LONG-HORIZON WEB AGENTS NEED A STATE BESIDE THE TRAJECTORY

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents struggle on long-horizon tasks for two reasons. First, the policy fails to attend to evidence that is scattered across a long trajectory yet needed for the next decision. It tends to commit to actions without grounding them in the current evidence. Second, when the current environment differs from training in ways the trajectory does not reveal, the policy falls back on training-time priors rather than checking the current environment. We propose CAST (*Coded Agent with STate*), an agent loop that maintains a small, structured state alongside the trajectory and exposes it to the policy at decision time. This state collects evidence relevant to the next decision as the trajectory unfolds, so the policy reads it from one place rather than recovering it from the trajectory. Environment-side fields are either filled from the current observation or marked as unknown, so an unresolved fact is visible to the policy before it acts. Deterministic code updates the state after every step with no language-model call, and a fixed contract ties a few state values to behavioral constraints. On WebArena, CAST improves success rate across backbones without any offline exploration corpus. Under targeted environment perturbations spanning latent shifts, execution noise, and structural changes, CAST outperforms the baseline. Our code is available at this link.

1 INTRODUCTION

Large language models have become the default policy for interactive agents operating in open, partially observable environments (Wang et al., 2023). Despite rapid progress on short-horizon tasks, the gap between nominal completion and true task success remains wide in long-horizon settings, even when utilizing state-of-the-art backbones (Zhou et al., 2024; Xue et al., 2025; Aghzal et al., 2026). In the standard agent paradigm, the policy iteratively observes the environment, outputs an action, and appends the new state to a growing history (Yao et al., 2023; Zheng et al., 2024). This paradigm leaves two critical structural failures unaddressed.

First, agents suffer from *trajectory-side* degradation. The standard loop shows the policy only the latest observation and the growing list of past actions, so evidence gathered at earlier steps is either dropped or scattered across the history. Language models struggle to integrate this dispersed information (Laban et al., 2026), often exhibiting degraded attention over long contexts (Liu et al., 2024a; Hsieh et al., 2024). Consequently, policies tend to act on fragmented recall rather than aggregated evidence. An agent might attempt the same flawed approach multiple times, simply because the pattern of failure is buried too deep in the log to prompt a strategic shift.

Second, agents face an *environment-side* misalignment. LLMs default to training priors, assuming that a click took effect or that an element keeps its id after the page changes. Lacking an explicit way to represent "unknown" state variables, a standard agent acts on what is typically true. This leads to silent, cascading failures when the live site differs from these assumptions.

To address these limitations, we introduce CAST (*Coded Agent with STate*), a framework that fundamentally alters how policies are conditioned at decision time. CAST maintains a compact, structured state alongside the standard trajectory. Crucially, this state is updated by deterministic code rather than the LLM itself. As the agent receives new observations, programmatic extraction functions reliably parse and write mechanical facts into named state fields.

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.

CAST places the state at a fixed position in the policy's input, next to the prediction site. For the trajectory-side failure, fields such as how many times the same action has been repeated without an observation change consolidate evidence that the history leaves scattered. For the environment-side failure, each field is either filled from the observation or marked as unknown, so a fact the episode has not yet revealed is visible to the policy before it commits to an action. Because the state informs the policy without constraining it, a fixed contract ties a few state values to behavioral constraints, such as ruling out an action class that has just been repeated with no effect. The backbone is unchanged, and CAST adds no call to the language model at each step.

Our core contributions are:

- We identify and formalize two structural bottlenecks that limit long-horizon LLM agents: trajectory-side evidence degradation and environment-side prior misalignment.
- We propose CAST, a framework that leverages deterministic code to extract and maintain a reliable, structured state, grounding the LLM policy in explicit facts rather than context-window recall.
- We validate CAST on WEBARENA, showing consistent gains across backbone scales without an offline exploration corpus and strong robustness to latent shifts and execution noise.

2 RELATED WORK

Long-horizon web agents. WEBARENA, MIND2WEB, and VISUALWEBARENA establish multi-site benchmarks with persistent authentication and site-specific dynamics (Zhou et al., 2024; Deng et al., 2023; Koh et al., 2024), and BROWSERGYM, AgentBench, and WebVoyager consolidate the execution layer (Le Sellier de Chezelles et al., 2025; Liu et al., 2024b; He et al., 2024). Recent evaluations document a persistent gap between offline benchmark scores and online task success, with nominally completed episodes often missing the underlying goal (Xue et al., 2025; Pan et al., 2024b). Yang et al. (2025) separately show that aligning the policy's observation and action format with the language model's pretraining distribution alone yields large WebArena gains. We treat the decision-time condition itself, not the input format alone, as the variable to reexamine.

Memory and scaffolding. A common response to long-horizon difficulty is to augment the trajectory with retrieved or distilled content, such as reflection, self-feedback, critique, and memory (Shinn et al., 2023; Madaan et al., 2023; Chen et al., 2023; Huang et al., 2023b; Pan et al., 2024a; Zheng et al., 2025; Park et al., 2023; Zhong et al., 2023; Tan et al., 2026; Ouyang et al., 2026; Cao et al., 2025; Huang et al., 2025). Augmentation expands what is available to the policy, but the policy still extracts every bookkeeping signal from text at each step, from model-generated items that the model cannot reliably critique post hoc (Huang et al., 2024). A second line routes reasoning through executable code or composes roles and tools into a state machine (Gao et al., 2023; Wang et al., 2024b; Xu et al., 2023; Hong et al., 2024; Wu et al., 2024a; Wang et al., 2024a). There code is the action the model produces, and per-step inference still rests on the model. Pre-task adaptation (Chen et al., 2025; Murty et al., 2024) probes the environment beforehand to modify the prior carried into the episode and is composable with our reformulation.

State factorization. CoALA (Sumers et al., 2024) proposes the working-memory and long-term-memory decomposition for language agents but routes both through language-model-mediated internal actions. StateFlow (Wu et al., 2024b) factors the agent loop along a control-flow boundary, representing it as a task-specific finite-state machine of sub-prompts; our state is runtime data, with typed slots tracking what has happened in the episode, and is task-agnostic. Two recent proposals place the entire belief update on the language model: PABU (Jiang et al., 2026) trains an end-to-end belief-update policy executed by the language model, and ABBEL (Lidayan et al., 2025) replaces the trajectory with a natural-language belief written and refined by the language model. We factor along an orthogonal axis: rather than training the language model to compress history better, we partition history-keeping by who updates what, so computable trajectory facts are advanced by deterministic code. Prospective simulation and search (Ha & Schmidhuber, 2018; Chae et al., 2025; Gu et al., 2024; Zhou et al., 2023) add deliberation rather than narrow the input and are orthogonal.

3 WHY LONG-HORIZON AGENTS FAIL

3.1 PROBLEM FORMULATION

A language-model agent interacts with a partially observable environment entirely through token sequences. At each step t, the agent emits an action a_t, which the environment executes and returns as an observation o_{t+1}, a serialization of the current environment state (e.g., the accessibility tree of a rendered web page). Let x_{ins} denote the task instruction provided at $t = 0$, o_t for the current observation, and $h_t^a = (a_0, a_1, \ldots, a_{t-1})$ for the past-action history.

The standard language-model agent uses a policy π_θ that conditions on the current observation and the past-action history,

$$a_t \sim \pi_\theta(\cdot \mid x_{\text{ins}},\, o_t,\, h_t^a), \tag{1}$$

This is typically implemented by prompting a pretrained language model at each step with $(x_{\text{ins}}, o_t, h_t^a)$ (Yao et al., 2023; Schick et al., 2023; Xie et al., 2024). We refer to this paradigm as *latest-observation conditioning*: only the latest observation is kept, since o_{t-1} is replaced by o_t; the action history h_t^a is the only part of the prompt that accumulates, growing by one action label per step; and the policy maintains no persistent state outside of the language-model call. Consequently, the only cross-step signal available to the policy is whatever survives in the prompt: the instruction, the current observation, and the full list of past action labels.

Both failures introduced in Section 1 stem from this conditioning, but in different ways. The *trajectory-side failure* (Section 3.2) is a structural *layout* problem: critical, load-bearing content exists somewhere in the prompt $(x_{\text{ins}}, o_t, h_t^a)$ but is difficult for the model to attend to because it is either distant from the prediction site or scattered across a lengthy history. The *environment-side failure* (Section 3.3) is a *representation* problem: the prompt lacks a dedicated mechanism to explicitly track latent facts about the current deployment. When a necessary fact remains unobserved, the policy silently substitutes a training-distribution default.

3.2 TRAJECTORY-SIDE FAILURE: EFFECTIVE INFORMATION DISTANCE

The position of decision-relevant content within the policy's input is as critical as its presence. Let L_t denote the tokenized length of the prompt at step t. The tokens that directly dictate a_t occupy only a fraction of this length, often scattered across the instruction, isolated cues within o_t, and patterns implicit in the action history h_t^a. The intervening tokens consist largely of routine page content and low-information action logs. Two phenomena subsequently drive up the *effective information distance* between this critical content and the prediction site, even without exceeding the nominal context window. The first is *distance*: as L_t increases, the token gap between critical evidence and the prediction site widens, and in-context retrieval performance over such long sequences is notoriously non-uniform (Hsieh et al., 2024; Bai et al., 2024). The second is *scattering*: when the evidence required for a single decision is fragmented across multiple turns, models tend to commit to early assumptions and fail to synthesize later evidence (Laban et al., 2026).

Figure 1 isolates these effects using a controlled probe built from real WEBARENA trajectories. We fix the decision-relevant content while varying its placement across two axes: distance (Short vs. Long) and concentration (Concentrated vs. Distributed). When the content is compact and adjacent to the decision site (SC), accuracy reaches 100%. Introducing distance alone (LC) degrades accuracy to 88.9%, while scattering the content across snippets (SD) causes a severe drop to 48.1%. The combination of both effects (LD, mimicking the long history of a failed baseline run) plummets accuracy to 11.1%. Crucially, the target content remains identical and within the context window across all conditions; performance degrades solely due to spatial layout. Standard architectural interventions fail to resolve this. Expanding the context window does not mitigate distance or scattering. Summarizing the prompt via text or learned compression (Mu et al., 2023; Jiang et al., 2023) risks either omitting load-bearing facts or re-introducing them far from the prediction site. However, appending a compact state block containing the required content directly adjacent to the

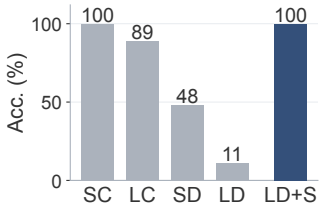


Figure 1: Probe of effective information distance.

prediction site (LD+S) restores accuracy to 100%. This result therefore indicates that the prompt must be augmented with a runtime state of bounded length, strictly positioned near the prediction site and managed via an explicit update mechanism at every decision step.

Takeaway: Latest-observation conditioning lets effective information distance grow with the horizon. A remedy must anchor load-bearing content at a fixed position near the prediction site.

3.3 Environment-Side Failure: Marginalization Trap

The second failure originates from the language model's pre-training objective rather than prompt layout. Trained via maximum likelihood over a vast text corpus, the language model essentially learns a marginal distribution over any unobserved variables x that influence a response:

$$P_\theta(b \mid a) \;=\; \int P(b \mid a, x)\, P(x \mid a)\, dx, \tag{2}$$

At deployment, however, the target behavior is governed by the specific instantiation of x in the current episode: $b = f(a, x)$. Because latest-observation conditioning lacks a representational slot to flag x as "unknown," the policy defaults to its pre-trained marginal distribution when confronted with an unobserved environmental fact. We term this the *marginalization trap*. While conceptually related to prior-driven hallucinations (Ji et al., 2023; Huang et al., 2023a), this failure specifically concerns dynamic deployment variables rather than static world knowledge. Standard output-side uncertainty estimates (Kuhn et al., 2023) cannot surface these missing inputs because they only calibrate the model's generated text, not its missing environment state variables.

Figure 2 evaluates this trap across four regimes. The control involves a *latent world fact* not yet retrieved (e.g., a product's price), which the policy naturally handles by emitting a search action, supported by its prior. The remaining three regimes involve episode-specific execution facts that the pre-training prior cannot predict: an *action error* (a silently failed click), a *stale locator* (a DOM ID reused post-render), and an *unverified write* (an unconfirmed form submission). We prompt the model at each failure point using the standard trace (*baseline*) versus the trace augmented with an explicit slot naming the missing variable (+ *oracle state*). The control scores 100% in both setups. For the execution regimes, the baseline input scores 0%, 40%, and 0%, whereas introducing the oracle state slot recovers performance to 100%, 100%, and 60%. The marginalization trap strictly penalizes scenarios where the prior cannot infer x. Neither stronger backbones nor longer historical trajectories can bypass this trap, as a refined default over x remains a marginal distribution. The fundamental remedy is to provide uncertainty with a representational locus: an explicit, structured slot in the prompt for unresolved environmental facts. These slots must be deterministically updated via recent interactions or external retrieval, rendering the unresolved fact visible to the policy before it commits to an action.



Figure 2: Oracle-state intervention probe on real WebArena failure points.

Takeaway: When a fact about the current environment has not yet been observed, the policy silently substitutes a training-distribution default; a remedy must give every such fact an explicit, named slot in the input, updated deterministically as the trajectory unfolds.

4 Cast: A Coded Agent with State

We address the structural failures identified in Section 3 by altering what the policy conditions on at decision time. Section 4.1 formulates this intervention at the level of the agent loop. The following sections develop it across three orthogonal axes: state representation (Section 4.2), transition dynamics (Section 4.3), and policy coupling (Section 4.4).

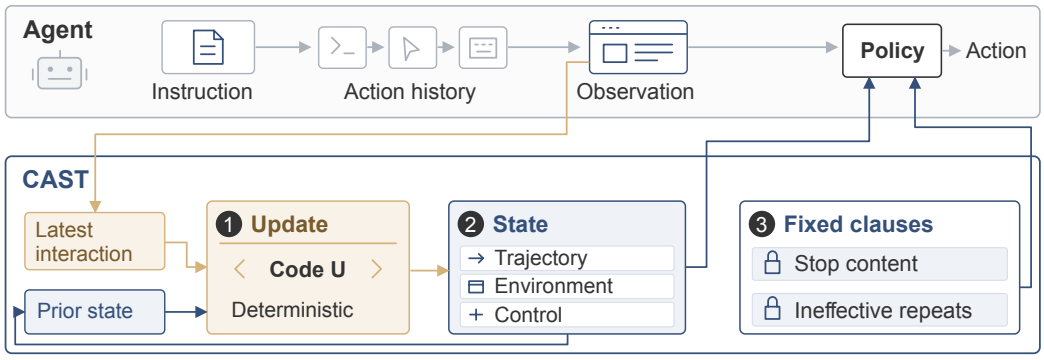


Figure 3: CAST in the agent loop. **Top:** the policy reads the instruction x_{ins}, past actions h_t^a, and the current observation o_t. **Bottom:** CAST adds two inputs to the same call. ❶ Deterministic update U combines the prior state with the latest interaction into ❷ a bounded runtime state s_t with trajectory, environment, and control fields; ❸ contract c_t constrains stop content and ineffective repeats.

4.1 OVERVIEW

CAST introduces a bounded *runtime state* $s_t \in \mathcal{S}$ that the policy reads alongside the current observation, and reformulates the agent loop as

$$s_{t+1} = U(s_t,\, o_{t+1},\, a_t,\, e_{t+1}), \tag{3}$$

$$a_t \sim \pi(\cdot \mid x_{\text{ins}},\, o_t,\, h_t^a,\, s_t,\, c_t), \tag{4}$$

where e_{t+1} collects structured signals from the most recent step, U is the *deterministic update*, and c_t is a fixed *contract* consumed by the policy alongside s_t. Equations (3)–(4) make three design choices visible: the shape of s_t, the form of U, and how c_t couples π to s_t. The remainder of this section takes them in turn; Figure 3 shows the resulting anatomy.

A single principle organizes our choices on all three axes: *facts that admit a deterministic extraction rule* belong in code, and *facts that require linguistic judgment* remain in the policy. Relative to Equation (1), the only thing that changes is what the single policy call conditions on.

4.2 RUNTIME STATE

The runtime state s_t is a *typed record of bounded size* whose fields hold facts the policy needs at decision time but the trajectory does not reliably surface. Each field has a fixed name and a fixed value type, so the policy reads s_t from a known location in its input instead of searching a growing history. The fields fall into three kinds, separated by what each is for.

Trajectory-side fields. These fields consolidate recent historical evidence, such as repeated clicks on one locator or a recoverable error from the last action. They target the Effective Information Distance failure of Section 3; for instance, `same_action_repeat_count` records how often the policy has repeated the same action class without an observation change.

Environment-side fields. These fields record what is true about the current environment. They target the Marginalization Trap by reserving named slots that are either filled from observation or marked as unknown: `visible_absence_evidence` is set when the page indicates the item is absent and `visible_completion_evidence` when a success indicator is shown, so the policy need not fall back on a training-default guess about the page state.

Control-side fields. A third set of fields is read mainly by the contract: the remaining step budget (`remaining_steps`); the task type (`task_mode`), which determines what evidence ends the task; the actions barred after repeating without effect (`forbidden_next_actions`); and `next_policy`, a short directive looked up from a fixed table (e.g., `seek_visible_evidence_before_stopping`). We list them separately because the contract turns their values into constraints on the next action.

4.3 DYNAMIC UPDATE

The update U in Equation (3) carries s_t to s_{t+1} from the most recent interaction. U is ordinary *deterministic code*: a short rule per field reads the current state, the new observation o_{t+1}, the action a_t, and structured signals e_{t+1}, and writes the new value of that field.

Where the rules come from. Every rule in U instantiates one of three principles. The first is *action effect*: after each action, the state records whether the action produced an observable effect, namely whether the environment returned an error, whether the URL changed, and whether the observation changed, and it counts actions repeated without effect. The second is *termination evidence*: each type of task ends on a type of visible evidence, an answer task on a visible value or a visible absence, a write task on a visible confirmation, and a navigation task on the target page; the state records the task type and whether its terminating evidence is visible. The third is *budget*: the state records the remaining steps. Rules under the first and third principles read only the signals the interface returns with each action, so they carry over unchanged to any environment with the same action–observation interface. The second principle needs a lexicon, the verbs that mark a task type and the phrases that mark absence, confirmation, or a validation error. In our instance the lexicon consists of short lists of generic web-interface phrases such as "no results" and "saved". No rule refers to a particular site, page, or task, and one schema, rule set, and lexicon serve all 812 tasks across the six sites. Under every principle a field's value is a comparison between consecutive steps, a count, or a lexicon lookup, so U needs no learned weight and no model call; the update is cheap, returns the same value for the same interaction, and every value the policy reads traces to the rule that set it.

For instance, `tried_no_effect_actions` adds an action that leaves both the URL and the observation unchanged, and `forbidden_next_actions` lists it once it has been repeated three times without effect. Neither rule reads page text or depends on the site.

One-time adaptation. The lexicon is the only component that depends on the environment, and its role is fixed by the termination-evidence principle, so a new environment calls for a new lexicon rather than a new schema. In this work the lexicon is written by hand. It can instead be written once by a language model: given the principle and a few pages and instructions from the new environment, the model produces the lexicon, which is then frozen as code, so the update at run time stays deterministic and adds no model call. We leave the evaluation of this adaptation to future work.

4.4 APPLYING CONTROL

The state s_t informs the policy but does not, on its own, constrain it. A policy reading s_t remains free to ignore what it shows: to retry an action class that has just produced no effect, or to stop with a placeholder answer when s_t directly indicates that a concrete one is available. We close this gap with a small, fixed contract c_t that the policy reads alongside s_t. c_t is a short set of prompt clauses, identical at every step, that tie specific state values to specific behavioral constraints. The clauses come in two kinds. One restricts what may appear in a stop action, ruling out placeholder and hedged answers and requiring a concrete value when s_t directly indicates one is available. The other forbids an action class that has just been shown ineffective, for example one that has been repeated several times with no observation change. The clauses stay fixed; what changes from step to step is the state they read, including the directive in `next_policy`. Neither clause refers to a particular site or task.

The forbidden-action clause fires on `same_action_repeat_count` ≥ 3 with no observation change. Under latest-observation conditioning the previous observation is no longer in the prompt, so the policy cannot establish this condition from its own input. The contract is therefore not a substitute for the state but the rule that turns the state's facts into behavior.

Takeaway: The information an agent acts on lives in parameters and the trajectory, and *attention* is the computation reaching into both. Attention is flexible but uneven: scattered across a long trajectory, the very facts a decision turns on are often the ones it misses. Many of these facts, such as whether the last action had any effect, are easy to compute but hard to attend to. CAST computes them with rules that follow from a few general principles rather than from any particular site, places them at a fixed input position, and lets the contract turn them into constraints on the next action.

Table 1: Success rate (%). GTTA is dynamics grounding (Chen et al., 2025); *Qwen2.5-14B-Instruct* baseline and GTTA (Hybrid) rows are from Chen et al. (2025). ✓: retrieves over an offline corpus of GPT-4.1 exploration trajectories. **Best** and **second-best** per column and model.

Model	Condition	Retrieval	GitLab	Shop Admin	Shop	Reddit	Map	Multi	**Avg**
Qwen2.5-14B-Instruct	baseline		20.0	**14.0**	**22.0**	13.0	13.0	**6.0**	17.0
	GTTA	✓	**26.0**	13.0	**24.0**	**31.0**	**19.0**	4.0	**21.0**
	CAST		**23.9**	**14.8**	21.9	**34.9**	**17.4**	**16.7**	**21.6**
GPT-5.4	baseline		**47.8**	31.3	34.8	31.1	**30.3**	**22.9**	35.1
	GTTA	✓	**46.7**	**31.9**	**36.4**	**38.7**	**30.3**	**20.8**	**36.2**
	CAST		36.1	**33.0**	**39.0**	**48.1**	**29.4**	18.8	**35.7**
GPT-OSS-20B	baseline		8.3	15.4	23.5	1.9	**22.0**	**12.5**	14.7
	GTTA	✓	**26.7**	**26.9**	**27.3**	**23.6**	**22.0**	**12.5**	**25.0**
	CAST		**27.8**	**25.3**	**25.7**	**32.1**	**22.9**	10.4	**25.6**

5 EXPERIMENTS

Evaluation overview. We frame the evaluation around three questions about CAST. *(i) Long-horizon performance.* Does CAST improve on a realistic long-horizon agentic benchmark? We use WEBARENA as the testbed. *(ii) Environment dynamics.* Does CAST hold up when the environment shifts across episodes? On long-horizon web tasks, the dominant source of failure is not the policy's reasoning capacity but the *environment dynamics*, and we focus on three regimes: *latent shift*, where the same task lands on different latent values (prices, account state) between episodes; *execution noise*, where actions fail transiently and recovery requires a different action class rather than a retry; and *structural shift*, where the same task template targets a different site, layout, or control vocabulary. *(iii) Ablation study.* Which parts of the design are doing the work?

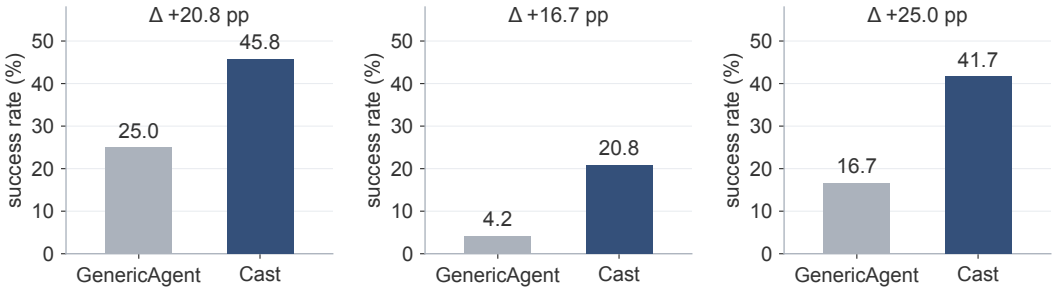
WebArena. We use WEBARENA (Zhou et al., 2024), a web-agent benchmark of 812 long-horizon tasks across six sites (*shopping*, *shopping admin*, *reddit*, *gitlab*, *map*, and a *multisite* split reported as *Multi*). Tasks emulate human web workflows that span multi-step navigation, form filling, and information retrieval across pages. Each task is judged by WebArena's functional-correctness evaluator, the environment is hard-reset between episodes, and the agent has a 30-step budget per episode.

Models and setup. We evaluate CAST on three backbones: Qwen2.5-14B-Instruct, gpt-5.4, and gpt-oss-20b. The baseline is the standard *GenericAgent* loop from BrowserGym/AgentLab (Le Sellier de Chezelles et al., 2025), which uses latest-observation conditioning: the prompt at step t contains the task, the current URL and observation, and the full list of past actions, while earlier observations are dropped (no observation summarization or sliding window). This is the same base agent against which Chen et al. (2025) report the gains of GTTA. For Qwen2.5-14B-Instruct we reuse the GenericAgent and dynamics-grounding (GTTA) numbers from Chen et al. (2025) and run only the CAST arm; for gpt-5.4 and gpt-oss-20b we run all three conditions end-to-end. On every backbone, GenericAgent and CAST share the same policy model and prompt scaffold; CAST only adds the runtime state s_t and the decision-time contract c_t from Section 4. Both are produced by code, so CAST issues no extra language-model calls per step. Appendix D reports the per-episode step counts and the constant per-step prompt overhead added by the runtime state block and by the contract block.

6 RESULTS

6.1 MAIN RESULTS ON WEBARENA

Table 1 compares the baseline, GTTA, and CAST. CAST gains on all three backbones: large on Qwen2.5-14B-Instruct (+4.6 pp, 17.0 → 21.6) and gpt-oss-20b (+10.9 pp, 14.7 → 25.6), small on gpt-5.4 (+0.6 pp, 35.1 → 35.7). On the first two, the gain is spread across the six sites (Appendix E), consistent with a design that changes the policy's decision-time input and targets no site-specific feature. Because CAST only modifies that input, positive gains on all three backbones suggest the effect is tied to the conditioning rather than to a particular model.



(a) Regime 1: latent shift. (b) Regime 2: execution noise. (c) Regime 3: structural shift.

Figure 4: Success rate on the three environment-dynamics stress subsets, GenericAgent versus CAST ($n = 24$ tasks per subset). CAST improves over the baseline in every regime, with the largest absolute gain on cross-site/control variant ($+25.0$) and the largest relative gain on transient action failure.

Matching dynamics grounding without retrieval or an offline exploration corpus. The most informative comparison is CAST against GTTA. GTTA adapts the prompt with offline-mined dynamics: at test time it retrieves over a corpus of exploration trajectories collected from *GPT-4.1*. CAST uses none of this. There is no exploration phase, no offline trajectory corpus, and no retrieval index. The state s_t and the contract c_t are produced from the agent's own per-step environment signals by code, so the decision-time input depends only on what the policy itself has just observed. Despite this leaner test-time setup, CAST matches or exceeds GTTA on the per-backbone average: 21.6 vs 21.0 on Qwen2.5-14B-Instruct, 35.7 vs 36.2 on gpt-5.4 (within noise), and 25.6 vs 25.0 on gpt-oss-20b. These numbers say that the gain credited to dynamics grounding can be recovered without an offline exploration corpus, simply by reshaping what the policy conditions on. CAST also issues no extra language-model calls per step (the slot update U_{code} runs in code), so the matchup is cost-neutral against the baseline and cheaper than retrieval-based dynamics grounding.

Where the per-backbone average comes from. On the gpt-oss-20b row of Table 1, CAST lifts SR on *Reddit* from 1.9 to 32.1 ($+30.2$ pp), on *GitLab* from 8.3 to 27.8 ($+19.5$ pp), and on *Shop Admin* from 15.4 to 25.3 ($+9.9$ pp). The two write-heavy sites (*GitLab* and *Shop Admin*) are exactly where the baseline most often falls into the no-effect $\rightarrow$ same-action loop pattern (see Section 4), and the contract block and scoped loop control are the components that target this failure. *Multi* is the only site where CAST loses ground ($12.5 \rightarrow 10.4$, -2.1 pp on $n = 48$). Figure 5 confirms this from the opposite angle: under CAST the gray "OK, no reward" band shrinks substantially on every site, the blue *Win* band grows, and on the loop-prone sites (*GitLab*, *Map*, *Multi*) part of the formerly-silent failure mass is converted into honest *Truncated* outcomes rather than soft, silently wrong final answers.

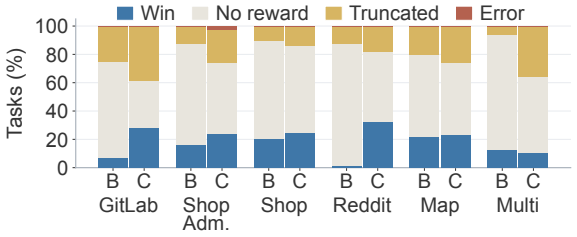
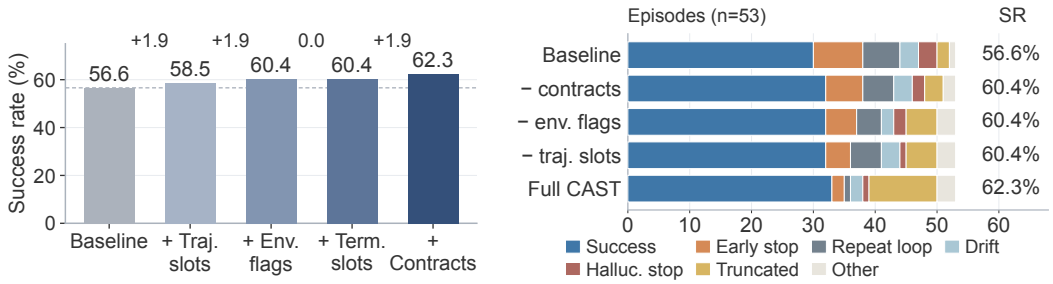


Figure 5: Per-site terminal status of the baseline (B) and CAST (C) on gpt-oss-20b, as a share of the site's tasks. Counts come from an earlier grading of these runs (baseline 110/812, CAST 204/812) than Table 1.

6.2 Environment dynamics for long-horizon agents

On long-horizon web tasks the dominant source of failure is not the policy's reasoning capacity but the environment dynamics it must condition on. To probe this directly, we construct three fixed 24-task stress subsets, each isolating one regime where the GenericAgent baseline is known to drift: *latent shift* (Regime 1), information-seeking tasks whose answer must be recomputed from the live page; *execution noise* (Regime 2), rollouts that hit errors, truncations, or long no-progress windows, where recovery requires switching action class rather than retrying; and *structural shift* (Regime 3),



(a) Cumulative ladder; rightmost bar is full CAST.

(b) Per-episode outcome decomposition.

Figure 6: Component-level ablation of CAST.

programmatic tasks spanning different sites, layouts, and control vocabularies. Each subset excludes polluted or missing paired trajectories; Appendix C describes their construction.

Figure 4 reports per-subset success rates. CAST improves over the GenericAgent baseline in every regime, with the largest absolute gain on cross-site/control variant ($+25.0$) and the largest relative gain on transient action failure ($4.2 \rightarrow 20.8$). The pattern tracks the design: a bounded state s_t that records grounded values, error signals, and locator hints lets the policy reground, recover, and relocate (one action per regime: latent shift, execution noise, structural shift) without scanning the full history.

6.3 ABLATIONS

We ablate CAST along the two component groups defined in Section 4: the runtime state slots s_t (split into trajectory bookkeeping, environment flags, and termination scaffolding) and the decision-time contract block c_t. We run a cumulative ladder and a failure-mode decomposition. All variants share the same backbone, prompt template up to the ablated component, and 30-step budget. On a held-out pressure subset of 53 tasks, full CAST reaches 62.3% from a 56.6% baseline.

Component-level ablation. Figure 6a reports the cumulative ladder over the four CAST component groups. Each of the three state-slot groups contributes a small additive gain over the baseline ($+1.9$ from trajectory bookkeeping, $+1.9$ from environment flags; termination scaffolding adds capacity for the contract block but does not move success rate by itself), and the contract block adds the final $+1.9$ to reach 62.3. No single slot is load-bearing alone, which tracks the design intent that each slot covers a distinct failure condition rather than a shared mechanism.

Failure-mode shift. Figure 6b decomposes the residual failures of each ablation variant by mechanism: premature stop (placeholder or hedged answer), repeat loop (same action class fired three times with no effect, or a two-action cycle), drift (URL unchanged across many clicks), hallucinated stop (concrete answer not supported by visible evidence), step truncation (budget exhausted), and other. The contract block primarily targets premature stop, repeat loop, and hallucinated stop; the matching $-$contracts row shows those three categories grow back. Removing state slots, by contrast, trades successes for step-truncated runs. Full CAST converts most of the non-truncation failure mass; the residual is dominated by step truncation and reflects a budget ceiling rather than a failure.

7 CONCLUSION

We argued that long-horizon LLM agents fail not because their backbones are too weak, but because of what the policy reads at decision time: the trajectory scatters the evidence each decision turns on, and leaves no place to mark what is still unknown about the environment. CAST addresses both by giving the policy a second input, a small, typed runtime state populated by deterministic code from each new observation and read alongside the trajectory, with a fixed contract tying a few state values to behavioral constraints. The backbone is unchanged and we add no extra language-model call per step. On WebArena, this loop-level change improves success across backbone scales and holds, often widening, under targeted perturbations. The underlying move, making the policy's decision-time input structured, bounded, and updated by code, should be productive beyond the web-agent setting.

AI USE STATEMENT

The research idea, the problem framing, and the method were conceived by the authors, who refined them through discussion with large language models. We used LLM-based coding assistants (Claude Code and OpenAI Codex) to implement the agent, the runtime-state update rules, the evaluation harness, and the experiment scripts under author direction. The assistants also proposed and executed experiment configurations and diagnostic probes, selected fixed WEBARENA task subsets used during development, and analyzed the resulting runs. We also used language models to help find related work. The authors drafted the manuscript and used these tools to revise the text, write plotting scripts, restyle figures, convert the citation style, and format the LaTeX source. The authors reviewed the AI-assisted outputs and take full responsibility for every claim, number, table, and figure reported in this paper and its appendices.

REFERENCES

Mohamed Aghzal, Gregory J. Stein, and Ziyu Yao. Why do LLM-based web agents fail? a hierarchical planning perspective. *arXiv preprint arXiv:2603.14248*, 2026.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. *arXiv preprint arXiv:2512.10696*, 2025.

Hyungjoo Chae, Namyoung Kim, Kai Tzu-iunn Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. Web agents with world models: Learning and leveraging environment dynamics in web navigation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=moWiYJuSGF.

Arthur Chen, Zuxin Liu, Jianguo Zhang, Akshara Prabhakar, Zhiwei Liu, Shelby Heinecke, Silvio Savarese, Victor Zhong, and Caiming Xiong. Test-time adaptation for LLM agents via environment interaction. *arXiv preprint arXiv:2511.04847*, 2025.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems*, volume 36, pp. 28091–28114. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/5950bf290a1570ea401bf98882128160-Paper-Datasets_and_Benchmarks.pdf.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799. PMLR, 2023. URL https://proceedings.mlr.press/v202/gao23f.html.

Yu Gu, Kai Zhang, Yuting Ning, Boyuan Zheng, Boyu Gou, Tianci Xue, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, Huan Sun, and Yu Su. Is your LLM secretly a world model of the internet? Model-Based planning for web agents. *arXiv preprint arXiv:2411.06559*, 2024.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

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, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.371. URL https://aclanthology.org/2024.acl-long.371/.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VtmBAGCN7o.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=IkmD3fKBPQ.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv preprint arXiv:2311.05232*, 2023a.

Tenghao Huang, Kinjal Basu, Ibrahim Abdelaziz, Pavan Kapanipathi, Jonathan May, and Muhao Chen. R2D2: Remembering, replaying and dynamic decision making with a reflective agentic memory. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 30318–30330. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1464. URL https://aclanthology.org/2025.acl-long.1464/.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and brian ichter. Inner monologue: Embodied reasoning through planning with language models. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1769–1782. PMLR, 2023b. URL https://proceedings.mlr.press/v205/huang23c.html.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, 2023. doi: 10.1145/3571730.

Haitao Jiang, Lin Ge, Hengrui Cai, and Rui Song. PABU: Progress-aware belief update for efficient LLM agents. *arXiv preprint arXiv:2602.09138*, 2026.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. *arXiv preprint arXiv:2310.06839*, 2023.

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, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.50. URL https://aclanthology.org/2024.acl-long.50/.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. *arXiv preprint arXiv:2302.09664*, 2023.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. LLMs get lost in multi-turn conversation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=VKGTGGcwl6.

Thibault Le Sellier de Chezelles, Maxime Gasse, Alexandre Lacoste, Massimo Caccia, Alexandre Drouin, 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, Graham Neubig, Quentin Cappart, Russ Salakhutdinov, and Nicolas Chapados. The BrowserGym ecosystem for web agent research. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=5298fKGmv3. Expert Certification.

Aly Lidayan, Jakob Bjorner, Satvik Golechha, Kartik Goyal, and Alane Suhr. ABBEL: LLM agents acting through belief bottlenecks expressed in language. *arXiv preprint arXiv:2512.20111*, 2025.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024a. doi: 10.1162/tacl_a_00638.

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

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. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf.

Jesse Mu, Xiang Li, and Noah Goodman. Learning to compress prompts with gist tokens. In *Advances in Neural Information Processing Systems*, volume 36, pp. 19327–19352. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/3d77c6dcc7f143aa2154e7f4d5e22d68-Paper-Conference.pdf.

Shikhar Murty, Christopher D. Manning, Peter Shaw, Mandar Joshi, and Kenton Lee. BAGEL: Bootstrapping agents by guiding exploration with language. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=VsvfSMI5bs.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=jL7fwchScm.

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. Autonomous evaluation and refinement of digital agents. In *First Conference on Language Modeling*, 2024a. URL https://openreview.net/forum?id=NPAQ6FKSmK.

Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, and Zhengyang Wu. WebCanvas: Benchmarking web agents in online environments. In *Trustworthy Multi-modal Foundation Models and AI Agents*, 2024b. URL https://openreview.net/forum?id=1TOW5WCtAW.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer:

Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/d842425e4bf79ba039352da0f658a906-Paper-Conference.pdf.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

Theodore Sumers, Shunyu Yao, Karthik R. Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=1i6ZCvflQJ. Survey Certification, Featured Certification.

Yunteng Tan, Zhi Gao, and Xinxiao Wu. Enhancing web agents with a hierarchical memory tree. *arXiv preprint arXiv:2603.07024*, 2026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024a. ISSN 2835-8856. URL https://openreview.net/forum?id=ehfRiF0R3a.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*, 2023.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 50208–50232. PMLR, 2024b. URL https://proceedings.mlr.press/v235/wang24h.html.

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. In *ICLR 2024 Workshop on Large Language Model Agents*, 2024a. URL https://openreview.net/forum?id=uAjxFFing2.

Yiran Wu, Tianwei Yue, Shaokun Zhang, Chi Wang, and Qingyun Wu. StateFlow: Enhancing LLM task-solving through state-driven workflows. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024b. URL https://openreview.net/forum?id=CZAs3WFw5r.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, volume 37, pp. 52040–52094. Curran Associates, Inc., 2024. doi: 10.52202/079017-1650. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5d413e48f84dc61244b6be550f1cd8f5-Paper-Datasets_and_Benchmarks_Track.pdf.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. ReWOO: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*, 2023.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? assessing the current state of web agents. *arXiv preprint arXiv:2504.01382*, 2025.

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. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=oWdzUpOlkX.

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

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. GPT-4V(ision) is a generalist web agent, if grounded. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=piecKJ2DlB.

Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, ZhongZhi Li, Yingying Zhang, Le Song, and Qianli Ma. LifelongAgentBench: Evaluating LLM agents as lifelong learners. *arXiv preprint arXiv:2505.11942*, 2025.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. *arXiv preprint arXiv:2305.10250*, 2023.

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.

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. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=oKn9c6ytLx.

A SLOT SCHEMA AND UPDATE RULES

This appendix specifies one concrete schema instance σ that we use to define the runtime state s_t in our experiments, together with the per-slot rules of the deterministic update U_{code}, the decision-time contract block c_t used by the policy, and an example of a serialized state that the policy reads. The slot list below is a representative instantiation of the framework rather than a closed specification: any field whose value can be advanced by a short deterministic rule from the recent interaction tuple is admissible, and we expect deployment-specific schemas to add or drop fields as the task family demands. CAST introduces no language-model updater and no retrieval over an offline corpus; every slot below is computed by code from the most recent interaction tuple.

A.1 SLOT GRAMMAR

The runtime state s_t in our implementation is a typed record of bounded size. Each slot in the instantiation we report has a documented semantics, value type, and per-slot length bound, but the schema itself is open in the sense that additional slots covering other deterministically extractable signals (for example, grounded values such as a price field, an account attribute, or an inventory indicator read off a known DOM region) fit the same framework. Table 2 lists the slots used in this paper, grouped by which Section 3 failure regime they target.

Table 2: Slots of the runtime state s_t in the instantiation used in our experiments. All listed slots are advanced by deterministic code; *length* is the per-slot bound in serialized tokens (—: a list slot capped at five entries, or a single directive, with no fixed token bound). The list is illustrative rather than exhaustive: schemas for other task families may add fields (for example, regex-extracted grounded values on pages with a known layout) or drop fields that are not needed.

Slot	Group	Type	Source signals	Length
`post_error_flag`	traj. bookkeeping	bool	last e	4
`locator_history`	traj. bookkeeping	(bid, int, int)	last (o, a, e)	32
`commit_affordance`	traj. bookkeeping	enum	current o, recent a	8
`tried_no_effect_actions`	traj. bookkeeping	action list	last (o, a, e)	—
`url_changed`	env. flag	enum	last two o	4
`observation_changed`	env. flag	enum	last two o	4
`visible_completion_evidence`	env. flag	enum	current o	4
`visible_absence_evidence`	env. flag	enum	current o	4
`remaining_steps`	termination	int	step counter	4
`task_mode`	termination	enum	x_{ins}	6
`same_action_repeat_count`	termination	int	last k actions	4
`two_action_cycle_count`	termination	int	last k actions	4
`forbidden_next_actions`	termination	action list	loop counters	—
`next_policy`	termination	enum	mode, flags, loops	—

A.2 CODED UPDATE RULES

Every slot is advanced by a short, slot-local rule over $(s_t, o_{t+1}, a_t, e_{t+1})$. Each rule is an ordinary deterministic procedure with no learned weights.

`post_error_flag`. Set to `true` when e_{t+1} contains a `last_action_error` field with a recognizable error class (timeout, stale element, navigation mismatch, request error). Cleared to `false` when a successful non-trivial action has been executed since the last error.

`locator_history`. A struct `(bid, click_count, steps_since_url_change)`. On each step: if a_t targets the same `bid` as the previous step, increment `click_count`; if the URL or page signature in o_{t+1} differs from the previous observation, reset `click_count` and `steps_since_url_change` to zero; otherwise increment `steps_since_url_change`.

`commit_affordance`. An enum over {`none`, `form_filled`, `submit_pending`}. Set to `form_filled` when a recent `type` or `select` action targeted a form field. Promoted to `submit_pending` when a form is filled and the current observation still exposes a submit-class control. Cleared to `none` when a submit-class action is observed in the recent history.

`url_changed`, `observation_changed`. Each is an enum over {`yes`, `no`, `unclear`}. Set to `yes` when the URL (resp. a stable page-signature hash of the accessibility tree) differs between the previous and current observation; set to `no` when both equal; set to `unclear` when only one observation is available (first step) or when a transient render is detected.

`visible_completion_evidence`, `visible_absence_evidence`. Each is an enum over {`yes`, `no`, `unclear`}. `visible_completion_evidence` is set by matching the current observation against a fixed regex bank for completion markers (saved-confirmation banners, success messages, created- resource URL paths, post-submit list rows containing the new content). `visible_absence_evidence` is set by matching against absence markers ("no records found", empty result tables, zero-result search responses).

`remaining_steps`. A small integer; the configured step budget minus the current step count.

`task_mode`. An enum over {`answer`, `write`, `navigate`, `unknown`}. Computed once per task by a fixed regex over x_{ins} (verbs like *tell, find, count* map to `answer`; *post, edit, change, notify* map to `write`; *open, show, navigate to* map to `navigate`).

`same_action_repeat_count`, `two_action_cycle_count`. Small integers tracking, respectively, the run length of the most-recent action class and the number of consecutive A/B alternations in the action stream when neither alternation produces an observation change. Both are reset on any action that produces an observation change.

A.3 Decision-time contract block

The contract block c_t is a fixed prompt segment plus a substrate-level parser rule. It does not vary with t (apart from small dynamic substitutions read off the state).

Stop-answer policy clause.

```
A valid stop content is the literal answer the task asks
for.
  - For numeric questions:  give the number, including 0.
  - When visible_absence_evidence is yes:  give N/A.
  - For yes/no questions:  give the answer.
The following are invalid stop contents:
  placeholders ('answer', 'placeholder', 'Done', 'Upvoted'),
  empty strings, hedged stops ('maybe...', 'I cannot...').
If visible evidence supports a concrete answer, commit to
it.
```

Forbidden-actions policy clause.

```
If same_action_repeat_count ≥ 3 with no observation change,
the most-recent action class is forbidden for the next step.
If two_action_cycle_count ≥ 3 with no observation change,
both members of the alternating pair are forbidden.
You must select a different action class or stop.
```

Parser-side guard rule. A substrate-level rule that rejects a stop attempt whose serialized content does not parse as a complete `stop [...]` expression (for example, a stop attempt truncated by a per-call output cap). A rejected stop becomes a no-op for the step and the policy is re-invoked next step. The guard does not change what the policy outputs; it only prevents an unintended substring from being parsed as the final answer.

A.4 EXAMPLE SERIALIZED STATE

Mid-episode on a shopping-admin task that asks for the order ID of the newest pending order, after the policy has navigated to `Sales > Orders` and applied a status filter, the serialized s_t that the policy reads is:

```
post_error_flag: false
locator_history: (bid=850, click_count=1,
steps_since_url_change=0)
commit_affordance: none
url_changed: yes
observation_changed: yes
visible_completion_evidence: no
visible_absence_evidence: no
remaining_steps: 23
task_mode: answer
same_action_repeat_count: 0
two_action_cycle_count: 0
```

The policy chooses the next action given this state, the current observation o_t, and the contract block c_t.

B DIAGNOSTIC PROBE DETAILS

This appendix gives the construction details of the two probes used in Section 3: the effective-information-distance probe (Figure 1) and the oracle-state intervention probe (Figure 2).

B.1 EFFECTIVE-INFORMATION-DISTANCE PROBE

Construction. The probe is built from real completed WEBARENA full-run trajectories. For each item we identify a paired *baseline-failed, method-succeeded* task, lift the method's final answer as the gold answer for the item, and construct five prompt variants of the same answer-extraction question.

Items. The final dataset has 27 item variants drawn from 12 distinct WEBARENA tasks. The selection rule: the strict baseline failed under the saved evaluator reward, a schema-equipped run succeeded on the same task, the schema-equipped final message contains an answer-like string, and the answer is not generic boilerplate (`done`, a bare integer, `unable to determine`). The items split per source: 24 items from distinct task IDs and 3 calibration variants of a single task whose ground-truth answer is read directly off the trajectory. The calibration variants are added so that LD is non-zero, avoiding a floor effect that would obscure the SD versus LD contrast.

Conditions. Each item is evaluated under five prompt layouts that share the same gold answer and the same model:

- **SC** (short, concentrated): only the current structured state with `final_answer` placed near the prediction site.
- **LC** (long, concentrated-but-far): early page-evidence containing the decisive content, followed by the long failed baseline history. Decisive content is concentrated but distant from the prediction site.
- **SD** (short, distributed): short snippets containing the answer pieces mixed with stale distractors; no full history.
- **LD** (long, distributed): the long failed baseline history alone.
- **LD+S** (long with state): the long failed history, followed by a compact structured state block carrying `final_answer`.

Correctness. The model's response is canonicalized (lowercased, punctuation, quotes, and trailing periods normalized, whitespace collapsed) and accepted if it exact-matches or contains the canonicalized gold answer. This is intentionally permissive: we measure whether the model retrieves and uses the right content, not whether it matches a display format exactly.

Model and result. We use `Qwen2.5-14B-Instruct` via a local OpenAI-compatible API; greedy decoding (`temperature`=0). The five conditions yield, in order: $27/27 = 100.0\%$ (SC), $24/27 = 88.9\%$ (LC), $13/27 = 48.1\%$ (SD), $3/27 = 11.1\%$ (LD), $27/27 = 100.0\%$ (LD+S).

Caveats. The 27 rows are item variants, not 27 distinct tasks. The source items are selected from baseline-failed and method-succeeded cases, so the probe targets a specific failure mode rather than measuring average WEBARENA performance. The calibration of LD through the three variants of one task is acknowledged: the figure should be read as a controlled diagnostic probe, not as an unbiased task sample.

B.2 ORACLE-STATE INTERVENTION PROBE

Construction. We collect 20 failure points from real strict-baseline WEBARENA trajectories. At each failure point, the model is asked to classify the correct next action class, twice. The two prompt conditions are:

- *baseline*: the task objective, the most recent trace, and the current observation. This mimics the policy's input under latest-observation conditioning.
- + *oracle state*: the same input, plus one additional block naming the unresolved environment fact for that case (an oracle slot, populated from ground truth).

The action-class taxonomy used for the classification target includes `PROBE`, `RECOVER`, `RELOCATE`, and `SUBMIT`, corresponding to the four regimes below.

Regimes and oracle slot content. The 20 cases are split evenly across four regimes, with five cases each:

- *Latent missing fact* (5 cases): the slot names the latent variable not yet retrieved (a price, an address, an account attribute), prompting a `PROBE`.
- *Execution noise* (5 cases): the slot reports the most recent action's error class (timeout, stale element, navigation mismatch), prompting a `RECOVER`.
- *Structural stale locator* (5 cases): the slot reports the locator's repeat-click count and missing URL change, prompting a `RELOCATE`.
- *Effect unverified* (5 cases): the slot reports that a form has been filled but no submit-class action has been issued, prompting a `SUBMIT`.

Model and result. We use `Qwen2.5-14B-Instruct` with greedy decoding. Per-regime accuracy (baseline $\rightarrow$ + oracle state): $5/5 \rightarrow 5/5$ (latent), $0/5 \rightarrow 5/5$ (execution), $2/5 \rightarrow 5/5$ (locator), $0/5 \rightarrow 3/5$ (unverified). Aggregated, $7/20 = 35\%$ versus $18/20 = 90\%$.

Caveats. The probe is a paired classification probe: it measures whether the correct next action class becomes selectable when the unresolved environment fact is exposed in a slot. It is not a measure of end-to-end task success, and it does not by itself establish that a deployed CAST pipeline can populate the same slot from observation alone (the probe uses an oracle slot). The latent-regime ceiling reflects the difficulty of the five sampled cases rather than absence of the marginalization trap in that regime.

C PERTURBATION CONSTRUCTION

This appendix gives the construction of the three controlled perturbation settings used in Section 6.2.

C.1 HIDDEN-FACT CHANGES (REGIME 1)

For a small set of WEBARENA tasks whose evaluator depends on a latent variable (a price, an inventory count, an account attribute), we vary the latent variable between episodes while holding the task instruction and the page DOM fixed. The agent's prior over the typical value is no longer authoritative. A gain on this setting is attributable to two pieces of the schema acting together. First,

the `task_mode`-conditioned absence flag `visible_absence_evidence` forces the policy to mark a fact as unresolved rather than committing on the prior whenever the page does not yet display it. Second, once the policy reaches a page that *does* display the latent value, the contract block requires the stop content to be the value that is visible at that point, so the policy is pushed to reground from the live page rather than fall back on a training-default guess. The schema instance reported here exposes absence directly and routes the grounded value through the contract clause; a schema with additional value-bearing slots (for example, a regex-extracted price field on pages with a known layout) is a natural extension within the same framework, and would be expected to lift the latent regime further. We therefore expect the gain here to be smaller than on regimes whose signal is fully captured by an existing slot, and we report the observed gain rather than commit to a closed account of where it comes from.

C.2 Transient action failures (Regime 2)

We inject one-step execution failures (timeouts, stale-element exceptions, race-condition redirects) into a fixed proportion of agent steps. The injection is observation-level: the environment returns the failed observation but does not block subsequent actions. Recovery requires a different action class (back, reload, relocate); pure retry does not unblock. A gain on this setting is attributable to the `post_error_flag` slot.

C.3 Cross-site variants (Regime 3)

For a subset of WebArena task templates, we map the task to a different site that exposes the same logical functionality with a different DOM layout and locator vocabulary. The schema is held fixed across sites; only the per-step environment signals e_{t+1} change. A gain on this setting is attributable to the `locator_history` slot, which is computed from e_{t+1} and is therefore site-agnostic by construction.

D Cost

CAST introduces no additional language-model call per step relative to the baseline: the runtime state is updated by deterministic code, and the policy issues a single language-model call per step in both arms. The per-step token budget is also comparable: CAST keeps the same policy-visible context as the baseline, $(x_{\text{ins}}, o_t, h_t^a)$, and adds the bounded state s_t and the fixed contract c_t near the prediction site, so the per-step input grows by at most a constant overhead determined by the schema. CAST does not expose the full per-step observation history $o_0, \ldots, o_{t-1}$ to the policy; the bounded state s_t is the only carrier of cross-step deterministic facts beyond the action labels in h_t^a. The remaining source of cost we report is episode length (number of steps to termination or budget exhaustion).

Hardware. Open-weight backbones (Qwen2.5-14B-Instruct and gpt-oss-20b) are served locally with vLLM on a single node with $4\times$ NVIDIA RTX PRO 6000 Blackwell GPUs (98 GB each); gpt-5.4 is accessed through its hosted API and uses no local GPU. The web environment runs in Docker on a separate CPU host. CAST uses the same hardware as the GenericAgent baseline on every backbone.

Episode length. Across the gpt-oss-20b runs reported in Table 1, the baseline takes 11.2 steps per episode on average, CAST takes 12.4, and GTTA (Chen et al., 2025) takes 15.8 (each capped at the 30-step budget). CAST therefore adds 1.2 steps per episode ($+11\%$) over the baseline while improving overall success rate by $+10.9$ pp on the same model. Compared to GTTA, CAST uses 3.4 fewer steps per episode (-22%) at a comparable success-rate level, because its update operator is in-loop and does not require a separate exploration phase before the task begins.

State block vs. baseline context in the prompt. The baseline prompt has four fields, in this order: `OBSERVATION` (the accessibility tree of the current page), `URL`, `OBJECTIVE` (the task instruction), and `PREVIOUS ACTIONS` (the numbered list of all past actions); the observation from the previous step is not kept. Its per-step input is therefore $(x_{\text{ins}}, o_t, h_t^a)$: the current observation o_t dominates per step, while the action history h_t^a grows linearly with t. CAST reads exactly this context and adds the bounded state s_t and contract c_t near the prediction site, so the per-step input is the baseline's

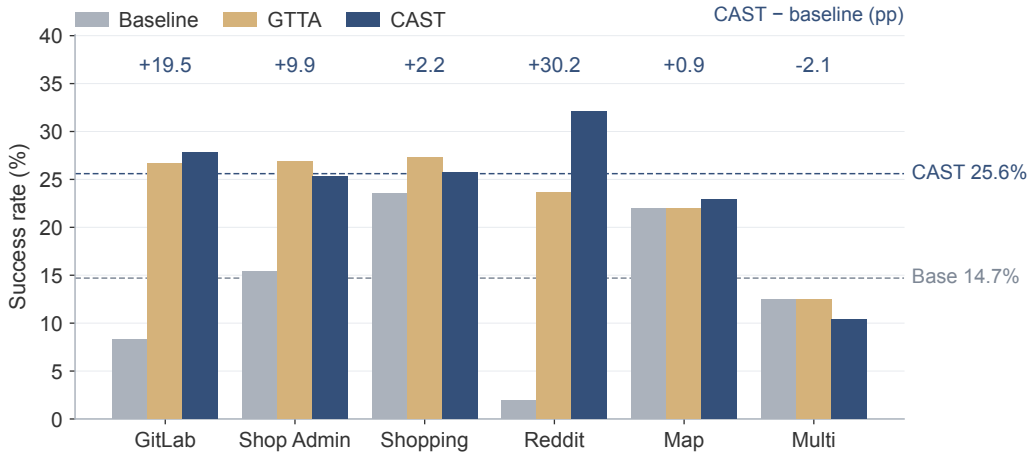


Figure 7: Per-site success rate on WEBARENA for gpt-oss-20b. Bars correspond to the gpt-oss-20b row of Table 1. Annotations show CAST − baseline in percentage points; horizontal dashed lines mark the corresponding per-backbone averages.

prompt plus a constant overhead determined by the schema. The added block does not change the asymptotic input-length scaling with the horizon; its role is to relocate load-bearing facts to a fixed position relative to the prediction site rather than to shorten the prompt. We do not separately ablate this scaling because all WEBARENA episodes terminate within 30 steps in our runs.

E PER-SITE MAIN-TABLE BREAKDOWN FOR GPT-OSS-20B

Figure 7 visualizes the gpt-oss-20b row of Table 1 site by site. The two largest absolute gains of CAST over the GenericAgent baseline come from *Reddit* ($1.9 \rightarrow 32.1$, $+30.2$ pp) and *GitLab* ($8.3 \rightarrow 27.8$, $+19.5$ pp), the two write-heavy sites where the baseline most often falls into the no-effect $\rightarrow$ same-action loop pattern (Section 4). *Multi* is the only site on which CAST loses ground (-2.1 pp on $n = 48$), so the per-backbone average gain is not driven by a single outlier site. Relative to GTTA the per-site picture is mixed: GTTA wins on *Shop Admin* and *Shopping*, CAST wins on *GitLab*, *Reddit*, and *Map*, and the per-backbone averages match within 0.6 pp (25.6 vs 25.0).

F SOCIAL IMPACT AND LIMITATIONS

CAST is a loop-level wrapper around an existing LLM policy and adds no extra language-model call per step, so its societal footprint largely inherits from the underlying model; all experiments are run inside the sandboxed WEBARENA environment, with no interaction against live production sites. Two scope notes are worth flagging. First, the runtime-state schema in this paper is shared across the six WEBARENA sites and was authored once for the benchmark; extending it to a qualitatively different agentic setting (for example, an OS-control or tool-using agent) is a natural follow-up rather than a re-derivation of the method. The update rules read generic signals (errors, URL and observation changes, repeated actions, remaining budget) and common page wording rather than anything specific to WEBARENA, so they can be reused as they are in a new environment; a one-time LLM query that follows the same design principles can further refine them for that environment. The set of state fields is fixed before deployment: CAST updates their values at every step but does not add new fields during an episode. Second, our empirical study covers three representative backbones spanning open and closed-weight models at different scales, but we do not sweep across every recent frontier model, and broader backbone coverage is left to future work.