FAC-CUA: FACTORIZED TRANSITION-LATENT REPRESENTATIONS FOR EFFICIENT LOCAL GUI CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

Computer-use agents repeatedly reason about what local action is intended, where that action should be grounded in the current interface, and whether the resulting GUI transition is consistent with task progress. Performing these decisions through large language or vision-language model inference at every interaction step can accumulate substantial latency, token usage, memory demand, and compute.

We introduce FAC-CUA, a factorized transition-latent representation learned from logged GUI interactions. Rather than compressing local-control state into a single embedding, FAC-CUA separates three control-functional roles: goal consistency, target grounding, and action-conditioned interface change. These factors support selective target grounding, post-action transition verification, and bounded self-recovery before LLM fallback.

We evaluate representation quality separately from system-level resource efficiency. Under frozen diagnostic probes, the evaluated factorized representation achieves 64.7% 4-way local-failure Macro-F1, compared with 39.3% for a monolithic representation trained on the same transition data. In a controlled 286-task evaluation with frozen planner outputs, FAC-CUA reaches 79.0% task success and 52.8% within-budget success (WBS), compared with 47.6% and 16.4% for the monolithic latent controller, while reducing median rollout wall-clock time from 89.55 s to 49.29 s. Without retraining or threshold calibration, the same controller further completes 17/40 tasks on a fixed WindowsAgentArena subset.

These results support the evaluated control-functional factorized architecture as a lightweight local-control substrate for GUI agents with structured actionable observations, while WBS provides a complementary system-level view of whether successful execution remains feasible under shared resource constraints.

1 INTRODUCTION

Computer-use agents (CUAs) have made substantial progress on realistic web and desktop tasks (Deng et al., 2023; Zhou et al., 2024; Xie et al., 2024; He et al., 2024). Yet interacting with a GUI requires repeated local reasoning even when an agent already has a high-level plan. At each interaction step, the agent must determine what local action is intended, resolve that intent to an executable target in the current interface, and assess whether the resulting interface transition is consistent with task progress. Repeatedly performing these decisions with large language or vision-language models can accumulate substantial latency, token usage, memory demand, and inference cost over a rollout.

This motivates a representation question rather than only a model-scaling question: can experience from prior GUI interactions be compressed into a reusable local-control state that preserves the information needed for grounding and transition verification? Local GUI control couples several distinct sources of information. The same subgoal may require different targets under different interface states, the same target may serve different subgoals, and the same action may induce different post-action changes. A single monolithic transition embedding can therefore make it difficult to expose which part of a local interaction is inconsistent.

We study a control-functional factorization of this transition state into three roles: *goal*, which captures progress consistency with the active subgoal; *ground*, which supports resolution of semantic actions to executable GUI targets; and *delta*, which captures action-conditioned changes in the interface. We do not assume that these factors are statistically independent or claim unsupervised disentanglement.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

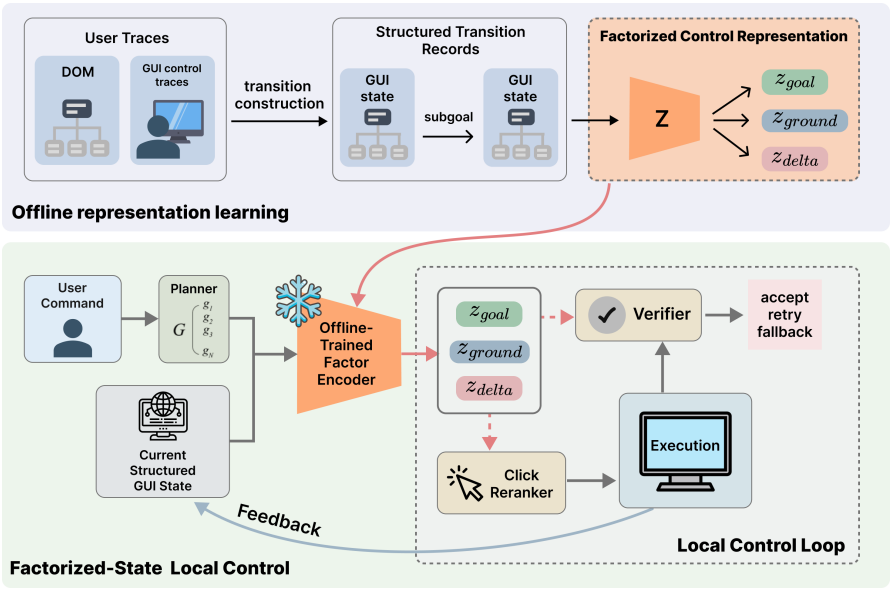

Figure 1: Overview of FAC-CUA. Offline, logged GUI interactions are converted into structured transition records and used to learn goal, ground, and delta factors. At runtime, pre-action context factors support target grounding. After execution, the observed post-action state and transition difference are encoded by frozen outcome-side branches for local consistency verification, which can trigger bounded recovery or LLM fallback.

Instead, we ask whether the resulting role-specific representation exposes failure-relevant information more accessibly than a monolithic representation trained on the same transition data.

We introduce FAC-CUA, a factorized transition-latent controller that uses these representations for selective target grounding, transition-consistency scoring, and verifier-guided self-recovery (Figure 1). FAC-CUA is not intended to replace an end-to-end CUA or its high-level reasoning. Rather, it studies a complementary local-control layer for settings in which an upstream agent provides a semantic action intent and the environment exposes structured actionable elements. Our focus is the repeated grounding and transition-validation computation that follows a proposed local intent.

We evaluate the representation and the resulting controller separately. First, under frozen diagnostic probes, the evaluated factorized representation achieves 64.7% 4-way local-failure Macro-F1, compared with 39.3% for a monolithic representation trained on the same transition data. Controlled interventions further show that incorrect subgoals, GUI targets, and post-action outcomes preferentially affect the Goal, Ground, and Delta signals, respectively. Second, in a controlled evaluation over 286 GUI tasks with frozen planner outputs, FAC-CUA achieves 79.0% task success and 52.8% WBS@Ref, compared with 47.6% task success and 16.4% WBS@Ref for the monolithic latent controller. Median rollout wall-clock time decreases from 89.55 s to 49.29 s.

We additionally evaluate transfer beyond the original task suite. Using the unchanged checkpoint, verifier, routing thresholds, recovery rules, and fallback policy, FAC-CUA completes 17/40 tasks on a fixed WindowsAgentArena subset spanning Web, System, Utilities, Coding, Media/Video, and Office tasks. We treat this result as evidence of transfer to a standardized OS-level benchmark rather than as a full-suite WindowsAgentArena estimate. Our current scope remains structured-observation GUI control; screenshot-only and canvas-heavy settings are outside the main evaluation regime.

In summary, this paper makes four contributions:

- We formulate repeated GUI local control as a transition-representation problem involving goal consistency, target grounding, and action-conditioned interface change.
- We introduce FAC-CUA, a factorized transition-latent representation with goal, ground, and delta factors, trained using context–outcome compatibility and counterfactual selectivity objectives.

- We provide representation-level evidence that the evaluated factorized representation exposes local failure identity more accessibly than the monolithic baseline under our diagnostic protocol, using controlled interventions and frozen diagnostic probes.
- We introduce within-budget success (WBS) as a resource-aware system-level evaluation protocol, explicitly separate it from representation-level diagnostics, and evaluate FAC-CUA under controlled resource budgets and a fixed WindowsAgentArena transfer setting.

2 PROBLEM SETUP: LOCAL GUI TRANSITION CONTROL

We study local GUI control given a semantic action intent from an upstream policy. At step t, the controller receives a local action specification $\bar{a}_t$, the current GUI state S_t^{pre}, an active subgoal g_t, and a short execution history. The action specification contains an event type together with an object or payload description from a finite action dictionary (Appendix B), but does not identify a concrete GUI coordinate, accessibility node, or executable target.

The controller must ground and parameterize $\bar{a}_t$ into an executable operation $\tilde{a}_t$, or abstain when grounding confidence is insufficient. After execution, the environment transitions to S_t^{post}. The controller may then use the observed post-action state to assess whether the transition is locally consistent with the intended subgoal and action.

This interface separates high-level action intent from local execution. Even when the semantic action is correct, its runtime target may differ across sessions because of layout changes, accessibility-name changes, focus state, dynamic content, or element availability. The local controller must therefore re-ground the semantic intent against the current interface.

In addition to task success, we record per-step resource cost $\rho_t = (T_t^{\text{step}}, N_t^{\text{tok}}, M_t^{\text{peak}}, C_t^{\text{TFLOPs}})$, denoting latency, LLM prompt-plus-completion token I/O, model-inclusive peak memory, and profiled compute. Given budget $B = (B_T, B_N, B_M, B_C)$, a rollout is within-budget successful iff the task succeeds and every step satisfies all four component-wise constraints. WBS is therefore a system-level metric reflecting execution reliability, fallback frequency, and resource use; representation quality is evaluated separately in Section 7. The exact indicator definition and accounting boundary are given in Appendix D.

3 RELATED WORK AND POSITIONING

Computer-use agents and GUI grounding. Recent CUA benchmarks and agents have advanced web, desktop, and mobile GUI automation through LLM/VLM reasoning, structured observations, visual grounding, and action execution (Deng et al., 2023; Zhou et al., 2024; Xie et al., 2024; He et al., 2024; Lù et al., 2024; Rawles et al., 2023; Drouin et al., 2024; Agashe et al., 2025; Wang et al., 2025). Grounding-oriented systems further map language or visual references to GUI targets using screenshots, element representations, or specialized parsers (Zheng et al., 2024; Cheng et al., 2024; Lu et al., 2024; Gou et al., 2025). Recent OS-level agents increasingly integrate perception, planning, grounding, and action generation in model-centric agent loops (Agashe et al., 2025; Wang et al., 2025). WindowsAgentArena and its Navi agent provide a standardized Windows OS benchmark and agent implementation (Bonatti et al., 2025), while UI-TARS studies native end-to-end GUI interaction (Qin et al., 2025).

FAC-CUA addresses a complementary question: given a semantic local intent, can repeated grounding and transition validation be handled through a reusable learned control representation rather than a full model call at every step? This local-control interface isolates execution-time grounding and verification, which remain necessary whenever a semantic action must be resolved against a changing GUI state.

Transition-latent and factorized representations. Logged GUI trajectories have been used for procedure mining, agent memory, and automation datasets (Wu et al., 2024; Song et al., 2024). Our use is different: we construct local transition records for representation learning. The objective is related to contrastive predictive representation learning, action-conditioned structured world models, and compact history representations (van den Oord et al., 2018; Kipf et al., 2020; Zhu et al., 2025).

Rather than learning one transition embedding, FAC-CUA assigns distinct control functions to goal, ground, and delta factors. This is related to factorized and controllable representation learning (Liang et al., 2023; Bengio et al., 2017; Thomas et al., 2017), but we do not claim unsupervised statistical disentanglement. Our factors are weakly supervised by control-relevant GUI fields, and we evaluate their role separation directly through controlled failure interventions and frozen diagnostic probes.

Resource-aware evaluation. Model and agent evaluation increasingly considers cost, latency, memory, and serving constraints alongside accuracy (Wang et al., 2024a; Reddi et al., 2020; Zhang et al., 2019; Kwon et al., 2023; Wang et al., 2024b). For CUAs, execution efficiency is likewise important in addition to task completion (**?**). We use WBS to summarize whether successful local-control rollouts satisfy shared resource limits, while reporting representation-level diagnostics and direct wall-clock measurements separately.

4 METHOD: FAC-CUA

FAC-CUA has four stages: transition-record construction, context/outcome splitting, factorized goal–ground–delta learning, and runtime grounding with verifier-guided recovery. Figure 1 summarizes the overall pipeline, while Figure 2 details the factorized representation. The full transition-record construction pipeline is shown in Appendix A.3 (Figure 5).

4.1 STRUCTURED GUI TRANSITION RECORDS

We convert raw GUI logs into structured transition records centered on local-control steps. Each record aligns a pre-action GUI state, a logged execution, and a post-action GUI state. The complete construction pipeline is provided in Appendix A.3 (Figure 5).

From each GUI state, we extract a compact slot representation $U_t = \phi(S_t)$, containing visible and actionable UI elements such as labels, control types, enabled/visible status, hierarchy context, and identity hints. We also construct a short UI-grounded state summary $s_t = \psi(S_t)$, and compute a symbolic transition diff $\Delta_t = \delta(U_t^{\mathrm{pre}}, U_t^{\mathrm{post}})$ over controls that appear, disappear, or change after the action.

The resulting record is $r_t = (e_t, g_t, a_t, s_t^{\mathrm{pre}}, s_t^{\mathrm{post}}, U_t^{\mathrm{pre}}, U_t^{\mathrm{post}}, \Delta_t)$, where e_t, g_t, and a_t denote the environment, local subgoal, and semantic action specification derived from the logged execution.

Clicked-target evidence from collected logs is used only for offline supervision. At runtime, the pre-action selector observes only the current pre-action state, planner-proposed semantic action, subgoal context, and candidate slots. After an action is executed, the newly observed post-action state and slot-level transition diff become available to the verifier for post-action consistency scoring; they are never available to the selector before acting.

4.2 FACTORIZED TRANSITION-LATENT REPRESENTATION

As illustrated in Figure 2, each transition is split into context $C_t = (e_t, g_t, a_t, s_t^{\mathrm{pre}}, U_t^{\mathrm{pre}})$ and outcome $O_t = (e_t, s_t^{\mathrm{post}}, U_t^{\mathrm{post}}, \Delta_t)$ views. The source logs contain grounded executions, but the encoder receives a canonical event–argument semantic action rather than an executable coordinate or accessibility-node identifier; at runtime the same pathway receives planner action $\bar{a}_t$. Only context is available before execution, while observed post-action state and diff enter frozen outcome-side branches afterward.

FAC-CUA represents each transition as $z_t = (z_t^{\mathrm{goal}}, z_t^{\mathrm{ground}}, z_t^{\Delta})$. The goal factor represents consistency with the active subgoal, the ground factor represents information for target selection, and the delta factor represents action-conditioned interface change. These roles are control-functional rather than statistically independent.

Each factor has context-side and outcome-side branches with role-specific inputs. Goal uses subgoal and state-summary fields; ground uses structured GUI and candidate-element fields; and delta uses action-conditioned state and transition-difference fields. The Ground factor encodes candidate-local GUI evidence rather than action semantics itself; action compatibility enters Stage 2 through action-conditioned Delta features and selector interaction features.

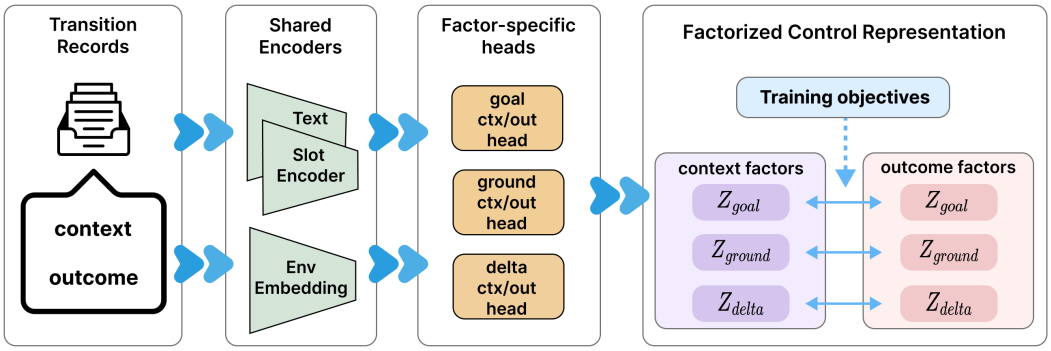

Figure 2: Factorized transition-latent representation. Each transition is separated into context-side and outcome-side views and encoded into goal, ground, and delta factors. Pre-action target selection uses context-side factors only, while post-action verification additionally encodes the observed post-action state and transition difference through frozen outcome-side branches.

Pre-action target selection uses only context-side representations and has no access to post-action information. After execution, however, the observed post-action state and transition diff are processed by the same frozen outcome-side factor branches used during training. Goal- and delta-side outcome representations are then consumed by post-action consistency scoring. The shared outcome encoding also materializes the ground-side factor, although it is not consumed by a post-action verifier scoring head. Thus, outcome-side computation is part of runtime verification but never enters pre-action target selection.

Full field definitions are provided in Appendix C.

4.3 Learning Objectives

We train the factorized representation with $\mathcal{L} = \mathcal{L}_{\text{comp}} + \lambda_{\text{sel}}\mathcal{L}_{\text{sel}}$. The factor-wise compatibility term $\mathcal{L}_{\text{comp}}$ is an InfoNCE-style objective that retrieves the matching outcome embedding from a context embedding. The selectivity term $\mathcal{L}_{\text{sel}}$ applies factor-specific counterfactual edits: subgoal, slot-set, and action edits target Goal, Ground, and Delta, respectively. With $d_f(r, r') = 1 - \cos(z_f^{\text{ctx}}(r), z_f^{\text{ctx}}(r'))$, the hinge encourages the targeted factor to change more than non-target factors by margin m. We additionally use control-confusable negatives such as mismatched outcomes, wrong actions under the same subgoal, and similar targets with different post-action changes. Full equations, negative construction, and hyperparameters are in Appendix C.3.

4.4 Runtime Grounding and Verifier-Guided Self-Recovery

At runtime, the planner provides a semantic local action $\bar{a}_t$, which FAC-CUA re-grounds against the current GUI state rather than treating as an executable target identifier. The controller then executes, abstains, attempts bounded recovery, or routes the step to fallback.

Selective target grounding. For target-dependent actions, FAC-CUA uses a two-stage selector over the current candidate set $C_t = \{c_{t,i}\}_{i=1}^{N_t}$. Stage 1 applies a high-recall gate to retain a compact set $K_t \subseteq C_t$, while Stage 2 ranks the retained candidates using grounding-side interaction features, action-conditioned delta features, and lightweight runtime metadata. The selector executes the top-ranked candidate only when both its normalized score and top1–top2 margin pass fixed routing thresholds; otherwise, the step is passed to the recovery router.

Verifier-guided self-recovery. After executing an action, FAC-CUA observes the resulting GUI state and constructs the corresponding post-action slot representation and transition diff. The frozen outcome-side factor branches encode this post-action evidence for local consistency verification.

The recovery router combines three signals: a pre-execution grounding score, a post-execution goal-consistency score, and a delta-consistency score over the observed interface change. These scores are routing signals rather than task-completion certificates. When the transition is locally inconsistent, the controller may wait and re-observe or select a safe retained alternative before invoking the LLM fallback.

Fallback routing. If grounding evidence is weak before execution, the controller abstains. If post-action delta evidence is inconsistent, the router first performs bounded waiting or one safe retry. If a retained alternative has high grounding evidence and sufficient margin, the controller may retry it without invoking the LLM. Remaining unresolved, unsupported, or low-confidence cases use fallback. Routine steps are handled by the learned selector and verifier, which reduces LLM-mediated local-control calls. For side-effectful or user-sensitive actions, self-recovery is disabled and the controller routes to fallback or user intervention. Thresholds, retry limits, fallback prompts, and the action-level safety policy are provided in Appendix D.2

5 EXPERIMENTS

5.1 EVALUATION PROTOCOL

We evaluate FAC-CUA along three complementary axes.

First, we run a controlled planner–controller evaluation over 286 GUI tasks. High-level planner outputs are generated once and cached, so all controller variants receive the same semantic action sequence and differ only in local representation, grounding, verification, recovery, and fallback behavior. This is our primary system-level comparison.

Second, we evaluate representation quality independently of LLM invocation policy. We measure target grounding on held-out target-dependent transitions, perform controlled subgoal/target/outcome interventions, and compare the factorized representation with a monolithic representation trained on the same transition data under a common frozen diagnostic protocol.

Third, we evaluate transfer beyond the original task suite. We run the unchanged FAC-CUA checkpoint and routing configuration on a fixed 40-task WindowsAgentArena subset using an environment-level observation/action adapter. We also report a task-level comparison against an end-to-end UI-TARS baseline in a locally instrumented execution-cost audit in the Appendix.

For resource-aware evaluation, WBS@Ref uses per-step budgets $B_T = 10$s, $B_N = 8$k, $B_M = 8$GB, and $B_C = 100$ TFLOPs-proxy. All selector and verifier thresholds were selected on held-out development data and frozen before the 286-task and transfer evaluations.

5.2 TASKS, BASELINES, AND METRICS

The primary controlled evaluation contains 286 GUI tasks: 100 in-house tasks, 116 WebVoyager tasks (He et al., 2024), and 70 ScreenAgent tasks (Niu et al., 2024). The tasks span web interfaces, desktop applications, productivity workflows, and cross-application interactions.

The representation-learning logs were manually collected by the authors from 175 continuous Windows GUI sessions. Using an episode-level split, the reported checkpoint is trained on 1,175 structured transition records and validated on 444 held-out records, with no episode-ID overlap between the two splits. These source logs were collected independently of the WebVoyager and ScreenAgent evaluation rollouts.

For the controlled comparison, all methods receive the same frozen planner outputs. We compare: (1) an every-step Qwen2.5-VL-7B controller with Qwen-based transition verification; (2) a monolithic latent controller trained on the same transition data with LLM fallback; (3) FAC-CUA without verifier-guided self-recovery; and (4) full FAC-CUA.

The every-step Qwen baseline uses the model as the primary local controller: given the semantic planner action and current structured GUI state, it generates and grounds an executable local action and separately verifies the transition. The learned latent controllers instead invoke Qwen only when a step is routed to fallback.

Table 1: Controlled local-execution comparison over 286 tasks with identical frozen planner outputs. Wall-clock and max-step latency are medians over task rollouts. WBS@Ref uses the fixed reference resource budget.

Method	Success (%) ↑	WBS@Ref (%) ↑	Task Prog. (%) ↑	LLM Calls per Task ↓	Median Wall-clock (s) ↓	Median Max-step Latency (s) ↓
Every-step Qwen controller + verifier	29.4	1.7	47.7	9.67	104.90	9.50
Monolithic latent + fallback	47.6	16.4	75.6	2.50	89.55	7.93
FAC-CUA w/o self-recovery	73.1	23.8	89.3	5.30	63.65	6.80
FAC-CUA	**79.0**	**52.8**	**92.7**	**2.10**	**49.29**	**5.15**

Evaluation rollouts do not provide clicked-target labels, post-action states, or transition diffs to the selector before execution. For the in-house test tasks, evaluation episodes are collected separately from the training episodes; WebVoyager and ScreenAgent evaluation rollouts are never used for selector or verifier supervision.

We report task success, task progress, WBS@Ref, LLM-mediated local-control calls, median total rollout wall-clock time, and median maximum step latency. WBS is interpreted as a system-level outcome. Representation quality is evaluated separately through target-selection metrics, controlled intervention diagnostics, and frozen-probe failure classification.

6 RESULTS

6.1 CONTROLLED LOCAL-EXECUTION PERFORMANCE

Table 1 reports the primary controlled comparison. All methods receive identical frozen planner outputs and therefore differ only in local representation, grounding, verification, recovery, and fallback behavior rather than high-level planning.

The every-step Qwen controller reaches 29.4% task success, while the monolithic latent controller reaches 47.6%. FAC-CUA increases task success to 79.0% and task progress to 92.7%. It also achieves 52.8% WBS@Ref, compared with 16.4% for the monolithic latent controller.

Importantly, WBS is not our sole efficiency indicator. Relative to the every-step Qwen controller, FAC-CUA reduces median total rollout wall-clock time from 104.90 s to 49.29 s and median maximum step latency from 9.50 s to 5.15 s. The self-recovery ablation further shows that bounded local recovery increases task progress while substantially reducing fallback usage: full FAC-CUA uses 2.10 LLM-mediated calls per task, compared with 5.30 calls without self-recovery.

At the reference budget, full FAC-CUA violates latency, token, memory, and compute constraints in 10.8%, 36.4%, 35.3%, and 35.7% of rollouts, with 46.5% violating at least one. Because WBS combines representation, execution, recovery, and resource use, we do not interpret its gap as a pure representation effect; the Appendix reports the full constraint decomposition and budget sensitivity.

6.2 TRANSFER TO WINDOWSAGENTARENA

We next evaluate whether the same local controller transfers beyond the original task suite. We integrate FAC-CUA into WindowsAgentArena (Bonatti et al., 2025) through an environment-level observation/action adapter while keeping the trained checkpoint, verifier, selector thresholds, recovery rules, and fallback policy unchanged.

We evaluate a fixed 40-task subset containing all 30 Web tasks and two tasks from each of five additional domains: System, Utilities, Coding, Media/Video, and Office. Task outcomes are measured using the official WindowsAgentArena evaluators.

FAC-CUA completes 17/40 tasks without retraining or threshold calibration. We interpret this experiment as evidence that the learned local controller can transfer to a standardized Windows benchmark under a fixed configuration. Because the evaluation covers only 40 tasks and the non-Web domains contain two tasks each, it should not be interpreted as a full-suite WindowsAgentArena performance estimate.

Table 2: Fixed 40-task WindowsAgentArena transfer. FAC-CUA is unchanged; Navi (Bonatti et al., 2025) uses GPT-4o in place of its unavailable GPT-4V preview backend, so this is not an exact reproduction.

Method	Task Success ↑
Navi (mixed-omni + UIA, GPT-4o)	11/40 (27.5%)
FAC-CUA	**17/40 (42.5%)**

Table 3: Target-selection ablation on strict target-dependent steps. All methods use only pre-action context at selection time. *Aux* denotes compact runtime candidate metadata; clicked-target evidence from training logs is used only for offline supervision.

Method	Top-1 (%) ↑	MRR (%) ↑	Mean rank ↓
Aux only	70.9	74.2	8.04
Monolithic latent only	54.1	55.4	8.12
Factorized latent only	65.9	77.1	7.50
Monolithic latent + aux	76.8	83.3	4.91
Factorized latent + aux (FAC-CUA)	**88.8**	**90.3**	**3.10**

A complementary task-level comparison against locally hosted UI-TARS-1.5-7B is reported as a locally instrumented execution-cost diagnostic in Appendix E.6.

7 Analysis and Ablations

7.1 Local Target Selection

We isolate target-dependent steps in which the upstream semantic action must be resolved to one executable element from the pre-action candidate set. Evaluation uses only pre-action information; clicked-target evidence from training logs is used only for offline supervision.

Table 3 compares the factorized representation with the monolithic baseline, with *aux* denoting lightweight pre-action candidate metadata. The evaluated factorized representation achieves 65.9% Top-1 in the latent-only setting versus 54.1% for the monolithic baseline and, with auxiliary features, 88.8% versus 76.8%. As with the frozen-probe diagnostic, this is an architecture-level representation comparison rather than a parameter-controlled factorization ablation.

7.2 Failure-Source Accessibility

The motivation for the factorized design is not only target accuracy, but whether distinct local failure sources are accessible in the learned representation. We test this through two complementary diagnostics: controlled interventions and frozen diagnostic probes.

For controlled interventions, 15 held-out contexts are randomly selected before outcomes are inspected: five each with an incorrect subgoal, GUI target, or post-action outcome. Only the failure-relevant component changes, and we measure target-factor score drop versus the largest off-target change.

For frozen probes, factor-specific synthetic counterfactual failures are constructed from structured transitions. The 4-way task distinguishes *clean*/Goal/Ground/Delta failures and the 3-way task excludes clean. Factorized and monolithic conditions use the same examples, selected-candidate identities, split indices, and binary-head architecture, but representation-specific feature construction differs.

Interventions produce the largest change in the intended factor for 12/15 pairs (mean target drop 0.0706; off-target change 0.0007). Frozen probes reach 64.7% versus 39.3% 4-way Macro-F1 and 71.6% versus 58.2% 3-way Macro-F1 for factorized versus monolithic representations. We interpret these as targeted failure-source accessibility diagnostics, not population estimates, natural-failure

Table 4: Failure-source accessibility through **(A)** controlled paired interventions and **(B)** frozen diagnostic probes. Probe values are mean ± standard deviation over three probe-training seeds (42, 43, 44).

(A) Controlled interventions

Intervention	Factor	Target Drop ↑	Positive Cases	Off-target Change ↓
Wrong subgoal	Goal	0.0727	4/5	0.0008
Wrong GUI target	Ground	0.0613	4/5	0.0006
Wrong post-action outcome	Delta	0.0779	4/5	0.0007
Overall	–	**0.0706**	**12/15**	**0.0007**

(B) Frozen diagnostic probes

Representation	4-way Macro-F1 ↑	3-way Failure Macro-F1 ↑
Monolithic latent	39.3 ± 2.6	58.2 ± 5.3
FAC-CUA	**64.7 ± 3.2**	**71.6 ± 4.8**

frequencies, or dimension-/parameter-controlled causal evidence. Full probe and objective-ablation details are in the Appendix.

8 SCOPE AND LIMITATIONS

FAC-CUA assumes structured actionable GUI elements and addresses local execution rather than high-level planning; screenshot-only/canvas-heavy interfaces remain outside scope. WindowsAgentArena, UI-TARS, and long-horizon results are limited diagnostics rather than full-benchmark estimates. WBS depends on fixed budgets and manually specified routing/safety structure. The frozen probe uses synthetic role-aligned counterfactuals and non-matched feature constructions, so it is not natural-failure or capacity-controlled causal evidence. The UI-TARS audit excludes upstream planner-generation cost, and FAC-CUA still relies on LLM/VLM components.

AI USE STATEMENT

Generative AI tools were used in several roles in this work. First, LLMs were used as components of the studied system, including offline GUI annotation, high-level semantic planning, and fallback local-control decisions, as described in the main paper and Appendix. Second, generative AI tools were used to assist with implementation of standard methods and with manuscript editing, including grammar, wording, and formatting. AI-assisted annotations, code, analyses, and manuscript edits were reviewed, validated, or revised by the authors as appropriate. The authors take full responsibility for the experimental methodology, reported results, claims, code, and final manuscript.

REFERENCES

Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s2: A compositional generalist-specialist framework for computer use agents. *arXiv preprint arXiv:2504.00906*, 2025. URL https://arxiv.org/abs/2504.00906.

Emmanuel Bengio, Valentin Thomas, Joelle Pineau, Doina Precup, and Yoshua Bengio. Independently controllable features. *arXiv preprint arXiv:1703.07718*, 2017.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Keunho Jang, and Zheng Hui. Windows agent arena: Evaluating multi-modal OS agents at scale. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 4874–4910. PMLR, 2025. URL https://proceedings.mlr.press/v267/bonatti25a.html.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, YanTao Li, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9313–9332, 2024.

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. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=kiYqbO3wqw.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable are web agents at solving common knowledge work tasks? In *International Conference on Machine Learning*, pp. 11642–11662, 2024. URL https://proceedings.mlr.press/v235/drouin24a.html.

Boyu Gou, Demi Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for GUI agents. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=kxnoqaisCT.

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. doi: 10.18653/v1/2024.acl-long.371. URL https://aclanthology.org/2024.acl-long.371/.

Thomas Kipf, Elise van der Pol, and Max Welling. Contrastive learning of structured world models. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1gax6VtDB.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Paul Pu Liang, Zihao Deng, Martin Q. Ma, James Zou, Louis-Philippe Morency, and Ruslan Salakhutdinov. Factorized contrastive learning: Going beyond multi-view redundancy. In *Advances in Neural Information Processing Systems*, 2023.

Xing Han Lù, Zdenek Kasner, and Siva Reddy. WebLINX: Real-world website navigation with multi-turn dialogue. In *International Conference on Machine Learning*, 2024.

Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. OmniParser for pure vision based GUI agent. *arXiv preprint arXiv:2408.00203*, 2024.

Runliang Niu, Jindong Li, Shiqi Wang, Yali Fu, Xiyu Hu, Xueyuan Leng, He Kong, Yi Chang, and Qi Wang. ScreenAgent: A vision language model-driven computer control agent. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, 2024. doi: 10.24963/ijcai.2024/711. URL https://doi.org/10.24963/ijcai.2024/711.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. UI-TARS: Pioneering automated GUI interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the wild: A large-scale dataset for android device control. In *Advances in Neural Information Processing Systems*, 2023.

Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. MLPerf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 446–459, 2020.

Yifan Song, Weimin Xiong, Xiutian Zhao, Dawei Zhu, Wenhao Wu, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. AgentBank: Towards generalized LLM agents via fine-tuning on 50000+ interaction trajectories. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 2124–2141, 2024. doi: 10.18653/v1/2024.findings-emnlp.116. URL https://aclanthology.org/2024.findings-emnlp.116/.

Valentin Thomas, Jules Pondard, Emmanuel Bengio, Marc Sarfati, Philippe Beaudoin, Marie-Jean Meurs, Joelle Pineau, Doina Precup, and Yoshua Bengio. Independently controllable factors. *arXiv preprint arXiv:1708.01289*, 2017.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Junlin Wang, Siddhartha Jain, Dejiao Zhang, Baishakhi Ray, Varun Kumar, and Ben Athiwaratkun. Reasoning in token economies: Budget-aware evaluation of LLM reasoning strategies. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 19916–19939, 2024a.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Boyuan Zheng, Peihang Li, Fangyu Lei, Ruisheng Cao, Yeqiao Fu, Dongchan Shin, Martin Shin, Jiarui Hu, Yuyan Wang, Jixuan Chen, Yuxiao Ye, Danyang Zhang, Yipu Wang, Heng Wang, Diyi Yang, Victor Zhong, Y. Charles, Zhilin Yang, and Tao Yu. OpenCUA: Open foundations for computer-use agents. In *Advances in Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=6iRZvJiC9Q.

Zhibin Wang, Shipeng Li, Yuhang Zhou, Xue Li, Zhonghui Zhang, Cam-Tu Nguyen, Rong Gu, Chen Tian, Guihai Chen, and Sheng Zhong. Revisiting service level objectives and system level metrics in large language model serving. *arXiv preprint arXiv:2410.14257*, 2024b.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and Yu Qiao. OS-Atlas: A foundation action model for generalist GUI agents. *arXiv preprint arXiv:2410.23218*, 2024. URL https://arxiv.org/abs/2410.23218.

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 Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=tN61DTr4Ed.

Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. MArk: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pp. 1049–1062, 2019. URL https://www.usenix.org/conference/atc19/presentation/zhang-chengliang.

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

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

Xiyue Zhu, Peng Tang, Haofu Liao, and Srikar Appalaraju. Turbocharging web automation: The impact of compressed history states. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 3644–3651, 2025. doi: 10.18653/v1/2025.findings-acl.187. URL https://aclanthology.org/2025.findings-acl.187/.

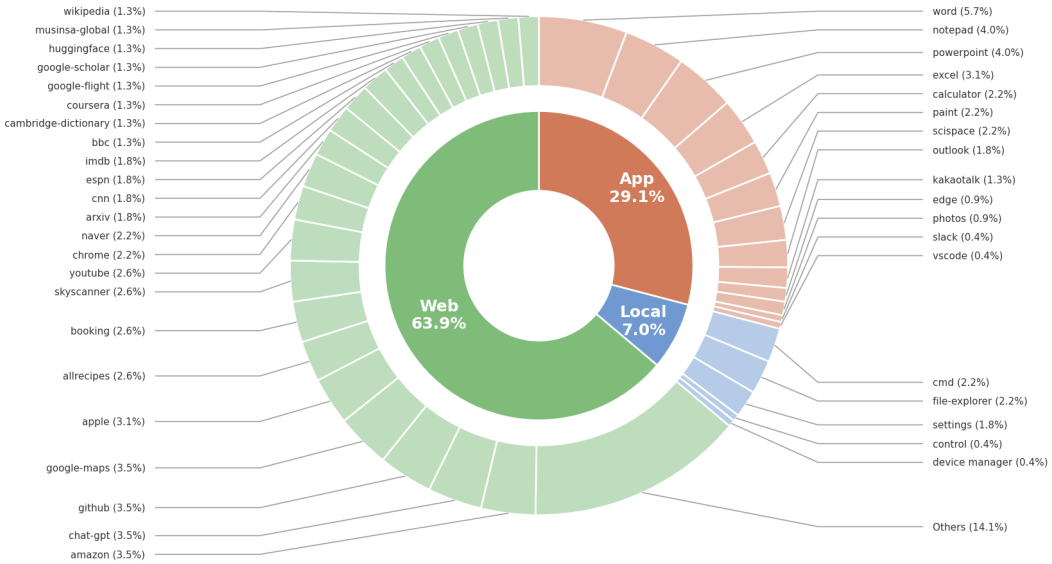

Figure 3: Domain composition of the collected training logs. The inner ring summarizes the environment-level distribution, while the outer ring shows the platform-level distribution. Platform counts are computed at the occurrence level; when multiple platforms appear in one log entry, each platform is counted separately. Low-frequency Web platforms are grouped into Others for readability.

A Data Construction and Leakage Control

This appendix describes the construction of the structured GUI transition records used for representation learning and clarifies the separation between offline supervision and runtime observations.

A.1 Data Collection and Recorded Signals

The source interaction logs were manually collected by the authors from 175 continuous Windows GUI sessions spanning browser workflows, desktop applications, and local-system interactions. A session may contain multiple workflows and may cross application or domain boundaries.

Each logged episode contains a user command, a sequence of primitive keyboard and pointer events, active-window metadata, and UIA snapshots captured around state-changing or action-centered boundaries. For pointer actions, the collector additionally stores a best-effort local subtree around the clicked element. This click-centered evidence is used only to construct offline grounding supervision.

The logged workflows include common local-control patterns such as opening applications, navigating websites, selecting menus, entering text, copying or saving content, and switching between windows. Evaluation rollouts are executed separately from the collected source sessions and are not replayed from logged trajectories.

A.2 Environment Distributions

Figures 3 and 4 summarize the environment composition of the collected training logs and the held-out 286-task evaluation set. These figures are descriptive diagnostics: the training logs were not constructed to match the evaluation distribution or to specialize the controller to the test cases.

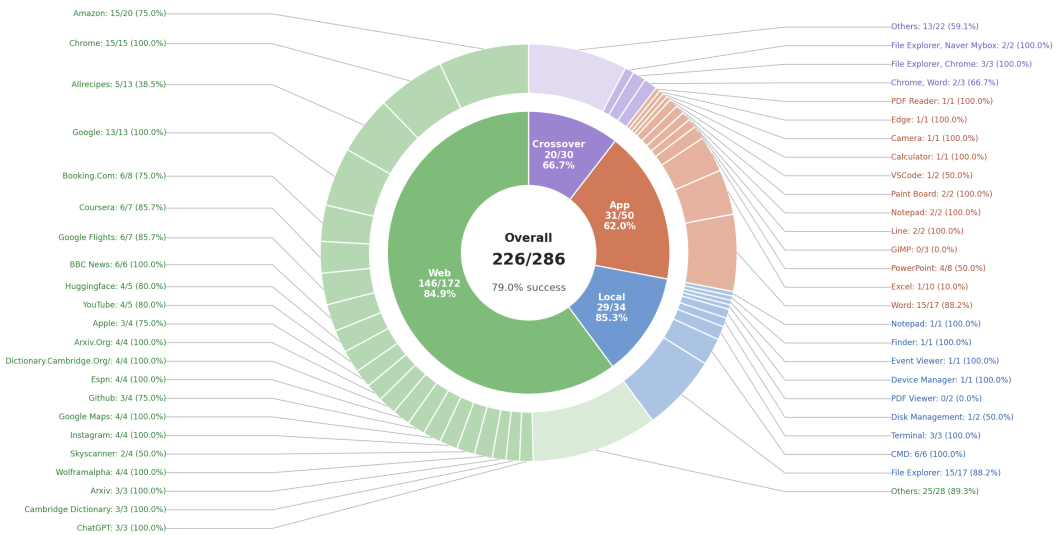

Figure 4: Environment- and platform-level success rates on the 286-task evaluation set. The inner ring shows the success rate for each environment category, and the outer ring shows the success rate for each platform within that environment. Each outer-ring label reports success count / platform occurrence count together with the corresponding success rate. Low-frequency platforms are grouped into Others where appropriate.

A.3 Transition-Record Construction

Each raw interaction sequence is converted into structured local transitions. The preprocessing pipeline consists of action/UI capture, stable-state normalization, control-salient pruning, slot abstraction, pre/post alignment, and symbolic transition-diff construction.

For each GUI state S_t, we extract a compact set of actionable slots

$$U_t = \phi(S_t).$$

A slot contains a normalized label, control type, visibility and enabled status, hierarchy context, stable identity hints when available, and optional geometry.

Given pre- and post-action slot sets, we construct

$$\Delta_t = \delta(U_t^{\text{pre}}, U_t^{\text{post}}),$$

which summarizes controls that appear, disappear, or change after an action.

The final transition record is

$$r_t = (e_t, g_t, a_t, s_t^{\text{pre}}, s_t^{\text{post}}, U_t^{\text{pre}}, U_t^{\text{post}}, \Delta_t),$$

where e_t denotes the environment, g_t the local subgoal, a_t the semantic action specification derived from the logged execution, and $s_t^{\text{pre/post}}$ UI-grounded state summaries.

The corresponding training views are

$$C_t^{\text{train}} = (e_t, g_t, a_t, s_t^{\text{pre}}, U_t^{\text{pre}})$$

and

$$O_t = (e_t, s_t^{\text{post}}, U_t^{\text{post}}, \Delta_t).$$

During training, clicked-target evidence and post-action observations provide offline supervision. At runtime, the logged action a_t is replaced by the upstream semantic action $\bar{a}_t$.

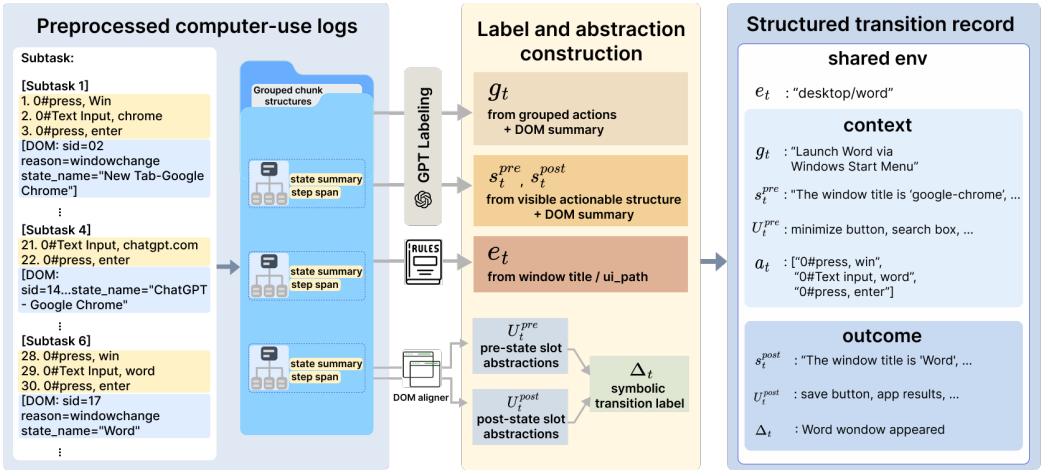

Figure 5: Data construction pipeline for structured GUI transition records. Preprocessed computer-use logs are segmented into grouped action chunks, then labeled with subgoal, action, environment, slot-abstraction, and transition-diff information using GPT labeling, rules, and DOM alignment. The resulting record organizes each transition into a shared environment, pre-action context, and post-action outcome for control-oriented representation learning.

A.4 Train–Evaluation Separation and Information Boundary

Using an episode-level split, the reported checkpoint uses 1,175 structured transition records for training and 444 held-out records for representation-level validation. There is no episode-ID overlap between the two splits.

The source sessions used to construct these records were collected independently of the WebVoyager and ScreenAgent evaluation rollouts. For the in-house evaluation tasks, test episodes are collected separately from the training episodes.

At runtime, the pre-action selector observes only information available before execution: the current structured GUI state, the planner-proposed semantic action, the active subgoal context, and the current candidate slots. It does not observe the clicked-target subtree, post-action state, or transition diff before selecting an action.

After execution, the newly observed post-action state and slot-level diff become available to the verifier. These observations are processed through the frozen outcome-side factor branches for post-action consistency scoring. Thus, post-action evidence participates in runtime verification but never enters pre-action target selection.

This separation ensures that evaluation measures re-grounding of semantic actions in the current GUI state rather than replay of logged clicked targets or historical interface paths.

A.5 Privacy and Sensitive Interaction Filtering

Raw GUI interaction logs may contain private or application-specific interface text. We therefore remove or mask credentials, payment information, personal identifiers, private document contents, and other user-sensitive fields when constructing structured transition records.

Actions involving credentials, CAPTCHA, payment approval, account changes, or irreversible external side effects are marked as user-sensitive. Such actions are not eligible for autonomous verifier-guided self-recovery and are routed to fallback, explicit confirmation, or user intervention.

Table 5: Summary of the planner-level action space. The planner specifies semantic events, while the local controller grounds target-dependent operations against the current GUI state.

Event group	Events and execution meaning
Pointer actions	`click`, `doubleclick`, `rightclick`, `drag`, and `scroll`; require grounding to a visible target or region.
Keyboard/input actions	`text input` and `press`; perform text entry or key sequences, optionally after resolving a target field.
Navigation/window actions	`open`, `close`, `switch focus`, and `go-to`; launch applications, navigate to destinations, or change the active window.
File/object operations	`save`, `copy`, `paste`, `delete`, and `rename`; combine target resolution with keyboard or menu operations.

B Planner Interface and Action Space

The upstream planner produces semantic local actions rather than executable GUI targets. Each planner step specifies an event type together with an object, payload, or destination description. The local controller must then resolve this semantic intent against the current GUI state.

Planner steps follow the representation

```
assist_bit# Event, object.
```

This interface separates high-level intent from local execution. A planner may specify that a particular field should be edited or a particular item opened, but it does not provide a runtime coordinate, accessibility node, or executable target identifier.

User-sensitive or irreversible steps marked through the interaction policy are not autonomously recovered and are handled according to the safety policy in Appendix D.2.

C Model and Training Details

C.1 Encoder Architecture

We initialize the factorized model from a pretrained monolithic transition encoder. The initialization model uses `sentence-transformers/all-MiniLM-L6-v2` as the text backbone and provides shared text, environment, control-type, and slot encoders.

The factorized student copies these shared components and adds separate goal, ground, and delta factor heads. In the reported configuration, the shared encoder is not frozen. The monolithic encoder emits a 256-dimensional transition latent. The factorized student emits three 128-dimensional role-specific factors and additionally projects their concatenated context and outcome states through separate pooling modules to 256-dimensional summaries. Because the factorized model exposes multiple role-specific states and additional role-specific heads, we do not describe the comparison as dimension- or parameter-matched. The monolithic representation is instead a baseline trained on the same structured transition data and evaluated under the same controlled planner-output setting.

Each factor contains a context-side adapter and an outcome-side adapter followed by a shared projection. The adapters are small MLPs with layer normalization and ReLU activations. Factor embeddings are ℓ_2-normalized before similarity-based objectives are evaluated.

The ground factor deliberately excludes action text from its context-side input, encouraging target evidence to depend on the current GUI and candidate elements, while the delta factor captures action-conditioned state change.

Before an action is executed, only context-side representations are available to target selection. After execution, the newly observed post-action state and transition diff are processed through the same frozen outcome-side factor branches used during training. Goal- and delta-side outcome representations are consumed by post-action verifier computations. The shared outcome encoding also materializes the ground-side factor, although it is not consumed by a post-action verifier scoring head.

Table 6: Role-specific input fields of the factorized transition encoder.

Factor	Context-side fields	Outcome-side fields
Goal	Subgoal text, pre-action state summary, environment	Post-action state summary, environment
Ground	Pre-action state summary, pre-action slot set, environment	Post-action state summary, post-action slot set, environment
Delta	Pre-action state summary, semantic action specification, pre-action slot set, environment	Slot-level transition diff, post-action slot set, environment

C.2 Monolithic Baseline Construction

The monolithic transition encoder is trained first on the same 1,175 structured transition records using a *structured symmetric multi-positive InfoNCE* objective with explicit hard negatives. It uses the same text-backbone family and shared text/environment/control-type/slot encoder components and emits a single 256-dimensional transition latent. The monolithic run uses 8 epochs, batch size 16, learning rate 10^{-4}, and the held-out `module_avg` checkpoint-selection criterion defined below.

The resulting monolithic checkpoint serves two roles: it is the monolithic baseline reported in our comparisons and the initialization source for FAC-CUA. FAC-CUA copies the shared encoder components from this checkpoint, introduces separate Goal, Ground, and Delta heads, and then undergoes an additional 8 epochs of factorized training. The shared components remain trainable during this stage. Thus, FAC-CUA uses monolithic pretraining followed by 8 additional epochs of factorized optimization; we do not treat the comparison as training-compute matched.

Because the local architectures and diagnostic feature construction differ, the controlled 286-task result is interpreted as a full local-controller comparison rather than a factorization-only causal ablation. Representation-specific evidence is reported separately through target selection, objective ablations, controlled interventions, and frozen diagnostic probes.

C.3 Training Objectives

The monolithic encoder is optimized with structured symmetric multi-positive InfoNCE with explicit hard negatives. After initialization from the resulting monolithic checkpoint, the factorized model is optimized with

$$\mathcal{L}_{\text{train}} = \mathcal{L}_{\text{comp}} + \lambda_{\text{sel}} \mathcal{L}_{\text{sel}}.$$

For factor f, the compatibility term is the minibatch InfoNCE objective

$$\mathcal{L}_{\text{comp}} = \frac{1}{B} \sum_{f} \sum_{t=1}^{B} -\log \frac{\exp(\text{sim}(z_{f,t}^{\text{ctx}}, z_{f,t}^{\text{out}})/\tau)}{\sum_{j=1}^{B} \exp(\text{sim}(z_{f,t}^{\text{ctx}}, z_{f,j}^{\text{out}})/\tau)}.$$

For a counterfactual r' targeting factor $f^\star$, define $d_f(r, r') = 1 - \cos(z_f^{\text{ctx}}(r), z_f^{\text{ctx}}(r'))$ and use

$$\mathcal{L}_{\text{sel}} = \sum_{f \neq f^\star} \max(0,\, m + d_f(r, r') - d_{f^\star}(r, r')).$$

Subgoal, slot-set, and action perturbations target Goal, Ground, and Delta, respectively. The training pipeline also constructs control-confusable negatives, including mismatched outcomes for similar pre-action states, wrong actions under a similar subgoal, and transitions with similar targets but different post-action effects. These examples enter through the same two core objectives rather than additional standalone losses.

The factorized stage uses AdamW with weight decay 10^{-2} and random seed 42. The checkpoint-selection score `module_avg` is the unweighted mean of four held-out module metrics:

$$\text{module_avg} = \frac{\text{Acc}_{\text{goal}} + \text{Acc}_{\text{slot}} + \text{EM}_{\Delta} + \text{Acc}_{\text{action}}}{4},$$

where the terms denote Goal classification accuracy, slot-reranking accuracy, Delta exact-match accuracy, and action classification accuracy, respectively. For both the monolithic and factorized training runs, the checkpoint with the highest validation `module_avg` is selected as the best checkpoint.

Table 7: Training configuration for the factorized transition-latent encoder.

Item	Value
Training records	1,175
Held-out representation records	444
Text backbone	`sentence-transformers/all-MiniLM-L6-v2`
Initialization checkpoint	Pretrained monolithic transition encoder
Monolithic / pooled latent dimension	256
Factor dimension	128
Factor-adapter hidden dimension	256
Shared encoder frozen?	No
Additional factorized training epochs	8
Text micro-batch size	16
Learning rate	1×10^{-4}
Optimizer	AdamW
Weight decay	1×10^{-2}
Training seed	42
Goal / Ground / Delta InfoNCE temperature	0.07 / 0.07 / 0.07
Compatibility loss weight	1.0
Counterfactual selectivity weight	0.5
Selectivity margin	0.05
Checkpoint selection metric	`module_avg`

D Runtime, Safety, and Resource Accounting

All selector and verifier thresholds were selected on held-out development data and frozen before the 286-task and transfer evaluations.

D.1 Selective Runtime Grounding and Recovery

For target-dependent local actions, the runtime constructs a pre-action candidate set

$$\mathcal{C}_t = \{c_{t,i}\}_{i=1}^{N_t}$$

from the current structured GUI state.

Stage-1 filtering and Stage-2 selection. Stage 1 applies a sigmoid keep/drop head. The reported runtime uses the validation-selected Stage-1 threshold stored in the checkpoint, a top-5 retention floor, and a maximum retained-set size of 30. Latent-safety union is disabled in the reported selector configuration.

Stage 2 jointly reranks the retained set $\mathcal{K}_t$ with factorized interaction features and compact pre-action metadata. If $s_{t,i}$ is the raw Stage-2 score, the runtime normalizes scores over the retained set as

$$p_{t,i} = \frac{\exp(s_{t,i})}{\sum_{j \in \mathcal{K}_t} \exp(s_{t,j})}.$$

Let

$$m_t = s_{t,(1)} - s_{t,(2)}$$

be the raw-score margin between the two highest-ranked candidates. In the reported configuration, the selector executes locally only when

$$p_{t,(1)} \geq 0.60 \qquad \text{and} \qquad m_t \geq 0.75.$$

Otherwise, the step is routed to the LocalPlanner fallback. These quantities are local routing signals rather than calibrated task-success probabilities.

Verifier score computation. Let

$$\phi(x, y) = [x,\ y,\ x \odot y,\ |x - y|,\ \cos(x, y)]$$

Table 8: Action-level safety policy for verifier-guided recovery.

Category	Runtime policy
Allowed bounded recovery	Wait and re-observe; or a retained-alternative click when the alternative remains visible and enabled and the previous transition does not indicate an irreversible state change.
Blocked autonomous recovery	Submit, send, publish, delete, overwrite, purchase, payment approval, credential entry, CAPTCHA, account changes, private-document operations, and other irreversible or user-sensitive actions. These are routed to confirmation or user intervention.
Executor validation	Check candidate validity, visibility, enabled state, geometry, supported action type, retry budget, duplicate text input, and repeated unsafe actions before execution.

for ℓ_2-normalized vectors. The Goal compatibility score is

$$s_t^{\text{goal}} = \sigma\big(h_{\text{goal}}\big(\phi(z_{t,\text{goal}}^{\text{ctx}}, z_{t,\text{goal}}^{\text{out}})\big)\big)\,,$$

where h_{goal} is the learned Goal MLP head.

For selected-target grounding, the verifier forms a grounding query from the context-side Goal and Ground factors and compares it with the selected candidate-slot embedding u_t:

$$q_t = \text{norm}\big(h_q([z_{t,\text{goal}}^{\text{ctx}}; z_{t,\text{ground}}^{\text{ctx}}])\big)\,, \qquad s_t^{\text{ground}} = \sigma(h_{\text{ground}}(\phi(q_t, u_t)))\,.$$

This score uses the selected pre-action target and does not require a post-action Ground-outcome scoring head.

For Delta consistency, the head first predicts the expected outcome-side Delta representation

$$\hat{z}_t^{\Delta} = \text{norm}\big(P_{\Delta}(z_{t,\Delta}^{\text{ctx}})\big)\,,$$

and then computes

$$s_t^{\Delta} = \sigma\big(h_{\Delta}([z_{t,\Delta}^{\text{ctx}}, \hat{z}_t^{\Delta}, z_{t,\Delta}^{\text{out}}, \hat{z}_t^{\Delta} \odot z_{t,\Delta}^{\text{out}}, |\hat{z}_t^{\Delta} - z_{t,\Delta}^{\text{out}}|, \cos(\hat{z}_t^{\Delta}, z_{t,\Delta}^{\text{out}}), \cos(z_{t,\Delta}^{\text{ctx}}, z_{t,\Delta}^{\text{out}})])\big)\,.$$

Cosine similarity is therefore an input feature to the learned heads rather than the final verifier score.

Recovery routing. The reported verifier configuration uses fixed score thresholds

$$\tau_{\text{goal}} = 0.33, \qquad \tau_{\text{ground}} = 0.0772, \qquad \tau_{\Delta} = 0.125.$$

The router computes factor margins $s_t^f - \tau_f$ and diagnoses the weakest factor. If all required margins are non-negative, execution continues. If Ground is weakest, the controller may try the next retained candidate, with at most one target retry. If Delta is weakest, the controller may wait and re-observe when the observed interface change is small or the Delta margin is only mildly negative. A weak Goal score, a strongly inconsistent Delta score, selector abstention, or exhaustion of the overall retry budget routes to fallback/replanning. The reported policy allows at most two local retries in total, disables soft-continue, and applies the safety policy in Appendix D.2 before any autonomous recovery action is executed.

This recovery mechanism explains the difference between the two FAC-CUA variants in the main experiment. Without self-recovery, unresolved states are sent directly to Qwen. Full FAC-CUA instead first attempts bounded local recovery. Across the 286-task evaluation, enabling recovery increases completed local-control steps from 2,778 to 2,864 while reducing total LLM-mediated calls from 1,516 to 601.

D.2 Action-Level Safety Policy

Verifier-guided self-recovery is intentionally restricted to local corrections with low side effect. A low verifier score alone is not sufficient to authorize another GUI operation.

The recovery policy is therefore a bounded local correction mechanism rather than an open-ended autonomous repair procedure.

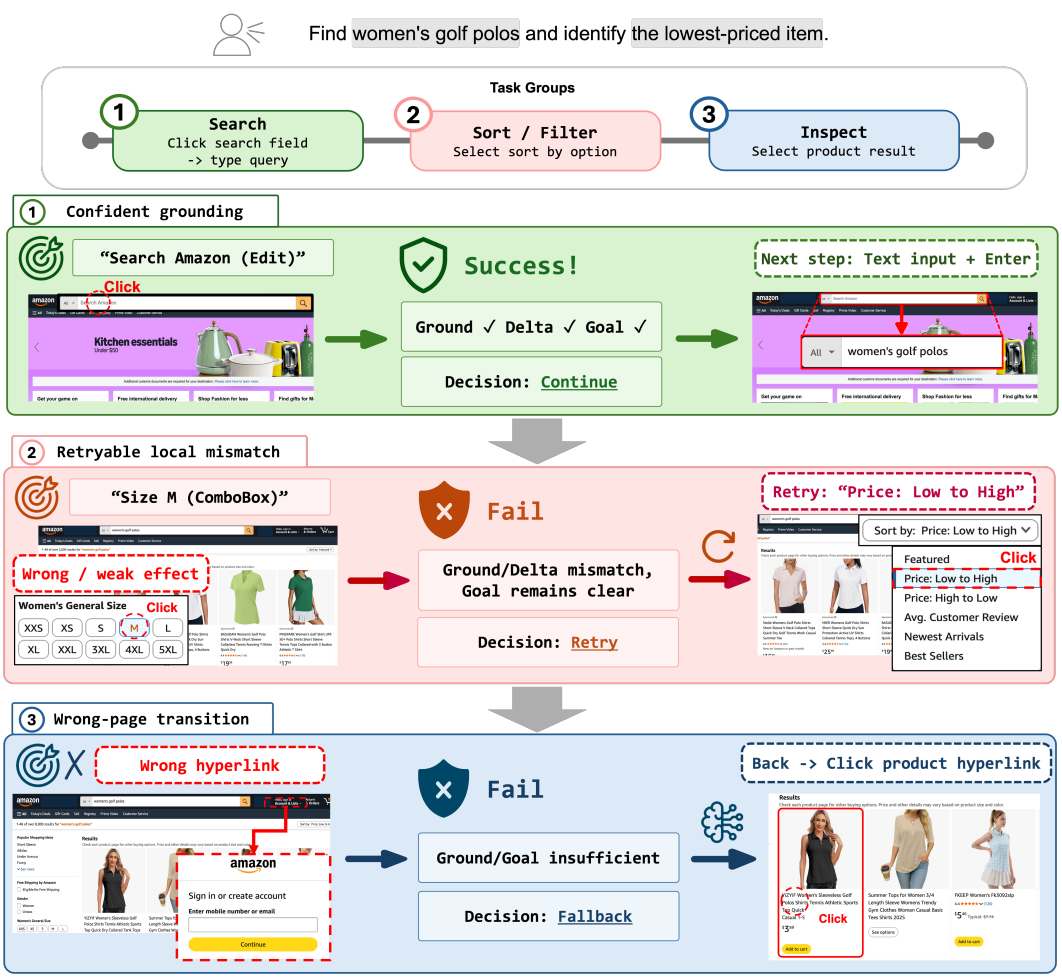

Figure 6: Selective runtime grounding and recovery. A planner step provides a semantic action prior rather than an executable target identifier. For target-dependent steps, the runtime constructs a pre-action candidate set and invokes the element selector. The selector executes locally only when the top candidate score and top1–top2 margin pass the routing thresholds; otherwise, it abstains and falls back. After execution, the verifier provides a local transition-consistency signal used to continue, wait/reobserve, retry a retained alternative, or request an LLM-backed fallback.

D.3 Resource Measurement and Corrected Memory Accounting

A local-control step begins when the controller receives a semantic action and the current pre-action GUI state. It ends when the controller executes a local GUI operation, abstains, completes bounded recovery, or completes an LLM-mediated fallback call.

High-level planner-generation time is excluded from the primary controlled comparison because planner outputs are frozen and shared across all local controller variants.

Latency. T_t^{step} is wall-clock time within the local-control boundary. For learned-controller steps, it includes state encoding, candidate scoring, target selection, post-action verifier computation when applicable, and recovery routing. For fallback-routed steps, it additionally includes the local LLM call and response parsing.

Table 9: Reported compute and profiling environment. We list only values retained in the run metadata and do not reconstruct missing software-version strings or precision settings retrospectively.

Item	Reported setting
Representation/verifier training GPU	NVIDIA RTX A6000 (48 GB)
Training CPU	Intel Core i7-10700 @ 2.90 GHz
Training host RAM	64 GB
Controlled evaluation / local profiling GPU	NVIDIA RTX A6000 (48 GB)
FLOP profiler	PyTorch profiler
Local model serving	vLLM

Token I/O. N_t^{tok} counts prompt-plus-completion tokens consumed by generative LLM calls inside the local-control loop. Tokenization performed inside the non-generative encoder, selector, or verifier is not counted as LLM token I/O and is instead reflected in latency, memory, and compute.

Model-inclusive peak memory. Both host-CPU RSS and GPU VRAM are recorded for locally profiled processes. For each local-control step, we operationalize the single memory constraint as

$$M_t^{\text{peak}} = \max\left(M_{t,\text{CPU}}^{\text{RSS}}, M_{t,\text{GPU}}^{\text{VRAM}}\right).$$

CPU RSS and GPU VRAM are not summed because they correspond to distinct physical memory pools.

For rollout-level WBS, we use

$$\max_t M_t^{\text{peak}}.$$

The model-inclusive boundary covers the controller, local inference-serving process, GPU-resident model weights, KV cache, and serving-time runtime allocations. Memory already resident before task execution is included rather than counting only incremental task-time allocation.

For FAC-CUA, Qwen/vLLM is instantiated only for fallback-routed execution, and its observed serving footprint is included in the corresponding runtime trace. For systems that maintain a local model service continuously, the resident model footprint is included throughout execution.

Compute. C_t^{TFLOPs} is a TFLOPs proxy obtained using the PyTorch profiler's FLOP estimation over operations executed within each profiled local-control step. For model-backed calls, the profile reflects the observed prompt-plus-completion workload. For learned-controller steps, it covers the encoder, selector, verifier, and routing computations inside the same local-control boundary. The same PyTorch-profiler configuration and supported-operation accounting are used for all locally instrumented methods. We use this quantity as a consistent within-study compute proxy rather than as a hardware-level measurement of realized floating-point throughput.

Unless otherwise stated, WBS@Ref uses

$$B_T = 10\text{s}, \qquad B_N = 8\text{k tokens/step}, \qquad B_M = 8\text{GB}, \qquad B_C = 100\ \text{TFLOPs-proxy/step}.$$

D.4 Metric Aggregation and Confidence Intervals

Task success is a binary indicator of whether the rollout completes the requested task. Task progress measures the fraction of planner/reference local-control steps reached or completed before termination.

A task contributes to WBS only when it succeeds and every local-control step satisfies all four resource constraints:

$$\text{WBS}(\tau; B) = \mathbf{1}\left[\text{Succ}(\tau) = 1 \wedge \forall t,\ T_t^{\text{step}} \le B_T,\ N_t^{\text{tok}} \le B_N,\ M_t^{\text{peak}} \le B_M,\ C_t^{\text{TFLOPs}} \le B_C\right].$$

For task-level uncertainty estimates, we bootstrap the 286 evaluation task IDs with replacement. Paired comparisons use the same resampled task IDs for all compared methods. Confidence intervals are 95% percentile bootstrap intervals.

Table 10: Task-level 95% bootstrap confidence intervals for the controlled planner–controller comparison.

Method	Success (%)	WBS@Ref (%)	Task Prog. (%)	LLM Calls / Task
Every-step Qwen controller + verifier	29.4 [24.4, 34.9]	1.7 [0.7, 4.0]	47.7 [41.9, 53.5]	9.67 [9.31, 10.03]
Monolithic latent + fallback	47.6 [41.8, 53.3]	16.4 [12.6, 21.2]	75.6 [70.6, 80.6]	2.50 [2.32, 2.68]
FAC-CUA w/o self-recovery	73.1 [67.7, 77.9]	23.8 [19.2, 29.0]	89.3 [85.7, 92.9]	5.30 [5.03, 5.57]
FAC-CUA	79.0 [73.9, 83.3]	52.8 [47.0, 58.5]	92.7 [89.7, 95.7]	2.10 [1.93, 2.27]

Table 11: Paired 95% bootstrap confidence intervals for WBS@Ref improvements, in percentage points.

Comparison	WBS@Ref difference
FAC-CUA – Every-step Qwen	51.1 [45.3, 56.8]
FAC-CUA – Monolithic latent + fallback	36.4 [31.0, 42.1]
FAC-CUA – FAC-CUA w/o self-recovery	29.0 [24.1, 34.5]

E ADDITIONAL EXPERIMENTS AND DIAGNOSTICS

E.1 REPRESENTATION AND OBJECTIVE ABLATIONS

We evaluate the contribution of the two representation-learning objectives using held-out structured transitions.

Removing $\mathcal{L}_{\text{comp}}$ primarily degrades context–outcome retrieval, whereas removing $\mathcal{L}_{\text{sel}}$ reduces factor selectivity and target selection accuracy. The two objectives therefore support complementary properties of the learned representation.

E.2 FROZEN FAILURE-PROBE DIAGNOSTIC

The failure-source probe is a representation diagnostic rather than a runtime component. Diagnostic failures are synthetically constructed factor-specific counterfactuals from structured transition records; they are not annotations of naturally occurring rollout-failure frequencies. The factorized and monolithic representations remain frozen throughout probe training, and no probe output is used to tune selector thresholds, verifier thresholds, fallback policy, or WBS.

Data and splits. The diagnostic contains four classes: clean, Ground failure, Goal failure, and Delta failure. The probe-training set contains 188 examples (47 per class), the validation set contains 68 examples (17 per class), and the held-out evaluation set contains 308 examples (77 per class). The construction uses the existing representation-learning train/evaluation transition splits. A validation split is created from the eligible probe-training records with a configured validation ratio of 0.2 and split seed 2026; after eligibility filtering this yields the counts above.

Pair features and probe inputs. For two 128-dimensional vectors a and b, the diagnostic pair feature is

$$\psi(a, b) = [\hat{a},\, \hat{b},\, |\hat{a} - \hat{b}|,\, \hat{a} \odot \hat{b},\, \hat{a}^\top \hat{b}],$$

where $\hat{a}$ and $\hat{b}$ are ℓ_2-normalized. The resulting head input is 513-dimensional.

For FAC-CUA, the Ground head uses $\psi(z^{\text{ctx}}_{\text{ground}}, u_{\text{fac}})$, where u_{fac} is the frozen factorized embedding of the selected slot. The Goal and Delta heads use $\psi(z^{\text{ctx}}_{\text{goal}}, z^{\text{out}}_{\text{goal}})$ and $\psi(z^{\text{ctx}}_{\Delta}, z^{\text{out}}_{\Delta})$, respectively.

For the monolithic condition, the frozen context and outcome latents are 256-dimensional and are mapped by independent trainable role-specific adapters to 128 dimensions before pair construction. For Ground, the same selected candidate identity is represented from its raw candidate features: a 384-dimensional text embedding concatenated with a 64-dimensional control-type embedding, followed by a trainable $448 \to 128$ candidate adapter. Goal and Delta analogously use independent $256 \to 128$ context/outcome adapters. Thus both conditions feed 513-dimensional pair features to the

Table 12: Representation and objective ablations. Context–outcome retrieval measures matching between context and outcome views, while selectivity JSD measures factor-specific response to control-field perturbations.

Model	Ctx–Out Recall@1 ↑	Ctx–Out MRR ↑	Selectivity JSD ↑	Target Top-1 ↑
Monolithic latent	40.9	51.8	–	54.1
FAC-CUA w/o $\mathcal{L}_{\text{comp}}$	41.8	55.6	0.132	76.4
FAC-CUA w/o $\mathcal{L}_{\text{sel}}$	58.7	71.9	0.071	81.2
FAC-CUA full	**64.9**	**78.4**	**0.179**	**88.8**

Table 13: Transition-consistency diagnostics. Balanced accuracy is evaluated at the best validation threshold and is not the fixed runtime routing threshold.

Head	AUROC ↑	Pair Win ↑	Best Bal. Acc. ↑	Score Margin ↑
Goal	0.686	0.686	0.655	0.151
Ground	0.784	0.784	0.743	0.269
Delta	0.726	0.726	0.678	0.052

same binary-head architecture, but they do not use the same underlying candidate representation and the overall representations are not dimension- or parameter-matched.

Probe optimization and decision rule. Probe training uses three seeds $\{42, 43, 44\}$, learning rate 10^{-3}, weight decay 10^{-4}, training batch size 128, evaluation batch size 256, hidden and adapter dimensions of 128, dropout 0.1, gradient clipping at 5.0, a maximum of 150 epochs, and early-stopping patience of 20. For each seed, three binary heads produce $p_{\text{ground}}, p_{\text{goal}}, p_{\Delta}$. For 4-way classification,

$$\hat{y} = \begin{cases} \text{clean}, & \max_f p_f < \tau, \\ \arg\max_{f \in \{\text{ground},\text{goal},\Delta\}} p_f, & \text{otherwise}, \end{cases}$$

where τ is selected separately for each seed by searching 0.05 to 0.95 in increments of 0.005 and maximizing validation 4-way Macro-F1. When multiple heads exceed the threshold, the highest probability determines the class; exact ties follow implementation order Ground, Goal, Delta. For the 3-way failure-only metric, clean examples are removed and the prediction is the direct argmax over the three failure-head probabilities, without the clean threshold.

Reported probe results are mean ± standard deviation over the three probe-training seeds. Because the failure examples are synthetic factor-specific perturbations related to the intended factor roles, and because the representation-specific feature construction differs across the two conditions, this experiment is interpreted as a controlled diagnostic of failure-source accessibility rather than a capacity-controlled causal ablation or an estimate of natural-failure performance.

E.3 VERIFIER-HEAD DIAGNOSTICS

We evaluate whether each verifier head can rank compatible transitions above negative or counterfactual mismatches. These diagnostics evaluate local transition consistency rather than task completion.

E.4 CORRECTED WBS CONSTRAINT DECOMPOSITION

Table 14 decomposes the reference-budget result into constraint-specific rollout violations over the same 286-task controlled evaluation. Categories are non-exclusive, so a rollout may violate multiple constraints.

The corrected memory accounting increases the measured memory-violation rate relative to the original derived field. However, every newly identified memory violation already co-occurs with at least one latency, token, or compute violation. Consequently, the rollout-level Any-violation sets and WBS@Ref values remain unchanged.

Table 14: Corrected constraint-specific rollout violation rates under WBS@Ref. Memory uses the model-inclusive CPU/GPU accounting described in Appendix D.3.

Method	Latency	Token	Memory	Compute	Any
Every-step Qwen controller + verifier	48.6% (139/286)	50.0% (143/286)	96.9% (277/286)	49.0% (140/286)	98.3% (281/286)
Monolithic latent + fallback	42.0% (120/286)	42.7% (122/286)	73.8% (211/286)	42.7% (122/286)	83.6% (239/286)
FAC-CUA w/o self-recovery	24.5% (70/286)	52.8% (151/286)	66.8% (191/286)	58.0% (166/286)	71.0% (203/286)
FAC-CUA	**10.8% (31/286)**	**36.4% (104/286)**	**35.3% (101/286)**	**35.7% (102/286)**	**46.5% (133/286)**

Table 15: Budget sensitivity for the controlled local-controller comparison. WBS count is the number of rollouts that both complete the task and satisfy all per-step resource constraints. Budgets are reported as latency / token I/O / memory / TFLOPs-proxy per local-control step.

Method	Strict 5s / 2k / 4GB / 50	Reference 10s / 8k / 8GB / 100	Relaxed 20s / 32k / 16GB / 350
Every-step Qwen controller + verifier	0/286 (0.0%)	5/286 (1.7%)	16/286 (5.6%)
Monolithic latent + fallback	5/286 (1.7%)	47/286 (16.4%)	66/286 (23.1%)
FAC-CUA w/o self-recovery	16/286 (5.6%)	68/286 (23.8%)	123/286 (43.0%)
FAC-CUA	**61/286 (21.3%)**	**151/286 (52.8%)**	**175/286 (61.2%)**

E.5 BUDGET SENSITIVITY

Table 15 reports within-budget success under strict, reference, and relaxed resource budgets. All values use the corrected model-inclusive memory accounting described in Appendix D.3.

As expected, WBS increases as the resource budget is relaxed. FAC-CUA achieves the highest WBS under all three operating points, increasing from 21.3% under the strict budget to 52.8% under the reference budget and 61.2% under the relaxed budget. The corresponding values for the monolithic controller are 1.7%, 16.4%, and 23.1%, respectively.

These results show that the relative advantage of FAC-CUA is not specific to a single WBS threshold. At the same time, WBS remains below raw task success even under the relaxed setting, indicating that some successful rollouts still contain at least one resource-violating local-control step.

E.6 COMPARISON WITH AN END-TO-END UI-TARS BASELINE

Audit-set construction. The UI-TARS diagnostic set was constructed by manually screening the full 286-task evaluation pool for compatibility with the capabilities of the locally evaluated UI-TARS-1.5-7B configuration (Qin et al., 2025). A task was considered executable when (i) UI-TARS could access the application or environment required by the task and (ii) the action types required to complete the task were supported by the UI-TARS executor. Exactly 30 tasks satisfied these criteria, and we include all 30 in the audit.

The executability criteria were applied before inspecting the reported FAC-CUA or UI-TARS task outcomes. No task was added, removed, or replaced based on either method's success or resource measurements, and the same 30 tasks are used for task success and all reported resource measurements. Because the subset is conditioned on compatibility with the evaluated UI-TARS configuration, we treat this comparison as a task-matched diagnostic rather than a population-level estimate over the full 286-task suite.

Resource-accounting boundary. This audit compares task-level success and locally instrumented execution costs rather than complete end-to-end system cost. FAC-CUA's upstream planner-generation cost lies outside the profiled local-control boundary, whereas UI-TARS performs planning and action generation within its model-backed agent loop. The two architectures therefore do not expose an identical end-to-end resource boundary, and the resource results below should be interpreted as a diagnostic rather than a full-system efficiency comparison.

Exact audit commands. For transparency, the 30 fixed commands are listed below.

1. open folder 'LLM-UI'

Table 16: Corrected 30-task task-level comparison. Token, memory, and compute columns are means of rollout-level maxima; latency is mean total rollout latency.

Method	Success	Mean Total Latency	Mean Rollout-max Token I/O	Mean Rollout-max Memory	Mean Rollout-max TFLOPs
UI-TARS-1.5-7B	16/30 (53.3%)	28.78 s	13.68k	45.21 GB	230.36
FAC-CUA	30/30 (100.0%)	37.02 s	2.16k	14.16 GB	32.44

Table 17: Constraint-specific violations for the same 30-task comparison. Latency violations use maximum step latency, not total rollout latency.

Method	Latency	Token	Memory	Compute	Any	WBS@Ref
UI-TARS-1.5-7B	1/30	29/30	30/30	29/30	30/30	0/30
FAC-CUA	0/30	0/30	9/30	0/30	9/30	**21/30**

2. copy file ‘test.txt’
3. open ‘apple.jpg’
4. copy ‘IBM.png’
5. get the path of ‘LLM-UI’
6. create a new folder named ‘log’ in folder ‘LLM-UI’
7. change name of folder ‘LLM-UI’ to ‘UI-LLM’
8. Image search for ‘dogs’ on Chrome
9. search up about ‘LLM’
10. ask ‘what is LLM?’ to chat-GPT
11. location search for ‘Niagara Falls’ on Chrome
12. execute chat-gpt on Chrome
13. serach for papers about LLM on Google Scholar
14. product search for ‘basil pesto’, price over 10000 on Chrome
15. search the meaning of word ‘synta0’
16. Product search ‘earlgray teabag’ on Chrome
17. translate the chosen page on Chrome
18. location search for ‘Seoul Station’ on Chrome and show a way to go there
19. write ‘LLM-UI’ on notepad
20. Search Baidu Baike for an introduction to RNN (Recurrent Neural Network)
21. Open Event Viewer
22. Find and read the book “Notre-Dame de Paris” on Douban Read
23. Learn how to make braised intestines in brown sauce
24. Bookmark the current webpage
25. Find and add the cheapest chocolate to the cart on Amazon
26. Search for symptoms of pneumonia on Baidu
27. Open Device Manager
28. Align document content to the left
29. Type “Hello, world!” and bold it
30. Search for children’s clothes on Amazon and select the appropriate size and color to add to the cart

Within this local execution-cost audit, UI-TARS keeps its model-backed service resident throughout execution. FAC-CUA instantiates Qwen/vLLM only on fallback-routed execution. Across these 30 rollouts, FAC-CUA invokes Qwen 14 times across 9 rollouts. Because WBS operates on rollout-level maxima, those 9 rollouts account for the 9 memory violations.

Table 18: Constraint violations over 87 normalized logical operations. Categories are non-exclusive. FAC-CUA's memory value reflects the corrected model-inclusive audit.

Method	Latency	Token	Memory	Compute	Any
Agent-S	81.6% (71/87)	92.0% (80/87)	21.8% (19/87)	88.5% (77/87)	92.0% (80/87)
OpenCUA	2.3% (2/87)	50.6% (44/87)	34.5% (30/87)	63.2% (55/87)	77.0% (67/87)
FAC-CUA	0.0% (0/87)	4.6% (4/87)	4.6% (4/87)	6.9% (6/87)	6.9% (6/87)

Table 19: Baseline-success-conditioned rollout audit on the fixed 11-task set.

Method	Task Success	Any Violating Rollout	WBS@Ref
Agent-S	7/11 (63.6%)	11/11 (100.0%)	0/11 (0.0%)
OpenCUA	7/11 (63.6%)	9/11 (81.8%)	2/11 (18.2%)
FAC-CUA	11/11 (100.0%)	1/11 (9.1%)	**10/11 (90.9%)**

E.7 AGENT-S AND OPENCUA COMMON-BACKEND AUDIT

Official/API execution of Agent-S and OpenCUA does not expose a directly comparable server-side memory and compute boundary. We therefore perform an additional local diagnostic using a shared locally hosted Qwen backend for model-backed components. Agent-S retains its UI-TARS grounding component, while FAC-CUA invokes Qwen only for fallback-routed operations.

We normalize complete traces from the fixed diagnostic set into 87 logical GUI operations. Adjacent micro-actions that implement one logical operation are merged without discarding resource records: latency, token I/O, and compute are summed, while peak memory is maximized.

We additionally compute rollout-level WBS on the fixed 11-task set for which both external baselines had complete, previously successful trajectories suitable for local instrumentation. Task selection therefore depends on the original baseline outcomes and should be interpreted as a *baseline-success-conditioned audit*, not a full-benchmark estimate.

The 87-operation result is therefore used as a cross-agent resource diagnostic, while the 11-task result is a conditional rollout audit. The 286-task frozen-planner experiment in the main paper remains the primary controlled comparison.

E.8 WINDOWSAGENTARENA DOMAIN BREAKDOWN

Subset construction. The 40-task subset was fixed before evaluating FAC-CUA and Navi (Bonatti et al., 2025). It contains all 30 Web tasks and two tasks from each of the System, Utilities, Coding, Media/Video, and Office domains.

The WindowsAgentArena experiment uses the unchanged FAC-CUA checkpoint, selector/verifier thresholds, recovery rules, and fallback policy.

Because each non-Web domain contains only two tasks, domain-specific rates should not be interpreted as population-level estimates. The overall 40-task result is transfer evidence rather than a full-suite WindowsAgentArena estimate.

E.9 ADDITIONAL UNSEEN-DOMAIN STRESS TEST

As a complementary transfer diagnostic, we evaluate the unchanged FAC-CUA configuration on 30 newly collected tasks from 12 domains absent from the training and original evaluation domain lists. Example domains include Google Arts & Culture, Canva, WPS, Reddit, Notion, Trello, Instacart, and Google Docs.

FAC-CUA completes 19/30 tasks (63.3%). The additional-domain tasks also require more fallback and resource usage than the original evaluation set.

This experiment is an author-constructed stress test rather than a standardized benchmark and is therefore used only as complementary evidence of cross-domain transfer.

Table 20: Domain-level outcomes on the fixed 40-task WindowsAgentArena subset. Navi uses mixed-omni + UIA with GPT-4o because its originally reported GPT-4V preview backend was unavailable; this is an updated-backbone re-evaluation rather than an exact reproduction.

Method	Web	System	Utilities	Coding	Media/Video	Office	Overall
FAC-CUA	11/30 (36.7%)	2/2 (100%)	2/2 (100%)	1/2 (50%)	1/2 (50%)	0/2 (0%)	**17/40 (42.5%)**
Navi (mixed-omni + UIA, GPT-4o)	8/30 (26.7%)	1/2 (50%)	1/2 (50%)	0/2 (0%)	1/2 (50%)	0/2 (0%)	11/40 (27.5%)

Table 21: Comparison between the original evaluation set and the additional author-constructed unseen-domain stress test.

Evaluation	Success	Latency/step	Token I/O/step	TFLOPs/step
Original evaluation	79.0%	4.98 s	3.49k	48.88
Additional domains	63.3%	7.37 s	8.46k	142.49

E.10 Cross-Accessibility-Backend Proxy Evaluation

FAC-CUA consumes a canonical structured-element representation rather than Windows UIA fields directly. To test whether this representation can be constructed from another accessibility backend, we collect 12 tasks and 84 action transitions across six macOS applications or web domains: IMDb, YouTube, GitHub, Hugging Face, Apple Music, and Calendar.

A deterministic adapter maps macOS Accessibility (AX) fields into the same canonical semantic role, label, state, hierarchy, geometry, and actionability schema used by FAC-CUA. No model retraining or threshold calibration is performed.

The adapter retains the annotated target in all 84 transitions. After quality filtering, 68 transitions are eligible for selector evaluation and 72 for verifier evaluation.

This experiment is an offline sparse-context cross-backend proxy rather than an end-to-end macOS rollout. Complete pre/post AX-tree captures are unavailable, so candidate context is reconstructed from the available AX target packet, breadcrumb, siblings, and target subtree. The experiment therefore supports schema compatibility and local selector/verifier transfer, but does not establish full-tree or end-to-end macOS transfer.

E.11 Longer-Horizon Stress Tests

Figure 7 reports the original 286-task controlled comparison stratified by planned local-control horizon. The additional stress-test analysis below extends the horizon analysis with eight composite tasks containing 24–37 planned actions.

We evaluate eight additional composite tasks containing 24–37 frozen-planner semantic actions, with a median of 28.5 actions. FAC-CUA completes 5/8 tasks (62.5%), but none satisfies WBS@Ref. These tasks require a median total rollout time of 170.57 s, a median maximum step latency of 17.715 s, and a median of 13.5 LLM-mediated calls per task.

We further stratify all 294 FAC-CUA rollouts, including the additional composite stress-test tasks, by planned horizon.

Raw success remains nontrivial in the 21–30-action range, but both success and WBS decline as interaction horizon grows. The degradation is associated with accumulated latency, compute, and fallback costs. Because the ≥ 31 group contains only five tasks, it should be interpreted as a descriptive stress test rather than a precise estimate of very-long-horizon performance.

F LLM and Prompt Details

LLMs are used in three bounded roles: GPT-5 for offline annotation, Qwen2.5-VL-7B for high-level planning, and Qwen2.5-VL-7B for unresolved local-control fallback.

Table 22: Frozen selector and verifier performance under the macOS AX structured-context adapter.

Metric	Result
Selector Top-1	71.0%
Selector Top-5	80.9%
Verifier AUROC	0.72
Verifier AUPRC	0.77
Verifier balanced accuracy	0.81

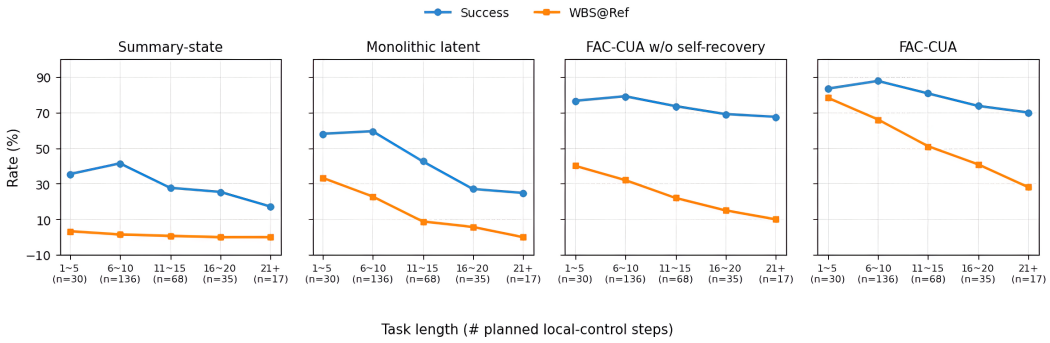

Figure 7: Success and WBS@Ref across planned task-length buckets in the original 286-task controlled comparison. The success–WBS gap widens with task length, while FAC-CUA retains the highest WBS@Ref across the displayed buckets.

Offline annotation prompts are constrained to describe only controls supported by the supplied GUI evidence. The high-level planner may generate semantic action descriptions but does not provide executable target identifiers.

Fallback prompts operate only within the current local-control boundary and are instructed not to invent hidden candidates or re-plan the complete task. For credentials, payment, CAPTCHA, account changes, private-document operations, and other user-sensitive cases, the fallback policy routes to confirmation or user intervention rather than autonomous recovery.

G Reproducibility and Released Artifacts

The supplementary release provides code, configuration files, prompts, learned checkpoints, and representative processed examples for inspecting and reproducing the released components of FAC-CUA.

Representation-learning and runtime code. The release contains structured transition-record construction code, monolithic transition-encoder training code, factorized goal/ground/delta encoder training code, selector/verifier components, runtime code, configuration files, learned checkpoints, and representative processed examples. These artifacts support inspection and module-level reproduction of the representation and local-control pipeline.

Prompts and configuration. The supplementary `prompts.json` contains prompt templates, input variables, expected output schemas, decoding settings, and safety constraints. Configuration files summarize model settings, routing thresholds, recovery rules, and resource budgets.

Privacy and release scope. We do not release unredacted raw GUI traces, screenshots, complete private UIA trees, credentials, private document contents, or user-specific task data. Full live replay of all 286 tasks would require private desktop states, application-specific interface states, and executor state that cannot be released safely. The provided artifacts therefore support inspection and module-

Table 23: FAC-CUA performance as a function of the number of planner-proposed local actions. The ≥ 31 group contains only five tasks and is descriptive.

Planned Actions	Tasks	Success	WBS	Median Wall-clock	Median Max-step Latency	Median LLM Calls
≤ 10	166	84.3%	66.9%	34.09 s	4.38 s	2
11–20	103	72.8%	33.0%	86.88 s	6.26 s	2
21–30	20	70.0%	25.0%	170.57 s	9.11 s	3
≥ 31	5	40.0%	20.0%	224.80 s	18.13 s	13

Table 24: Summary of LLM roles and model backends. Full system/user templates, input variables, output schemas, and decoding settings are included in the supplementary `prompts.json`.

Role	Model	Input	Output
Offline annotation	`GPT-5`	Transition segments, primitive actions, and compact structured GUI summaries	Structured local subgoal and UI-grounded state annotations
High-level planning	`Qwen2.5-VL-7B`	User instruction and execution context	Semantic task units and planner-level local actions
Local fallback	`Qwen2.5-VL-7B`	Current semantic action, local subgoal, pre-action structured GUI state, indexed candidates, and already observed context	One bounded local-control decision or abstention/user handoff

level reproduction of the released components rather than exact public replay of every live GUI rollout.

Third-party resources. Public benchmarks, models, and agent systems are used under their respective licenses and are cited in the main paper.