BUDGET-AWARE COUNTERFACTUAL RECOVERY FOR LONG-HORIZON AI AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon AI agents act through learned policies, tools, and external environments, but the action proposed by a base agent can be unreliable and runtime self-correction is not free. An external reliability controller can challenge a suspect action by restoring the current state, executing alternative recoveries, and comparing their realized continuations before committing. Because these counterfactual checks consume model, tool, and environment resources, they cannot be applied indiscriminately. We study when an agent action is worth challenging under a finite fork-execution budget, while tracking replay and critic costs separately, and learn state-conditioned recovery value for budget-aware runtime intervention. In a fault-injected tool-agent sandbox with noisy diagnostics, learned allocation raises held-out episode success from 9.3% without runtime forks and 14.1% with equal allocation to 17.0% at a fork budget of 12. The same learned allocation idea transfers to Taxi-v4 without latent-state access: at budget 24 it improves deadline-constrained success over direct execution by 13.9, 10.1, and 6.5 percentage points at action-error rates 0.10, 0.20, and 0.30 while using fewer forks than a hand-designed reserve-aware policy; the 0.40 condition is inconclusive and reserve-aware retains higher success overall. On a pre-locked 24-task ALFWorld OOD set with Gemini 3 Flash, direct execution succeeds on 21/24 tasks and selective recovery policies on 22/24, with fewer interventions than fixed forking but no population-level superiority claim. We then freeze an agent-specific learned challenge model on the development set and evaluate it on a second untouched 24-task OOD set with exact repeated base-policy prompts shared across methods. Direct, reserve-aware, and learned challenge each succeed on 23/24 tasks; learned challenge uses 1.50 forks per task versus 2.50 for reserve-aware, but does not rescue the sole direct failure and incurs 16.75 replay steps versus 14.92. Recovery can also harm otherwise successful trajectories, and fewer forks need not imply lower total runtime cost when restoration requires replay. The evidence supports counterfactual challenge as a bounded inference-time reliability mechanism rather than an unlimited local repair operation.

1 INTRODUCTION

Long-horizon AI agents repeatedly propose actions that change an external state. Even a strong language model, reinforcement-learning policy, planner, or hybrid agent can produce a locally plausible action that is wrong for the current trajectory because of stale state, malformed tool use, failed handoffs, incorrect plans, or accumulated execution error. A runtime reliability system therefore faces a question that the base policy itself does not answer: should the proposed action be trusted, or is it worth spending additional inference and environment execution to challenge it before commitment?

Existing work supplies important pieces of such runtime self-correction. Prefix monitors estimate whether a partial trajectory is heading toward failure (Huang et al., 2026; Li, 2026); recovery orchestrators trigger retry or replanning actions under bounded resources (Babu & Agrawal, 2026); and state-aware runtimes provide checkpoint, rollback, and audit mechanisms (Chen, 2026; Dong et al., 2026; Wu et al., 2026). Branching agent training and counterfactual replay further show that executable environments can support sibling continuations from a shared prefix (He et al., 2026; Shah, 2026). Yet these capabilities do not determine when a runtime should actively challenge the

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.

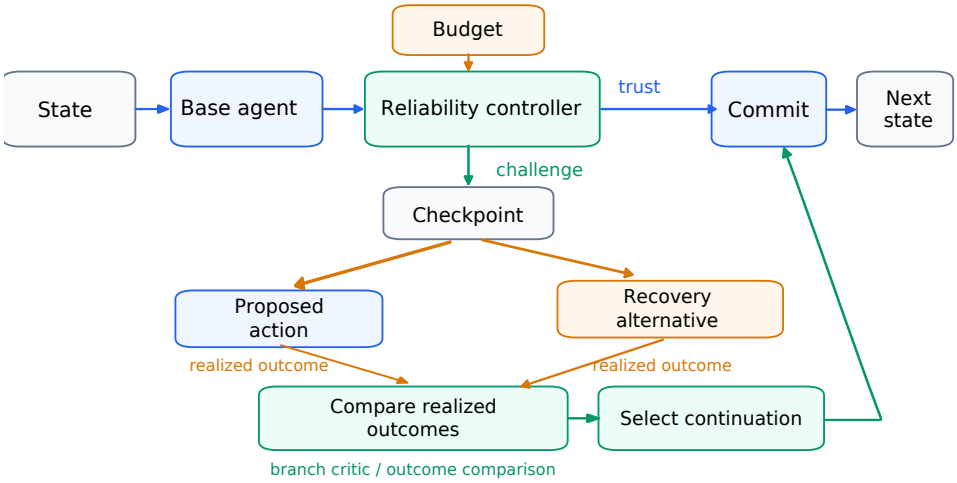


Figure 1: **Budget-aware counterfactual challenge before commitment.** The reliability controller can trust the base action and commit immediately, or spend remaining budget to checkpoint the current environment state and execute the proposed action and a recovery alternative as sibling continuations. It compares their realized outcomes, selects a continuation, and commits the selected branch back to the main trajectory. Colors encode function: blue denotes the base-agent/main trajectory, green reliability-control operations, orange counterfactual experimentation, and gray recoverable environment state.

base agent's current action. Challenging every action wastes inference and can itself alter a successful trajectory, while never challenging leaves recoverable agent errors untouched.

We study **budget-aware counterfactual challenge** as a runtime self-correction mechanism. Given a base-agent action, an external reliability controller can commit immediately or checkpoint the current state, execute the proposed action and plausible alternatives in counterfactual branches, compare their realized outcomes, and then commit the selected continuation. The runtime therefore decides not only *which* recovery to use but *whether the current agent decision is worth challenging at all*. The constraint is sequential: computation spent challenging the current action is unavailable when a later, potentially more consequential agent error appears.

Our contribution is not a new checkpointing primitive, self-correction loop, test-time compute allocator, best-arm problem, or dynamic-programming theorem. We study a narrower runtime object: *pre-commit executable challenge of a concrete agent action from the same recoverable environment state*. Specifically, we: (i) separate the opaque base agent from an external reliability controller that can restore the active state and execute the proposal and recovery alternatives before commitment; (ii) learn a state-conditioned proxy for when this branch evidence is likely to change the committed action and budget such challenges across the remaining trajectory; (iii) isolate the resource-allocation mechanism in controlled settings and Taxi-v4; and (iv) evaluate counterfactual challenge with a hosted LLM in ALFWorld while explicitly accounting for fork and replay cost and negative intervention effects. Figure 1 summarizes the runtime procedure.

2 Related Work

Runtime monitoring and recovery. Prefix-risk monitoring, self-healing orchestration, and state-aware runtimes motivate recovery as a first-class execution concern (Huang et al., 2026; Babu & Agrawal, 2026; Chen, 2026). We start one step later: once a recovery opportunity is identified, should the runtime spend additional execution before choosing an intervention, and how should that expenditure compete with future opportunities?

Self-correction and agent test-time compute. Language-agent self-correction already includes verbal reflection (Shinn et al., 2023), iterative self-feedback (Madaan et al., 2023), tool-interactive

critique (Gou et al., 2023), and environment-level tree search (Zhou et al., 2023; Koh et al., 2024). Verifier-guided embodied agents sample candidate actions and learn verifiers to choose among them (Singhi et al., 2026). More directly, prior work studies when long-horizon agents should plan (Paglieri et al., 2025), when to reflect under test-time scaling (Zhu et al., 2025), how search agents should allocate token/tool budgets using value-of-information estimates (Fang et al., 2026), and how learned orchestrators should dispatch additional solver calls (Qin et al., 2026). Recent harness work makes the overlap sharper: one line formalizes the external agent harness itself as a learnable finite-horizon control layer trained from offline rollouts (Yi & Song, 2026), while CHILL-Harness learns counterfactual intervention-relative workflow advantage for adaptive long-horizon orchestration (Fu et al., 2026). We therefore do not claim novelty for selective reflection, dynamic planning, learned harness control, causal workflow adaptation, verification, or learned test-time compute allocation. Our narrower object is a pre-commit intervention on an already active checkpointable environment trajectory: restore the same ephemeral state, execute the proposed action and recovery alternatives, observe their actual next states, and decide whether the base action should be overridden while accounting explicitly for fork and restore/replay cost.

Checkpoint, fork, and replay. Branching Policy Optimization uses snapshottable sandboxes and sibling rollouts for training-time policy optimization (He et al., 2026); Causal Agent Replay re-executes intervened trajectories for post-hoc attribution (Shah, 2026); DeltaBox and Crab improve checkpoint/restore infrastructure (Dong et al., 2026; Wu et al., 2026). We use these established primitives prospectively for selective runtime challenge rather than claiming branching itself as new.

Pure exploration and value of computation. Local recovery comparison is closely related to contextual best-arm identification and multiple-policy evaluation (Kato & Ariu, 2026; Dann et al., 2023; Erez et al., 2026). The episode-level problem is also related to classical value-of-computation and stochastic dynamic programming. We use these theories as foundations; the contribution is their operational instantiation for sequential agent recovery under a shared execution budget.

3 Runtime Self-Correction by Counterfactual Challenge

3.1 When should the runtime challenge the agent?

Let H_t denote the trajectory history at the t-th recovery decision point, $X_t = \phi(H_t)$ its observable runtime features, and $\mathcal{U}_t$ the candidate recovery actions. The runtime has remaining counterfactual execution budget B_t. For a recovery $u \in \mathcal{U}_t$, define continuation loss

$$\mu_t(u) = \mathbb{E}[L_t(u) \mid H_t, \mathrm{do}(U_t = u)]. \tag{1}$$

At state H_t, the runtime may spend m_t fork executions to compare candidates before committing to $\hat{u}_t$, with episode-level constraint

$$\sum_{t=1}^{T} m_t \leq B. \tag{2}$$

Here, B is specifically a counterfactual fork-execution budget, not a complete runtime-cost budget. Replay/restore work, critic calls, model or tool calls, and wall-clock cost are measured separately because policies with similar fork counts can incur different restoration and verification costs. The key online difficulty is that current experimentation consumes fork budget before future recovery states are known.

3.2 Learning the value of counterfactual challenge

Offline calibration estimates whether additional execution at an observable agent state is likely to improve the eventual commitment. In the controlled recovery setting, traces estimate a recovery-success prior $\hat{p}(u \mid x)$ for each diagnostic state and recovery action, together with a *counterfactual challenge-value curve* $\hat{q}_x(m)$: the residual failure probability after spending m local forks and committing using the resulting evidence. In public agent settings, x can instead contain trajectory signals such as action ambiguity, repeated actions, environment error cues, or state stagnation. These quantities summarize the empirical value of challenging the current decision; they are not oracle transition models.

3.3 BUDGETING FUTURE CHALLENGE OPPORTUNITIES

Let $V(r, b)$ be expected future risk when r recovery opportunities remain and b fork units are available. Given an estimated distribution over observable decision-state types, the controlled reference uses

$$V(r,b) = \mathbb{E}_X \left[\min_{0 \le m \le \min(m_{\max}, b)} \{ C_X(m) + V(r-1, b-m) \} \right], \tag{3}$$

where $C_X(m)$ is derived from learned residual-risk or challenge-value estimates and optional severity weighting. This Bellman recursion is classical; its role is to prevent a weakly suspicious action from automatically consuming evidence that may be more valuable for a later agent error. The learned controller uses sampled calibration estimates rather than latent environment parameters.

3.4 PUBLIC DEPLOYMENT AND LEARNED TRANSFER

The public benchmarks use the same checkpoint/fork/commit architecture with two kinds of allocators. First, `risk-gated` and `reserve-aware` are lightweight hand-designed baselines. `risk-gated` spends two fork executions when an observable runtime-risk score exceeds 0.42. `reserve-aware` uses threshold

$$0.58 - 0.22 \min\left(\frac{B_t}{2R_t}, 1 \right), \tag{4}$$

where R_t is remaining main-trajectory steps, making intervention more conservative when budget is scarce early in an episode. Fork candidates are the base action and one admissible alternative; a branch critic selects between their observed next states.

For Taxi-v4 we additionally deploy a learned budget-aware allocator. For each training/evaluation seed and action-error condition, 300 disjoint calibration episodes are used to collect only observable state-risk scores and counterfactual continuation outcomes. The risk score is quantile-discretized into five bins. Calibration estimates each bin's frequency and residual continuation-failure curve after $m = 0, \ldots, 4$ local forks, using two evaluation rollouts per calibrated state/fork count. A finite-horizon dynamic program then allocates the episode fork budget using only the current risk bin, remaining main-trajectory steps, remaining budget, learned bin frequencies, and learned fork-value curves. The allocator does not read an optimal Taxi policy, latent fault label, or future evaluation trajectory; transition dynamics are accessed only through the same counterfactual rollout primitive used by the recovery benchmark.

For ALFWorld we evaluate a more agent-specific learned challenge controller. On the previously exposed 12-task development set, we first execute non-intervening Gemini trajectories and audit selected states only after recording their direct prefixes. At each audited state, the runtime restores the exact prefix, executes the base action and one admissible alternative, and asks the branch critic to choose between the two realized next states. A locally fitted L2-regularized logistic model predicts the probability that this state-matched critic would overturn the base action from pre-fork trajectory features: action ambiguity, environment-error cues, action repetition, recent observation stagnation, action-set size, task–action lexical overlap, and normalized trajectory position. The fitted score estimates the probability that acquiring branch evidence changes the critic's decision. It is therefore a proxy for challenge value, not an estimate of the expected downstream task improvement caused by performing that challenge. A finite-horizon allocator then decides whether to spend a two-fork challenge from this score and the remaining challenge budget. The model is frozen before evaluation on a separately locked 24-task OOD set that excludes every ALFWorld game index appearing in historical result files.

4 EXPERIMENTS

We evaluate four questions: **Q1** does future-opportunity-aware allocation help when recovery budget is scarce? **Q2** does a learned recovery-value model retain the advantage without access to latent fault parameters? **Q3** can learned budget-aware allocation transfer to a public non-LLM sequential agent using only observable calibration data, and how does it compare with hand-designed selective policies? **Q4** what benefits and failure modes appear in a public LLM-agent environment? Figure 2 summarizes this evidence progression. Full configurations, additional sweeps, development experiments, and reproducibility details are in the appendix.

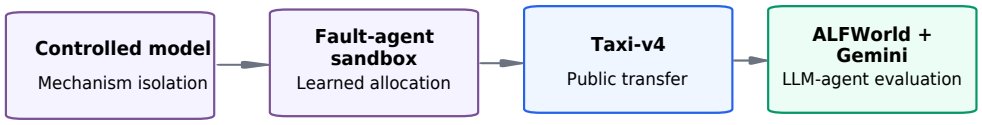


Figure 2: **Evidence progression across evaluation settings.** We move from controlled mechanism isolation to learned allocation in a fault-agent sandbox, then test public transfer in Taxi-v4 and finally evaluate the same selective-challenge principle with a hosted LLM agent in ALFWorld. Each stage supports a different claim rather than serving as a substitute for the next.

4.1 Controlled learned recovery

Our controlled tool-agent sandbox contains four latent orchestration faults (transient failure, malformed arguments, stale state, and tool outage), five recovery actions, and noisy diagnostic labels. Offline randomized calibration estimates diagnostic frequencies, recovery priors, and fork-value curves; held-out allocation never observes the latent fault class or true recovery matrix.

With six recovery decision points per episode and budget 12, the larger evaluation sweep gives 9.25% success for direct routing, 14.11% for equal allocation, and 17.00% for learned budget-aware allocation over 8,000 held-out episodes. At budget 32 the corresponding rates are 9.35%, 18.26%, and 20.96%. Across five independent training/evaluation seeds at the main calibration setting, success is 0.0918 ± 0.0033 for direct, 0.1367 ± 0.0050 for equal, and 0.1710 ± 0.0038 for learned allocation (mean $\pm$ s.e.). The learned controller therefore improves recovery at essentially matched intervention volume rather than by simply spending more forks.

A simpler sequential opportunity-cost simulator independently isolates the mechanism: at budget 20, severity-weighted risk is 1.956 for equal allocation, 1.908 for a myopic local rule, 1.852 for stochastic allocation, and 1.836 for a clairvoyant lower bound. We treat this as mechanism validation, not a new DP result.

4.2 Taxi-v4: non-LLM public benchmark

We train a tabular Q-learning agent for 1,200 episodes and evaluate it under a 40-step deadline. Executed actions are replaced by a random admissible action with probability 0.10, 0.20, 0.30, or 0.40. Each condition uses five training/evaluation seeds and 300 evaluation episodes per seed. For the final evaluation, base-action stochasticity is keyed by seed, episode, and main-trajectory step, while fork-rollout randomness uses a disjoint stream; methods that spend different numbers of forks therefore cannot shift future base-policy noise merely by consuming extra random numbers. At fork budget 24, reserve-aware allocation improves deadline-constrained success relative to no recovery by $+0.173$ [95% CI $0.134, 0.213$], $+0.141$ [$0.119, 0.164$], $+0.116$ [$0.076, 0.156$], and $+0.105$ [$0.076, 0.135$] as the action-error rate increases from 0.10 to 0.40. Equal allocation becomes harmful at 0.30 (-0.085 [$-0.133, -0.038$]) and 0.40 (-0.183 [$-0.215, -0.151$]). The inferential unit is the seed, not the individual episode.

The learned allocator provides a direct public-transfer test of the method. At budget 24, learned allocation achieves success rates of 0.886, 0.878, 0.851, and 0.813 at error rates 0.10, 0.20, 0.30, and 0.40. Relative to direct execution, the paired seed-level changes are +0.139 [95% CI 0.097, 0.182], +0.101 [0.072, 0.129], +0.065 [0.017, 0.114], and +0.051 [-0.007, 0.109], respectively. Thus the learned allocator shows positive public transfer through error 0.30 under the pre-locked protocol, while the highest-error condition remains inconclusive.

The learned allocator is also intervention-efficient, using 4.81, 4.91, 6.17, and 7.86 forks per episode across the four error rates, compared with 7.08, 7.97, 8.98, and 9.94 for reserve-aware. This efficiency does not translate into the highest success: reserve-aware remains better by 3.4–5.4 percentage points at budget 24, with the paired difference excluding zero at error 0.20 and narrowly at 0.30. The public result therefore closes the controlled-to-public deployment gap without establishing that

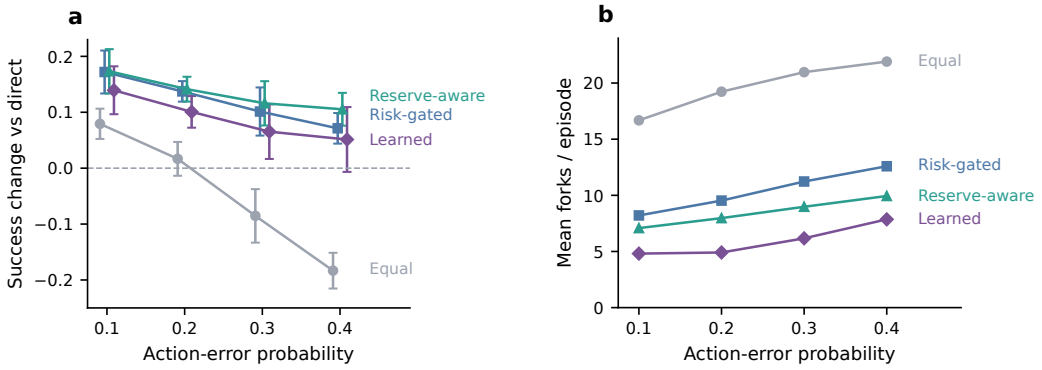


Figure 3: **Taxi-v4 reliability–intervention trade-off.** (a) Paired success changes versus direct execution at budget 24, with 95% t intervals over five paired seeds. (b) Mean fork use under the same conditions. Learned allocation transfers positively through action-error 0.30 while using fewer forks than reserve-aware, but reserve-aware retains higher success overall.

the current learned calibration dominates a tuned heuristic. At error 0.30, equal allocation averages 21.0 forks per episode, versus 11.2 for risk-gated, 9.0 for reserve-aware, and 6.2 for learned allocation; at error 0.40 the corresponding values are 21.9, 12.6, 9.9, and 7.9. Figure 3 summarizes the reliability/intervention trade-off.

4.3 ALFWorld: public LLM-agent evaluation

We use ALFWorld 0.4.2 with TextWorld 1.7.0 and Gemini 3 Flash as an opaque base policy. The environment's admissible commands are indexed and the model returns a numeric action index, avoiding surface-form copying errors. Episodes use a 50-step horizon and fork budget 12. Because the packaged interface lacks native checkpointing, restoration uses deterministic reset plus replay of committed action history; replay steps are counted separately from forks.

We first pre-lock 24 OOD tasks across all six ALFWorld task families after excluding every development task. Direct execution succeeds on 21/24 tasks (87.5%), while fixed-fork, risk-gated, and reserve-aware recovery each succeed on 22/24 (91.7%). Only a few task outcomes are discordant, so we make no population-level superiority claim. Because each policy independently queries the hosted model, task-level direct/recovery differences are also descriptive rather than matched causal effects. Nevertheless, the intervention profile is informative: fixed forking uses 11.42 forks per task, risk-gated 8.25, and reserve-aware 4.25 at the same aggregate success. Reserve-aware, however, averages 40.25 replay steps versus 27.42 for fixed forking because some interventions occur later. Thus fewer forks do not imply lower total runtime cost.

We next freeze the learned ALFWorld challenge model after 86 audited development states, including 16 critic overturns (18.6%); leave-one-game-out challenge prediction has AUC 0.687 and Brier score 0.144. We then evaluate direct, reserve-aware, and learned challenge on a second 24-task OOD set that excludes every game index appearing in previous ALFWorld result files. Exact repeated base-policy prompts are shared through a prompt-hash cache so common prefixes receive the same hosted-model response, while diverged states still require independent calls. Because all three policies achieve the same task-level success on this confirmatory set (23/24; 95.8%), this comparison isolates intervention burden rather than efficacy. Learned challenge uses 1.50 forks and 0.75 critic calls per task, compared with 2.50 forks and 1.25 critic calls for reserve-aware, while replay increases from 14.92 to 16.75 steps per task. It issues 18 challenges and the critic overturns five proposed actions, but the sole direct failure is not rescued. Thus selective challenge changes where and how often the runtime intervenes without establishing an improvement in task success, and replay timing can offset savings in fork count. Figure 4 makes this distinction explicit.

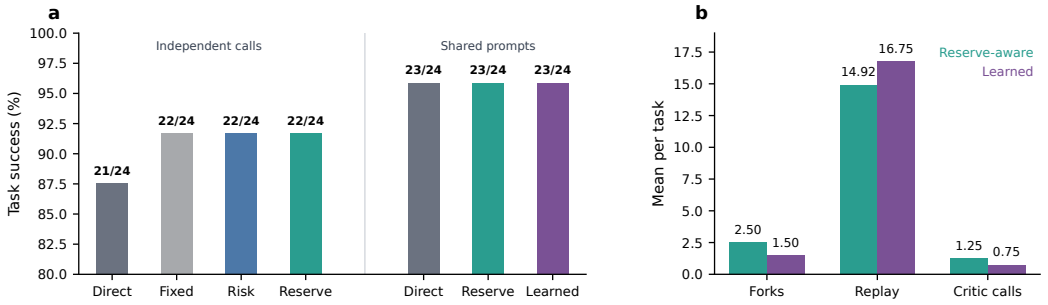


Figure 4: **ALFWorld success and intervention burden.** (a) Task success on the independently queried heuristic set and the shared-prompt learned-challenge set. (b) On the shared-prompt confirmatory set, learned challenge reduces forks and critic calls relative to reserve-aware while replay steps rise slightly. The result concerns intervention allocation, not a task-success gain.

Table 1: Matched-success intervention burden on the second untouched 24-task ALFWorld confirmatory set. All policies succeed on the same 23 tasks; the informative comparison is therefore how much and where each recovery policy intervenes.

Policy	Forks	Critic calls	Replay	Success
Direct	0.00	0.00	0.00	23/24
Reserve-aware	2.50	1.25	14.92	23/24
Learned challenge	1.50	0.75	16.75	23/24

5 Limitations and Conclusion

The evidence has three important boundaries. First, the learned recovery-value/DP allocator transfers directly to Taxi-v4 but does not outperform the hand-designed reserve-aware policy in success rate. The agent-specific learned challenge controller is also deployed end to end in ALFWorld, but on the second untouched 24-task set it matches direct and reserve-aware success rather than improving it; its development-set challenge predictor is only moderately discriminative. Second, the first ALFWorld confirmatory comparison independently queries a hosted model across policies and remains descriptive. The second set improves common-prefix control through exact-prompt sharing, but only one direct task fails, leaving little natural failure headroom for an efficacy estimate. Third, checkpoint fidelity and cost are environment dependent: replay-based restoration cannot undo arbitrary external side effects, and the learned controller's lower fork count on the second ALFWorld set still produces slightly more replay than reserve-aware because interventions occur at different depths. The present formulation also budgets fork executions rather than a unified heterogeneous runtime cost, and the ALFWorld challenge score predicts critic overturns rather than expected downstream outcome improvement. A fuller decision-theoretic formulation could jointly price fork, replay/restore, critic, model, and tool costs while learning the expected downstream benefit of each challenge.

Within these boundaries, the results support a limited runtime principle: counterfactual recovery execution has opportunity cost. Counterfactual experiments spent at one state cannot be reused at future recovery opportunities, indiscriminate intervention can waste resources or harm trajectories, and learned controllers can reduce intervention frequency in public sequential environments. The evidence does not establish that the current learned challenge policy improves a strong LLM agent's task success, and lower fork count is not equivalent to lower restoration cost. The present method should therefore be read as a selective executable-challenge mechanism with measured reliability-cost trade-offs, not as a general solution to agent self-correction.

Reproducibility Statement

The anonymous supplementary material includes canonical commands, fixed seeds and task indices, software versions, complete dependency snapshots, SHA-256 artifact anchors, source-data files for the main figures, and scripts for all controlled, Taxi-v4, and ALFWorld evaluations. The main text reports the independent statistical unit for each experiment and separates development, stress-test, and pre-locked confirmatory evidence.

AI Use Statement

Generative AI tools were used to assist manuscript drafting and language polishing, literature retrieval, hypothesis refinement and research-method feedback, software and figure preparation, and result interpretation. The authors executed and audited the experiments, verified the reported analyses and results, and take responsibility for all scientific claims and the final content of this work.

References

R. S. Babu and A. Agrawal. Self-healing agentic orchestrators for reliable tool-augmented large language model systems. *arXiv preprint arXiv:2606.01416*, 2026.

Xiwei Chen. State-aware runtime for long-horizon llm agents: A conceptual framework and research agenda. *Cambridge Open Engage*, 2026. doi: 10.33774/coe-2026-vt9t2-v2.

Christoph Dann, Mohammad Ghavamzadeh, and Teodor Vanislavov Marinov. Multiple-policy high-confidence policy evaluation. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics*, volume 206, pp. 9470–9487. PMLR, 2023.

Y. Dong, J. He, Y. Hou, D. Du, Z. Xu, S. Yu, Y. Xia, and H. Chen. Deltabox: Scaling stateful ai agents with millisecond-level sandbox checkpoint/rollback. *arXiv preprint arXiv:2605.22781*, 2026.

L. Erez, F. Chen, A. Cohen, T. Koren, Y. Mansour, S. Moran, and A. Rakhlin. The sample complexity of multiclass and sparse contextual bandits. In *Proceedings of the 39th Conference on Learning Theory*, volume 336, pp. 2307–2338. PMLR, 2026.

Zhengru Fang, Senkang Forest Hu, Zhonghao Chang, Yu Guo, Yihang Tao, Hongyao Liu, Mengzhe Ruan, Jun Huang, and Yuguang Fang. Inference-time budget control for llm search agents. *arXiv preprint arXiv:2605.05701*, 2026.

Jiarun Fu, Lizhong Ding, Sida Chen, Honglei Xin, Chunhui Zhang, Pengqi Li, Qiuning Wei, Ye Yuan, and Guoren Wang. Chill-harness: Counterfactual harness learning for efficient reasoning in long-horizon agents. *arXiv preprint arXiv:2607.25825*, 2026.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023.

B. He, Y. Chen, X. Zhang, and X. Liu. Branching policy optimization: Sandbox-native language agent reinforcement learning. *arXiv preprint arXiv:2607.14171*, 2026.

X. Huang, J. Hu, R. Roy, C. Wu, Y. Dong, and X. Huang. Prefixguard: From llm-agent traces to online failure-warning monitors. *arXiv preprint arXiv:2605.06455*, 2026.

Masahiro Kato and Kaito Ariu. The role of contextual information in best arm identification. *Journal of Machine Learning Research*, 27(51):1–61, 2026.

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents. *arXiv preprint arXiv:2407.01476*, 2024.

Zhenpeng Li. Post-hoc trajectory-risk certification for modular llm-based security agents. *arXiv preprint arXiv:2608.05199*, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

Davide Paglieri, Bartłomiej Cupiał, Jonathan Cook, Ulyana Piterbarg, Jens Tuyls, Edward Grefenstette, Jakob Nicolaus Foerster, Jack Parker-Holder, and Tim Rocktäschel. Learning when to plan: Efficiently allocating test-time compute for llm agents. *arXiv preprint arXiv:2509.03581*, 2025.

Peijia Qin, Qi Cao, and Pengtao Xie. Atlas: Agentic test-time learning-to-allocate scaling. *arXiv preprint arXiv:2606.01667*, 2026.

J. Shah. Causal agent replay: Counterfactual attribution for llm-agent failures. *arXiv preprint arXiv:2606.08275*, 2026.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

Nishad Singhi, Christian Bialas, Snehal Jauhri, Vignesh Prasad, Georgia Chalvatzaki, Marcus Rohrbach, and Anna Rohrbach. Think twice, act once: Verifier-guided action selection for embodied agents. *arXiv preprint arXiv:2605.12620*, 2026.

T. Wu, C. Chang, L. Cao, W. Gao, and W. Wang. Crab: A semantics-aware checkpoint/restore runtime for agent sandboxes. *arXiv preprint arXiv:2604.28138*, 2026.

Haiwen Yi and Xinyuan Song. Learning to control llm agent harnesses with offline reinforcement learning. *arXiv preprint arXiv:2607.05458*, 2026.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

King Zhu, Hanhao Li, Siwei Wu, Tianshun Xing, Dehua Ma, Xiangru Tang, Minghao Liu, Jian Yang, Jiaheng Liu, Yuchen Eleanor Jiang, Changwang Zhang, Chenghua Lin, Jun Wang, Ge Zhang, and Wangchunshu Zhou. Scaling test-time compute for llm agents. *arXiv preprint arXiv:2506.12928*, 2025.

A Additional Experimental Detail

A.1 Static diagnostic rejected as the main formulation

A static pool of 50 heterogeneous prefix classes with three candidate recoveries was used early in development. At fork budget 100, a risk-aware allocation reduced mean failure from 0.1104 under uniform allocation and 0.1069 under importance-weighted best-arm allocation to 0.1019. At budgets 400 and 800, importance-weighted allocation matched or slightly outperformed the risk-aware rule. This diagnostic confirmed that a static pool largely behaves like contextual pure exploration; we therefore do not use it as the paper's main formulation.

A.2 Controlled budget sweeps and robustness

The sequential opportunity-cost model uses ten decision points, four decision-state types, up to eight local forks per state, and 5,000 evaluation episodes per budget. The stochastic allocator has lower severity-weighted risk than equal and myopic allocation at every tested nonzero budget from 4 to 48. The learned fault-runtime main sweep uses 30,000 randomized prior samples, 50,000 value-curve calibration samples, up to eight local forks, six recovery points per episode, and 8,000 held-out episodes per budget. Additional archived experiments vary training seed, calibration size, fault mixture, and diagnostic shift. Severe diagnostic shift reduces absolute performance, motivating the calibration-transfer limitation in the main text.

B TAXI-V4 DETAILS

The five seed values are generated as $20260808 + 1009i$ for $i = 0, \ldots, 4$. The main sweep evaluates action-error probabilities 0.10, 0.20, 0.30, and 0.40 at fork budgets 12 and 24. Each seed summarizes 300 evaluation episodes. Confidence intervals in the main text are two-sided t intervals on paired seed-level success differences with four degrees of freedom; individual episodes are not treated as independent inferential replicates.

Future-opportunity ablation. To isolate the role of budget reservation, we compare the learned DP allocator with a learned-myopic variant that uses the identical calibration model, local fork-selection primitive, budget, and keyed exogenous random streams but removes the future-value term. The myopic controller generally attains slightly higher raw success by spending nearly all available forks, whereas the DP controller reduces mean fork use by 64.8–78.0% across the tested budget/error conditions. At budget 24, the DP-minus-myopic success differences are -0.034 [95% CI $-0.080, 0.012$], -0.027 [$-0.057, 0.002$], -0.035 [$-0.106, 0.035$], and -0.039 [$-0.107, 0.029$] for action-error rates 0.10–0.40, while mean fork use falls from 21.0–22.4 to 4.8–7.9. We therefore interpret future-opportunity-aware allocation as a resource-efficiency mechanism rather than a guarantee of higher unconstrained success.

C ALFWORLD PROTOCOL AND ADDITIONAL RESULTS

The pre-locked confirmatory indices are

> 69, 97, 106, 112, 7, 9, 49, 79, 13, 23, 86, 114, 54, 123, 131, 132, 15, 16, 73, 74, 3, 34, 43, 45.

They contain four tasks from each of six OOD task families and exclude all 12 development tasks. Gemini uses temperature 0 and minimal thinking. The runtime maps a numeric model output to the exact environment-provided admissible command. Confirmatory runs recorded zero parser failures and zero API errors.

In the clean 12-task development set, direct, risk-gated, and reserve-aware execution each succeed on 10/12 tasks, while fixed forking succeeds on 9/12. In a separate 35% decision-fault stress test, direct and fixed forking succeed on 9/12 tasks while risk-gated and reserve-aware succeed on 10/12. These development and stress results are not pooled with the untouched confirmatory set.

Second learned-challenge confirmatory set. After fitting the agent-specific challenge model exclusively on the historical 12-task development set, we locked a second 24-task OOD evaluation set that excludes every ALFWorld game index appearing in prior result files. Exact repeated base-policy prompts are cached and shared across methods, so common prefixes receive the same Gemini response while diverged states still require distinct hosted-model calls. Direct, reserve-aware, and learned challenge each succeed on 23/24 tasks. Mean forks are 0.00, 2.50, and 1.50; mean replay steps are 0.00, 14.92, and 16.75; and mean critic calls are 0.00, 1.25, and 0.75, respectively. The learned controller issues 18 total challenges with five critic overrides but does not change any task-level success outcome relative to direct execution. This set is therefore evidence about selective intervention behavior and cost, not a success-rate gain.

D COST ACCOUNTING

A complete implementation should distinguish main-trajectory cost, fork execution, checkpoint/restore, tool calls, and verifier/critic calls. Fork count is therefore an intervention-frequency measure, not a universal cost proxy. In the current ALFWorld replay implementation, later forks can be more expensive because restoring a deeper state requires replaying a longer committed action history.

E Reproducibility Manifest

The supplementary artifact records canonical reproduction commands, environment versions, dependency snapshots, task locks, seed rules, and SHA-256 checksums. The project directory used during development was not a Git checkout, so no historical commit hash is claimed; checksums anchor the frozen script and result versions instead. Before final ICLR submission, the anonymized artifact should be placed in a version-controlled release or uploaded as anonymous supplementary code.