TRAINING NEEDS TRUSTWORTHY WORLDS: VERIFIED SYNTHETIC WEB ENVIRONMENTS FOR AGENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Synthetic web environments can scale agent training, but plausible pages may hide unreachable goals, contradictory records, or invalid state updates. These defects confound policy failure with environment failure. We study environment reliability as a source of supervision for compact web agents. Our framework represents a website as a scaffold of pages, navigation, backend records, tasks, and state-change markers. Bounded verification and local repair establish executable task traces; marker-constrained updates and backend-derived rewards connect visible actions to persistent state. Across 500 environments in six domains, task feasibility increases from 48.6% to 94.8%. With terminal rewards fixed, verification raises held-out PPO success from 19.4% to 42.6%; adding dense state rewards reaches 58.7%. Controls separate these effects from feasible-task filtering and reward granularity. The resulting policies also improve on WebArena-compatible tasks, WebShop, and MiniWoB++ through a shared DOM interface, without policy-side LLM calls. The evidence supports bounded, DOM-grounded interaction learning while exposing remaining limitations in noisy web dynamics and verification coverage.

1 INTRODUCTION

Web agents must connect language instructions to actions whose consequences persist across pages. Buying a product, updating a profile, or submitting an application requires navigation, element grounding, and consistent backend changes. Benchmarks such as WebShop and WebArena make these dependencies explicit through executable websites and programmatic success evaluators (Yao et al., 2023a; Zhou et al., 2024). Constructing similarly diverse training worlds is expensive, motivating automatic generation of websites and interaction experience.

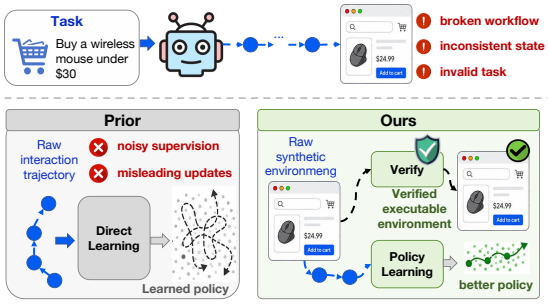


Figure 1: **Verified worlds for policy learning.** Broken workflows, inconsistent state, and invalid tasks can turn generated interactions into misleading supervision. Verification and repair produce executable environments for agent training.

A generated website can appear coherent while providing invalid supervision. A purchase task may link to a missing checkout page; a button may change the display without updating the cart; a completion predicate may require a state that no action can reach. In each case, a failed rollout is compatible with a competent policy (Figure 1). More interaction does not resolve a broken transition system, and a denser reward may amplify its inconsistencies.

We ask how to make synthetic environments sufficiently reliable to support agent learning. The central object is the whole stateful environment: reachable pages, valid updates, and task rewards must agree. We call this objective *attributable supervision*: reducing environment-induced ambiguity

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.

in the interpretation of observed policy outcomes. This objective links policy evaluation to an explicit environment specification and executable checks.

Our framework generates a structured scaffold, verifies and locally repairs it, and trains a compact agent through an event-driven simulator. Ordinary interface operations are deterministic; persistent writes pass marker-specific validation. The same backend state defines task completion and training progress. This connects three obligations—reachability, transition validity, and reward consistency—within one inspectable representation.

Our contribution is a unified environment-level formulation of reliable training supervision and a controlled evaluation of its effect on agent learning. We specify the bounded certificate and its failure cases, distinguish verification from reward shaping through controlled comparisons, and evaluate transfer under a shared Document Object Model (DOM) interface. Additional audits characterize repair locality, fallible verifiers, human executability, benchmark overlap, and policy-backbone dependence. The resulting evidence connects improved execution semantics to better supervision for DOM-grounded policies.

2 Related Work

Web environments and agent learning. World of Bits and workflow-guided exploration established web interaction as language-conditioned control (Shi et al., 2017; Liu et al., 2018). WebShop, Mind2Web, WebArena, VisualWebArena, WorkArena, and BrowserGym broaden task coverage and evaluation realism (Yao et al., 2023a; Deng et al., 2023; Zhou et al., 2024; Koh et al., 2024; Drouin et al., 2024; Le Sellier De Chezelles et al., 2024). ReAct, WebVoyager, and AgentOccam improve agent-side reasoning or interfaces (Yao et al., 2023b; He et al., 2024; Yang et al., 2024). Our target is the validity of automatically constructed training environments. We evaluate external transfer through a common DOM interface.

Synthetic supervision and rewards. EnvGen generates and adapts environments for embodied agents, while DreamGym synthesizes interactive experience (Zala et al., 2024; Chen et al., 2025). Web-agent self-improvement, Agent Q, and WebRL use synthetic trajectories, search, or evolving curricula (Patel et al., 2024; Putta et al., 2024; Qi et al., 2024). Autonomous evaluators and WebShepherd supply learned feedback (Pan et al., 2024; Chae et al., 2025). We instead expose persistent records and task constraints within the generated world, so progress can be computed from state. This supplies privileged training rewards while the actor observes only the task and DOM.

Testing and program repair. Model-based testing derives tests from system models, symbolic execution explores program paths, and automated repair uses tests or validation feedback to modify faulty programs (Utting et al., 2012; Cadar et al., 2008; Le Goues et al., 2012; Xia & Zhang, 2023). Our framework shares these foundations. The setting differs because the synthesized system supplies both the policy's transitions and its reward signal. We therefore evaluate not only whether repairs pass checks, but also whether the resulting environments yield better policy learning under controlled reward and task-selection conditions.

3 Problem and Consistency Contract

A task t comprises an instruction x_t, initial-state distribution ρ_t, and completion predicate C_t. The agent sees a rendered DOM and chooses a currently available action. We represent the environment and interaction as

$$\mathcal{E} = (\mathcal{P}, \mathcal{L}, \mathcal{D}, \mathcal{M}, \mathcal{Q}), \tag{1}$$

$$s_k = (p_k, b_k, \sigma_k, h_k), \quad o_k = \mathrm{Render}(p_k, b_k, \sigma_k), \quad a_k \in \mathcal{A}(o_k). \tag{2}$$

Here $\mathcal{P}$ contains page templates, $\mathcal{L}$ navigation links, $\mathcal{D}$ typed database records, $\mathcal{M}$ state-change markers, and $\mathcal{Q}$ tasks. State s_k contains the current page p_k, database b_k, session σ_k, and relevant history h_k. The actor receives only $(x_t, o_k, \mathcal{A}(o_k))$.

We impose three linked obligations. **Reachability:** a task marked feasible has a replayable successful trace within horizon $H_{\max}$. **Transition validity:** each accepted persistent write respects its marker,

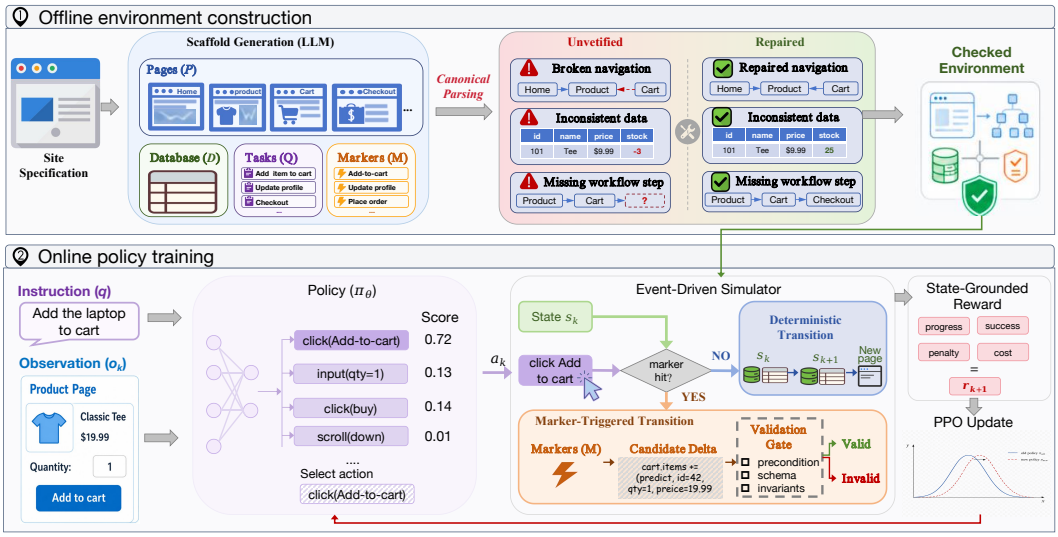


Figure 2: **From generated scaffolds to state-grounded learning.** Offline verification repairs navigation, data bindings, and task workflows. During synthetic interaction, marked events propose constrained writes; accepted state changes define PPO rewards. The learned actor uses only the task and DOM.

schema, and declared invariants. **Reward consistency:** success and intermediate progress are evaluated on the resulting backend state. These obligations are relative to the generated specification: an incorrect goal or omitted invariant can still pass all checks.

Motivating diagnostics. The original diagnostic pool contains 500 environments and 6,842 tasks across six domains. Raw scaffolds have 12.4 defects per environment, including 4.9 task-blocking defects, and 48.6% bounded task feasibility. The diagnostic protocol also considers 1,800 successful trajectories to study marker sparsity (Appendix B.1). Frequency alone is not the repair objective: a rare missing submit action can block an entire workflow. The representation consequently exposes both structural errors and their consequences for task execution.

4 Method

4.1 Canonical Scaffolds and Bounded Verification

The authors specify a site's domain, purpose, user roles, entity types, allowed operation families, and global invariants. The generator supplies pages, links, records, task text, and workflows. Canonical parsing extracts DOM nodes, database bindings, and marker instances. A marker is

$$m = (e_m, \mathrm{pre}_m, R_m, W_m, \Omega_m), \tag{3}$$

where e_m triggers the operation, pre_m is its precondition, R_m and W_m are permitted read and write fields, and Ω_m specifies the operation, entity, arguments, and output schema.

Deterministic checks test link targets, entry-page reachability, schema validity, DOM–database bindings, and marker signatures. Specialized semantic verifiers examine structural omissions, content validity, cross-page consistency, and workflow feasibility. A report contains a type, location, object, severity, evidence, and confidence. Reports are deduplicated by $(\mathrm{type}, \mathrm{loc}, \mathrm{obj})$; defect-triggered coordination (DTCP) sends one targeted cross-check to dependent verifier roles. For a report d, confidence is aggregated over reporting verifiers:

$$\mathrm{conf}(d) = \frac{\sum_{j:d\in\Delta_j} \omega_j c_j(d)}{\sum_{j:d\in\Delta_j} \omega_j}, \qquad \Delta_{\mathrm{acc}} = \{d : \mathrm{conf}(d) \geq \tau_c \ \vee \ \mathrm{sev}(d) \geq \tau_s\}. \tag{4}$$

The high-severity branch admits a candidate for repair; it does not bypass validation. Appendix A.2 specifies the checks, and Appendix A.3 gives report routing.

Algorithm 1 Verification-guided construction (bounded specification)

Require: Site specification q, repair budget B, horizon $H_{\max} = 40$
$\mathcal{E} \leftarrow \mathrm{Canonicalize}(\mathrm{Generate}(q))$
for $r = 1, \ldots, B$ **do**
  Run symbolic checks and semantic reports; deduplicate and cross-check dependencies
  Form Δ_{acc} and the defect dependency graph
  if no accepted critical defects remain **then**
    Search and replay task witnesses; stop if every required task is certified
  end if
  Select a ready defect by Eq. (6); stop unresolved if none exists
  Propose a local edit and recheck affected bindings, markers, invariants, and traces
  Commit a validated edit; otherwise retain the previous scaffold and record failure
end for
return Current scaffold, task certificate statuses, residual defects, and repair log

Feasibility requires a witness, not an LLM assertion. A proposed or searched action sequence is replayed against the current scaffold, retaining its states and deltas. For the tested initial state s_0, a positive certificate is

$$\exists H \le 40,\ a_{0:H-1}: \quad a_k \in \mathcal{A}(o_k), \quad s_{k+1} = F(s_k, a_k), \quad C_t(s_H) = 1. \tag{5}$$

The certificate is specific to the tested initial state, scaffold version, and realized transitions. Failure to find a trace with pruned actions or a finite search budget is *unresolved*; bounded infeasibility requires exhaustive coverage of the declared finite search space. Worst-case search remains exponential in horizon. We distinguish these statuses in Appendix A.4.

4.2 Local Repair and Revalidation

Accepted reports form a dependency graph: schema repairs precede affected bindings; marker and navigation repairs precede task rechecks. Priority-weighted repair scheduling (PWRS) selects a ready defect using

$$\mathrm{score}(d) = \lambda_1 \mathrm{sev}(d) + \lambda_2 \log(1 + \mathrm{dep}(d)) + \lambda_3 \mathrm{scope}(d) - \lambda_4 \mathrm{cost}(d). \tag{6}$$

Here dep counts downstream dependencies, scope counts affected pages and tasks, and cost estimates edit complexity. The nonnegative weights are fixed scheduling parameters. Repairs restore links or controls, reconcile values with canonical records, and repair marker schemas. The task's intended goal is a constraint on repair, not an objective to weaken for easier certification.

After a local edit, revalidation covers navigation neighbors and all affected data bindings, markers, and tasks, including dependencies outside the immediate page neighborhood. Previously stored witnesses are valid only for the corresponding scaffold version. The output records residual defects and unresolved tasks when a dependency cycle or the repair budget prevents completion.

4.3 Event-Driven Simulation and State Rewards

Ordinary navigation, scrolling, and local input editing use deterministic transitions. A persistent event invokes a writer with the marker's declared context. A candidate delta $\hat{\delta}_k$ is accepted only when the precondition holds, reads stay within R_m, writes stay within W_m, and the resulting state satisfies the schema and invariants:

$$\delta_k = \mathrm{Validate}_{m_k}(\hat{\delta}_k; s_k, a_k) \in \mathcal{U}_{m_k}(s_k, a_k) \cup \{\bot\}. \tag{7}$$

Rejection ($\bot$) preserves the previous state and sets $\epsilon_k = 1$; accepted writes are logged and rendered. Marker validation checks declared constraints; independent replay additionally tests their consequences for task execution.

Task constraints are decomposed into backend or session predicates $\phi_{t,i}$, such as a valid form, the correct cart entity, or a completed order. With nonnegative weights whose sum is positive,

$$\Psi_t(s) = \frac{\sum_i w_i \phi_{t,i}(s)}{\sum_i w_i}, \tag{8}$$

$$r_k = \mathbf{1}[C_t(s_{k+1}) = 1] + \alpha(\Psi_t(s_{k+1}) - \Psi_t(s_k)) - \beta\epsilon_k - \eta. \tag{9}$$

Method	Def.↓	Block↓	Feasible↑	Author SR↑	State viol.↓	Min.↓
No verification	12.4	4.9	48.6	36.9	18.7	–
Rule-based	9.8	4.1	53.2	27.2	14.6	**0.5**
Single-LLM (OpenAI, 2024)	6.2	2.7	70.4	54.3	9.8	<u>12</u>
Self-consistency (Wang et al., 2023)	5.8	2.1	79.6	69.1	7.2	35
AutoGen (Wu et al., 2023)	<u>5.1</u>	<u>1.6</u>	<u>88.7</u>	<u>89.2</u>	<u>5.5</u>	28
Ours	**3.3**$_{-1.8}$	**0.7**$_{-0.9}$	**94.8**$_{+6.1}$	**91.4**$_{+2.2}$	**2.8**$_{-2.7}$	18

Table 1: **Environment quality.** Rates are percentages; defects are counts per site; time is minutes per site. Bold and underline indicate best and second-best values in each column. Green subscripts are changes from the strongest comparison method for that metric.

Episodes terminate on success or at the horizon, so the success bonus is paid once. We use $\alpha = 0.5$, $\beta = 0.2$, and $\eta = 0.01$. The progress difference supplies dense shaping, whose effect is measured through the reward controls below.

PPO (Schulman et al., 2017) trains $\pi_\theta(a \mid o_k, x_t)$ over available DOM actions. Backend records, predicates, and verifier reports are hidden from the actor. LLM calls may occur in the synthetic simulator at marked writes, while the trained actor performs action selection without LLM calls. On external benchmarks the benchmark environment supplies transitions and its original evaluator supplies success.

5 Experiments

5.1 Setup and Measurement

The suite contains 500 sites in e-commerce, social media, banking, education, healthcare, and government, with 15–30 pages per site. The split is 350/75/75 sites for training/validation/test, grouped by site specification. Generation and model selection exclude external benchmark artifacts. The protocols below distinguish construction quality, held-out policy learning, and external transfer.

The primary policy has 8.9M parameters, a two-layer bidirectional GRU instruction encoder, an attribute-aware DOM encoder, and an MLP action scorer. The generator is `gpt-4-0613` at temperature 0.7; verifiers and the state writer use the same model at temperature 0. All construction methods start from the same raw scaffolds. PPO uses three seeds, discount 0.99, GAE 0.95, clipping 0.2, learning rate 3×10^{-4}, 4,096 rollout steps per update, minibatches of 512, and four epochs. The training budget is 10^6 steps; the sample-efficiency comparison reports checkpoints through 600K steps. Appendix B.2 collects the implementation settings.

We compare raw scaffolds, deterministic checks, a single LLM verifier, five-sample self-consistency, and an adapted AutoGen verifier. These are construction baselines, not policy architectures. Environment metrics count defects, blocking defects, witnessed feasibility, state violations, and curation time. Author task SR measures task completion by an author using the visible interface; a separate blinded human study is described in Appendix B.5. Policy success uses the frozen completion evaluator. Intervals denote 95% confidence intervals where available; the remaining tables report point estimates.

5.2 Environment Executability

Table 1 shows a 46.2-point feasibility increase over raw generation, accompanied by fewer blocking defects and state violations. Rule-based checks improve graph and schema validity but leave workflow errors; their author task SR decreases despite higher witnessed feasibility. These metrics answer different questions: a trace search can exploit a route that is difficult to discover through the visible interface.

The repair trajectories in Figure 3 decrease defects from 12.4 to 4.2 at the reported intermediate stage and to 3.3 finally; feasibility rises from 48.6% through 90.7% to 94.8%. Separately implemented trace analysis reports 92.9% feasibility, with 91.7% author-audited replay and 2.3% false-feasibility

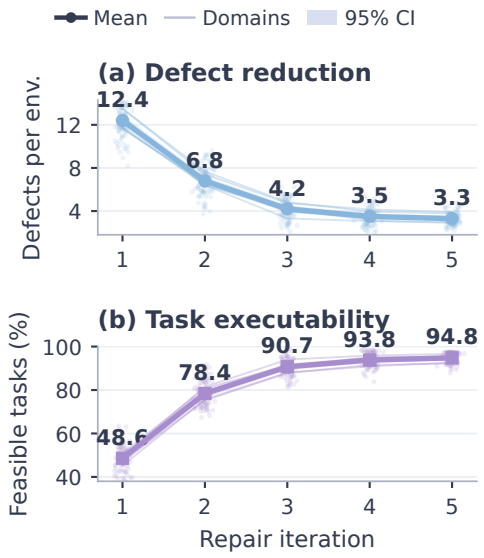


Figure 3: **Verification–repair convergence.** Defect counts decrease and witnessed task feasibility increases across repair iterations. The curves show the rapid early improvement and subsequent saturation.

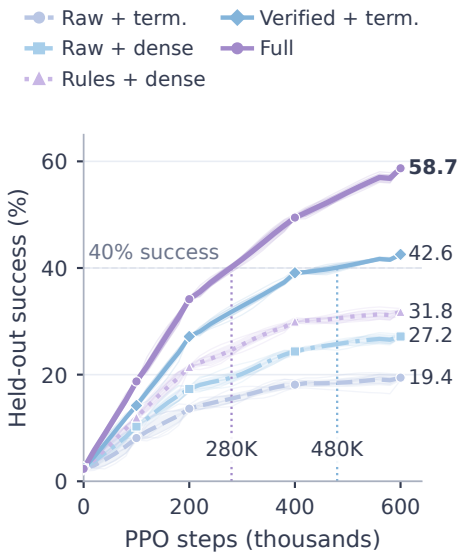


Figure 4: **Policy learning on synthetic environments.** Verification improves learning under terminal rewards; state-grounded dense rewards further improve sample efficiency and final success.

Training condition	Synthetic SR↑	Steps to 40% SR↓	External macro SR↑	HPF↓
Raw + terminal	19.4	> 600K	24.7	–
Raw + dense	27.2	> 600K	27.0	0.27
Feasible-filtered raw + dense	36.4	> 600K	31.4	0.20
Verified + terminal	42.6	480K	33.5	–
Verified + endpoint-only	49.8	370K	36.6	0.18
Verified + coarse milestones	54.2	320K	38.0	0.14
Full	**58.7**	**280K**	**39.3**	**0.12**
Full + 10% predicate corruption	53.0	335K	37.2	0.19

Table 2: **Reward and task-selection controls.** Task counts are matched. The feasible-filtered condition additionally matches reported path complexity. External macro SR averages the three external benchmarks; HPF is the fraction of high-progress rollouts that fail. All values use the same train/validation/test split.

(Appendix C.2). The agreement across execution procedures supports the improvement in task executability.

5.3 Verification, Reward, and Repair Locality

Figure 4 compares learning dynamics. With terminal rewards fixed, verified environments improve PPO SR from 19.4 to 42.6 (+23.2 points). On raw environments, dense reward adds 7.8 points; on verified environments it adds 16.1. Their difference of differences is 8.3 points, indicating interaction between the training substrate and shaping. The full 39.3-point improvement combines the verification effect, shaping effect, and their interaction.

Feasible-task filtering reaches 36.4 SR (Table 2), above raw+dense but below verified+terminal. Repair consequently contributes beyond removing tasks that lack executable traces. Endpoint-only reward reaches 49.8, and two or three order-free milestones reach 54.2. Thus, the benefit does not

require the finest workflow predicates, while the full predicate set still adds 4.5 points over coarse shaping.

Paired raw/repaired summaries show edits to 18.6% of pages, 5.9% of DOM nodes, and 2.4% of task constraints. Among tasks executable both before and after repair, shortest successful traces increase from 8.8 to 9.1 steps; branching factor changes from 5.74 to 5.69. The paired results indicate local corrections with modest changes in workflow structure. Appendix C.12 reports structural diversity and the boundaries of this comparison.

Figure 5 separates the contribution of each component. Without feasibility verification, feasible tasks fall to 75.6% and PPO SR to 38.5. Without marker validation, violations rise from 2.8% to 14.3%; PPO SR is 48.1 despite 93.9% feasibility. Reachability is therefore insufficient when state changes remain unreliable. Scheduling and verifier coordination affect curation efficiency as well as final quality; the complete summary is in Appendix C.5.

	Defects ↓	Feasible (%) ↑	PPO SR (%) ↑	Viol. (%) ↓	Time (min) ↓
Full method	**3.3 ±0.28**	**94.8 ±0.74**	**58.7 ±1.36**	**2.8 ±0.31**	**18.0 ±0.82**
No symbolic checks	6.0 ±0.68	82.1 ±1.48	47.3 ±2.18	8.7 ±0.81	16.0 ±0.92
No semantic verifiers	6.7 ±0.53	80.4 ±1.63	44.9 ±2.32	7.9 ±0.57	10.0 ±1.35
No consistency verifier	5.8 ±0.38	84.3 ±1.09	48.8 ±2.24	6.8 ±1.27	15.0 ±1.67
No feasibility verifier	4.9 ±0.37	75.6 ±1.91	38.5 ±2.42	5.2 ±1.24	14.0 ±1.91
No DTCP	4.5 ±0.50	88.1 ±1.62	52.6 ±2.06	4.9 ±1.24	22.0 ±1.06
No PWRS	3.9 ±0.68	90.2 ±1.13	54.0 ±2.07	3.7 ±0.83	25.0 ±2.26
No marker validation	3.4 ±0.55	93.9 ±2.20	48.1 ±1.86	14.3 ±1.74	16.0 ±0.77
No dense reward	3.3 ±0.34	94.8 ±0.93	42.6 ±2.08	2.8 ±0.43	18.0 ±0.96

Direction-corrected degradation

Best — Worst

Figure 5: **Component ablation matrix.** Each row removes one component from the full method. Cells show means and 95% confidence intervals; blue-to-purple shading indicates better-to-worse values within each metric. Feasibility checks, marker validation, and dense rewards affect distinct parts of the training signal.

5.4 Defect Impact on Policy Learning

Component ablations identify which checks matter; controlled defect reintroduction examines the learning damage associated with the errors they address. Figure 6 relates each defect type's occurrence frequency in raw scaffolds and task-blocking rate to its decrease in PPO success.

Feasibility defects produce the largest average success decrease, at 24.3 percentage points, followed by marker defects (17.3), structural defects (14.4), consistency defects (10.3), and semantic defects (4.7). Occurrence frequency alone does not track learning damage: semantic defects are more frequent in the plotted observations but cause smaller drops, whereas feasibility and marker defects are less frequent and more damaging. This pattern complements the component ablations: task reachability and reliable state writes are central to useful training supervision. It also motivates the repair scheduler's emphasis on blocking defects and their dependencies, beyond raw defect counts.

5.5 External DOM Transfer and Policy Capacity

We evaluate 180 WebArena-compatible, 500 WebShop, and 560 MiniWoB++ tasks with the same serialized-DOM action interface and original success evaluators. The WebArena subset excludes private credentials, file uploads, and visual-only information. Compact policies receive no external fine-tuning. These conditions test reusable DOM interaction skills.

Relative to raw+dense PPO, the full method gains 6.2, 14.2, and 16.5 points on the three benchmarks (Table 3). Their unweighted macro gain is 12.3 points. The smaller WebArena gain is consistent with a harder transfer setting: its reported successful traces average 15.2 actions, versus 9.1 in the repaired synthetic suite. The common interface keeps action representation fixed across construction conditions.

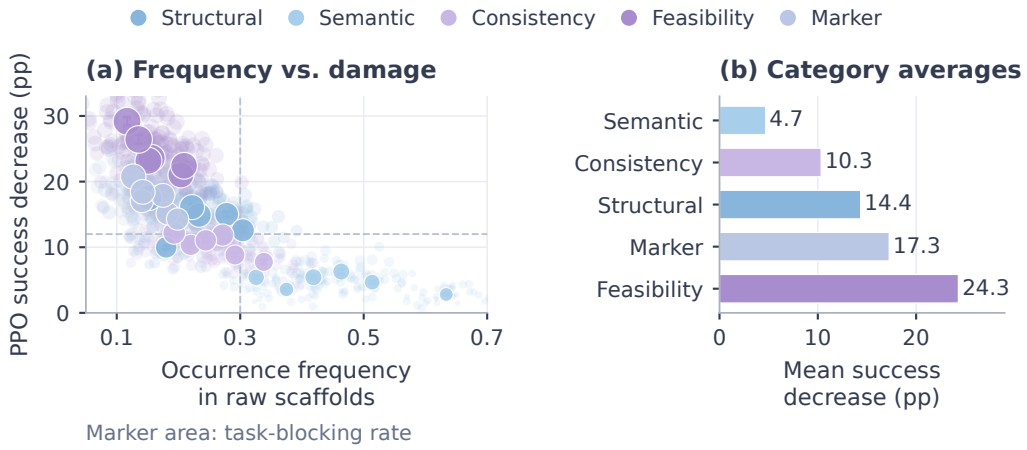


Figure 6: **Defect impact on policy learning.** (a) Defect occurrence frequency versus PPO success-rate decrease under defect injection, with marker size indicating task-blocking rate and color denoting category. (b) Average success-rate decrease by defect category.

Method	WebArena-compatible	WebShop	MiniWoB++
GPT-4 direct (OpenAI, 2024)	$10.6^{\pm 2.1}$	$32.5^{\pm 2.7}$	$41.2^{\pm 2.9}$
GPT-4 ReAct (Yao et al., 2023b)	$\underline{15.3}^{\pm 2.6}$	$\underline{38.7}^{\pm 2.9}$	$47.6^{\pm 3.1}$
Synthetic behavior cloning	$11.9^{\pm 2.3}$	$30.4^{\pm 2.5}$	$39.8^{\pm 2.8}$
Raw + dense PPO	$12.4^{\pm 2.4}$	$29.6^{\pm 2.5}$	$38.9^{\pm 2.8}$
Verified + terminal PPO	$14.7^{\pm 2.5}$	$37.2^{\pm 2.8}$	$\underline{48.5}^{\pm 3.1}$
Full PPO	$\mathbf{18.6}^{\pm 2.8}_{+3.3}$	$\mathbf{43.8}^{\pm 3.0}_{+5.1}$	$\mathbf{55.4}^{\pm 3.1}_{+6.9}$

Table 3: **External success rates (%).** Superscripts give 95% intervals; green subscripts compare with the strongest baseline in each column. All compact policies use zero policy-side LLM calls.

The same raw/full comparison extends to a 15M BiLSTM policy and a 42M four-layer Transformer policy. Their external macro gains are 12.2 and 12.1 points; the 8.9M policy gains 12.3. This consistency supports an environment-quality effect across the three compact architectures. Architecture details and per-benchmark results are in Appendices B.2 and C.16.

The artifact audit finds no exact task matches, no flagged 5-word-shingle near duplicates, and no exact canonical DOM collisions against the synthetic training set. The near-duplicate threshold is Jaccard similarity 0.8. Appendix B.8 describes normalization, 128-permutation MinHash retrieval, exact candidate verification, and descriptive embedding similarity. The audit characterizes detectable overlap in the generated training artifacts.

Operation-level evaluation further characterizes the transferred skills. A primitive is identified by its operation name, entity type, arguments, output fields, and invariants; a workflow records dependencies among primitives. On tasks combining seen primitives in novel compositions, full PPO reaches 47.2 SR versus 31.5 for raw+dense. The corresponding rates are 35.0 versus 21.2 for partially unseen primitives and 30.8 versus 18.0 for held-out domains. Seen primitives with seen compositions reach 53.2 versus 40.8. The largest gain occurs when familiar operations must be composed in a new order, suggesting that coherent stateful training supports reusable workflow components. Appendices B.6 and C.15 give the bucket definitions and task counts.

5.6 Execution Reliability and Simulation Cost

Reliability involves both identifying defective scaffolds and executing repaired ones. On the labeled audit of 80 raw environments and 1,286 defects, the full verifier obtains 91.5% precision, 87.2% recall, and 89.3 F1. Rule-based checks attain 96.1% precision but 39.4% recall. Combining deterministic checks with semantic reports therefore increases coverage while retaining a high rate of valid findings.

Simulator	Calls↓	Tokens (K)↓	Fidelity↑	Viol.↓	Episodes/h↑
Step-wise LLM	17.8	22.4	**97.1**	2.5	210
Deterministic-only	**0**	**0**	71.5	**0**	**5,400**
Unconstrained event-driven	2.3	3.6	88.2	10.7	1,550
Constrained event-driven	2.3	3.9	96.3	2.8	1,820

Table 4: **Synthetic rollout cost and fidelity.** Calls and tokens are per episode; fidelity and state violations are percentages. Throughput measures complete rollout execution, including simulator work.

Repair success is 90.8%, compared with 84.6% for AutoGen. Appendices C.3 and C.2 report full accuracy and replay comparisons.

Under a comparable API budget, our pipeline reaches 87.4% feasibility with 42.5K tokens per environment, versus 70.4% with 41.8K tokens for the single-pass baseline. The full setting uses 116.4K tokens for 94.8% feasibility. Targeting typed defects and their dependencies improves curation quality within a similar token allowance; a larger budget enables further repairs.

During training, event-driven simulation concentrates LLM work at persistent writes. Table 4 compares this design with step-wise state generation and with two controls. Constrained event updates use 2.3 calls and 3.9K tokens per episode, reducing these costs by 87.1% and 82.6% relative to step-wise simulation. State fidelity is 96.3% versus 97.1%, while throughput rises from 210 to 1,820 episodes per hour. Removing validation produces 10.7% state violations and 88.2% fidelity, showing why sparse calls still need an explicit update contract. A deterministic-only simulator is faster but reaches only 71.5% fidelity because required backend changes are omitted.

Reward diagnostics provide a complementary check on the learning signal. State-grounded progress has AUROC 0.87 and Spearman correlation 0.58 with terminal success, compared with 0.78 and 0.45 for an LLM judge. The high-progress failure rate falls from 0.29 to 0.12. Together with the terminal-reward and predicate-granularity controls, these results connect coherent transitions to more informative intermediate supervision. Appendices C.5, C.7, and C.8 give the complete component, simulator, and reward diagnostics.

5.7 Fallible Verification and Noisy Interaction

On the 100-environment perturbation pool, dropping 10% of true defect reports reduces PPO SR to 53.6; adding 10% spurious reports gives 55.2; faulty local repairs give 52.5. At 20% mixed error, SR falls to 39.6 and feasibility to 77.1%. The decrease links accumulated construction errors to weaker learning outcomes. The error protocol and full results are in Appendices B.7 and C.10.

Controlled DOM perturbations likewise leave a substantial gap. The full policy's four-suite macro SR falls from 44.1 clean to 27.2 under combined noise. Noise-randomized training reaches 34.0 under the same combined condition, with clean SR 43.7. The largest remaining challenges involve delayed updates and stale observations; training with controlled noise improves every individual perturbation condition.

The scale study covers 20–200 pages per site. Curation grows from 18.0 to 181.6 minutes per environment, while time per page remains 0.81–0.91 minutes and feasibility decreases from 94.8% to 92.2%. The cost grows approximately with page count over this fixed-horizon workload.

6 Conclusion

Reliable synthetic training requires agreement among reachable goals, valid state changes, and success criteria. Our framework combines bounded scaffold verification, local repair, constrained persistent writes, and state-derived rewards. Controlled comparisons show improved task executability and compact-policy learning beyond terminal-reward training and feasible-task filtering. Transfer and stress tests support reusable DOM-grounded skills, while specification correctness and realistic web dynamics remain limitations. Richer transition models, auditable execution records, and stronger policies are needed to extend the approach.

AI USE STATEMENT

Large language models are components of the proposed system: they generate synthetic web scaffolds, support semantic verification and repair, and propose constrained state deltas during synthetic rollouts. We also used large language models to refine method exposition and algorithm specifications, formulate and write the replay-certificate claim and its proof, organize and interpret experimental results, and polish manuscript language. The authors are responsible for the final content, including AI-assisted text and presentation of results.

REFERENCES

Cristian Cadar, Daniel Dunbar, and Dawson Engler. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *USENIX Symposium on Operating Systems Design and Implementation*, 2008. URL https://static.usenix.org/events/osdi08/tech/full_papers/cadar/cadar_html/index.html.

Hyungjoo Chae, Sunghwan Kim, Junhee Cho, Seungone Kim, Seungjun Moon, Gyeom Hwangbo, Dongha Lim, Minjin Kim, Yeonjun Hwang, Minju Gwak, et al. Web-shepherd: Advancing prms for reinforcing web agents. *arXiv preprint arXiv:2505.15277*, 2025.

Zhaorun Chen, Zhuokai Zhao, Kai Zhang, Bo Liu, Qi Qi, Yifan Wu, Tarun Kalluri, Sara Cao, Yuanhao Xiong, Haibo Tong, Huaxiu Yao, Hengduo Li, Jiacheng Zhu, Xian Li, Dawn Song, Bo Li, Jason Weston, and Dat Huynh. Scaling agent learning via experience synthesis, 2025. URL https://arxiv.org/abs/2511.03773.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web, 2023. URL https://arxiv.org/abs/2306.06070.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*, 2024.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6864–6890, 2024.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 881–905, 2024.

Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. GenProg: A generic method for automatic software repair. *IEEE Transactions on Software Engineering*, 38(1):54–72, 2012. doi: 10.1109/TSE.2011.104.

Thibault Le Sellier De Chezelles, Maxime Gasse, Alexandre Drouin, Massimo Caccia, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Quentin Cappart, Graham Neubig, Ruslan Salakhutdinov, Nicolas Chapados, and Alexandre Lacoste. The BrowserGym ecosystem for web agent research. *arXiv preprint arXiv:2412.05467*, 2024. URL https://arxiv.org/abs/2412.05467.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration, 2018. URL https://arxiv.org/abs/1802.08802.

OpenAI. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. Autonomous evaluation and refinement of digital agents. *arXiv preprint arXiv:2404.06474*, 2024.

Ajay Patel, Markus Hofmarcher, Claudiu Leoveanu-Condrei, Marius-Constantin Dinu, Chris Callison-Burch, and Sepp Hochreiter. Large language models can self-improve at web agent tasks, 2024. URL https://arxiv.org/abs/2405.20309.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. Agent q: Advanced reasoning and learning for autonomous ai agents. *arXiv preprint arXiv:2408.07199*, 2024.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. *arXiv preprint arXiv:2411.02337*, 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3135–3144. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/shi17a.html.

Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*, 22(5):297–312, 2012. doi: 10.1002/stvr.456.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL https://arxiv.org/abs/2203.11171.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023. URL https://arxiv.org/abs/2308.08155.

Chunqiu Steven Xia and Lingming Zhang. Conversational automated program repair, 2023. URL https://arxiv.org/abs/2301.13246.

Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor, Pratik Chaudhari, George Karypis, and Huzefa Rangwala. Agentoccam: A simple yet strong baseline for llm-based web agents. *arXiv preprint arXiv:2410.13825*, 2024.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents, 2023a. URL https://arxiv.org/abs/2207.01206.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023b. URL https://arxiv.org/abs/2210.03629.

Abhay Zala, Jaemin Cho, Han Lin, Jaehong Yoon, and Mohit Bansal. Envgen: Generating and adapting environments via llms for training embodied agents, 2024. URL https://arxiv.org/abs/2403.12014.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024. URL https://arxiv.org/abs/2307.13854.

APPENDIX CONTENTS

Technical specifications, evaluation protocols, additional results, and the replay-certificate proof are organized below. Click any entry to navigate.

A Technical Details 13

A.1 Defect Taxonomy . . . 13
A.2 Canonical Parsing and Check Interfaces . . . 13
A.3 Verifier Coordination and Local Repair . . . 13
A.4 Bounded Trace Analysis . . . 14
A.5 Event-Driven State Updates . . . 14
A.6 Reward Compilation and PPO . . . 15
A.7 Verifier Prompt Interfaces . . . 16

B Experimental Details 18

B.1 Diagnostic Pool and Measurements . . . 18
B.2 Models, Hyperparameters, and Compute . . . 18
B.3 Construction Baselines and Evaluation Metrics . . . 19
B.4 Reward and Task-Selection Controls . . . 19
B.5 Independent Execution and Defect Audits . . . 19
B.6 External Benchmark and Transfer Protocols . . . 20
B.7 Perturbation and Scale Protocols . . . 20
B.8 Structural and Overlap Audit Protocols . . . 20
B.9 Reward and Failure Diagnostic Metrics . . . 21

C Additional Experimental Results 22

C.1 Motivating Diagnostics . . . 22
C.2 Independent Execution and Replay Audits . . . 22
C.3 Verifier and Repair Reliability . . . 23
C.4 LLM Budget Comparison . . . 23
C.5 Complete Component Comparison . . . 24
C.6 Reward-Control Effects . . . 24
C.7 Simulator Cost and State Fidelity . . . 24
C.8 Reward Alignment . . . 25
C.9 Policy Failure Attribution . . . 26
C.10 Verifier and Repair Error Injection . . . 27
C.11 Scaling Environment Size . . . 28
C.12 Repair Locality and Workflow Diversity . . . 28
C.13 Noisy DOM Interaction . . . 29
C.14 Artifact Overlap Audit . . . 30
C.15 Operation and Composition Analysis . . . 30
C.16 Policy Backbone Comparison . . . 31
C.17 Transfer Inference Cost . . . 31
C.18 Qualitative Repair Example . . . 32

D Proofs 32

D.1 Replay Certificate Guarantee . . . 32

A TECHNICAL DETAILS

A.1 DEFECT TAXONOMY

Defects are localized to a page, DOM node, database field, marker, or task. Table 5 gives the categories used throughout the paper.

Category	Example	Check or evidence
Structural	Broken checkout link; missing submit button	Resolve declared targets, traverse from the entry page, and check required controls.
Semantic	Invalid price, date, or content field	Compare field types, schema constraints, and page-level semantics.
Consistency	Different prices for one product	Compare every binding of an entity attribute with its canonical database record.
Feasibility	A required workflow cannot complete	Replay a bounded witness; retain search coverage and the first failing transition.
Marker	Undeclared write or invalid operation arguments	Validate the trigger, precondition, read/write sets, and operation schema.

Table 5: Defect categories and operational evidence. A failed non-exhaustive trace search leaves feasibility unresolved.

A.2 CANONICAL PARSING AND CHECK INTERFACES

A canonical page contains a DOM tree, interactive nodes, and bindings from rendered attributes to typed records. Stable page and entity identifiers connect the navigation graph, task predicates, and marker specifications. Parsing resolves each binding to its source field and each interaction to either a deterministic operation or a state-change marker.

The symbolic checker tests five properties: (i) every declared navigation target exists; (ii) required pages are reachable from the entry page; (iii) initialized records satisfy types and schema constraints; (iv) rendered bindings refer to valid entities and fields; and (v) marker triggers, argument schemas, and read/write declarations are well formed. Each failed check emits its location and a concrete counterexample. Dynamic execution checks complement these structural tests.

A semantic report is

$$d = (\mathrm{type}, \mathrm{loc}, \mathrm{obj}, \mathrm{sev}, \mathrm{ev}, c), \qquad \mathrm{sev}, c \in [0, 1]. \tag{10}$$

The evidence field may contain an invalid link, conflicting values, a missing workflow control, or a failed transition. The canonical key $(\mathrm{type}, \mathrm{loc}, \mathrm{obj})$ merges duplicate reports while retaining their evidence. Confidence is the weighted mean in Eq. (4), with nonnegative reliability weights and positive total weight for each report; a report accepted through the severity branch still passes the same repair validation.

A.3 VERIFIER COORDINATION AND LOCAL REPAIR

The four semantic roles inspect structure, content semantics, cross-page consistency, and task feasibility. Defect-triggered coordination sends a targeted cross-check when a report affects another role: structural defects trigger feasibility checks; semantic defects trigger consistency checks; consistency defects trigger feasibility checks; and feasibility defects trigger structural and consistency checks. The target receives the defect key, local evidence, affected objects, and proposed dependency. One cross-check round is used. Its response is merged into the original report set before Eq. (4) is evaluated.

The accepted defect graph orders repairs. A schema correction precedes dependent bindings; a marker or navigation correction precedes affected task replays. If $\mathcal{S}_{\mathrm{done}}$ is the set of resolved defects, the ready set is

$$\mathrm{Ready} = \{d \in \Delta_{\mathrm{acc}} : \mathrm{Pred}(d) \subseteq \mathcal{S}_{\mathrm{done}}\}. \tag{11}$$

The scheduler selects the highest score from Eq. (6) in this set. Structural repair restores links, pages, and controls; semantic repair fixes invalid content; consistency repair propagates a canonical record

value; feasibility repair restores required workflow steps; marker repair corrects preconditions and operation schemas. The task's intended operation constrains all five families.

Revalidation follows dependency closure over touched objects. It includes adjacent navigation edges, all dependent DOM bindings, markers, and tasks, even when a dependency lies on a distant page. A proposed edit is committed only after these checks; otherwise the previous scaffold is retained. Each certificate records its scaffold version so that a later edit cannot silently inherit a stale witness. An empty ready set with residual defects indicates an unresolved dependency cycle. The bounded repair loop returns the residual defects and task statuses when it reaches its budget.

A.4 Bounded Trace Analysis

A positive certificate stores the initial state, environment version, action sequence, realized state deltas, and terminal predicate result. Replaying the stored deltas makes the certified execution explicit even when a generative writer can produce different admissible outputs on a future call. Search explores candidate DOM actions to depth $H_{\max} = 40$; action pruning and state deduplication reduce the explored space.

Algorithm 2 Bounded witness search with explicit coverage status

Require: Environment $\mathcal{E}$, task t, tested initial states $\mathcal{S}_0$, horizon H, budget K
Initialize FIFO queue with $(s_0, [], 0)$ for $s_0 \in \mathcal{S}_0$; set coverage flag $q \leftarrow \mathrm{true}$
while queue nonempty and expansion budget remains **do**
  Pop (s, π, h); return WITNESS and replay log if $C_t(s) = 1$
  if $h < H$ **then**
    Enumerate candidate actions; set $q \leftarrow \mathrm{false}$ if an admissible action is pruned
    for each retained action a **do**
      Execute and log (s, a, s'); set $q \leftarrow \mathrm{false}$ if transition alternatives are omitted
      Enqueue $(s', \pi \circ a, h + 1)$ unless its complete task-relevant key was visited at an equal or smaller depth
    end for
  end if
end while
if queue empty and $q = \mathrm{true}$ and all declared starts are covered **then**
  return BOUNDED INFEASIBLE over the declared finite transition space
else
  return UNRESOLVED, expansion log, and coverage limits
end if

A deduplication key must preserve the database, session, current page, and every history component used by transitions or the completion predicate. Two identical rendered DOMs may represent different backend states and cannot in general be merged. Breadth-first search may discard a repeated complete state reached at greater depth, because the earlier visit retains at least as much remaining horizon. For history-dependent tasks, history belongs in the complete state.

With branching factor at most b, exhaustive search can require $\sum_{h=0}^{H} b^h$ states. Deduplication and pruning improve practical cost but do not remove this worst case. Exhaustive negative certification additionally requires a finite, fully enumerated set of starts and transitions. For sampled starts or a stochastic writer with unenumerated outputs, the positive witness remains useful while unsuccessful search has the unresolved status.

Proposition 1 formalizes the successful-replay guarantee; its proof is given in Appendix D.1.

A.5 Event-Driven State Updates

The deterministic transition handles page navigation, scroll position, menu state, local input editing, and rendering. For a persistent event, the writer receives

$$z_k = \mathrm{Pack}(p_k, b_{k,R_m}, \sigma_k, a_k, \Omega_m), \qquad \hat{\delta}_k = G_\psi(z_k), \tag{12}$$

where b_{k,R_m} contains the declared readable records. The structured output contains field-level writes and declared session updates. Validation uses separate read and write restrictions:

$$\mathcal{U}_m(s,a) = \{\delta : \mathrm{pre}_m(s,a) = 1, \quad \mathrm{Read}(\delta) \subseteq R_m, \quad \mathrm{Write}(\delta) \subseteq W_m, \\ b \oplus \delta \models \mathcal{D}, \quad \mathrm{Inv}(s \oplus \delta) = 1\}. \tag{13}$$

Read access alone never grants write access. Invariants include type validity, entity identity, permissions, and task-independent workflow constraints. An accepted delta is applied atomically and rendered; a rejected delta leaves the previous state unchanged. The log records the marker, candidate write, validation outcome, and resulting backend state.

A.6 Reward Compilation and PPO

Task predicates query database fields, session state, and bindings between the rendered page and backend records. A purchase task may check the target item, quantity, valid delivery fields, and placed-order status. The actor receives DOM observations and task text; these predicates serve as privileged training feedback.

Given candidate scores ℓ_θ, the policy normalizes over currently available actions:

$$\pi_\theta(a \mid o, x) = \frac{\exp \ell_\theta(a; o, x)}{\sum_{a' \in \mathcal{A}(o)} \exp \ell_\theta(a'; o, x)}. \tag{14}$$

The action mask removes nonexistent DOM actions. A valid candidate can still cause a rejected persistent write, which incurs the penalty in Eq. (9).

Let ξ be the discount factor and d_k the episode termination indicator. Generalized advantage estimation uses

$$\delta_k^V = r_k + \xi(1 - d_k) V_{\theta_{\mathrm{old}}}(o_{k+1}, x) - V_{\theta_{\mathrm{old}}}(o_k, x), \tag{15}$$

$$\hat{A}_k = \sum_{\ell \geq 0} (\xi \lambda_{\mathrm{GAE}})^\ell \delta_{k+\ell}^V, \tag{16}$$

with the sum stopped at episode termination. PPO optimizes the clipped objective

$$L^{\mathrm{clip}}(\theta) = \mathbb{E}_k \Big[\min \Big(r_k^\pi \hat{A}_k, \mathrm{clip}(r_k^\pi, 1 - \varepsilon, 1 + \varepsilon) \hat{A}_k \Big) \Big], \quad r_k^\pi = \frac{\pi_\theta(a_k \mid o_k, x)}{\pi_{\theta_{\mathrm{old}}}(a_k \mid o_k, x)}. \tag{17}$$

A value loss and entropy term accompany this objective. The ratio r_k^π is distinct from the environment reward r_k. The shaping term in Eq. (9) is an undiscounted potential difference. Because training uses $\xi = 0.99$, its effect is evaluated through reward ablations rather than a discounted policy-invariance claim.

A.7 VERIFIER PROMPT INTERFACES

The verifier roles consume canonical pages, referenced records, markers, and task specifications. They return localized defect reports with evidence; coordination passes the affected objects and report key. Figures 7–10 preserve the supplied role prompt templates for structural, data-consistency, cross-page navigation, and task checks. Their reports enter the same canonical validation and repair interface.

```
You are a web structure verification expert.
Analyze the DOM tree and identify:
1. Broken links (href pointing to non-existent pages)
2. Missing required elements (forms without submit buttons)
3. Invalid HTML nesting (interactive elements inside links)
4. Orphaned elements (elements with invalid parent refs)

DOM Tree:
{dom_tree}

Page Graph:
{page_graph}

Output format: JSON with defect_type, location, severity, evidence
```

Figure 7: Structural verifier prompt template.

```
You are a data consistency verification expert.
Check for contradictions between:
1. Displayed prices vs. cart totals
2. Stock status vs. add-to-cart availability
3. User session data vs. displayed user info
4. Form field constraints vs. submitted values

Page State:
{page_state}

Data Elements:
{data_elements}

Output format: JSON with inconsistency_type, elements, expected, actual
```

Figure 8: Data-consistency verifier prompt template.

Cross-page and workflow checks. These templates check navigation consistency and task-level completion paths.

```
You are a cross-page navigation verification expert.
Verify consistency across page transitions:
1. Navigation links lead to correct destinations
2. Breadcrumb trails are accurate
3. Back button behavior is correct
4. Session state persists correctly across pages

Source Page: {source_page}
Target Page: {target_page}
Navigation Action: {action}

Output format: JSON with navigation_issue, source, target, expected_behavior
```

Figure 9: Cross-page navigation verifier prompt template.

```
You are a task completability verification expert.
Verify that tasks can be completed end-to-end:
1. All required steps are reachable
2. No blocking errors in critical paths
3. Form submissions lead to expected outcomes
4. Task success criteria can be achieved

Task Description: {task}
Available Actions: {actions}
Current State: {state}
Goal State: {goal}

Output format: JSON with blocking_issue, step, reason, suggested_fix
```

Figure 10: Task-feasibility verifier prompt template.

B Experimental Details

B.1 Diagnostic Pool and Measurements

The diagnostic pool consists of 500 generated environments and 6,842 tasks across e-commerce, social media, banking, education, healthcare, and government. Each environment contains initialized records, a navigation graph, interactive elements, and task completion predicates.

For environment $\mathcal{E}$, the count in category c is $N_c(\mathcal{E}) = \sum_{d \in \Delta(\mathcal{E})} \mathbf{1}[\mathrm{type}(d) = c]$. Blocking attribution examines the earliest defective transition on an attempted task workflow. When several categories apply, the diagnostic order is feasibility, structural, consistency, and semantic. A category-level blocking rate divides attributed tasks by the number of tasks in that environment; environment averages give each site equal weight. This attribution order assigns a primary label and is distinct from a causal decomposition of interacting defects.

The marker diagnostic uses 1,800 successful trajectories. For $\tau = (s_0, a_0, \ldots, s_H)$ and the marker lookup $\mu(p_k, a_k)$, it records

$$U(\tau) = \sum_{k=0}^{H-1} \mathbf{1}[\mu(p_k, a_k) \neq \varnothing], \qquad M(\tau) = U(\tau)/H. \tag{18}$$

Navigation, scrolling, menu expansion, and local text input contribute to the trajectory length without requiring persistent writes. These statistics motivate event-driven simulation. Conditioning this diagnostic on successful trajectories makes it a description of completed workflows.

B.2 Models, Hyperparameters, and Compute

The generator, parser, checkers, repairs, simulator, and training pipeline use Python. The main actor contains 8.9M parameters: a two-layer bidirectional GRU instruction encoder, an attribute-aware DOM encoder, and an MLP action scorer. All construction baselines start from the same raw scaffolds and train this actor unless a backbone comparison is specified. The generator uses `gpt-4-0613` at temperature 0.7; semantic verification, cross-checks, and state writing use temperature 0. The writer returns structured JSON updates constrained by the marker schema.

Setting	Value	Setting	Value
Training steps	10^6	Rollout steps/update	4,096
Minibatch size	512	PPO epochs/update	4
Learning rate	3×10^{-4}	Optimizer	Adam
Discount ξ	0.99	GAE λ_{GAE}	0.95
PPO clipping ε	0.2	Gradient clipping	0.5
Value-loss coefficient	0.5	Entropy coefficient	0.01
Reward α, β, η	$0.5, 0.2, 0.01$	Episode horizon	40
Policy seeds	3	Train/validation/test sites	350/75/75

Table 6: Fixed policy-learning configuration. Advantages are normalized within each PPO batch. Hyperparameters are selected on validation environments.

A training job uses one NVIDIA A10G (24GB), 16 CPU cores, 64GB RAM, and about 200GB scratch storage. One run takes approximately 3.2 GPU-hours. Generation and verification use CPU workers and LLM APIs; actor optimization uses the GPU with CPU rollout workers. The full method's curation cost is 18 minutes and 116.4K input/output tokens per environment. Simulator costs are measured separately in Table 14.

Policy backbones. The backbone study keeps raw/full construction conditions and evaluation tasks fixed. Policy A is the 8.9M BiGRU actor used in the main experiments. Policy B is a 15M, two-layer bidirectional LSTM with 256 hidden units per direction and a bilinear action scorer. Policy C is a 42M, four-layer Transformer with eight attention heads and cross-attention between instruction and DOM representations. Each actor scores the same candidate action set and is trained under the common PPO protocol.

B.3 Construction Baselines and Evaluation Metrics

No verification uses the raw generated scaffold. Rule-based verification applies deterministic graph, schema, and binding checks. Single-LLM verification uses one inspection-and-repair pass. Self-consistency aggregates repeated checks. The AutoGen baseline uses multi-agent inspection and repair (Wu et al., 2023). Our method combines symbolic checks, role-specific reports, defect-triggered cross-checks, dependency-aware scheduling, and local revalidation. All conditions share the same site specifications and raw inputs.

Environment metrics include defects per site, task-blocking defects per site, witnessed feasible-task percentage, author-executed success, state-violation percentage, and curation time. Policy SR is the percentage of tasks meeting the terminal predicate. The original policy evaluations use three seeds and present 95% intervals where available. Bold/underline ranks point estimates; interval overlap is not a significance test. The figures in the main paper display aggregate anchors and ablation summaries.

Simulator variants. The simulator comparison uses the same task distribution and distinguishes four transition implementations. Step-wise LLM regenerates state at every interaction; deterministic-only suppresses generative persistent updates; unconstrained event-driven simulation invokes the writer at marked events without validation; the full event-driven simulator validates each proposed delta.

B.4 Reward and Task-Selection Controls

Table 2 compares task-count-matched conditions on the same site split. Raw + terminal and verified + terminal retain the same terminal success signal. Raw + dense and full use the state-progress reward in Eq. (9). Feasible-filtered raw removes tasks without a witness and additionally matches the path-complexity distribution; it leaves the retained scaffolds unrepaired.

Endpoint-only shaping evaluates backend endpoint subconditions. Coarse shaping uses two or three order-free state milestones. The full reward uses the finer compiled predicate set. Predicate corruption perturbs 10% of that set while retaining the original completion evaluator. These variants change the training feedback without exposing predicate state to the actor. Steps to 40% SR are measured within the 600K-step learning comparison; > 600K denotes a threshold not attained in that window.

B.5 Independent Execution and Defect Audits

The original author-executed diagnostic in Table 1 is complemented by an independent blinded study. Three paid evaluators who were not paper authors execute 120 tasks per construction method, balanced across the six domains. Two evaluators independently execute each task. They see the instruction and visible interface, but not the construction method, backend completion predicate, reference trace, or study hypothesis. Each attempt has a budget of 40 actions and 10 minutes. Success is computed automatically from the frozen backend predicate.

The third evaluator adjudicates infrastructure or protocol issues; 5.4% of runs require such adjudication. This procedure separates human execution from the author diagnostic and uses a fixed outcome evaluator.

Trace and replay audit. The executability audit evaluates 120 environments and 720 tasks with a separately implemented bounded analyzer, using a different action enumerator and search ordering. Author-audited replay covers a randomly selected subset of 240 tasks. Neither audit is used for repair or model selection. The false-feasible diagnostic records internal positive decisions rejected by both the held-out analyzer and the author audit.

Labeled defect audit. The labeled defect audit contains 80 raw environments and 1,286 defects. Precision, recall, and F1 measure defect identification. Repair success measures whether a proposed correction resolves its target under validation; false repair measures erroneous modifications.

B.6 External Benchmark and Transfer Protocols

A shared adapter serializes the DOM and enumerates available element-grounded actions. Task instructions, benchmark transitions, and original completion evaluators are preserved. Compact policies are evaluated without external fine-tuning, backend predicates, verifier feedback, or policy-side LLM calls. GPT-4 direct and ReAct receive the same task and serialized DOM with temperature 0.

Suite	Tasks	Evaluation scope
WebArena-compatible	180	DOM-compatible tasks; excludes private credentials, file uploads, and visual-only tasks. The protocol uses a 20-action cap.
WebShop	500	Product search, navigation, and purchase under the original success evaluator.
MiniWoB++	560	Short-horizon DOM control tasks with original task termination and success checks.

Table 7: External evaluation subsets. The common interface defines the scope of the transfer comparison.

Operation and composition strata. An operation signature records its name, entity type, arguments, output fields, and invariants. A workflow is an ordered dependency graph of these signatures. The diagnostic separates seen primitives with seen compositions, seen primitives with novel compositions, partially unseen primitives, and held-out domains. Domain status takes precedence when assigning the mutually exclusive buckets.

B.7 Perturbation and Scale Protocols

Verifier and repair errors. The perturbation study uses the same pool of 100 environments for all conditions. False-negative (FN) injection drops a specified fraction of true defect reports; false-positive (FP) injection adds spurious reports relative to the corresponding report candidate set; faulty-repair injection corrupts a fraction of local repair candidates. Mixed injection combines these interventions at the stated rate in each candidate set. Environment checks and policy evaluation follow the injected construction run. Precision and recall concern defect reports, while repair success and false repair concern attempted edits.

Environment size. The scale experiment increases pages and tasks per site with a fixed trace horizon, candidate pruning, state deduplication, and incremental revalidation.

DOM noise. The stress suite perturbs the observation or interaction layer while preserving the underlying task objective. Conditions include DOM reordering, identifier renaming, irrelevant elements, modal interruptions, delayed updates, stale observations, and their combination. The full + noise-training variant adds perturbations during training. Evaluation averages success over held-out synthetic tasks, WebArena-compatible tasks, WebShop, and MiniWoB++; this four-suite macro differs from the external-only macro in Table 2.

B.8 Structural and Overlap Audit Protocols

Paired structural analysis. The paired comparison evaluates the same sites before and after repair. Edit fractions measure the share of pages, DOM nodes, navigation edges, or task constraints touched by repair. Template ratio and pairwise structural distances describe diversity. Path-related statistics use tasks executable in both versions, so that a missing raw path is not treated as a short path.

Artifact overlap. External benchmark tasks, DOM snapshots, records, and evaluator artifacts are excluded from environment generation, repair, reward design, hyperparameter selection, and model selection. The artifact audit compares the synthetic training set against the evaluated benchmark tasks using the following fixed representations.

Text. Task text is normalized with Unicode NFKC, lowercasing, HTML and punctuation removal, and whitespace normalization. Exact normalized matches are counted directly. Near-duplicate retrieval uses 5-word shingles and 128-permutation MinHash; retrieved candidates are checked with exact Jaccard similarity and flagged at similarity at least 0.8. Held-out paraphrase and unrelated pairs calibrate the near-duplicate check.

DOM and entities. DOM canonicalization removes visible text and volatile identifiers/classes while retaining tag names, ARIA roles, action types, and parent edges. SHA-256 hashes detect exact canonical-tree collisions. Entity matching normalizes benchmark-specific site URLs and entity identifiers, excluding generic terms such as "shirt" or "email."

Semantic similarity. A frozen `all-mpnet-base-v2` encoder produces 768-dimensional text embeddings. Cosine similarity supplies a descriptive nearest-neighbor statistic. It is not used as an exact-duplicate criterion.

B.9 Reward and Failure Diagnostic Metrics

Figure 13 compares surface interaction heuristics, an LLM judge, and state-grounded progress. It evaluates whether trajectory progress ranks terminal outcomes (AUROC and Spearman correlation), how closely progress scores track success frequencies (ECE), and how often high-progress trajectories fail:

$$\mathrm{HPF} = \Pr(C_t(s_H) = 0 \mid \Psi_t(s_H) > 0.8). \tag{19}$$

A progress score is a task-completion fraction; calibration evaluates its empirical association with success without assuming it is intrinsically a probability.

Failure diagnosis first checks certificate status, then state-write violations, then high-progress terminal failures. Remaining cases are reviewed for element grounding, workflow ordering, and horizon exhaustion. An unresolved search remains unresolved rather than being converted to environment invalidity. Likewise, a high-progress failure is an alignment warning rather than proof of a faulty reward. These labels help localize problems while preserving the distinctions between execution evidence and policy behavior.

The diagnostic assigns one primary category to each failed trajectory. The ordering checks environment validity, state-update violations, and high-progress terminal failures before inspecting grounding, planning, and timeout errors. The high-progress threshold is $\Psi_t > 0.8$.

C Additional Experimental Results

C.1 Motivating Diagnostics

Figure 11 summarizes defect prevalence, blocking rates, and update sparsity. Protocols and measurements are specified in Appendix B.1.

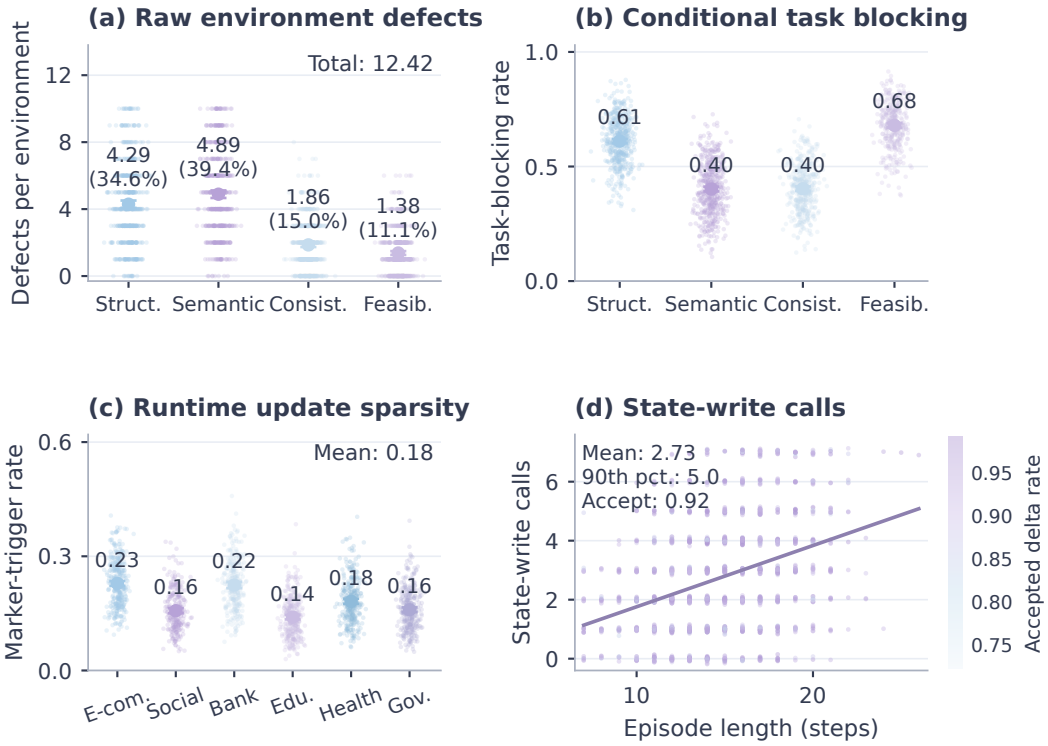


Figure 11: **Motivating diagnostics.** (a) Defect counts in raw generated environments. (b) Conditional task-blocking rates by defect category. (c) Marker-trigger rate across domains. (d) State-write calls by episode length. These diagnostics motivate checking task executability and invoking state writers only for marked events.

C.2 Independent Execution and Replay Audits

Table 8 reports the blinded human study, and Table 9 reports the separate trace and replay audit with 95% intervals. Both protocols are specified in Appendix B.5.

Method	Raw	Rule-based	Single-LLM	Self-consistency	AutoGen	Ours
Human SR (%)	38.3	29.2	55.8	70.0	87.5	**90.8**

Table 8: Independent visible-interface execution. Rates use the blinded study's protocol rather than the author-executed diagnostic.

Method	Internal feasible↑	Held-out feasible↑	Author replay↑	False feasible↓	Trace agreement↑
Raw	$48.6^{\pm 1.4}$	$46.1^{\pm 1.6}$	$44.8^{\pm 2.4}$	$12.4^{\pm 1.5}$	$83.2^{\pm 2.1}$
Rule-based	$53.2^{\pm 1.5}$	$51.4^{\pm 1.7}$	$49.6^{\pm 2.6}$	$10.7^{\pm 1.4}$	$84.9^{\pm 1.9}$
Single-LLM	$70.4^{\pm 1.3}$	$67.8^{\pm 1.5}$	$65.9^{\pm 2.2}$	$6.8^{\pm 1.1}$	$89.4^{\pm 1.6}$
Self-consistency	$79.6^{\pm 1.1}$	$77.2^{\pm 1.3}$	$75.4^{\pm 2.0}$	$5.2^{\pm 0.9}$	$91.1^{\pm 1.4}$
AutoGen	$\underline{88.7}^{\pm 0.9}$	$\underline{86.9}^{\pm 1.0}$	$\underline{84.8}^{\pm 1.9}$	$\underline{3.9}^{\pm 0.7}$	$\underline{93.8}^{\pm 1.1}$
Ours	$\mathbf{94.8}^{\pm 0.7}$	$\mathbf{92.9}^{\pm 0.9}$	$\mathbf{91.7}^{\pm 1.8}$	$\mathbf{2.3}^{\pm 0.6}$	$\mathbf{96.4}^{\pm 0.8}$

Table 9: Executability audit (%). The two analyzers and human replay share task specifications but use distinct execution procedures.

C.3 Verifier and Repair Reliability

Deterministic checks have the highest precision but substantially lower coverage, while the combined system improves recall and repair success. The labeled audit and metrics are described in Appendix B.5.

Method	Precision↑	Recall↑	F1↑	Repair success↑	False repair↓
Rule-based	$\mathbf{96.1}^{\pm 1.2}$	$39.4^{\pm 2.1}$	$56.0^{\pm 2.0}$	$71.8^{\pm 2.8}$	$\mathbf{1.2}^{\pm 0.4}$
Single-LLM	$78.4^{\pm 2.0}$	$66.7^{\pm 2.4}$	$72.1^{\pm 2.2}$	$74.3^{\pm 2.6}$	$8.9^{\pm 1.1}$
Self-consistency	$83.1^{\pm 1.8}$	$73.6^{\pm 2.2}$	$78.1^{\pm 2.0}$	$80.2^{\pm 2.4}$	$6.1^{\pm 0.9}$
AutoGen	$86.7^{\pm 1.6}$	$\underline{79.5}^{\pm 2.0}$	$\underline{82.9}^{\pm 1.8}$	$\underline{84.6}^{\pm 2.1}$	$4.8^{\pm 0.8}$
Ours	$\underline{91.5}^{\pm 1.4}$	$\mathbf{87.2}^{\pm 1.7}$	$\mathbf{89.3}^{\pm 1.5}$	$\mathbf{90.8}^{\pm 1.8}$	$\underline{2.7}^{\pm 0.6}$

Table 10: Verifier and repair reliability (%; 95% intervals). Ranking respects the direction of each metric.

Category	Precision	Recall	F1	Repair success
Symbolic	97.4	93.1	95.2	96.5
Structural	91.2	86.7	88.9	90.5
Semantic	88.6	84.2	86.3	87.1
Consistency	90.4	85.5	87.9	88.3
Feasibility	87.8	91.6	89.7	92.1
Marker	92.3	88.9	90.6	93.4

Table 11: Reliability by defect category for the full method (%). Categories are diagnostic strata rather than competing methods.

C.4 LLM Budget Comparison

At approximately the single-pass token budget, the combined checker obtains 87.4% feasibility with 42.5K tokens per environment, compared with 70.4% at 41.8K tokens for Single-LLM. The full setting increases the budget to 116.4K tokens and obtains 94.8%. Thus, both how checks are allocated and the total available budget affect curation quality.

Method	Calls	Tokens (K)	Minutes	Defects	Feasible	False feasible
Rule-based	0	0	0.5	9.8	53.2	10.7
Single-LLM	8.1	41.8	12.0	6.2	70.4	6.8
Self-consistency	40.0	207.4	35.0	5.8	79.6	5.2
AutoGen	29.4	154.2	28.0	5.1	88.7	3.9
Ours, comparable budget	8.3	42.5	11.5	4.7	87.4	3.4
Ours, full	22.6	116.4	18.0	3.3	94.8	2.3

Table 12: Per-environment curation cost and quality. The comparable budget differs from Single-LLM by 1.7% in tokens; feasible and false-feasible rates are percentages.

C.5 Complete Component Comparison

Figure 5 in the main text visualizes the component comparison; Table 13 lists the aggregate values. Each row removes one component while retaining the remaining pipeline. Symbolic, semantic, cross-page consistency, and task-feasibility checks address different error sources. DTCP controls cross-check routing; PWRS controls dependency-aware repair order. Marker validation and dense rewards operate during interaction and learning.

Variant	Defects↓	Feasible↑	PPO SR↑	State viol.↓	Minutes↓
Full	**3.3**	**94.8**	**58.7**	**2.8**	18
No symbolic checks	6.0	82.1	47.3	8.7	16
No semantic verification	6.7	80.4	44.9	7.9	**10**
No consistency verification	5.8	84.3	48.8	6.8	15
No feasibility verification	4.9	75.6	38.5	5.2	14
No DTCP	4.5	88.1	52.6	4.9	22
No PWRS	3.9	90.2	54.0	3.7	25
No marker validation	3.4	93.9	48.1	14.3	16
No dense reward	**3.3**	**94.8**	42.6	**2.8**	18

Table 13: Full component summary. Rates are percentages, defects are per environment, and curation time is in minutes. Tied values share the same rank.

The feasibility ablation produces the largest loss in PPO success among the listed variants. The marker ablation leaves static scaffold metrics nearly unchanged while raising dynamic violations. Removing coordination or scheduling increases curation time to 22 or 25 minutes and leaves more defects. This distinguishes the quality contribution of checks from the efficiency contribution of their orchestration.

C.6 Reward-Control Effects

Under the matched conditions in Appendix B.4, the two-by-two raw/verified and terminal/dense comparison gives

$$\Delta_{\text{verification,terminal}} = 42.6 - 19.4 = 23.2, \tag{20}$$

$$\Delta_{\text{dense,raw}} = 27.2 - 19.4 = 7.8, \tag{21}$$

$$\Delta_{\text{dense,verified}} = 58.7 - 42.6 = 16.1. \tag{22}$$

The 8.3-point difference between the last two quantities indicates that the effect of shaping depends on the training environment. These are differences of point estimates. Filtering achieves 36.4 SR, endpoint-only shaping 49.8, and coarse shaping 54.2, locating the full method's benefit across repaired transitions and progressively richer feedback.

C.7 Simulator Cost and State Fidelity

Figure 12 visualizes the simulator comparison. The constrained simulator reduces LLM calls from 17.8 to 2.3 and tokens from 22.4K to 3.9K relative to step-wise simulation. Its state fidelity is 96.3%, versus 97.1%, and throughput rises from 210 to 1,820 episodes per hour. Compared with unconstrained event updates, validation reduces the violation rate from 10.7% to 2.8%.

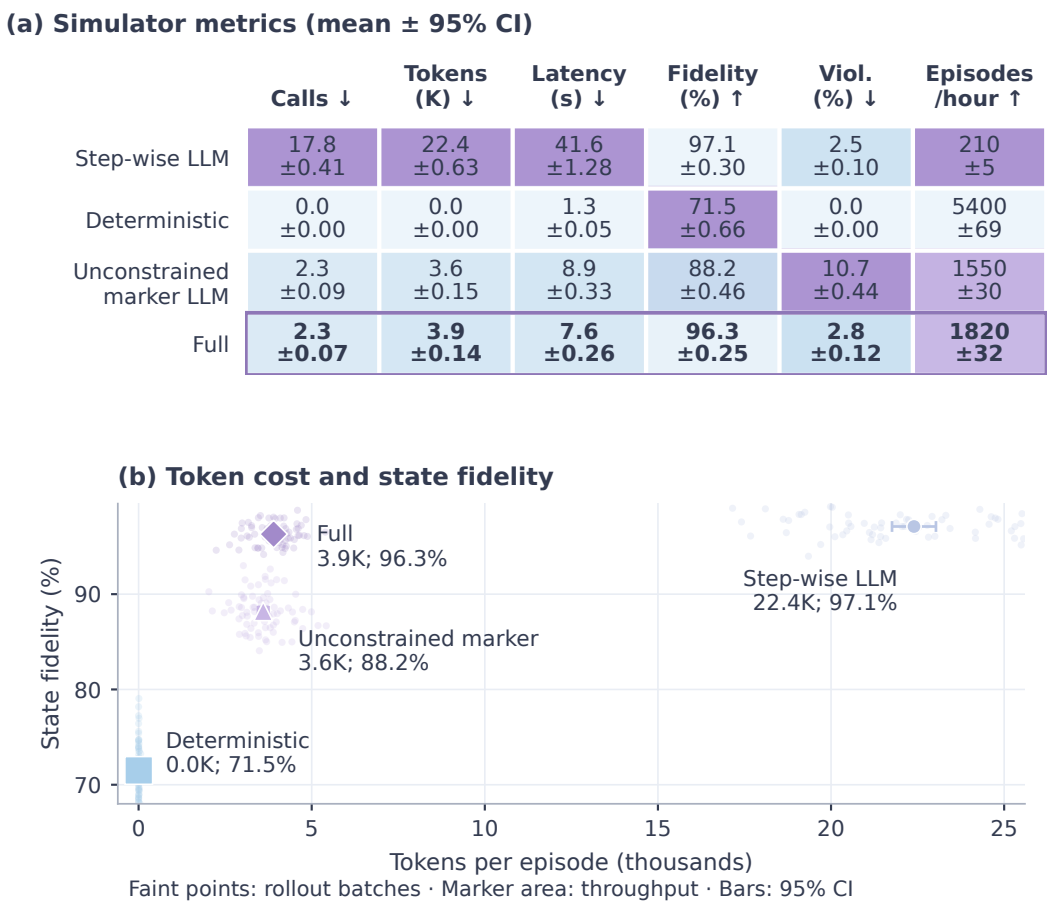


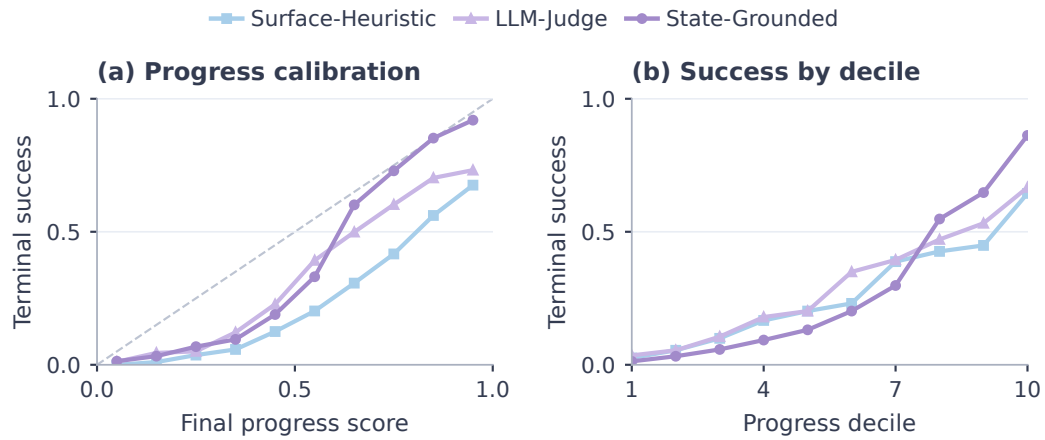
Figure 12: **Simulator cost–fidelity trade-off.** (a) Metric summary with 95% confidence intervals. (b) Token cost versus state fidelity; faint points denote rollout-batch observations, large markers show means, and marker size encodes throughput.

Simulator	Calls	Tokens (K)	Latency (s)	Fidelity (%)	Viol. (%)	Episodes/h
Step-wise LLM	17.8	22.4	41.6	**97.1**	2.5	210
Deterministic-only	**0**	**0**	**1.3**	71.5	**0**	**5,400**
Unconstrained event-driven	2.3	3.6	8.9	88.2	10.7	1,550
Constrained event-driven	2.3	3.9	7.6	96.3	2.8	1,820

Table 14: Simulator trade-off. Calls, tokens, and latency are per episode. State fidelity evaluates agreement with the intended backend transition. Zero writes can avoid violations while failing to implement required updates.

C.8 Reward Alignment

The alignment metrics and diagnostic protocol are defined in Appendix B.9.



(c) Alignment	AUROC ↑	Spearman ↑	ECE ↓	HPF ↓
Surface-Heuristic	0.77	0.42	0.31	0.40
LLM-Judge	0.78	0.45	0.18	0.29
State-Grounded	**0.87**	**0.58**	**0.15**	**0.12**

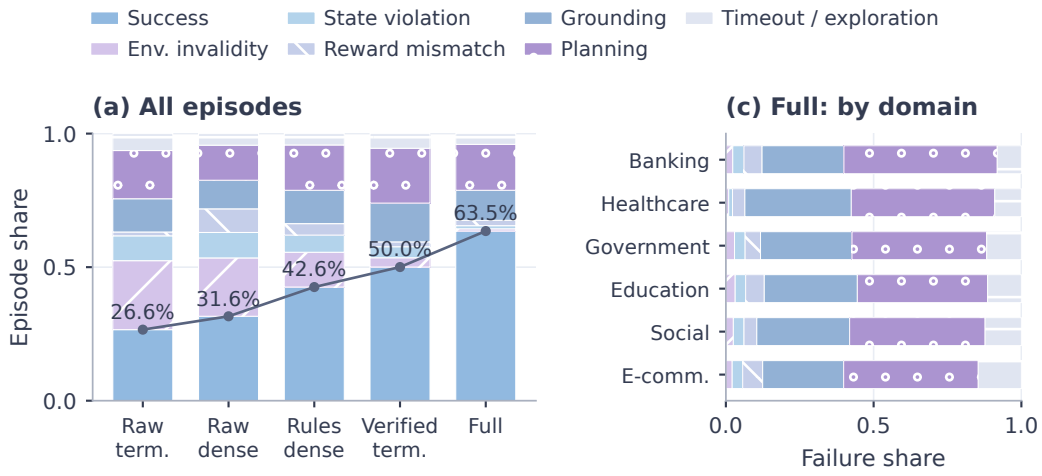
Figure 13: **Dense-reward alignment.** (a) Terminal success versus final progress score. (b) Success by progress decile. (c) AUROC, Spearman correlation, expected calibration error, and high-progress failure rate.

Reward	AUROC↑	Spearman↑	ECE↓	HPF↓
Surface heuristic	0.77	0.42	0.31	0.40
LLM judge	0.78	0.45	0.18	0.29
State-grounded	**0.87**	**0.58**	**0.15**	**0.12**

Table 15: Aggregate reward-alignment diagnostics. HPF in this diagnostic uses its own trajectory pool; Table 2 evaluates training-condition controls.

C.9 Policy Failure Attribution

Figure 14 shows the outcome decomposition, the category distribution conditioned on failure, and residual failures by domain. Planning and grounding account for a larger fraction of remaining failures after verification and marker validation. The attribution procedure is specified in Appendix B.9.



(b) Attribution among failed episodes (%)

	Env. invalid	State viol.	Reward mismatch	Ground-ing	Planning	Timeout
Raw + terminal	35	13	2	17	25	9
Raw + dense	32	14	13	16	19	6
Rules + dense	23	11	7	22	30	7
Verified + terminal	7	8	4	29	41	11
Full	2	3	5	31	47	11

Figure 14: **Policy failure attribution.** (a) Outcome decomposition over the episodes in the attribution analysis. (b) Category proportions conditioned on failed episodes. (c) Domain-level composition of residual failures for the full method.

C.10 Verifier and Repair Error Injection

The interventions defined in Appendix B.7 quantify sensitivity to missed reports, spurious reports, and faulty repairs.

Intervention	Rate	Prec.	Recall	Repair success	False repair	Feasible	State viol.	PPO SR
None	0	91.5	87.2	90.8	2.7	94.8	2.8	58.7
Drop true reports	5	91.6	82.8	90.7	2.7	93.2	3.2	56.8
Drop true reports	10	91.8	78.5	90.5	2.8	90.6	3.9	53.6
Drop true reports	20	92.0	69.8	90.2	2.9	85.1	5.2	48.3
Add spurious reports	5	87.1	87.2	89.1	6.6	93.9	3.2	57.0
Add spurious reports	10	83.2	87.2	87.4	10.4	92.6	3.9	55.2
Add spurious reports	20	76.5	87.2	83.5	17.6	89.3	5.3	51.4
Faulty repairs	10	91.5	87.2	81.9	2.7	90.8	4.7	52.5
Faulty repairs	20	91.5	87.2	72.8	2.7	85.5	7.1	46.7
Mixed errors	10	83.9	78.5	79.2	10.3	87.9	6.3	48.8
Mixed errors	20	76.9	69.8	67.8	17.4	77.1	10.8	39.6

Table 16: Error injection (all entries in %). Rates refer to the candidate set for each intervention.

At 10% injection, missing true reports reduces feasibility to 90.6%, whereas spurious reports leave it at 92.6%. Faulty repairs produce 90.8% feasibility and 4.7% state violations. Mixed errors accumulate

across these stages: at 20%, feasibility is 77.1% and policy SR is 39.6. The results show the practical value of validation gates and also the loss of useful supervision as construction errors accumulate.

C.11 Scaling Environment Size

Table 17 reports runtime and API cost; Table 18 reports residual quality and memory. The 20-page setting provides the paper-scale anchor.

Pages/site	Tasks/site	Sites	Min./site	Min./page	Calls/site	Tokens/site	Iterations
20	14	100	18.0	0.90	22.6	116.4K	3.5
50	30	100	40.7	0.81	49.8	258.7K	3.8
100	60	100	85.9	0.86	104.2	543.1K	4.1
200	120	50	181.6	0.91	219.5	1.14M	4.4

Table 17: Curation cost as site size increases. Entries are aggregate values at each workload size.

Pages/site	Residual defects/site	Residual defects/page	Feasible (%)	Peak memory (GB)
20	3.3	0.165	94.8	1.2
50	7.2	0.144	94.1	2.4
100	13.8	0.138	93.4	4.5
200	27.1	0.136	92.2	8.6

Table 18: Quality and memory at the same four scales as Table 17.

Time per page remains between 0.81 and 0.91 minutes, while peak memory rises from 1.2GB to 8.6GB. Absolute residual defects grow with site size, but defects per page decrease from 0.165 to 0.136. The feasible-task rate falls by 2.6 points over the tested range. These measurements describe practical scaling for this workload and horizon; the search complexity remains as characterized in Appendix A.4.

C.12 Repair Locality and Workflow Diversity

Table 19 gives edit locality and Table 20 gives paired structural changes. The paired protocol is specified in Appendix B.8.

Object modified	Fraction (%)	95% interval
Pages	18.6	[17.7, 19.5]
DOM nodes	5.9	[5.5, 6.3]
Navigation edges	8.7	[8.0, 9.4]
Task constraints	2.4	[2.0, 2.8]

Table 19: Locality of repair. Task-constraint edits are included in the paired accounting.

Repair improves feasibility by 46.2 points while modifying 5.9% of DOM nodes. The small changes in template ratio and graph distance suggest that repair preserves much of the generated structural variety. The increase in shortest successful trace length from 8.8 to 9.1 is inconsistent with a general path-shortening account on the jointly executable subset. The 2.4% task-constraint edit rate and conditioning on joint executability remain relevant when interpreting the policy comparison.

Metric	Raw	Repaired	Change	95% interval of change
Feasible tasks (%)	48.6	94.8	+46.2 pp	[44.8, 47.6] pp
Unique page-template ratio	0.847	0.839	−0.8 pp	[−1.2, −0.4] pp
Pairwise DOM edit distance	0.421	0.415	−1.4%	[−2.1, −0.7]%
Workflow graph distance	0.468	0.459	−1.9%	[−2.8, −1.0]%
Path entropy	1.78	1.74	−2.2%	[−3.5, −0.9]%
Valid-path multiplicity	2.46	2.41	−2.0%	[−3.4, −0.6]%
Shortest successful trace	8.8	9.1	+3.4%	[2.1, 4.8]%
Branching factor	5.74	5.69	−0.9%	[−1.8, 0.1]%
Task semantic diversity	0.618	0.614	−0.6%	[−1.1, −0.1]%

Table 20: Paired raw/repaired summaries. Relative changes follow the paired analysis; pp denotes percentage points. The path metrics condition on joint executability.

Metric	Repaired synthetic	WebArena subset	WebShop	MiniWoB++
DOM depth	11.7	13.9	11.4	8.3
Actionable elements/page	19.2	25.6	21.3	12.7
Successful trace length	9.1	15.2	10.9	6.6
Branching factor	5.69	7.80	6.40	4.50
Operation-type entropy	2.24	2.41	2.16	2.08

Table 21: Structural and interaction profiles. These descriptors characterize workload differences rather than a perceptual-realism score.

C.13 Noisy DOM Interaction

Figure 15 summarizes the stress tests specified in Appendix B.7.

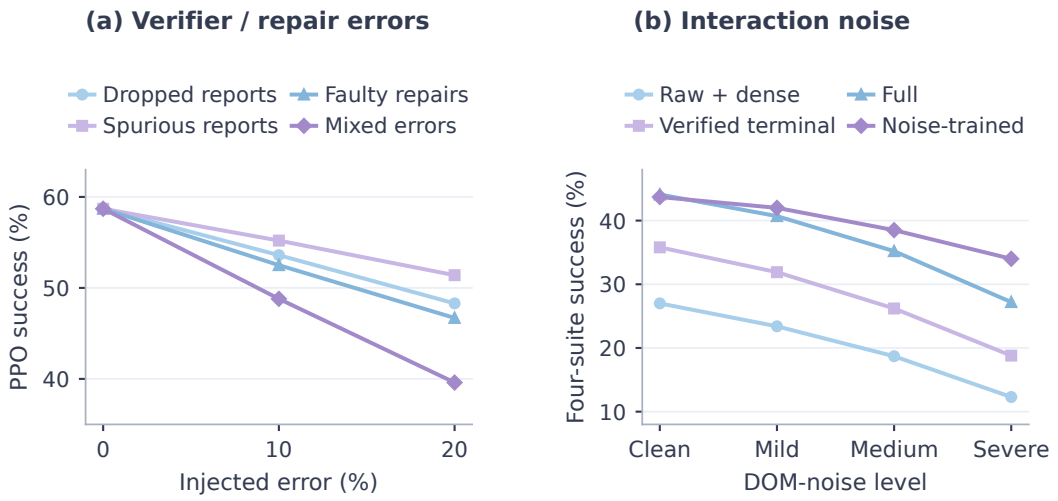


Figure 15: **Additional stress-test summaries.** Left: PPO degradation under verifier and repair errors. Right: four-suite macro success under controlled DOM-noise levels. The two panels summarize different interventions.

Noise training improves combined-condition success by 6.8 points over full, with a 0.4-point reduction in clean success. It also improves every individual perturbation condition. The remaining severe-noise gap identifies delayed and stale interaction as useful targets for richer training dynamics.

Perturbation	Raw + dense	Verified + terminal	Full	Full + noise training
Clean	27.0	35.8	**44.1**	43.7
DOM reorder	23.6	32.1	40.9	**42.1**
Identifier rename	23.1	31.7	40.4	**41.8**
Irrelevant elements	20.8	28.9	38.0	**40.4**
Modal interruption	19.2	26.7	35.9	**38.9**
Delayed update	17.8	25.0	33.8	**37.6**
Stale observation	17.0	24.2	33.1	**37.0**
Combined	12.3	18.8	27.2	**34.0**
Relative combined drop↓	54.4%	47.5%	38.3%	**22.2%**

Table 22: Controlled perturbation success rates (%). Relative drop is $(\text{clean} - \text{combined})/\text{clean}$.

Training condition	Clean	Mild	Medium	Severe	AURC↑
Raw + dense	27.0	23.4	18.7	12.3	20.6
Verified + terminal	35.8	31.9	26.2	18.8	28.5
Full	**44.1**	40.7	35.2	27.2	37.2
Full + noise training	43.7	**42.0**	**38.5**	**34.0**	**39.8**

Table 23: Success across ordered noise-severity levels. AURC is normalized trapezoidal area over the four equally spaced levels, in success-rate units.

C.14 Artifact Overlap Audit

Table 24 reports the artifact comparisons under the fixed representations in Appendix B.8.

Benchmark	Exact text	Near text	DOM collision	Entity match	NN mean	NN max
WebArena-compatible	0	0	0	0	0.37	0.74
WebShop	0	0	0	0	0.42	0.78
MiniWoB++	0	0	0	0	0.35	0.71

Table 24: Artifact audit against synthetic training data. The first four columns count detected matches; NN columns are embedding cosine similarities.

The audit detects no matches under these rules. Its scope is the accessible artifacts and retrieval procedure. Closed-source generator pretraining and semantically related cases outside the retrieval threshold are outside that scope.

C.15 Operation and Composition Analysis

Table 25 stratifies transfer by the operation and composition buckets defined in Appendix B.6.

Evaluation bucket	Tasks	Raw + dense	Verified + terminal	Full	Full − raw
Seen primitives and compositions	384	40.8	48.5	**53.2**	+12.4
Seen primitives, novel compositions	513	31.5	39.8	**47.2**	+15.7
Partially unseen primitives	223	21.2	29.3	**35.0**	+13.8
Held-out domains	120	18.0	25.2	**30.8**	+12.8

Table 25: Operation-stratified diagnostic success rates (%). Bucket-level results characterize transfer structure; benchmark-level results are given in Table 3.

The largest raw-to-full difference is in novel compositions of seen primitives. Gains also occur in the partially unseen and held-out-domain buckets. These patterns support transfer of reusable interaction components within the evaluated DOM interface.

C.16 Policy Backbone Comparison

Table 26 compares the three policy architectures specified in Appendix B.2.

Policy	Construction	Synthetic	WebArena	WebShop	MiniWoB++	External macro
A: BiGRU, 8.9M	Raw + dense	27.2	12.4	29.6	38.9	27.0
	Full	58.7	18.6	43.8	55.4	39.3
B: BiLSTM, 15M	Raw + dense	29.8	14.0	31.2	40.5	28.6
	Full	60.9	20.1	45.2	57.1	40.8
C: Transformer, 42M	Raw + dense	33.7	15.6	34.2	44.3	31.4
	Full	64.2	22.3	48.1	60.0	43.5

Table 26: Success rates (%) across policy backbones. External macro averages the three external suites. The raw-to-full gains are 12.3, 12.2, and 12.1 percentage points for A, B, and C.

Larger actors improve absolute success in both construction conditions. The similar external macro gains across the three architectures indicate that the benefit of the training environments persists across this tested capacity range.

C.17 Transfer Inference Cost

Table 27 supplements the benchmark success rates with policy-side LLM calls and successful-episode lengths.

Method	Policy LLM calls/task↓	WebArena steps on success↓
GPT-4 direct (OpenAI, 2024)	17.5	$17.5^{\pm 0.6}$
GPT-4 ReAct (Yao et al., 2023b)	21.8	$\mathbf{14.3}^{\pm 0.5}$
Synthetic behavior cloning	**0**	$16.8^{\pm 0.7}$
Raw + dense PPO	**0**	$17.2^{\pm 0.6}$
Verified + terminal PPO	**0**	$16.0^{\pm 0.5}$
Full PPO	**0**	$\underline{15.1}^{\pm 0.5}$

Table 27: Inference cost accompanying Table 3. LLM calls and environment actions are different counts. Successful-episode length conditions on each method's own successes.

C.18 Qualitative Repair Example

A task asks the agent to buy a wireless mouse priced below $30. The raw listing displays $24.99 while the detail page shows $34.99; the cart route is broken, and the add-to-cart control lacks a valid persistent update. Consistency checking resolves the price against the canonical record. Structural repair reconnects the cart route. Marker repair binds the selected item to `cart.items`; checkout updates `order.status` to `placed` after validation. The resulting witness follows home → search results → item detail → cart → checkout → confirmation. The example illustrates how several local corrections jointly restore a stateful workflow while retaining the purchase instruction.

D Proofs

D.1 Replay Certificate Guarantee

Proposition 1 (Scope of a replay certificate)**.** *Fix a scaffold version, a tested initial state, and its recorded transition realization. If replay checks action admissibility and the transition relation at every step of a stored length-H trace and verifies $C_t(s_H) = 1$, that initial state admits a successful execution of at most H steps under the recorded realization.*

Proof. The recorded initial state supplies the base case. At step k, admissibility and transition validation establish that the recorded a_k takes the recorded s_k to s_{k+1}. Induction constructs the entire valid execution. Its final predicate check establishes success. No quantification over other initial states or writer realizations is required. □

This proposition states the semantics of the certificate; empirical feasibility rates count the tasks for which the procedure obtains such evidence.