238.v1.

Jun Nie, Yonggang Zhang, Jun Song, Qianshu Cai, Dahai Yu, Yike Guo, Xinmei Tian, and Bo Han. Tthe: Test-time harness evolution. *arXiv preprint arXiv:2607.08124*, 2026.

Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, Zhining Liu, Ting-Wei Li, et al. Code as agent harness. *arXiv preprint arXiv:2605.18747*, 2026.

OpenAI. GPT-5.4 Model. https://developers.openai.com/api/docs/models/gpt-5.4, 2026. Accessed: 2026-07.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. Adapt: As-needed decomposition and planning with language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 4226–4252, 2024.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 7957–7968, 2023.

Jon Saad-Falcon, Adrian Gamarra Lafuente, Shlok Natarajan, Nahum Maru, Hristo Todorov, Etash Guha, E Kelly Buchanan, Mayee Chen, Neel Guha, Christopher Ré, et al. Archon: An architecture search framework for inference-time techniques. *arXiv preprint arXiv:2409.15254*, 2024.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic llm agent search in modular design space. In *International Conference on Learning Representations*, volume 2025, pp. 3841–3865, 2025.

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

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Tencent Hy. Introducing Hy3. `https://hy.tencent.com/research/hy3`, 2026. Accessed: 2026-07.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024a.

Ruhan Wang, Yucheng Shi, Zongxia Li, Zhongzhi Li, Yue Yu, Junyao Yang, Kishan Panaganti, Haitao Mi, Dongruo Zhou, et al. Harness handbook: Making evolving agent harnesses readable, navigable, and editable. *arXiv preprint arXiv:2607.13285*, 2026a.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, 2022.

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`.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric Xing, and Zhiting Hu. Promptagent: Strategic planning with language models enables expert-level prompt optimization. In *International Conference on Learning Representations*, volume 2024, pp. 23967–24001, 2024b.

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. *arXiv preprint arXiv:2607.12227*, 2026c.

Lilian Weng. Harness engineering for self-improvement. *lilianweng.github.io*, July 2026. URL `https://lilianweng.github.io/posts/2026-07-04-harness/`.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, volume 2024, pp. 12028–12068, 2024.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, et al. Skillopt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. $\tau$-bench: A benchmark for tool-agent-user interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*, 2025.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel agent: A self-referential agent framework for recursively self-improvement. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27890–27913, 2025.

Ling Yue, Kushal Raj Bhandari, Ching-Yun Ko, Dhaval Patel, Shuxin Lin, Nianjun Zhou, Jianxi Gao, Pin-Yu Chen, and Shaowu Pan. From static templates to dynamic runtime graphs: a survey of workflow optimization for llm agents. *arXiv preprint arXiv:2603.22386*, 2026.

Z.ai. GLM-5.2 Model. `https://docs.bigmodel.cn/cn/guide/models/text/glm-5.2`, 2026. Accessed: 2026-07.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In *International Conference on Machine Learning*, pp. 75834–75852. PMLR, 2025a.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 104223–104294, 2026a. URL `https://proceedings.iclr.cc/paper_files/paper/2026/file/aa5f5e6eb6f613ec412f1d948dfa21a5-Paper-Conference.pdf`.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents. *arXiv preprint arXiv:2603.19461*, 2026b.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025b.

Jiayi Zhang, Yongfeng Gu, Jianhao Ruan, Maojia Song, Yiran Peng, Zhiguang Han, Jinyu Xiang, Zhitao Wang, Caiyin Yang, Yixi Ouyang, et al. Harnessing agentic evolution. *arXiv preprint arXiv:2605.13821*, 2026c.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, 2026d.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

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

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*, 2024.

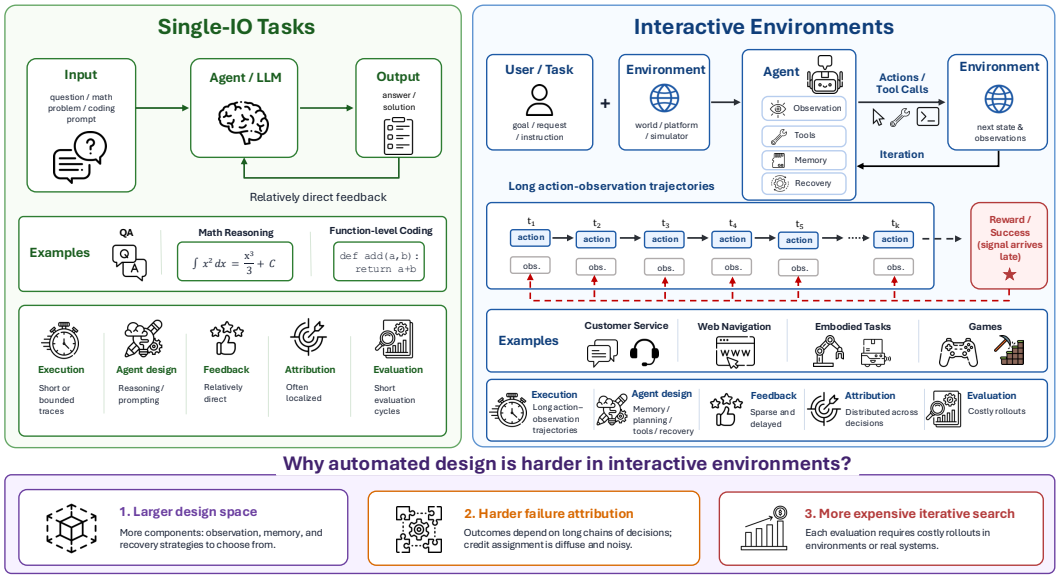



Figure 5: Task comparison between single-IO tasks and long-horizon interactive environments.

## A Interactive Agent

Interactive environments make automated agent design substantially more difficult in three ways.

First, interactive environments induce a larger and more task-dependent design space than single-input-output tasks.. Successful agents may require different combinations of observation processing, memory, planning, tool use, verification, and recovery mechanisms (Huang et al., 2024; Wang et al., 2024a). Full-code design provides the necessary expressiveness, but must search over a much larger and less structured space of executable implementations (Hu et al., 2025).

Second, interaction feedback is sparse, delayed, and difficult to attribute. Interactive agents may make many dependent decisions over multiple environment states before a final success or failure signal is observed (Liu et al., 2024; Yao et al., 2025). A scalar score reveals *whether* an agent succeeded, but rarely explains *why*: the failure may originate from an early misunderstanding, lost state, incorrect tool use, a policy violation, or ineffective recovery. Simply providing complete trajectories to an LLM is also problematic, as long and heavy traces contain substantial irrelevant context and make decisive failure evidence difficult to isolate. Effective design search therefore requires precise, evidence-grounded failure attribution.

Third, interactive evaluation is expensive, which makes search efficiency critical. Each candidate must be assessed through environment rollouts that may require extensive model calls and interaction steps (Yao et al., 2022; 2025). Existing full-code design and self-improvement methods commonly retain and select evaluated agent variants based primarily on empirical task scores (Hu et al., 2025; Zhang et al., 2026a;b). Without a persistent view of search progress, however, the designer may revisit unsuccessful modifications, overlook the recurrence of previously mitigated failures, and repeatedly spend evaluation budget on already explored directions. Efficient optimization therefore requires tracking what has been tried, what has improved or regressed, and which unresolved bottleneck should guide the next design round.

## B Detailed Related Work

Recent work has increasingly shifted from manually designing agentic systems toward automatically optimizing them from execution feedback. Automated Design of Agentic Systems (ADAS) formalizes this process as searching over agentic system designs under an evaluation objective (Madžar & Mekterović, 2026). Related surveys study this broader direction through workflow optimization, self-evolving agents, and adaptive agent harnesses (ang Gao et al., 2026; Yue et al., 2026; Ning et al.,

2026). We organize prior work into three levels according to the expressiveness of the optimization target: prompt (text)-level, architecture-level, and full-code-level design.

**Prompt-level Design.** Prompt-level methods optimize textual artifacts based on fixed models and an agent execution process. Early automatic prompt optimization methods search directly over natural-language instructions, using LLM-generated candidates (Zhou et al., 2023), scored optimization histories (Yang et al., 2024), textual gradients (Pryzant et al., 2023), evolutionary search (Guo et al., 2024; Fernando et al., 2024), or Monte Carlo tree search (Wang et al., 2024b). More recent methods exploit richer execution feedback: GEPA evolves prompts through trajectory reflection and Pareto-based selection, while DSPy optimizes instructions within modular language-model pipelines (Agrawal et al., 2026; Khattab et al., 2024). ACE and SkillOpt further treat persistent context playbooks or skill documents as editable textual state, accumulating reusable procedures, heuristics, and failure-handling knowledge from experience (Zhang et al., 2026d; Yang et al., 2026).

**Architecture-level Design.** Architecture-level methods design the agent's overall structure, while remaining constrained by a predefined graph, module set, or harness abstraction. Existing approaches broadly follow two directions. The first direction directly searches over agent architectures or workflows, including AutoFlow, AFlow, and GPTSwarm (Li et al., 2024; Zhang et al., 2025b; Zhuge et al., 2024). Other methods compose predefined reasoning, planning, tool-use, memory, and inference-time components, as in AgentSquare, Archon, MaAS, and AgentSwift (Shang et al., 2025; Saad-Falcon et al., 2024; Zhang et al., 2025a; Li et al., 2026). More recent systems expose runtime harnesses themselves as structured, editable artifacts: A-Evolve maintains versioned knowledge, workflow, tool, and validation assets; AHE edits explicit harness components using trajectory observability; and HarnessX evolves compositions of typed harness primitives (Lin et al., 2026b;a; Chen et al., 2026).

**Full-code-level Design.** At the most expressive level, agents are represented as executable programs and optimized directly in code space. Hu et al. (2025) introduced this direction by using a fixed meta-agent to iteratively propose new task agent implementations and retain successful variants in an archive. DGM combines self-modification with open-ended evolutionary exploration over a branching archive (Zhang et al., 2026a), while HGM further improves parent selection using estimated descendant potential (Wang et al., 2026b). In addition, Meta-Harness searches harness code using selective access to previous code, scores, and traces (Lee et al., 2026); AutoHarness synthesizes executable harnesses from environment feedback (Lou et al., 2026); MOSS performs source-level rewriting from curated failures (Cai et al., 2026); and TTHE evolves harnesses at test time from unlabeled trajectories (Nie et al., 2026). A further line of work makes parts of the improvement mechanism itself editable. Gödel Agent supports self-referential modification of task behavior and improvement logic; DGM-H jointly evolves the task agent and meta-level modification mechanism; and AEvo revises the procedure or context that supports subsequent evolution (Yin et al., 2025; Zhang et al., 2026b;c). These methods extend automated design from optimizing an agent artifact to optimizing how future agent modifications are designed.

# C ADIAS Details

Algorithm 1 summarizes the complete procedure.

## C.1 External Prior Search

Full-code agent design exposes an extremely large space of possible implementations. However, the task protocol and public task-level knowledge often provide useful priors about likely interaction requirements and failure modes. ADIAS uses a one-shot external searcher to exploit this information before iterative design begins.

Given the task protocol $\mathcal{P}$, the external searcher retrieves public information such as benchmark descriptions, rule explanations, common failure modes, tool-use strategies, and candidate agent mechanisms. The resulting external prior $P$ is incorporated into the initial issue state as *unverified task prior knowledge*. It is explicitly separated from empirical experience and is treated only as a set of design hypotheses. This distinction is important because retrieved information may be incomplete or unsuitable for the current agent. A prior is promoted into accumulated task experience only when supported by trajectory diagnosis, patch outcomes, runtime evidence, or score trends. The exter-

**Algorithm 1** Automated Design of Interactive Agentic Systems

**Require:** Task protocol $\mathcal{P}$, initial agent $A_0$, optimization round $T$

$P \leftarrow \text{ExternalSearch}(\mathcal{P})$
$\mathcal{T}_0, M_0 \leftarrow \text{Evaluate}(A_0)$
$E_0 \leftarrow \text{InitializeIssueState}(P)$
$H_0 \leftarrow \text{InitializeHistory}(A_0, \mathcal{T}_0, M_0)$
**for** $t = 1, \ldots, T$ **do**
  $D_t \leftarrow \text{Diagnose}(\mathcal{T}_{t-1}, M_{t-1})$
  $H_t \leftarrow \text{UpdateHistory}(H_{t-1}, A_{t-1}, \mathcal{T}_{t-1}, M_{t-1}, D_t)$
  $E_t \leftarrow \text{UpdateIssueState}(E_{t-1}, H_t)$
  $A_{p_t}, R_t \leftarrow \text{Plan}(H_t, E_t)$
  $A_t \leftarrow \text{Modify}(A_{p_t}, R_t)$
  $\mathcal{T}_t, M_t \leftarrow \text{Evaluate}(A_t)$
**end for**
**return** $A^\star = \arg\max_{A_i \in \mathcal{A}_T} M_{val}(A_i)$

nal searcher is executed exactly once and is not changed during late design iterations. We restrict retrieval to public task-level information and exclude held-out labels, instance-specific solutions, hidden evaluation data, and hard-coded benchmark outputs. Overall, external search narrows the effective cold-start search space without restricting the expressiveness of the underlying full-code design space.

### C.2 Evidence-Grounded Failure Diagnosis

A scalar evaluation score reveals whether an agent succeeds, but rarely identifies which design deficiency caused a failure. This attribution problem is difficult in interactive tasks because a single episode may contain many state transitions, tool calls, and dependent decisions. Directly placing all trajectories into the context of a designer model also introduces substantial irrelevant information.

ADIAS therefore introduces a dedicated diagnostic agent that analyzes all training trajectories collected in one optimization round. Rather than consuming complete trajectories as a flat prompt, the diagnostic agent interacts with a structured trajectory database through targeted query tools. It can inspect aggregate failure patterns, compare successful and failed episodes, retrieve compact action sequences or local step windows, search for repeated and invalid actions, and examine tool errors, visible final states, etc. The diagnostic agent uses these queries to iteratively examine failures based on trajectory evidence. The output is a diagnostic report

$$D_t = (I_t, S_t, C_t), \tag{4}$$

where $I_t$ denotes observed issue categories, $S_t$ contains supporting evidence, and $C_t$ describes the inferred causes. The objective is not merely to locate a failed action, but to identify recurring behavioral patterns that indicate deficiencies in the current agent design. Diagnosis is deliberately separated from modification, which means the diagnostic agent does not propose suggestions or implement code patches.

### C.3 Persistent Issue Control

This section describes how ADIAS operationalizes that state for full-code agent design. In our implementation, $E_t$ is maintained as a structured profile containing a global optimization summary and a set of issue records. Each issue record instantiates the fields in Section 3.3.1 with a concise issue description, priority, lifecycle status, evidence references, previously attempted interventions, observed outcomes, and the candidate generations in which the issue was observed or absent. The candidate archive remains available separately and stores complete programs, trajectories, scores, and parent relations. Thus, $E_t$ does not replace the archive; it provides an issue-indexed control layer over candidate-level history.

After each evaluation, the Issue Manager compares the diagnostic report with the existing issue records. Observations are associated with an existing issue when they describe the same underlying

Table 4: Summary of the five interactive benchmarks used in our experiments.

| Benchmark | Domain | Capability | #Train | #Val | #Test |
|---|---|---|---|---|---|
| $\tau$-Bench-Return | Customer Service | Policy-aware tool use and user interaction | 55 | 15 | 32 |
| $\tau$-Bench Retail | Customer Service | Policy-aware tool use and user interaction | 500 | 20 | 115 |
| ALFWorld | Embodied AI | Long-horizon planning and state tracking | 153 | 15 | 134 |
| TextCraft | Game | Hierarchical decomposition and dependency planning | 60 | 15 | 100 |
| WebShop | Web | Information seeking and constraint-aware navigation | 84 | 25 | 50 |
| ScienceWorld | Scientific Scenes | Grounded procedural and scientific reasoning | 66 | 22 | 102 |

behavioral failure, affected capability, and execution context, even if their surface descriptions differ across trajectories. Otherwise, a new stable issue identity is created. Supporting evidence stores references to the relevant trajectory segments and diagnostic findings rather than only an aggregate score change. External task priors are handled as provisional hypotheses: they may initialize issue records, but do not become confirmed failures unless supported by observed behavior.

The persistent issue state controls which problem should be addressed and where optimization should continue, but it does not prescribe a fixed implementation. Conditioned on the repair plan, the designer may add, remove, or rewrite any agent-side logic, such as prompts, tools, memory, and task-specific modules. After the revised candidate is evaluated, its diagnostic evidence and measured outcomes are written back to the corresponding issue records, forming an issue–intervention–outcome chain for the next round. This separation preserves the expressiveness of full-code optimization while supporting continuous repair across candidate generations.

# D Experimental Details

## D.1 Benchmark Details

To control computational costs, we construct representative task subsets for the main experiments. We preserve the native task structure and evaluator of each benchmark, and select subsets according to task semantics or trajectory complexity rather than sampling arbitrary individual episodes.

**$\tau$-Bench.** $\tau$-Bench evaluates agents in tool–agent–user interactions, requiring them to communicate with a simulated user, comply with domain-specific policies, and invoke tools that modify an underlying database state (Yao et al., 2025). We use the retail domain at two evaluation scales.

For the main experiments, we use all qualified return-related tasks, comprising 55 training, 15 validation, and 32 test examples. Return requests form a coherent but nontrivial intent family: agents must identify the relevant order and items, verify policy constraints, collect missing user information, and execute the appropriate tool sequence. We adopt this setting primarily for computational efficiency. Restricting the optimization set to one complete intent family enables controlled comparison under a feasible and uniform budget.

To verify that the conclusions are not specific to return requests, we additionally repeat the comparison on the full $\tau$-Bench retail distribution, containing 500 training, 20 validation, and 115 test tasks. This expanded setting covers heterogeneous customer-service requests beyond returns and substantially increases the cost and diversity of optimization. The purpose is to show that the return subset provides a cost-efficient proxy for controlled comparison rather than an artificially favorable evaluation setting.

**ALFWorld.** ALFWorld uses text-based environments to simulate embodied household tasks and evaluates instruction understanding, planning, navigation, and object interaction (Shridhar et al., 2021). For example, the *Clean-and-Place* task family requires an agent to locate a target object, identify and navigate to the required locations, clean the object, and place it in a specified place. Successful execution therefore requires long-horizon planning, environment state tracking, and recovery from incorrect intermediate actions. We randomly sample 153 training and 15 validation examples from the *Clean-and-Place* task family, together with 30 held-out *Clean-and-Place* examples for in-distribution testing. To evaluate whether the optimized agent generalizes beyond the task family used during optimization, we additionally evaluate it on the official ALFWorld test set

containing 134 examples across all task families. The results are based on the official test set by default.

**TextCraft.** TextCraft is a text-based gaming environment designed to evaluate compositional planning over Minecraft-inspired crafting recipes (Prasad et al., 2024). Given a target item and a set of available crafting operations, the agent must reason recursively about prerequisite materials and execute a valid sequence of gathering and crafting actions. Recipe depth naturally controls task complexity, with deeper recipes requiring longer chains of dependent subgoals. To obtain a complexity-balanced subset, we stratify examples by recipe depth and sample tasks from depths 2, 3, and 4. Our final split contains 60 training and 15 validation examples, where examples are evenly distributed across the three recipe depths. We use the official test set with 100 samples.

**WebShop.** WebShop is a simulated e-commerce environment in which an agent must satisfy a natural-language shopping instruction by searching products, navigating result and product pages, inspecting attributes, and selecting an appropriate item for purchase (Yao et al., 2022). The environment evaluates information seeking, constraint tracking, long-horizon web navigation, and comparison among multiple candidate products. To emphasize genuinely interactive tasks, we construct a subset from successful human trajectories and retain instances requiring more than 10 non-redundant interaction steps. This filtering removes short or trivial purchases and focuses evaluation on tasks involving sustained navigation and decision-making. The resulting subset contains 84 training, 25 validation, and 50 test instances.

**ScienceWorld.** ScienceWorld evaluates scientific reasoning in an interactive text environment grounded in elementary-school science concepts (Wang et al., 2022). Agents must understand a scientific objective and carry out a valid sequence of observations, navigation actions, object manipulations, and experimental procedures. The benchmark tests procedural reasoning, state tracking, and the ability to translate scientific knowledge into executable actions. To reduce experimental cost, we first retain ScienceWorld instances whose successful reference trajectories can be completed within 30 interaction steps, and obtain 22 task types containing more than five eligible instances. For each retained task type, we sample three instances for training, one for validation, and up to five for testing. If fewer than five eligible test instances remain for a task type, we include all available instances without resampling or duplication. The resulting dataset contains 66 training, 22 validation, and 102 test examples.

### D.2 Baselines

**Handcrafted.** We use a manually designed ReAct-style agent (Yao et al., 2023) with task-level memory as the fixed, non-automated baseline. The agent follows a standard reasoning-and-acting loop and retains relevant interaction history across steps, but its strategy and execution structure are specified by humans and remain unchanged throughout evaluation.

**SkillOpt.** SkillOpt (Yang et al., 2026) performs text-level agent optimization by treating a persistent natural-language skill document as the editable state of a frozen task agent. A separate optimizer converts scored trajectories into bounded textual edits, and candidate updates are accepted through held-out validation evaluations. The task model and execution harness remain fixed, so the search space is restricted to the agent's textual strategy representation.

**AHE.** Agentic Harness Engineering (AHE) (Lin et al., 2026a) performs architecture-level optimization over the agent harness. It exposes multiple editable components, including prompts, tools, middleware, skills, memory, and other harness modules, and iteratively modifies these components. Compared with SkillOpt, AHE searches over a broader agent architecture and harness.

**Meta-Harness.** Meta-Harness (Lee et al., 2026) performs end-to-end optimization over the entire model harness. It uses an agentic proposer with filesystem access to the source code, evaluation scores, and execution traces of all previous candidates, which allows arbitrary code-level revisions and organizes cross-round experience as a raw candidate archive.

**DGM-H.** DGM-Hyperagents (DGM-H) (Zhang et al., 2026b) represents the broadest full-code-level automated design baseline. It integrates the task agent and meta agent into a single editable program and allows the resulting hyperagent to modify both its task-solving logic and the mechanism used to generate future improvements. The method searches over arbitrary program logic, memory, and control flow, which provides a substantially less constrained code-level design space.

Table 5: Comparison of baseline design scope and cross-round experience organization.

| Method | Design Level | Editable Scope | Cross-Round Experience |
|---|---|---|---|
| Handcrafted | Fixed | None | None |
| SkillOpt (Yang et al., 2026) | Prompt | Skill document | Accepted/rejected edits and feedback |
| AHE (Lin et al., 2026a) | Harness | Pre-defined components | Records of edits, predictions, and outcomes |
| Meta-Harness (Lee et al., 2026) | Full code | Full agent program | Full candidate archive / raw history |
| DGM-H (Zhang et al., 2026b) | Full code | Full agent program | Full candidate archive / raw history |
| ADIAS (Ours) | Full code | Full agent program | Full candidate archive & Persistent issue state |

### D.3 Evaluation Protocol

We evaluate each method on held-out test tasks after selecting the best agent according to validation performance. All methods use the same task splits, environment interfaces, evaluation budgets, and benchmark-specific evaluators.

**Task Performance.** We retain the native evaluation signal of each benchmark and define the reported task score as

$$\text{Score} = \begin{cases} \dfrac{1}{N}\displaystyle\sum_{i=1}^{N} \mathbb{I}[y_i = 1], & \tau\text{-Bench, ALFWorld, TextCraft,} \\ \dfrac{1}{N}\displaystyle\sum_{i=1}^{N} r_i, & \text{WebShop, ScienceWorld,} \end{cases} \tag{5}$$

where $N$ denotes the number of evaluation tasks, $y_i \in \{0, 1\}$ is the binary task-success indicator, and $r_i \in [0, 1]$ is the normalized benchmark-specific task score. For $\tau$-Bench, ALFWorld, and TextCraft, task performance is measured by the success rate. For WebShop and ScienceWorld, we report the average native task score, which provides graded credit for partial task progress. The native reward of WebShop measures how well the purchased product satisfies the attributes and constraints specified in the shopping instruction (Yao et al., 2022). The score of ScienceWorld captures the degree of progress toward completing the required scientific procedure (Wang et al., 2022). For presentation, all reported scores are multiplied by 100. We report Pass 1 results by default.

**Task Interaction Efficiency.** We measure how effectively a task agent converts environment interactions into task performance. Let $\bar{L}$ denote the average number of environment interaction steps and $S$ denote the benchmark task score defined above. We define interaction efficiency as

$$\text{Eff.} = \frac{Score}{\bar{L}}. \tag{6}$$

This performance-per-interaction ratio jointly accounts for task effectiveness and environment interaction cost. We compare efficiency only within the same benchmark, since the semantics and granularity of an environment step differ across domains.

**Optimization Efficiency.** Beyond the performance of the final agent, we evaluate how rapidly an automated design method discovers stronger agents under a fixed optimization budget. Let $V_t$ denote the validation score of the agent generated at optimization iteration $t$. We track the best-so-far validation score

$$B_t = \max_{1 \le j \le t} V_j, \tag{7}$$

which measures the quality of the agent available after $t$ optimization iterations. A method with a rapidly increasing $B_t$ discovers high-performing designs with fewer search iterations and is therefore more optimization-efficient.

## E More results

We further examine robustness to execution randomness, optimization variability, finite test-set sampling, and broader task distributions.

Table 6: Multiple runs of evaluation on $\tau$-Bench-Return. Pass@2 counts a task as successful if either rollout succeeds, whereas Pass$^2$ requires both rollouts to succeed. Best and second-best results are highlighted in green and blue, respectively.

| Method | Pass 1 | Pass 2 | Pass@2 | Pass$^2$ | Avg. $\pm$ Std. |
|---|---|---|---|---|---|
| Handcrafted | 75.0 | 68.8 | 87.5 | 56.3 | $71.9 \pm 4.4$ |
| SkillOpt | 12.5 | 6.3 | 15.6 | 3.1 | $9.4 \pm 4.4$ |
| Meta-Harness | 43.8 | 43.8 | 53.1 | 34.4 | $43.8 \pm 0.0$ |
| AHE | 56.2 | 37.5 | 68.8 | 25.0 | $46.9 \pm 13.2$ |
| DGM-H | 59.4 | 43.8 | 65.6 | 37.5 | $51.6 \pm 11.0$ |
| ADIAS (Ours) | 81.3 | 81.3 | 90.6 | 71.9 | $81.3 \pm 0.0$ |
| *Ablations* | | | | | |
| w/o External Prior | 75.0 | 75.0 | 90.6 | 59.4 | $75.0 \pm 0.0$ |
| w/o Round-Level Diagnosis | 78.1 | 71.9 | 87.5 | 62.5 | $75.0 \pm 4.4$ |
| w/ Archive-Wide Synthesis | 65.6 | 62.6 | 78.1 | 50.0 | $64.1 \pm 2.1$ |
| w/ Best-Candidate Revision | 71.9 | 71.9 | 81.3 | 62.5 | $71.9 \pm 0.0$ |
| w/ Latest-Candidate Continuation | 62.5 | 62.5 | 75.0 | 50.0 | $62.5 \pm 0.0$ |

### E.1 Evaluation Robustness

We examine whether the performance of a fixed selected agent is robust to execution randomness by repeating its evaluation on the same held-out test instances. The main experiments report the result of the first evaluation pass for a consistent single-rollout comparison across benchmarks. We conduct a second independent rollout on $\tau$-Bench to examine whether the reported performance is robust to execution randomness. Both passes use the same agents, task instances, and evaluation protocol.

For task $i$, let $y_i^{(j)} \in \{0, 1\}$ denote whether the $j$-th rollout succeeds. We report two complementary repeated-execution metrics. Pass@$k$ measures whether a task succeeds in at least one of the $k$ rollouts:

$$\text{Pass@}k = \frac{1}{N} \sum_{i=1}^{N} \left( 1 - \prod_{j=1}^{k} \left( 1 - y_i^{(j)} \right) \right). \tag{8}$$

In contrast, Pass$^k$ measures consistent success across all $k$ rollouts (Yao et al., 2025):

$$\text{Pass}^k = \frac{1}{N} \sum_{i=1}^{N} \prod_{j=1}^{k} y_i^{(j)}. \tag{9}$$

Pass@$k$ therefore captures whether the agent can solve a task in multiple runs, whereas Pass$^k$ provides a stricter measure of whether it solves the task reliably. We additionally report the arithmetic mean of the two individual pass scores. The same score of two passes merely indicates that the success rates of the two aggregations are the same; it does not imply that the two aggregations resolved exactly the same set of tasks. The consistency of the specific tasks is more accurately reflected by Pass$^k$.

Overall, the repeated-evaluation results preserve the conclusions of the main experiments and show that ADIAS illustrates robust execution performance rather than a favorable outcome from a single rollout. As shown in Table 6, ADIAS obtains identical success rates of 81.3 across the two independent evaluation passes, exceeding the strongest baseline average of 71.9 by 9.4 points. More importantly, ADIAS achieves a Pass$^2$ score of 71.9, outperforming the strongest non-ADIAS baseline by 15.6 points. This indicates that its advantage is not driven solely by tasks that succeed under favorable sampling; a substantially larger fraction of tasks are solved consistently across both executions. The ablations further support the importance of the proposed design. Removing either the external prior or round-level diagnosis reduces the two-pass average to 75.0 and lowers Pass$^2$ to 59.4 and 62.5, respectively. Replacing the persistent issue-centric state with candidate-based revision policies produces larger reductions in both average performance and consistent success.

Table 7: Optimization-run robustness on ALFWorld across three independent complete optimization runs. Avg. ± Std. reports the mean and standard deviation of test scores across runs.

| Method | Avg. ± Std. ↑ | Pass@3 ↑ | Pass$^3$ ↑ |
|---|---|---|---|
| SkillOpt | 0.0 ± 0.0 | 0.0 | 0.0 |
| Meta-Harness | 54.7 ± 19.2 | 89.6 | 23.1 |
| AHE | 81.1 ± 2.4 | 88.8 | 68.7 |
| DGM-H | 71.6 ± 3.3 | 93.3 | 46.3 |
| ADIAS (Ours) | **93.3 ± 1.3** | **98.5** | **86.6** |

Table 8: Generalization from the *Clean-and-Place* task family to the broader ALFWorld task distribution. All methods are optimized only on *Clean-and-Place* training and validation examples.

| | Clean-and-Place Test | | Official Test Set | |
|---|---|---|---|---|
| Method | Score ↑ | Eff. ↑ | Score ↑ | Eff. ↑ |
| Handcrafted | 76.7 | 4.26 | 70.9 | 3.60 |
| SkillOpt | 5.0 | 0.17 | 0.0 | 0.00 |
| Meta-Harness | 83.3 | 6.08 | 59.7 | 3.32 |
| AHE | 60.0 | 3.49 | 78.4 | 4.48 |
| DGM-H | 83.3 | 5.05 | 69.4 | 3.42 |
| ADIAS (Ours) | **86.7** | **8.03** | **94.0** | **8.95** |

### E.2 Optimization-Run Robustness

We examine whether the complete optimization procedure consistently discovers strong agent designs by repeating the full optimization process in three independent runs on ALFWorld. Figure 3 reports the mean best-so-far validation trajectories, while Table 7 summarizes the held-out test performance of the three independently optimized agents. Avg. ± Std. measures variation in overall test performance across optimization runs. ADIAS achieves both the highest average test performance and the lowest variability among the competitive methods, reaching 93.3 ± 1.3 across three independent optimization runs. It also attains 98.5% Pass@3 and 86.6% Pass$^3$, which shows that its optimization procedure not only discovers agents with broad task coverage, but also reproduces successful behavior on most test tasks across independently optimized agents.

### E.3 Task-Family Generalization

We examine whether agents optimized on a single ALFWorld task family generalize to a broader task distribution. All methods are optimized using 153 training and 15 validation examples from *Clean-and-Place*. We evaluate the resulting agents on two test sets: 30 held-out *Clean-and-Place* examples for in-distribution evaluation, and the official ALFWorld test set of 134 examples spanning multiple task families for cross-task evaluation.

ADIAS achieves the highest success rate on both the held-out *Clean-and-Place* set and the broader official test set, while also requiring the fewest interaction steps. Notably, although optimization is performed exclusively on *Clean-and-Place*, ADIAS reaches 94.0% success on the official multi-task test distribution, compared with 78.4% for the strongest baseline. These results indicate that the optimized harness does not merely specialize to the task family seen during optimization, but transfers effectively to other ALFWorld task types.

### E.4 Expanded-Distribution Evaluation

We examine whether the observed advantage persists when optimization and evaluation are expanded from the main task subset to a broader task distribution. The main experiments use the return-intent subset of $\tau$-Bench to reduce the substantial costs. To verify that the observed gains are not specific to this task family, we further optimize and evaluate all methods on the complete $\tau$-Bench retail distribution, which contains 500 training, 20 validation, and 115 test tasks spanning

Table 9: Results on the complete Tau-Bench retail distribution. Pass@2 counts a task as successful if either evaluation rollout succeeds, whereas $\text{Pass}^2$ requires both rollouts to succeed. Best and second-best results are highlighted in green and blue, respectively.

| Method | Pass 1 ↑ | Eff. ↑ | Pass 2 ↑ | Pass@2 ↑ | $\text{Pass}^2$ ↑ | Avg. ± Std. ↑ |
|---|---|---|---|---|---|---|
| Handcrafted | 68.7 | 5.13 | 67.0 | 81.7 | 53.9 | $67.8 \pm 1.2$ |
| SkillOpt | 6.1 | 0.23 | 5.2 | 7.8 | 3.5 | $5.7 \pm 0.6$ |
| Meta-Harness | 58.1 | 4.12 | 58.1 | 69.5 | 46.7 | $58.1 \pm 0.0$ |
| AHE | 67.8 | 4.13 | 64.3 | 80.9 | 51.3 | $66.1 \pm 2.5$ |
| DGM-H | 63.5 | 3.94 | 59.1 | 77.4 | 45.2 | $61.3 \pm 3.1$ |
| ADIAS (Ours) | 81.7 | 6.24 | 79.1 | 92.2 | 68.7 | $80.4 \pm 1.8$ |

Table 10: Test-set uncertainty across three benchmarks. Each cell reports the task score followed by its 95% confidence interval. The strongest baseline is Handcrafted for Tau-Bench Retail, AHE for ALFWorld, and DGM-H for TextCraft. The difference row reports ADIAS minus the strongest baseline with paired 95% confidence intervals.

| Method | Tau-Bench Retail | ALFWorld | TextCraft |
|---|---|---|---|
| Strongest Baseline | 68.70 [59.73, 76.45] | 78.36 [70.65, 84.49] | 76.00 [66.77, 83.31] |
| ADIAS (Ours) | **81.74 [73.69, 87.73]** | **94.03 [88.66, 96.94]** | **91.00 [83.77, 95.19]** |
| $\Delta$ (Ours − Baseline) | **+13.04 [4.35, 21.74]** | **+15.67 [8.96, 23.13]** | **+15.00 [7.00, 24.00]** |

a broader range of customer-service requests and interaction patterns. Under this expanded setting, each method is optimized for 15 iterations using 20 training episodes per iteration.

As shown in Table 9, ADIAS remains the strongest method when optimization is scaled to the complete retail distribution. It achieves a Pass-1 success rate of 81.7, outperforming the strongest baseline by 13.0 percentage points. The improvement is not obtained at the expense of interaction efficiency: ADIAS reaches an efficiency of 6.24, compared with 5.13 for the strongest baseline.

The advantage also persists across repeated executions. ADIAS obtains an average success rate of $80.4 \pm 1.8$, exceeding the strongest baseline average of $67.8 \pm 1.2$ by 12.6 points. Its $\text{Pass}^2$ score reaches 68.7, outperforming the best baseline by 14.8 points, while its Pass@2 score of 92.2 indicates broad task coverage across the two executions. These results show that ADIAS improves both task performance and execution consistency on the broader retail distribution.

### E.5 Test-Set Uncertainty

The main results report the score estimates on finite held-out test sets. We additionally quantify uncertainty due to test-set sampling using nonparametric task-level bootstrap resampling. For each benchmark, we compare ADIAS with the strongest competing method in Table 10. We resample test instances with replacement 10,000 times and use identical sampled indices for both methods. We report 95% confidence intervals for each method's task performance and for their paired performance difference. Across three evaluated benchmarks, the paired 95% confidence intervals remain entirely above zero, indicating that ADIAS's advantage over the strongest competing method is robust to finite test-set sampling.

## F Additional Analysis and Findings

### F.1 Trade-off between Expressiveness and Searchability

The automated agent design baselines exhibit different behaviors as the editable agent-design space expands from text/prompt to harness/architecture components and full code level.

SkillOpt operates at the text level by optimizing a persistent skill document while leaving the agent structure fixed. This restricted design space enables controlled updates, but prevents the optimizer from introducing new modules when the task requires capabilities absent from the initial agent. For example, on ALFWorld, effective long-horizon interaction requires persistent state tracking and

Table 11: Meta-level token consumption and held-out test performance on Tau-Bench and ALFWorld. Meta tokens include LLM calls used by the optimization procedure. All token values are in millions.

| | $\tau$-Bench-Return | | ALFWorld | |
|---|---|---|---|---|
| Method | Meta Tokens ↓ | Score ↑ | Meta Tokens ↓ | Score ↑ |
| SkillOpt | 1.0 | 12.5 | 1.0 | 0.0 |
| Meta-Harness | 3.1 | 43.8 | 4.0 | 59.7 |
| AHE | 4.4 | 56.2 | 3.9 | 78.4 |
| DGM-H | 4.7 | 59.4 | 3.2 | 69.4 |
| ADIAS (Ours) | 8.1 | 81.3 | 6.3 | 94.0 |
| *Ablations* | | | | |
| w/o External Prior | 12.5 | 75.0 | 9.1 | 82.8 |
| w/o Round-Level Diagnosis | 2.9 | 78.1 | 4.5 | 63.4 |
| w/ Archive-Wide Synthesis | 7.7 | 65.6 | 5.6 | 60.4 |
| w/ Best-Candidate Revision | 13.1 | 71.9 | 10.7 | 75.4 |
| w/ Latest-Candidate Continuation | 8.2 | 62.5 | 12.6 | 41.0 |

memory mechanisms. As SkillOpt cannot alter the agent architecture to introduce such mechanisms, its performance is significantly lower than that of the other methods.

AHE expands the editable space to the agent harness and can modify multiple components, such as prompts, tools, skills, and memory. This broader scope enables useful improvements across different environments, and AHE achieves non-trivial gains on all five benchmarks. However, its search remains constrained by a predefined set of harness components and interfaces. As a result, the performance of AHE generally falls to the middle of the performance range.

DGM-H removes this structural restriction by allowing full-code-level self-modification. Its larger design space provides a higher performance ceiling, as illustrated by its strong result on ALFWorld. However, unrestricted code search introduces a substantially more difficult optimization problem. Under a limited evaluation budget, useful agent designs may be extremely sparse in the space of arbitrary program modifications. Thus, DGM-H fails to consistently outperform other baselines.

ADIAS resolves this trade-off between *expressiveness* and *searchability*. Similar to DGM-H, ADIAS retains a full-code design space and can introduce new agent mechanisms when required by the task. Unlike unconstrained code-level evolution, however, ADIAS narrows the effective search space with task priors that guide the designer toward mechanisms likely to be useful for the target environment. In addition, the *diagnostic agent* performs fine-grained failure analysis and precisely attributes observed errors to actionable causes. Finally, a global *Issue Manager* aggregates recurring patterns across episodes and iterations to prioritize broadly useful modifications and improve optimization efficiency. Overall, ADIAS preserves a high-capacity search space while using task priors to guide exploration, precise diagnosis to localize failures, and global issue state to drive efficient improvement.

### F.2 Optimization of the Optimization Process Itself

Because ADIAS optimizes the full agent harness, its editable space is not restricted to the task agent. Components involved in diagnosing failures and planning subsequent revisions may also be modified, which allows the optimization process itself to become part of the search space. We observe several such modifications during optimization. The Issue Manager is revised to normalize the revision plan into a fixed schema containing the target failure mode, selected parent generation, and revision rationale. The code improver is also revised to simplify how the target failure mode is extracted from the revision plan. These edits indicate that ADIAS can identify and revise interfaces within the optimization machinery, rather than modifying only task-execution behavior.

### F.3 Token Consumption

We report *meta-level token consumption* in Table 11 together with held-out test performance on $\tau$-Bench and ALFWorld. It measures the LLM calls used by the optimization procedure itself. We do not use total token consumption for direct comparison because task-rollout cost is also affected by the behavior of the generated agents, and therefore does not isolate the computational overhead of the optimization method. ADIAS uses more meta-level tokens than other automated baselines, which reflects the additional reasoning required for diagnosis and persistent state maintenance. However, the controlled variants indicate that the performance gains cannot be explained simply by greater meta-level model usage. On $\tau$-Bench, Archive-Wide Synthesis and Latest-Candidate Continuation consume nearly the same number of tokens as ADIAS (7.7M and 8.2M vs. 8.1M), yet their scores drop from 81.3 to 65.6 and 62.5, respectively. The pattern is even clearer on ALFWorld: Archive-Wide Synthesis uses 5.6M tokens but scores only 60.4, while Best-Candidate Revision and Latest-Candidate Continuation consume substantially more tokens than ADIAS (10.7M and 12.6M) while reaching only 75.4 and 41.0. Removing round-level diagnosis reduces meta-level cost, but also reduces performance.

### F.4 Limitations and Future Work

First, issue-centric optimization relies on sufficiently accurate trajectory diagnosis and issue consolidation. In this work, diagnosis is treated as an intermediate component rather than a prediction task; its effectiveness is therefore evaluated through downstream optimization performance and qualitative case analysis with human inspection. The strong gains across benchmarks and controlled ablations suggest that the resulting diagnoses are sufficiently informative for guiding optimization, although a large-scale annotation study of diagnosis accuracy is beyond the scope of this work. Future work could further improve diagnosis and issue association, or develop explicit benchmarks for evaluating them. Second, our evaluation focuses on text-based interactive environments. While the experiments cover multiple benchmarks, backbone models, repeated optimization runs, and broader task distributions, extending issue-centric optimization to substantially longer-horizon tasks and multimodal agent environments is a promising direction for future work.