ONE STEP, FOUR CLOCKS: HOW GUI AGENTS FALL OUT OF STEP

Anonymous authors
Paper under double-blind review

ABSTRACT

GUI agents can choose the right action and still fail because the interface changes while they think. We model the timing of agent responses through four events: an environment change, observation availability, decision commitment, and action receipt. These events distinguish failures in which relevant evidence is missed, a decision uses outdated evidence, an action becomes invalid after commitment, and an action arrives after the window of opportunity closes. We introduce **ChronotopeGym** to make these timing conditions experimentally controllable. We evaluate agent behaviors in the same task under three conditions: the interface waits; observations change while the response remains valid; or the interface continues changing while the agent thinks. Agents that succeed when the response remains valid can fail when it expires before their action arrives. But response speed alone does not explain the failures. On a moving-target task, we keep the model call and the action's commitment and arrival times unchanged, changing only which observation supplies the click coordinate. Using the newer observation raises success from 0/100 to 100/100: the original click targets where the object was, while the revised click targets where it has moved. Guided by the framework, **Chronotope** allows new evidence to revise a decision before commitment, preserving useful reasoning while updating action variables that depend on the changing interface.

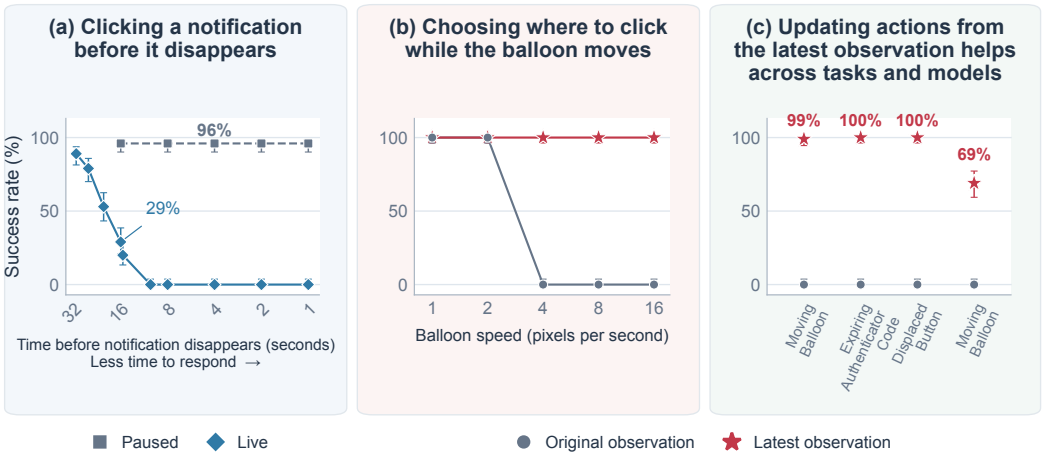


Figure 1: When a screen keeps changing, agents must act in time and update their actions to match it. **(a)** In Disappearing Notification, an Agent S3 model must click a notification before it disappears. We shorten this time limit and compare success when the interface waits while the agent thinks (Paused) against when it keeps running (Live). Success stays at 96% in Paused but falls in Live as notifications disappear sooner, showing how pausing can hide timing failures. **(b)** In Moving Balloon, a Qwen3.8-27B agent must click a moving balloon. We increase its speed and compare clicks aimed at its original versus latest observed positions. The model call takes the same time in both cases. At higher speeds (4, 8, and 16 pixels/s), updating the position raises success from 0% to 100%. **(c)** We test the same update across three tasks. With Qwen3.8-27B, success rises from 0% to 99–100%, showing that the update helps with both changing positions and changing text. The update method generalizes to another model, UI-TARS-1.5-7B, whose success increased from 0% to 69%. Each point represents 100 trials, and error bars 95% confidence intervals. See §5 for experiment details.

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.

1 INTRODUCTION

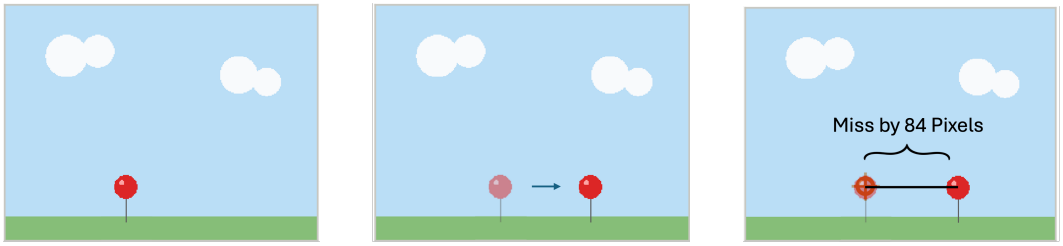
Graphical user interface (GUI) agents are typically evaluated as if the interface waits for them. The agent observes a screen, thinks, produces an action, and is judged on whether that action was correct (Deng et al., 2023; Zhou et al., 2024; Zheng et al., 2024a; Rawles et al., 2025; Xie et al., 2024; Qin et al., 2025; Li & Shi, 2026). Real interfaces continue evolving independently: a page finishes loading, a notification disappears, a target moves away from where it was observed, or an earlier click keeps the player character advancing while the agent reacts to enemy fire. A click can thus be right for the frame the agent saw and wrong for the state in which it lands. Figure 2 illustrates this with a moving balloon: the agent chooses the correct click position for its initial image, but the click misses because the balloon has moved by the time it arrives. This reveals a gap between solving a GUI task when the interface waits and solving it while the interface continues to evolve.

A natural explanation is that the agent is too slow. Sometimes it is: the action reaches the application after the deadline. But model latency alone does not tell us what went wrong, because a response does not fail at a single moment. It unfolds through a lifecycle in which the interface changes, information becomes available to the controller, a decision becomes fixed, and the resulting action reaches the application. Correctness can be lost at different points along this lifecycle, even when the model takes the same amount of time. Figure 3 shows that the the same model latency can cause different failures. Diagnosing a timing failure therefore requires knowing not only how long computation took, but where the response correctness is lost.

We describe this lifecycle using four events. E marks when a task-relevant environment change occurs; O, when an observation becomes available to the controller; D, when the action can no longer be changed; and A, when the application receives the action. Figure 3 illustrates four timing failure modes:

- **Missed**. Relevant evidence disappears before it is observed.
- **Stale**. The agent acts based on an outdated observation when newer evidence is available.
- **Invalidated**. A correct action at D becomes wrong before the application receives it at A.
- **Late**. A response reaches A after its opportunity has closed.

A single latency number tells how long a response took; it cannot tell us where correctness was lost.



(a) The agent sees the balloon at its original position

(b) The balloon moves while the agent reasons

(c) The agent's click misses the balloon by 84 pixels

Figure 2: Correct for the observed screen, wrong on arrival. In Moving Balloon, the agent must click a moving balloon. The agent chooses the correct position for its initial observation, but the interface continues evolving while it computes, so the same position is wrong when the click arrives. The faded balloon marks the initially observed position; the crosshair marks where the click lands.

The four failures require different remedies. A missed event requires observation to continue while the agent deliberates. But continued observation is not enough to prevent a stale decision if new evidence can affect only the next decision. Relevant evidence arriving before D must therefore be able to change the decision already underway. Once an action is committed, an unexpected change after D cannot be repaired by revising that decision, and a late response still has to reach A sooner. We investigate these failures and their remedies in three steps:

- **Control the temporal structure of GUI interaction**. ChronotopeGym evaluates tasks under three conditions: Paused, Replay, and Live. Paused freezes task progression during deliberation. Replay and Live deliver the same observations on the same schedule, but only Live allows the conditions for a successful response to change during deliberation.

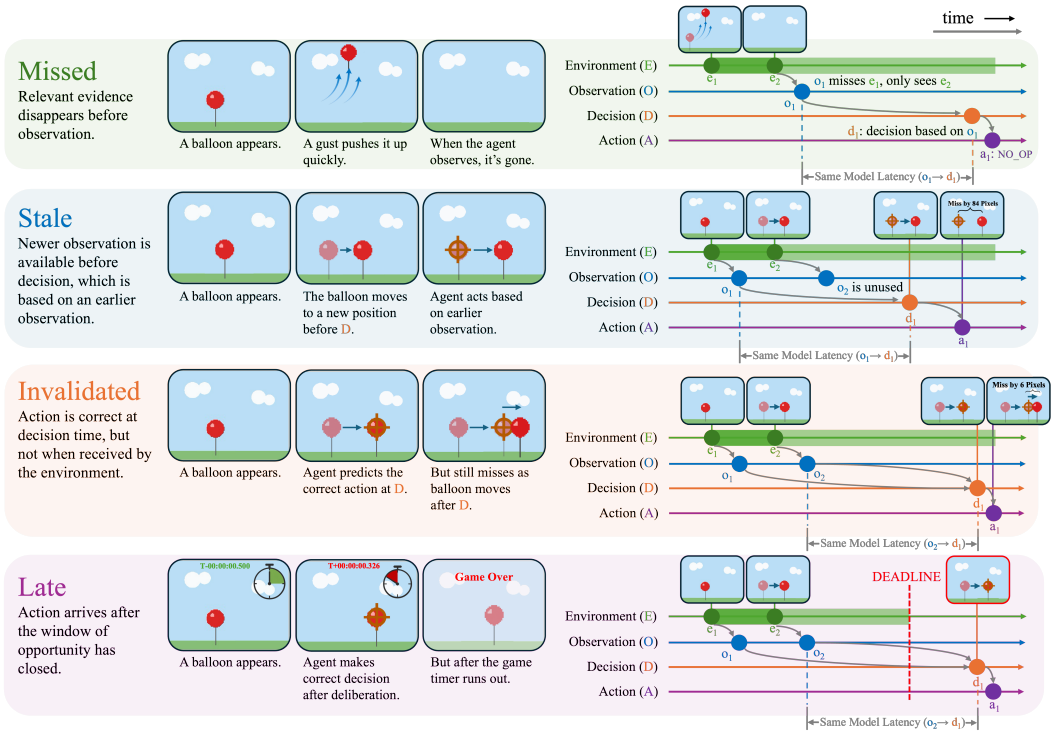


Figure 3: The same model latency conceals four timing failures. Four Moving Balloon examples share the same observation-to-decision interval but fail for different reasons. **Missed.** Relevant evidence disappears before the agent observes it. **Stale.** Newer evidence is available before commitment. The agent still uses an earlier observation. **Invalidated.** The action is correct when committed. It becomes wrong before the application receives it. **Late.** The action reaches the application after the response deadline. The timelines mark environment changes (E), observation availability (O), decision commitment (D), and application receipt (A). They show which observation supports each decision. ***Model latency alone does not decide timing failure.***

- **Isolate the effect of newer evidence**. We change the observation supporting a pending decision while holding decision start, commitment, and application receipt fixed. This separates the benefit of newer evidence from that of a faster response.
- **Check pending decisions before commitment**. Chronotope continues observing during deliberation, maintains the current control state and pending decision, and checks that decision against the newest relevant evidence before commitment.

Our diagnosis and control approach are both based on the EODA framework. Specifically, **ChronotopeGym** is a test bed for diagnosing and evaluating timing failures in GUI agents. It changes when evidence becomes available, when a decision becomes fixed, and when an action reaches the application, allowing us to test where correctness is lost. We release ChronotopeGym and its evaluation code. **Our approach, Chronotope,** uses the same EODA framework to keep an unfinished decision current as the interface changes, preserving what remains correct and updating information that has become outdated before commitment. **Our experimental results show that these timing distinctions matter.** A planner-grounder agent drops from ***96% success in Replay to 29% in Live*** when the same evidence arrives at the same times but the response opportunity continues advancing. With decision start, commitment, and receipt held fixed, updating only the observation used to locate a moving target ***raises success from 0% to 100%***.

2 Related Work

GUI agents solve computer tasks by observing the screen and taking actions (Liu et al., 2026; Yao et al., 2026), but an interface can change before an action takes effect. We therefore review how existing benchmarks evaluate this timing problem and how latency control methods address it.

Time in GUI-agent benchmarks. Recent works study timing in GUI interaction with benchmark-specific tasks and metrics (Liu et al., 2026; Yao et al., 2026; Li & Shi, 2026; Dong et al., 2026). However, a benchmark score alone does not tell us whether the agent failed due to missed changes, outdated information, or late actions. We address this gap with an event-stream view that diagnoses temporal failures across environment change, observation, decision commitment, and action receipt, enabling controlled studies under a common measurement scheme. §A.2 compares eighteen benchmarks on these criteria.

Latency control in GUI agents. Latency can cause GUI agents to act on outdated information, miss action windows, or respond too late to unexpected changes. Existing methods supply newer evidence through AOI (Li & Shi, 2026), predict future states through MobileDreamer and TraceR1 (Cao et al., 2026; Liang et al., 2026), or prepare and reuse computation before execution through UFO2 and AAPT (Zhang et al., 2025b; Dong et al., 2026). However, newer evidence may go unused, and planning ahead alone does not update where the agent clicks or types when the screen changes. Our event-stream view identifies where these failures arise and which interventions address them. §A.3 and Table 13 provide extended discussion and comparisons.

3 Identifying Where Correctness Is Lost

In order to address the limitations of existing GUI benchmarks and latency-control methods, we need a common way to identify where timing causes a GUI response to become incorrect and to control those timing relations experimentally. Existing benchmarks can show that timing matters, but their scores do not distinguish missed changes, outdated information, and late actions. Existing latency-control methods can supply newer evidence, predict future states, or prepare computation earlier, but they do not by themselves identify where correctness is lost during the response. We therefore model environment change, observation availability, decision commitment, and action receipt explicitly, then control their temporal relations in **ChronotopeGym**.

3.1 From Observation-Action to EODA

In order to identify why the same response time can produce different failures, we need to separate the events hidden within observation-to-action latency. Figure 4 shows two **Disappearing Notification** executions with the same $O \rightarrow A = 400$ ms latency. In the first, the decision is committed while the notification is still present, but the notification disappears before the click arrives. In the second, the notification disappears before commitment, but that change has not yet appeared in the latest observation available to the agent. Observation-to-action timing treats both executions alike, even though one is invalidated after commitment and the other misses relevant evidence. We therefore model four events explicitly: environment change E, observation availability O, decision commitment D, and action receipt A. As Figure 4(c) shows, different $O \rightarrow D$ and $D \rightarrow A$ intervals can produce different outcomes even when $O \rightarrow A$ is unchanged. Therefore, we use EODA to distinguish where correctness is lost within the same end-to-end response time.

3.2 Controlling Timing with Paused, Replay, and Live

In order to control timing within EODA, we introduce three execution conditions, **Paused**, **Replay**, and **Live**. In Paused, task-relevant state is held fixed while the agent computes and acts. In Replay, the agent receives the same changing observations as in a corresponding Live execution at the same cadence, but the response opportunity remains valid until the agent acts. In Live, both the observations and the underlying task state continue advancing while the agent computes and acts. These conditions let us vary whether task evolution is observed and whether that evolution can invalidate the response.

Comparing the three conditions separates different effects of task evolution. Paused versus Replay tests whether changing observations alone affect the agent when the response remains valid. Replay versus Live then tests what changes when the same evolving evidence is coupled to an underlying task that can invalidate the response before action receipt. Paused, Replay, and Live therefore let us distinguish failures caused by processing evolving evidence from failures caused by the task continuing to change while the agent responds.

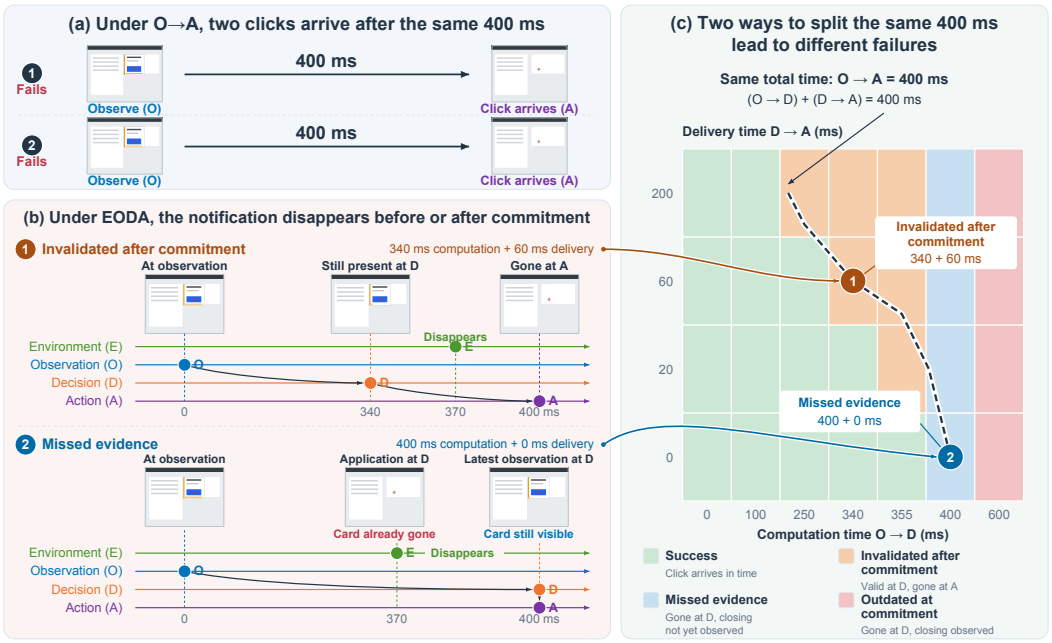


Figure 4: The same response time can hide different reasons for failure. **(a)** In Disappearing Notification, a scripted clicker observes a notification, and its click reaches the application 400 ms later. Both clicks arrive after the notification disappears. The observation–action view records the same response time but does not explain why each click fails. **(b)** EODA reveals when the notification disappears (E) and when the click becomes fixed (D). The notification disappears 370 ms after the initial observation. In Case 1, the click is committed after 340 ms, while the notification is still present, but arrives after another 60 ms, when it is gone: *invalidated after commitment*. In Case 2, the click is committed after 400 ms and arrives immediately. The notification has already disappeared, but the agent's latest observation still shows it: *missed evidence*. **(c)** The heatmap varies computation time ($O \to D$) and delivery time ($D \to A$). The arrows locate the two cases from (b) on the dashed line where these intervals sum to 400 ms. Observation, disappearance, and click arrival occur at the same times in both cases; changing when the click becomes fixed changes why it fails. Each cell shows one tested timing combination, with tested values spaced equally along each axis. Screenshots illustrate task states, and timeline spacing is schematic.

We choose five tasks to represent different ways a GUI response can become invalid over time: **Disappearing Notification**, where the opportunity to respond closes; **Moving Balloon**, where the target location changes; **Expiring Authenticator Code**, where an observed value becomes invalid; **Displaced Button**, where the location of the intended control changes; and **Reversing Trading Signal**, where the required action itself changes. Together, they cover changes to the opportunity, target, value, control location, and action rather than particular application domains. Appendix C provides detailed task definitions, action spaces, success criteria, and implementation details. After Paused, Replay, and Live establish whether task evolution changes the outcome, we use finer EODA interventions to vary which observation can affect the decision, when the decision is committed, or when the action is received while holding the remaining timing fixed. This lets us localize whether correctness is lost from missed evidence, an outdated decision at commitment, or invalidation after commitment.

4 Chronotope: Towards Control in Physical Time

§3 shows that GUI agents can fail because environment changes, observations, decisions, and action receipts occur at different times in physical execution. Diagnosing these failures does not by itself determine how an agent should respond when the interface continues to change during inference. We therefore develop Chronotope. Figure 5 summarizes its design. Chronotope maintains the recent EODA interaction history, reconciles unfinished decisions with newly arrived observations before commitment, and executes the latency-sensitive control path on the accelerator.

4.1 Online Interaction-State Modeling

A current screenshot does not always contain enough information to decide what the agent should do next. While the agent is reasoning, an earlier action may be received, the environment may change independently, or a newer observation may make part of the unfinished decision outdated. Chronotope therefore maintains the recent interaction history during inference rather than treating each screenshot as an isolated state.

Chronotope represents this history using the EODA events introduced in §3. Figure 5 shows this history as an event stream together with the dependencies between events. As new observations arrive, Chronotope records which interface state informed the unfinished decision and whether relevant state has changed since then. Later observations can also clarify whether a visible change followed an earlier action or occurred independently. The underlying model can therefore reason over the ongoing interaction rather than only the most recent screenshot.

4.2 Meta-Control over Unfinished Decisions

An unfinished decision does not necessarily become entirely invalid when the interface changes. In Moving Balloon, the decision to click the target may remain correct even though its location has changed. In Expiring Authenticator Code, the destination field may remain correct even though the value to enter has changed. Restarting the full decision process in these cases discards computation that is still useful. Chronotope therefore separates the parts of an unfinished decision that remain current from those that need to be updated before commitment.

Figure 5 shows this pre-commit control path. The planner produces a provisional action, and the updater reconciles it with the latest relevant observation before commitment. Chronotope can then preserve, revise, late bind, or re-decide the unfinished decision. Preservation keeps the decision unchanged when its relevant state remains current. Revision changes an argument that was specified earlier but has become outdated. Late binding leaves a state-dependent argument unresolved until fresher information is available. Re-decision restarts the underlying model when the intended operation itself is no longer appropriate.

4.3 Accelerator-Resident Fused Control

Updating an unfinished decision is useful only if the resulting action can still arrive in time. Maintaining interaction state and reconciling a decision introduce additional computation while the interface continues to change. A correct update can therefore still fail if this added work delays the action beyond its useful response window. Chronotope treats this execution cost as part of the control problem and keeps the latency sensitive control path on the accelerator.

As shown in Figure 5, Chronotope keeps the latency-sensitive intermediate state on the accelerator from model inference through action commitment. The provisional decision, interaction history, and update operations remain on this path until the action is finalized. This avoids repeatedly moving control state between GPU inference and expensive host-side orchestration for parsing, synchronization, and action construction. Reducing this overhead can keep an otherwise correct action within the available response window. Appendix Table 9 isolates this execution cost under a tight response window. At 200 pixels/s, host-side late binding succeeds in 9/100 trials, while accelerator-resident late binding succeeds in 35/100.

5 Experiments

In order to understand how interface changes during a response affect GUI-agent success, we study three questions. First, can an agent succeed when the interface waits but fail when it keeps changing? Second, can two responses that take the same amount of time have different outcomes? Third, can updating an action using the latest observation recover success without another model call? We describe the common experimental setup in §5.1 and answer these questions in §5.2–5.4.

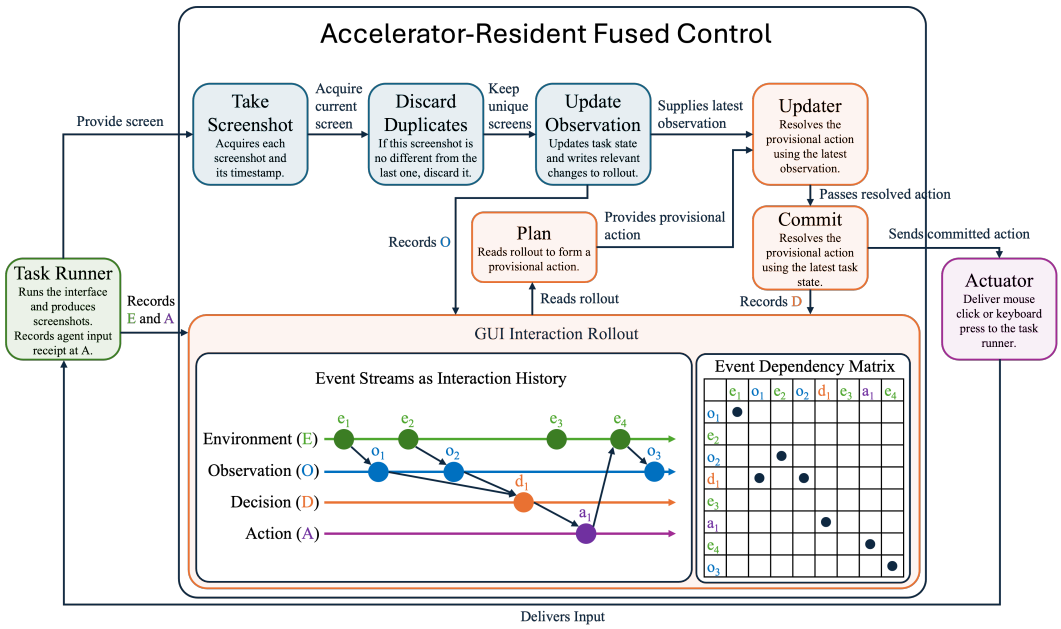


Figure 5: Chronotope's accelerator-resident fused control pipeline (§4.3). The controller discards duplicate screenshots and updates a GUI interaction rollout. The planner reads this history to propose an action, which the updater reconciles with the latest observation before commitment at D. The actuator delivers the committed input to the task runner, which records receipt at A. The rollout represents $EODA$ interaction history as event streams and a dependency matrix, where each filled cell indicates that the row event directly depends on the column event.

5.1 Experimental Setup

Tasks and agents. We evaluate Chronotope on five temporally dynamic GUI tasks introduced in §3: Disappearing Notification, Moving Balloon, Expiring Authenticator Code, Displaced Button, and Reversing Trading Signal. Our primary agent uses Qwen3.8-27B Qwen Team (2026) in BF16; additional evaluations use the same model in NVFP4 NVIDIA et al. (2025), Gemma-4-31B Gemma Team et al. (2026) in NVFP4, UI-TARS-1.5-7B Qin et al. (2025), and Agent S3 Gonzalez-Pumariega et al. (2026) with EvoCUA-8B Xue et al. (2026) and UI-TARS-1.5-7B. To separate temporal failure from ordinary task incompetence, each agent configuration enters a timing study only after passing the corresponding Paused qualification check.

Metrics. We evaluate each task under the three timing conditions introduced in §3. In **Paused**, the interface waits while the agent decides. In **Replay**, the agent receives the same sequence of observations as in Live, at the same times, but the response remains valid while it thinks. In **Live**, both the observations and the task continue advancing: a target may move, a value may change, or an opportunity may close before the action arrives. We also use **controlled comparisons** that rerun the same task instance while changing one part of the interaction—for example, which observation supplies the final action or how long a committed action takes to reach the application—while keeping the task, model call, and other event times unchanged. Each experiment is evaluated on 100 random seeds separate from seeds used in development. We report success and task return at action receipt, together with the $E \to O$, $O \to D$, $D \to A$, and $E \to A$ intervals.

Environment. All experiments run on one NVIDIA GB10 GPU with local models deployed via vLLM and TensorRT.

5.2 Claim 1. Agents That Succeed When the Interface Waits Can Fail When It Keeps Changing

GUI agents are commonly evaluated while the interface effectively waits for their response. We therefore ask whether the same agent remains successful when the interface continues changing

Task	Changed part	Frozen	Revised
Moving Balloon	coordinate	0/100	100/100
Expiring Authenticator Code	typed value	0/100	100/100
Displaced Button	button location	0/100	100/100

Table 1: Same-decision revision across different state-dependent arguments. The intended operation is preserved while the argument whose supporting state changed is resolved from the newest pre-commit state.

while it computes. We compare the same task instances under Paused, Replay, and Live execution. In Paused, the interface waits while the agent computes. Replay presents the same observations as Live at the same times, but keeps the response valid. Live allows both the observations and the task to keep changing.

Across the three task families in Table 2, agents that succeed in Paused and Replay can fail completely in Live execution. Figure 1(a) shows this transition as the response window is shortened, while Appendix Table 8 reports the complete results. With Qwen3.8-27B, the 4 s Disappearing Notification, Moving Balloon at 8 pixels/s, and Reversing Trading Signal all score 100/100 in Paused and Replay but 0/100 in Live. On Disappearing Notification, this transition follows the agent's response time: the BF16 model takes about 5.3 s and succeeds only when the response window exceeds this time. Faster NVFP4 and Gemma-4-31B configurations shift the transition earlier, while Agent S3 shows a more gradual transition because its response times are more variable.

5.3 Claim 2. The Same Response Time Can Lead to Different Outcomes

Claim 1 shows that Live execution can fail even when Paused and Replay succeed. A natural explanation is that the agent is simply too slow. To test whether response time alone explains these failures, we hold the model call, decision commitment time, and action receipt time fixed on Moving Balloon and change only which observation supplies the click coordinate. One response uses the balloon position from the initial observation, while the other uses its newest position available before commitment.

As shown in Figure 1(b), at 4, 8, and 16 pixels/s, the coordinate from the initial observation fails in 100/100 trials, while the coordinate from the newest pre-commit observation succeeds in 100/100. The model call, commitment time, and receipt time are unchanged. UI-TARS-1.5-7B shows the same effect at 8 pixels/s, improving from 0/100 to 69/100, compared with 70/100 on a stationary target. Thus, changing only the observation supporting the action can change the outcome even when the response takes the same amount of time. Appendix Figure 15 and Table 11 show the same principle without spatial grounding: with the committed order and receipt time fixed, moving only the signal reversal across D and A changes the same order from invalid to valid.

5.4 Claim 3. Updating Actions With the Latest Observation Can Recover Success

Claim 2 shows that an action can fail because it uses an outdated observation, even without taking longer. We therefore test whether the agent can keep its original decision while updating only the part that depends on the changing interface. We compare the original action, restarting the full model after a relevant change, and updating the state-dependent action argument from the latest observation before commitment.

Updating the action with the latest observation recovers success across different kinds of changing information. As shown in Figure 1(c) and Table 1, success rises from 0/100 to 100/100 on Moving Balloon, Expiring Authenticator Code, and Displaced Button when we update only the changed coordinate or value before commitment. On Balloon Select, the same update raises success from 0/100 to 99/100 while preserving the balloon selected by the model. Appendix Table 15 shows why: the model selects the correct named balloon in 100/100 trials, but its original coordinate hits that balloon in 0/100, while updating the coordinate recovers 99/100. These updates use the original model decision and require no additional model call.

Model	Task	Condition	Paused	Replay	Live	Replay $\bar{G}$	Live $\bar{G}$	Live failure
Qwen3.8-27B BF16	Disappearing Notification	window 4 s	100/100	100/100	0/100	-2.2 pixels	-400 pixels	late, outdated
Qwen3.8-27B BF16	Disappearing Notification	window 8 s	100/100	100/100	100/100	-2.2 pixels	-2.2 pixels	—
Qwen3.8-27B NVFP4	Disappearing Notification	window 2 s	100/100	100/100	4/100	-2.1 pixels	-384 pixels	late, outdated
Qwen3.8-27B NVFP4	Disappearing Notification	window 4 s	100/100	100/100	100/100	-2.1 pixels	-2.1 pixels	—
Gemma-4-31B NVFP4	Disappearing Notification	window 2 s	100/100	100/100	0/100	-2.2 pixels	-400 pixels	late, outdated
Gemma-4-31B NVFP4	Disappearing Notification	window 4 s	100/100	100/100	100/100	-2.2 pixels	-2.2 pixels	—
Agent S3 (EvoCUA-8B + UI-TARS-1.5-7B)	Disappearing Notification	window 8 s	96/100	96/100	0/100	-3.2 pixels	-400 pixels	late, outdated
Agent S3 (EvoCUA-8B + UI-TARS-1.5-7B)	Disappearing Notification	window 16 s	96/100	96/100	29/100	-3.2 pixels	-281 pixels	late, outdated
Qwen3.8-27B BF16	Disappearing Notification	ad window 4 s	100/100	100/100	0/100	-1.0 s	-4.0 s	late, outdated
Qwen3.8-27B BF16	Moving Balloon	drift 8 pixels/s	100/100	100/100	0/100	-2.1 pixels	-42 pixels	outdated
Qwen3.8-27B BF16	Moving Balloon	jump at 2 s	100/100	100/100	0/100	-2.0 pixels	-97 pixels	outdated
Qwen3.8-27B BF16	Reversing Trading Signal	reversal at 2.0 s	100/100	100/100	0/100	$+1$	-1	outdated
Qwen3.8-27B BF16	Reversing Trading Signal	no reversal	100/100	100/100	100/100	$+1$	$+1$	—

Table 2: Paused, Replay, and Live success on matched instances, 100 held-out seeds per cell. The table shows boundary cells and representative non-deadline failures; Table 8 reports the complete sweeps. $\bar{G}$ is mean return at application receipt in the unit shown by the row.

Restarting the full model can recover success after a one-time change, but it adds another deliberation. On Moving Balloon (Jump variant), restart succeeds in 99/100 trials but moves median commitment from 5.1 to 11.2 s, as shown in Appendix Figure 19. On Expiring Authenticator Code, it succeeds in 98/100 while moving commitment from 3.8 to 9.3 s. Under continuous Moving Balloon, however, restart fails in 100/100 trials because the target moves again during each replacement model call. Appendix Figure 16 shows this contrast directly: late binding commits after about 5.1 s, while restart first commits after 17.6 s.

Intervention	Change before D	Closing window	Drift through $D \to A$
None	0/100	0/100	0/100
Newest pre-commit state	100/100	0/100	0/100
Earlier receipt	0/100	99/100	0/100
Both	100/100	99/100	100/100

Table 3: The intervention that repairs a trial depends on where correctness is lost. Updating state repairs a pre-commit change but not a pure deadline failure; earlier receipt shows the opposite pattern; continuous drift through the irreversible interval requires both.

6 Conclusion

GUI agents can fail even when they choose the right action because the interface continues changing while they think. We model this interaction through environment change, observation, decision commitment, and action receipt, and introduce ChronotopeGym to vary these timing relations experimentally. Across five tasks, agents that succeed when the interface waits can fail in Live execution. Faster responses help when the task imposes a deadline, but equal response times can still produce different outcomes depending on which observation supports the action and when that action arrives. Chronotope addresses these cases by updating unfinished actions with newer evidence and moving time-sensitive control onto the GPU.

Limitations and future work. The decision updates used in our experiments are lightweight but task-specific. Another multimodal model could infer what information in an unfinished decision should be preserved or replaced, but repeatedly invoking such a model would add inference cost and latency. Our current design therefore favors inexpensive task-specific updates while demonstrating that revising unfinished decisions can recover Live performance. Future work can study more general update mechanisms that retain this low-latency property and can additionally predict changes that occur before the action reaches the interface.

AI USE STATEMENT

In this work, we used generative AI tools for design or provide feedback on research methodology or experiments, implement methods, support qualitative and thematic data analysis, interpret results. We have not used generative AI tools for help develop theoretical models or conceptual frameworks, propose or refine hypotheses and Generate synthetic data sets, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, assist with translation, clean and reformat dataset, are not applicable to this work.

Additionally, we used generative AI tools for create or modify scientific figures or images, create or edit software code, draft parts of a research paper, summarize or analyze existing literature, discover research topics or identify gaps, brainstorming, sourcing/searching for information, edit a research paper to improve readability, identify relevant literature, suggest a structure for a research paper, propose a title or keywords for a research paper.

We have reviewed all AI-assisted work. We checked AI-generated related literature for their accuracy and differentiation from our methodology. We reviewed AI-generated writing outline and writing edits for accuracy and fairness. We extensively reviewed and often rejected frontier AI model-generated code for correctness, accuracy, and faithfulness to our methodological design, or experimental design, and experimental parameters. We also carefully examined AI-suggested interpretation of results for accuracy, fairness, and scientific parsimony.

We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Chronotope studies software agents that can issue inputs to real GUI applications. While Chronotope's aim is to enable to address outdated, late, or repeated action, it is not guaranteed to succeed in all scenarios and can therefore cause unintended, real-world consequences if deployed unscrupulously. For example, in the quantitative, high-frequency trading scenario modeled by the trading task in ChronotopeGym, failures should likely result in substantial monetary losses for an otherwise reliable trading strategy not using AI GUI agents.

Chronotope, like other GUI agents, can also raise the question of ethical responsibility when used in real-time, real-world applications. In the algorithmic trading scenario, when the trading strategy is dynamic and pattern recognition is probabilistic rather than purely rule-based, Chronotope's enabling of real-time control may also complicates the regulation and compliance in applications, including but not limited to fiduciary duty (15 U.S. Code §§80b-1), best interest (17 C.F.R. §§240.15l-1), and more contemporaneously, AI conflict of interest (proposed 17 C.F.R. §§240.15l-2).

The ChronotopeGym environment proposed in this study providing a realistic, measurable, and controllable interface for studying GUI agent behaviors, efficacy, efficiency, and real-world impacts, while preventing real-world consequences.

The first author is the project lead. They are solely responsible for implementing the Chronotope methodology and the ChronotopeGym environment, running the experiments, and collecting the reported results. The first author worked closely with the second author on creating and refining concrete research questions. In addition, the second author provided concrete suggestions on methodological design and suggested related works.

REPRODUCIBILITY STATEMENT

We release the ChronotopeGym task suite, evaluators, and Chronotope controller at `https://anonymous.4open.science/r/chronotope_iclr-3FD6/`. Every experiment result in §5 was evaluated on 100 test-only random seeds. The experimental configurations and held-out seed ranges are reported in Appendix C.8.

REFERENCES

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s: An open agentic framework that uses computers like a human. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.08164.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Jang, and Zack Hui. Windows agent arena: Evaluating multi-modal OS agents at scale. *arXiv preprint arXiv:2409.08264*, 2024.

Lawrence D. Brown, T. Tony Cai, and Anirban DasGupta. Interval estimation for a binomial proportion. *Statistical Science*, 16(2):101–133, 2001.

Yilin Cao, Yufeng Zhong, Zhixiong Zeng, Liming Zheng, Jing Huang, Haibo Qiu, Peng Shi, Wenji Mao, and Wan Guanglu. Mobiledreamer: Generative sketch world model for gui agent. *arXiv preprint arXiv:2601.04035*, 2026.

Dongping Chen, Yue Huang, Siyuan Wu, Jingyu Tang, Huichi Zhou, Qihui Zhang, Zhigang He, Yilin Bai, Chujie Gao, Liuyi Chen, Yiqiang Li, Chenlong Wang, Yue Yu, Tianshuo Zhou, Zhen Li, Yi Gui, Yao Wan, Pan Zhou, Jianfeng Gao, and Lichao Sun. Gui-world: A video benchmark and dataset for multimodal gui-oriented understanding. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=QarKTT5brZ.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing GUI grounding for advanced visual GUI agents. *arXiv preprint arXiv:2401.10935*, 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*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/5950bf290a1570ea401bf98882128160-Abstract-Datasets_and_Benchmarks.html.

Zihan Dong, Rui Qian, Qishi Zhan, Dongshen Peng, Kaixin Li, and Yu Li. Why are gui agents correct but late? decode on the decision-time critical path, tested with pre-compiled policy trees. *arXiv preprint arXiv:2607.28399*, 2026. URL https://arxiv.org/abs/2607.28399.

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 *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11642–11662, 2024.

Difei Gao, Lei Ji, Zechen Bai, Mingyu Ouyang, Peiran Li, Dongxing Mao, Qinchen Wu, Weichen Zhang, Peiyi Wang, Xiangwu Guo, Hengxu Wang, Luowei Zhou, and Mike Zheng Shou. AssistGUI: Task-oriented pc graphical user interface automation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13289–13298, 2024.

Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, et al. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

Gonzalo Gonzalez-Pumariega, Vincent Tu, Chih-Lun Lee, Jiachen Yang, Ang Li, and Xin Eric Wang. Scaling agents for computer use. *Transactions on Machine Learning Research*, 2026.

Boyu Gou, 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. *arXiv preprint arXiv:2401.13919*, 2024.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. CogAgent: A visual language model for GUI agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14281–14290, 2024.

Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem Alshikh, and Ruslan Salakhutdinov. Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. *arXiv preprint arXiv:2402.17553*, 2024.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024.

Bojie Li and Noah Shi. Agent-computer observation interfaces enable dynamic computer use. *arXiv preprint arXiv:2606.29472*, 2026. doi: 10.48550/arXiv.2606.29472.

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. Screenspot-pro: GUI grounding for professional high-resolution computer use. *arXiv preprint arXiv:2504.07981*, 2025.

Yongyuan Liang, Shijie Zhou, Yu Gu, Hao Tan, Gang Wu, Franck Dernoncourt, Jihyung Kil, Ryan A. Rossi, and Ruiyi Zhang. Anticipatory planning for multimodal ai agents. *arXiv preprint arXiv:2603.16777*, 2026.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Qinchen Yang, Shiwei Wu, Zechen Bai, Weixian Lei, Lijuan Wang, and Mike Zheng Shou. Videogui: A benchmark for GUI automation from instructional videos. *arXiv preprint arXiv:2406.10227*, 2024.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. ShowUI: One vision-language-action model for GUI visual agent. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19498–19508, 2025. doi: 10.1109/CVPR52734.2025.01816.

Enqi Liu, Liyuan Pan, Zhi Gao, Yan Yang, Chenrui Shi, Yang Liu, Jingrong Wu, and Qing Li. Benchmarking and improving gui agents in high-dynamic environments. *arXiv preprint arXiv:2604.25380*, 2026. URL https://arxiv.org/abs/2604.25380.

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

NVIDIA, Felix Abecassis, Anjulie Agrusa, Dong Ahn, Jonah Alben, et al. Pretraining large language models with NVFP4. *arXiv preprint arXiv:2509.25149*, 2025.

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. *arXiv preprint arXiv:2406.12373*, 2024.

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. URL https://arxiv.org/abs/2501.12326.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026. URL https://qwen.ai/blog?id=qwen3.8.

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. URL https://arxiv.org/abs/2307.10088.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=il5yUQsrjC.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen 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*, volume 38, 2025. doi: 10.52202/085713-4669.

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. In *International Conference on Learning Representations*, 2025. 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*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5d413e48f84dc61244b6be550f1cd8f5-Abstract-Datasets_and_Benchmarks_Track.html.

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous GUI interaction. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 69772–69805, 2025.

Taofeng Xue, Chong Peng, Mianqiu Huang, Linsen Guo, Tiancheng Han, Haozhe Wang, Jianing Wang, Xiaocheng Zhang, Xin Yang, Dengchang Zhao, et al. EvoCUA: Evolving computer use agents via learning from scalable synthetic experience. *arXiv preprint arXiv:2601.15876*, 2026.

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.

Jiashu Yao, Heyan Huang, Daiqing Wu, Wangke Chen, Huaxi Ai, Haoyu Wen, Zeming Liu, and Yuhang Guo. Benchmarking living-screen-native gui agents on short-video platforms. *arXiv preprint arXiv:2606.04701*, 2026. URL https://arxiv.org/abs/2606.04701.

Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. Large language model-brained GUI agents: A survey. *Transactions on Machine Learning Research*, 2025a.

Chaoyun Zhang, He Huang, Chiming Ni, Jian Mu, Si Qin, Shilin He, Lu Wang, Fangkai Yang, Pu Zhao, Bo Qiao, Chao Du, Liqun Li, Yu Kang, Zhao Jiang, Suzhen Zheng, Rujia Wang, Jiaxu Qian, Minghua Ma, Jian-Guang Lou, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. UFO^2: The desktop agentos. *arXiv preprint arXiv:2504.14603*, 2025b.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. GPT-4V(ision) is a generalist web agent, if grounded. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 61349–61385. PMLR, 2024a. URL https://proceedings.mlr.press/v235/zheng24e.html.

Longtao Zheng, Zhiyuan Huang, Zhenghai Xue, Xinrun Wang, Bo An, and Shuicheng Yan. Agentstudio: A toolkit for building general virtual agents. *arXiv preprint arXiv:2403.17918*, 2024b.

Hongbin Zhong, Fazle Faisal, Luis França, Tanakorn Leesatapornwongsa, Adriana Szekeres, Kexin Rong, and Suman Nath. ActionEngine: From reactive to programmatic GUI agents via state machine memory, 2026.

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 Extended Related Works

A.1 GUI Agents

GUI agents translate high-level user goals into interactions with graphical interfaces. A typical system receives a screenshot, sometimes together with structured interface information such as the DOM or accessibility tree, reasons about the next step, grounds that step to an interface element, and emits mouse, keyboard, touch, or hybrid GUI–API actions. This interface-level abstraction is useful because one agent can operate heterogeneous software through the interfaces already exposed to users, rather than requiring a task-specific API for every application. Recent surveys accordingly organize GUI agents around perception and grounding, reasoning and planning, and action execution (Zhang et al., 2025a).

Existing systems realize this loop in several ways. SeeAct couples multimodal planning with interface grounding (Zheng et al., 2024a), while Agent S combines experience-augmented hierarchical planning with an agent–computer interface (Agashe et al., 2025). A complementary line trains GUI-specialized visual action models: CogAgent and UGround emphasize visual GUI understanding and grounding (Hong et al., 2024; Gou et al., 2025); OS-ATLAS, ShowUI, Aguvis, UI-TARS, and OpenCUA extend toward cross-platform or end-to-end action prediction (Wu et al., 2025; Lin et al., 2025; Xu et al., 2025; Qin et al., 2025; Wang et al., 2025). Despite these advances, recurring problems remain in localizing small or ambiguous controls, maintaining intent over long tasks, transferring across applications, and responding to interfaces that change during interaction. Chronotope focuses on the last issue: a decision may be semantically correct for the screen that supported it yet become outdated before it is committed or received.

A.2 GUI-Agent Benchmarks and Time

GUI-agent benchmarks cover different parts of this stack. MiniWoB++ provides controlled browser tasks (Liu et al., 2018); Mind2Web and Android in the Wild collect multi-step web or mobile trajectories (Deng et al., 2023; Rawles et al., 2023); ScreenSpot and ScreenSpot-Pro isolate GUI grounding (Cheng et al., 2024; Li et al., 2025); and OmniACT, VideoGUI, and GUI-World evaluate richer action or video-based understanding (Kapoor et al., 2024; Lin et al., 2024; Chen et al., 2025). Interactive environments then test task completion in running software, including WebArena and VisualWebArena for the web (Zhou et al., 2024; Koh et al., 2024), WorkArena for enterprise browser tasks (Drouin et al., 2024), AndroidWorld for mobile applications (Rawles et al., 2025), and OSWorld, Windows Agent Arena, AgentStudio, and AssistGUI for desktop or cross-application control (Xie et al., 2024; Bonatti et al., 2024; Zheng et al., 2024b; Gao et al., 2024). These benchmarks have been central to progress in grounding, planning, and end-to-end task success, but their use of a live application does not by itself specify what happens to task time while the agent is computing.

Time therefore needs to be reviewed separately from benchmark realism. Many interactive harnesses remain step-synchronous: they capture an observation, block while the agent returns an action, execute that action, and only then request the next observation. Mind2Web-Live evaluates agents against changing online websites (Pan et al., 2024), but does not expose within-decision observation updates or score action arrival time. MiniWoB++ is an earlier exception in which its default positive reward is scaled by remaining wall-clock time, although most prior publications use a binary reward that removes this time penalty (Liu et al., 2018). Recent benchmarks make temporal change more explicit: DynamicGUIBench scripts transient events (Liu et al., 2026), LivingScreen uses continuously playing interfaces (Yao et al., 2026), DynaCU-Bench includes timed dynamic tasks (Li & Shi, 2026), and AAPT evaluates deadline-sensitive reactions (Dong et al., 2026). Table 4 compares these timing semantics directly.

Benchmark	State advances during compute?	Observations during decision?	Wall-clock time measured?	Arrival time changes score?
Mind2Web (Deng et al., 2023); GUI-World (Chen et al., 2025); ScreenSpot (Cheng et al., 2024); ScreenSpot-Pro (Li et al., 2025); OmniACT (Kapoor et al., 2024); VideoGUI (Lin et al., 2024); Android in the Wild (Rawles et al., 2023)	no (static)	no	no	no
MiniWoB++ (Liu et al., 2018)	yes (task timer)	no (step API)	task timer	yes by default[g]
OSWorld (Xie et al., 2024); WebArena (Zhou et al., 2024); VisualWebArena (Koh et al., 2024); AndroidWorld (Rawles et al., 2025)	live, step-sync	no	log only[a]	no
Mind2Web-Live (Pan et al., 2024); Windows Agent Arena (Bonatti et al., 2024); WebVoyager (He et al., 2024); AgentStudio (Zheng et al., 2024b)	live, step-sync	no	log only[b]	no[c]
Online-Mind2Web (Xue et al., 2025)	live, agent-side step-sync[d]	no	no	no
DynamicGUIBench (Liu et al., 2026)	scripted timed events[e]	no (video per step)	no	no (final state)[f]
LivingScreen (Yao et al., 2026)	yes (autoplay)	pull-based clips	no	no (endpoint)
DynaCU-Bench (Li & Shi, 2026)	yes (timers)	bundled per step	decision latency	yes
AAPT testbed (Dong et al., 2026) (method)	yes (deadlines)	yes (30 FPS gate)	server receipt	yes

Table 4: Timing semantics of GUI-agent benchmarks and related testbeds. [a]OSWorld and AndroidWorld log timestamps or episode run time that enter no metric; WebArena and VisualWebArena record none. [b]Windows Agent Arena and AgentStudio log per-action timestamps; Mind2Web-Live and WebVoyager record none. [c]AgentStudio only through an optional task-level timeout. [d]No benchmark-owned harness; the agents reported in Online-Mind2Web ran in their own step-synchronous loops, and its evaluator scores task text, action history, and screenshots with no time input. [e]The released task specifications script transient events of 1.5–7 s; the runner is unreleased, so overlap with inference is unverified. [f]Evaluators sampled are OSWorld-style end-state checks with no clock input; not reviewed over all 500 tasks. [g]MiniWoB++'s original reward scales positive reward by the remaining task time; most prior publications instead use its binary reward without this time penalty.

For the temporal columns in Table 4, we reviewed the public harness or evaluator code when available and otherwise the paper, documentation, and released task specifications. The released artifact will include the consulted files, commit heads, and supporting excerpts for each row.

A.3 Latency and Temporal Control in GUI Agents

Recent methods address temporal failures at different points in the interaction path. AOI decouples observation from action so transient visual or audio evidence can be captured between model calls (Li & Shi, 2026). MobileDreamer and TraceR1 anticipate future interface states or short-horizon trajectories before execution (Cao et al., 2026; Liang et al., 2026). UFO2 amortizes decision cost through speculative multi-action planning (Zhang et al., 2025b), ShowUI reduces visual-token computation (Lin et al., 2025), ActionEngine reuses an explored state-machine representation to synthesize executable programs (Zhong et al., 2026), and AAPT pre-computes bounded conditional policies so expensive decoding need not occur after a transient event appears (Dong et al., 2026).

These approaches are complementary but target different failure sources. Faster inference and pre-computation expand the response windows an agent can meet; continuous observation reduces missed evidence; and prediction can compensate for future change. None of those quantities alone identifies whether a failure occurred because relevant evidence was never observed, because available evidence did not revise a pending decision, because a committed action became invalid before receipt, or because it arrived after the opportunity closed. Chronotope therefore uses environment change, observation availability, decision commitment, and action receipt as separate events, so temporal failures can be diagnosed and controlled according to where correctness is lost.

B Implementation Details

B.1 Physical Observation and Input Path

The intended physical boundary is the same as the software one: captured video of the target display enters the controller at O, committed actions leave it through a USB HID emulator, and the target application records A in its input handler. No result in this paper comes from that route; its $E \to O$ and $D \to A$ distributions are deferred to a physical study.

B.2 Controller State and Commitment

Each Balloon Drift frame receives an observation version, and the trace records the frame used at binding. The action-relevant control-state version advances when tracked displacement crosses the movement threshold. The binder reads the newest available frame, avoiding sequential processing of older frames before commitment. The Code Panel task defines the typed value as its action-relevant field: its control-state version advances when the code decoded from the panel region changes, and the binder fills the value from the newest available frame by decoding the panel with the task's fixed glyph font. The completed results exercise these two binders, the Balloon Drift coordinate and the Code Panel value.

The Balloon Drift provisional action stores the decision identifier, action kind, selected target, proposed coordinate, binder, and control-state version. Initial coordinate publishes the proposed point unchanged. Next-decision buffering stores later frames without altering the current action. Restart-on-update discards the proposal and invokes the full model. Late binding applies normalized cross-correlation to a 25×25 patch around the proposed point, translates (x, y) by the measured displacement, validates the target and action kind, and commits under the original decision identifier.

B.3 Source Time

Instrumented tasks associate each rendered state with an evaluator-only identifier and target-side timestamp. The identifier is recovered after capture, joined with the target log, and removed before the observation reaches the agent; the application records A in its input handler. Software experiments inject observations at the common O boundary, consume committed actions at D, and use one task clock for E, O, D, and simulated A, which enables exact replay and direct delay control. No claim in the paper depends on an unresolved cross-machine clock difference.

C Task Suite Reproducibility Details

Environment	Dynamic variable	Action space	Success	Return
Disappearing Notification	a card that disappears after a window	click, wait	click inside the card before it closes	$-$click error (pixels)
A, sponsor player	a sponsor that plays until skipped	click, wait	skip lands within the window	$-$seconds of sponsor watched
Reversing Trading Signal	a signal that appears, reverses, or expires	buy, sell, wait	valid order at receipt	order utility (± 1)
C, moving targets	a balloon that drifts, jumps, or is crossed	click, wait	click inside the target at receipt	$-$click error (pixels)
D, changing code	a code replaced during deliberation	type, wait	typed value equals the code shown at receipt	$-$seconds to a correct value

Table 5: The task suite: what changes, what the agent can do, and the two outcomes every episode reports.

C.1 Notification and Player: Expiring Response

the Notification task uses an instrumented synthetic video player (320×240) with a seeded sponsor interval. While the sponsor plays, the player shows a sponsor banner with a countdown and a plain, unlabeled skip button (60×24pixels, unique color); the notification rendering shows a card with a blue button instead. The action space is `click(x,y)` or `wait` in every study: a skip is a click received inside the button, and the scripted oracle clicks the same button. Sponsor onset defines E; the first published observation containing usable evidence defines O; commitment of the click defines D; and player receipt defines A. A later E records the visible seek. The scorer reports timely success, sponsor time watched, repeated skips before the visible effect, and actions outside the valid interval.

C.2 TRADING: SYNTHETIC LIVE CHART

the Trading task streams a seeded synthetic price path into a chart and order panel. Each episode specifies an evaluator-defined response rule. The simulator supplies exact event times, market state, valid action, and action window, allowing a signal to appear briefly, reverse before commitment, or expire before receipt. The task evaluates timed rule following, not financial forecasting.

C.3 BALLOON: SYNTHETIC MOVING TARGETS

the Balloon task logs exact target position at every update. Balloon Drift uses continuous horizontal drift. Balloon Jump moves the target once by 96 pixels and then holds it fixed; the jump time is a parameter, placed during deliberation (2 s) or after receipt (14 s) in the agent cells and swept across B, D, and A in the oracle cells. Balloon Select adds named targets and crossing distractors, logging target identity separately from committed coordinate and the object under the click at A. The binder may update only the coordinate associated with the model-selected target.

C.4 THE CHROMIUM-RENDERED PAGE

The reflow variant is a form page with a heading, text, and a Continue button, rendered once per seed by headless Chromium into two frames, before and after a notice banner is inserted above the button, with the button geometry read from the DOM. The renderer shows the first frame until the banner time (2 s or 14 s after the page appears) and the second afterwards, so the manifest stays a pure function of task time; a live browser through the O boundary is not used. The model sees the page at its native 640×480 (prompt 12); the binder is the Balloon Drift template matcher. Cells: banner at 2 s with frozen, late binding, and restart; banner at 14 s with frozen and late binding; seeds 1000–1099; Paused gate on the page without the banner.

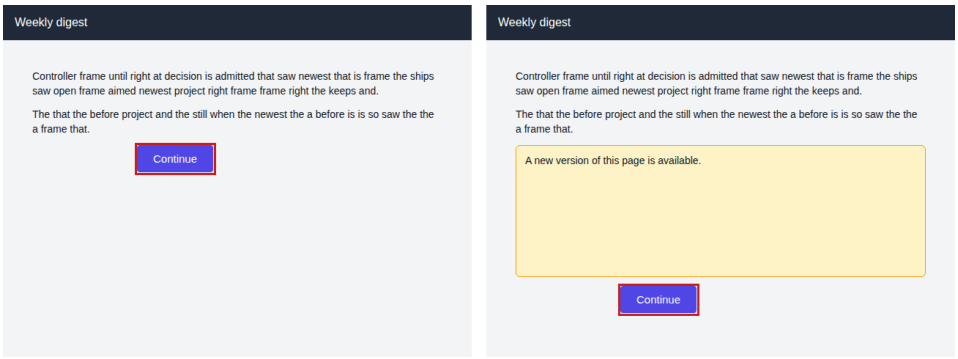


Figure 6: The Chromium-rendered page before (left) and after (right) the notice banner is inserted, seed 1000; the red box marks the Continue button, displaced by 192 pixels.

C.5 CODE PANEL: CHANGING CODE

the Code Panel task renders a code panel and a text field on the notification desktop (320×240). The four-digit code is seeded, drawn with a fixed glyph font, shown from $E = 0.5$ s, and replaced by a second seeded code at $E + 2$ s, $E + 14$ s, or never. The prompt asks the model to type the code currently shown; the schema admits `type` or `wait` with a value. Under late binding the value is left unresolved and filled just before D by decoding the panel region of the newest available frame with the same font, evaluator-side and without the manifest; restart-on-change discards the proposal when the decoded value changes. Both binders are specific to their rendering: the Balloon Drift template matcher and the Code Panel glyph decoder. Success requires the typed value to equal the code shown at A; the return is minus the seconds from E to the receipt of a correct value, with the window charged when none arrives. Cells: change at 2 s with frozen, late binding, and restart; change at 14 s and static with frozen and late binding; seeds 1000–1099; Paused gate on the static code.

C.6 Recorded Episodes

Figures 7–11 show one recorded episode of each task family as three frames, the frame the model saw at B, the newest frame at commitment D, and the first frame at or after receipt A, in two rows that share the model call. Every time, coordinate, and verdict is read from the trace; the model is Qwen3.8-27B (thinking off) served through vLLM, with the route of Appendix C.8. Figure 2 is the same construction for the drifting balloon.

C.7 Agent Interface and Shared Conditions

The unit of evaluation is an *agent configuration*: a checkpoint together with its observation policy, controller, serving arrangement, and delivery route; results are reported separately for each. The core adapter receives timestamped observations and publishes a complete action at D; the harness records application receipt at A. Instrumented adapters also expose the decision identifier, supporting observation identifiers, control-state version, and whether a relevant update was buffered, caused restart, or changed the current action. Direct comparisons fix the checkpoint, prompt, observation resolution, action grammar, decoding settings, task instance, and event schedule.

The standardized software track fixes the task generator, common O boundary, timing gates, and receipt simulator; an end-to-end track can record a system's own capture, serving, controller, and input route. All completed results in this paper come from the software track. Every main comparison reports success rate and task-specific return; Live retention, failure composition, interval distributions, and task-specific diagnostics provide the explanation.

C.8 Agent and Serving Configuration

Table 6 lists the configuration shared by every model study. Prompts will be reproduced verbatim in the released code (prompt versions 4, 5, 8, 9, 10, and 11 for the notification, balloon, balloon-select, player, chart, and code-panel renderings; 12 and 15 for the baked and live Chromium pages; 7 for the UI-TARS native grammar; 16 for EvoCUA's tool-call grammar; 17 for Agent S3); each states the task in two or three sentences, defines the coordinate convention, and asks for the JSON object only.

C.9 Which Failure Does Each Intervention Test?

Gates at the O, D, and A boundaries let the harness hold $E \to A$ fixed while changing the state used at commitment, redistribute a fixed delay among the intervals, or change only delivery. Task variants expose transient evidence before O, relevant change before D, a closing window, predictable change after D, and combinations, with static targets and irrelevant motion as negative controls; each is paired with the intervention designed for it (Appendix C, Table 7).

Because the conditions were constructed to match the interventions, the informative test is whether any intervention also corrects a neighboring condition. With a scripted controller whose only defect is the timing of its path, none does (Appendix C, Figure 12); §**??** repeats the test with the 27 B model.

C.10 Metrics and Statistical Analysis

Every episode reports application-time success and task return. The video-player rendering of the Notification task uses $G_A = -T_{\text{sponsor}}$; the notification rendering has no sponsor and, as a windowed-target task, reports the click-error return G_C defined below, which is why Table 2 lists the two renderings in different units. The Trading task uses evaluator-defined utility $G_B = U_{\text{order}}$. The Balloon task uses $G_C = -e_{\text{click}}$, the click's distance to the target at receipt, with the display diagonal charged when the target is absent at receipt or no click is received; it also reports coordinate translation, model calls, and response-window status at receipt. Timing reports include $E \to O$, $O \to D$, $D \to A$, total $E \to A$, receipt-time slack, and the source time of the observation supporting the committed action.

Binary proportions use Wilson 95% intervals, which remain informative at $0/n$ and n/n where the Wald interval collapses to zero width (Brown et al., 2001). Paired binary outcomes use exact McNe-

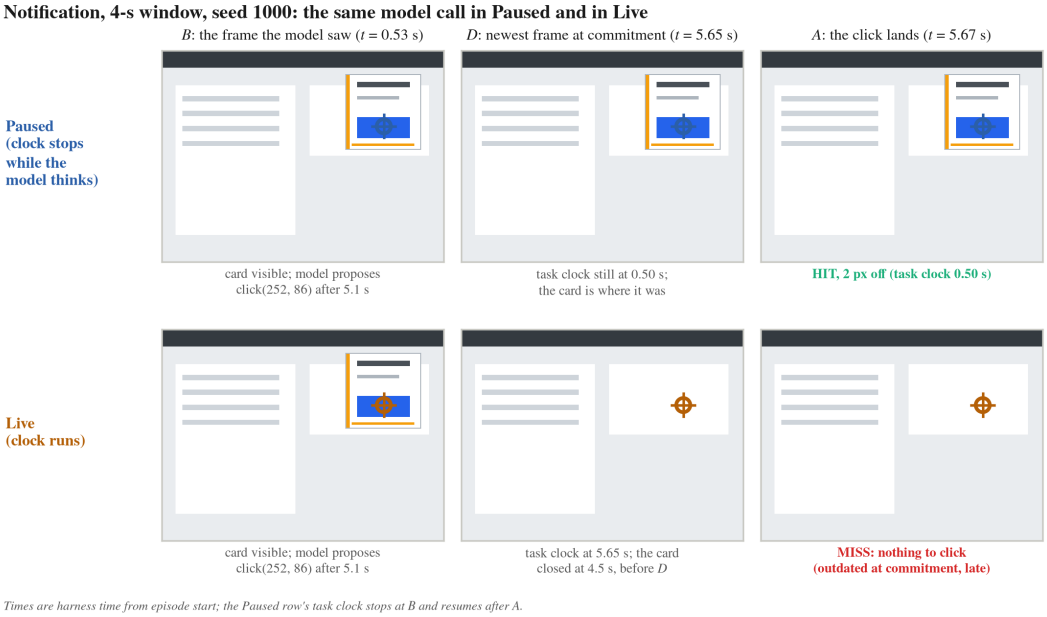


Figure 7: Notification, 4 s window, seed 1000: the same model call in Paused and in Live. The model proposes the click 5.1 s after B. In Paused the task clock stops at B, so the card is still present at D and the click is a hit. In Live the card closed at 4.5 s, before D; the same click arrives on an empty desktop and is labeled outdated at commitment and late (§5.2).

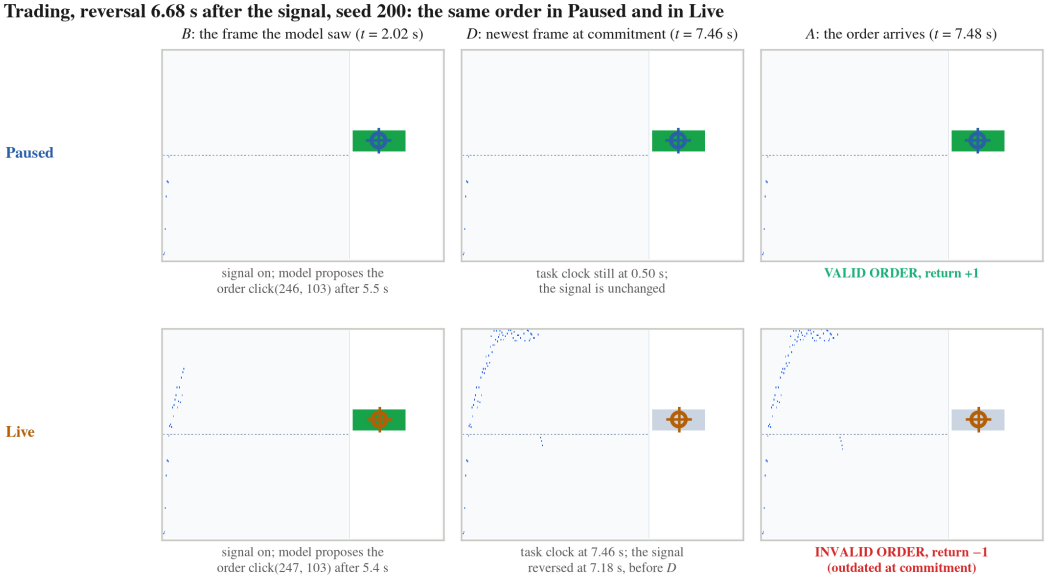


Figure 8: Trading, reversal 6.68 s after the signal, seed 200: the same order in Paused and in Live. The model proposes the order 5.5 s after B. In Paused the signal is unchanged at D and the order is valid (return $+1$). In Live the signal reversed at 7.18 s, before D; the same order is invalid (return -1) and labeled outdated at commitment (§5.2).

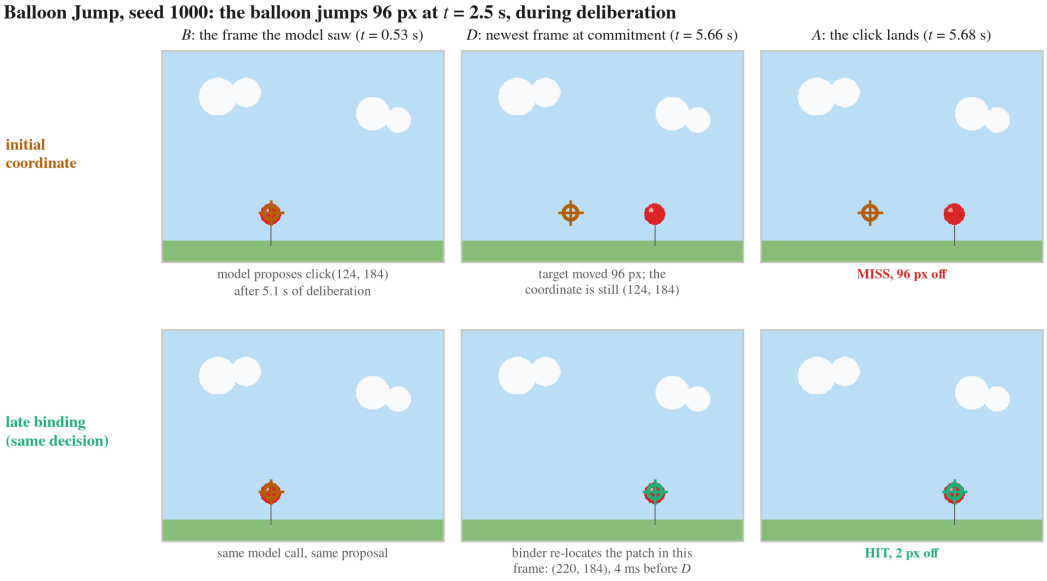


Figure 9: Balloon Jump, seed 1000: the balloon jumps 96 pixels at 2.5 s, during a 5.1 s deliberation. Top: the proposed coordinate (124, 184) is committed unchanged and misses by 96 pixels. Bottom: the same decision with late binding; the binder re-locates the patch in the newest frame 4 ms before D and the click lands 2 pixels from the center (§5.4).

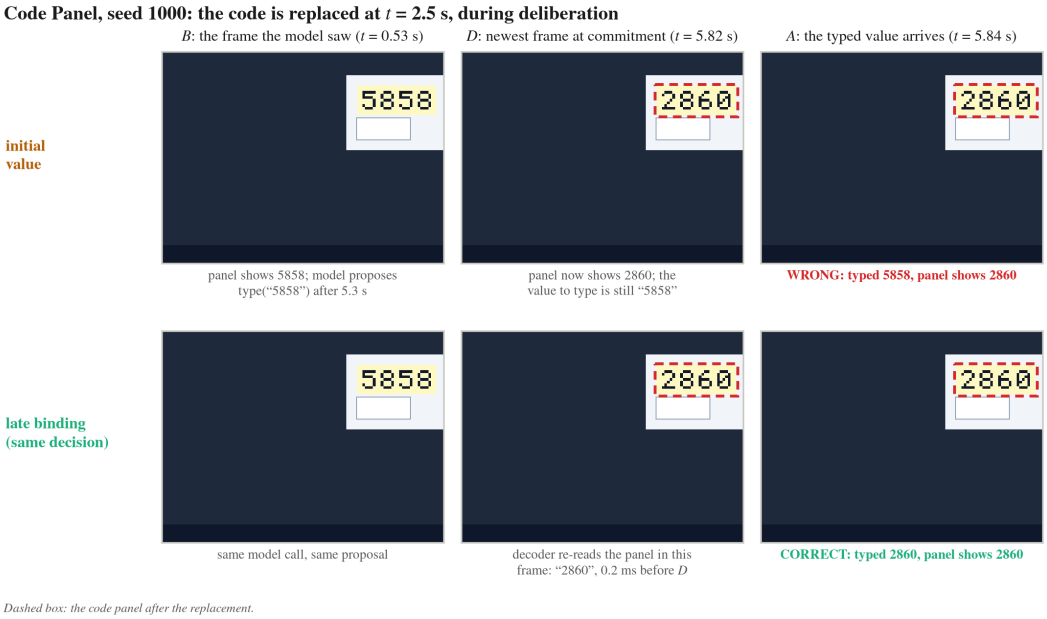


Figure 10: Code Panel, seed 1000: the code is replaced at 2.5 s, during a 5.3 s deliberation. Top: the proposed value 5858 is typed while the panel shows 2860. Bottom: the same decision with late binding; the decoder re-reads the panel in the newest frame 0.2 ms before D and types 2860. The open argument here is a typed value, not a coordinate (§5.4).

Item	Value
Primary agent	Qwen3.8-27B (Hugging Face `Qwen/Qwen3.8-27B`, snapshot `1d4bf0f2`); thinking disabled. NVFP4 rows: `unsloth/Qwen3.8-27B-NVFP4`, the same weights in 4-bit floating point, same prompt, grammar, and decoding; seeds 200–299
Other agents	Gemma-4-31B NVFP4 (`nvidia/Gemma-4-31B-IT-NVFP4`); Agent S3 (`gui-agents` 0.3, planner `meituan/EvoCUA-8B-20260105`, grounder UI-TARS-1.5-7B, both served on the same GPU for the whole cell, the one exception to one model at a time); tried and not passing: Qwen3-VL-8B-Instruct, UI-TARS-1.5-7B (`ByteDance-Seed/UI-TARS-1.5-7B`) outside the balloon, EvoCUA-8B/32B in their native grammar; Appendix E
Serving	vLLM 0.27.1, bfloat16 weights, no quantization, 8192-token context, one image per request, OpenAI-compatible endpoint on the same host as the controller and application
Hardware	one NVIDIA GB10 GPU
Decoding	temperature 0; at most 64 output tokens (256 for the UI-TARS native grammar); JSON structured output with schema {`action`∈{`click`,`wait`}, `x`, `y`} for the click tasks and {`action`∈{`type`,`wait`}, `value`} for the Code Panel task
Observation	320×240 canvas rendered at 60 Hz, published to the controller 30 ms after rendering, upscaled $2\times$ to 640×480 for the model; one frame per model call
Coordinates	0–1000 grid (Qwen family); absolute pixels of the resized image (UI-TARS native); calibrated per family
Delivery	20 ms fixed $D \to A$ in every model cell; uniform 0–40 ms in the jitter study only
Late binding	normalized cross-correlation, 25×25 template around the proposed point, newest available frame at D; restart movement threshold $\tau = 6$ pixels. Code Panel: fixed-font decode of the code panel in the newest available frame; restart when the decoded code changes
Seeds	Paused calibration 100–109; held-out 200–299 for Tasks A and B, 1000–1099 for Tasks C and D; development and held-out seeds disjoint
Windows and speeds	the Notification task: 1, 2, 4, 8, 16 s; Balloon Drift: 0, 1, 2, 4, 8, 16 pixels/s, 16 s window; Balloon Jump: 96 pixels jump at 2 s or 14 s, 16 s window; Balloon Select: 8 pixels/s, 16 s window; Chromium page: banner at 2 s or 14 s, 640×480, prompt 12, 16 s window; D: code replaced at 2 s, at 14 s, or never, 16 s window

Table 6: Configuration of the model studies. Every model cell in the main text uses the primary agent under this configuration.

Diagnostic condition	Failure exposed	Intervention tested	Tasks
Transient evidence before O	missed evidence	make the relevant observation available before expiry	A–B
Relevant change before D	outdated at commitment	use the newest relevant pre-D state or update a provisional field	B–D
Fixed action, closing window	late	deliver the same committed action earlier	A–B
Predictable change after D	invalidated after commitment	earlier receipt, receipt-state prediction, or both	B–C
Several conditions together	multiple labels	combine only the interventions required by the variant	A–D
Static or irrelevant change	negative control	leave the action and outcome unchanged	all

Table 7: Diagnostic conditions, the failure label each exposes, and the intervention designed for it; Figure 12 tests whether the interventions are selective.

mar tests on the discordant pairs; continuous outcomes use paired intervals and permutation tests. Resampling operates on task instances or templates, not frames. Development runs are not pooled with held-out results.

C.11 Scorer Validation

Hand-authored traces with known outcomes test valid actions, invalid actions, `wait`, timeout, no received action, late receipt, missed evidence, outdated commitment, post-commit invalidation, semantic error, actuation error, duplicate action, and unresolved-action status. A second minimal scorer recomputes the principal success rates, returns, and paired contrasts from immutable events.

D Additional Results

This appendix collects results that support the main text but are not needed to follow it.

	Paused		Replay		Live		Live	
Setting	SR	$\bar{G}$	SR	$\bar{G}$	SR	$\bar{G}$	$B \to D$ (s)	**Live labels**
Task Notification rendering ($\bar{G}$ in pixels)								
1 s	100/100	-2.2	100/100	-2.2	0/100	-400	5.12	late, outdated
2 s	100/100	-2.2	100/100	-2.2	0/100	-400	5.11	late, outdated
4 s	100/100	-2.2	100/100	-2.2	0/100	-400	5.12	late, outdated
8 s	100/100	-2.2	100/100	-2.2	100/100	-2.2	5.12	—
16 s	100/100	-2.2	100/100	-2.2	100/100	-2.2	5.11	—
Notification, Gemma-4-31B NVFP4 ($\bar{G}$ in pixels)								
1 s	100/100	-2.2	100/100	-2.2	0/100	-400	3.65	late, outdated
2 s	100/100	-2.2	100/100	-2.2	0/100	-400	3.37	late, outdated
4 s	100/100	-2.2	100/100	-2.2	100/100	-2.2	3.38	—
8 s	100/100	-2.2	100/100	-2.2	100/100	-2.2	3.36	—
16 s	100/100	-2.2	100/100	-2.2	100/100	-2.2	3.36	—
Notification, Agent S3: EvoCUA-8B planner, UI-TARS-1.5-7B grounder ($\bar{G}$ in pixels)								
1 s	96/100	-3.2	96/100	-3.2	0/100	-400	19.5	late, outdated; 4 aiming
2 s	96/100	-3.2	96/100	-3.2	0/100	-400	19.7	late, outdated; 4 aiming
4 s	96/100	-3.2	96/100	-3.2	0/100	-400	19.7	late, outdated; 4 aiming
8 s	96/100	-3.2	96/100	-3.2	0/100	-400	19.6	late, outdated; 4 aiming
16 s	96/100	-3.2	96/100	-3.2	29/100	-281	19.4	late 70, outdated 67; 4 aiming
Task Player ($\bar{G}$ in s)								
1 s	100/100	-1.0	0/100	-1.0	0/100	-1.0	—	outdated (`wait`)
2 s	100/100	-1.0	100/100	-1.0	0/100	-2.0	5.24	late, outdated
4 s	100/100	-1.0	100/100	-1.0	0/100	-4.0	5.24	late, outdated
8 s	100/100	-1.0	100/100	-1.0	100/100	-7.7	5.24	—
16 s	100/100	-1.0	100/100	-1.0	100/100	-7.7	5.24	—
Balloon ($\bar{G}$ in pixels)								
Balloon Drift 4 pixels/s	100/100	-2.1	100/100	-2.1	0/100	-21	5.12	outdated
Balloon Drift 8 pixels/s	100/100	-2.1	100/100	-2.1	0/100	-42	5.11	outdated
Balloon Drift 16 pixels/s	100/100	-2.2	100/100	-2.2	0/100	-82	5.11	outdated
Balloon Jump at 2 s	100/100	-2.0	100/100	-2.0	0/100	-97	5.11	outdated
Balloon Select, 8 pixels/s	100/100	-2.1	100/100	-2.1	0/100	-42	5.13	outdated
Task Trading ($\bar{G}$ in order utility)								
B, no reversal	100/100	$+1.0$	100/100	$+1.0$	100/100	$+1.0$	5.41	—
B, reversal 2.0 s after signal	100/100	$+1.0$	100/100	$+1.0$	0/100	-1.0	5.42	outdated
B, reversal 6.7 s after signal	100/100	$+1.0$	100/100	$+1.0$	0/100	-1.0	5.44	outdated
B, reversal 6.9 s, 0.5 s delivery	100/100	$+1.0$	100/100	$+1.0$	0/100	-1.0	5.44	mixed†
B, reversal 10 s after signal	100/100	$+1.0$	100/100	$+1.0$	100/100	$+1.0$	5.44	—

Table 8: Complete Paused, Replay, and Live outcomes on matched instances, one row per response window, 100 held-out seeds per cell (summarized in Table 2); the 27B model unless the block says otherwise. Agent S3 at windows of 0.53, 0.80, 1.06, 1.33, and 1.59 times its median deliberation (10.3, 15.5, 20.6, 25.8, and 30.9 s), Live only: 0, 20, 53, 79, and 89 of 100. SR: success at application receipt. $\bar{G}$: mean return at receipt; notification, click error in pixels (-400, the display diagonal, when no valid click arrives); player, seconds of sponsor watched (-1.0, the seek delay, at best; $-w$ when no skip lands in the w s window); The Balloon task, click error in pixels under the step-based controller; The Trading task, order utility ($+1$ valid, -1 invalid). $B \to D$: median deliberation of the Live click decision (none in the 1 s player row, §**??**). Live failures carry the listed labels in all 100 trials; outdated is *outdated at commitment*. †74 *invalidated after commitment*, 24 *missed evidence*, 2 *outdated at commitment* (§5.2). A 100/100 cell has Wilson interval $[0.96, 1.00]$, a 0/100 cell $[0.00, 0.04]$.

D.1 THE RESIDENT CORE ON THE SOFTWARE TRACK

Table 9 lists the cells of §D.1. The planner is a color detector with a 5.2 s wall-clock sleep after B (1.0 s in the short-deliberation rows), so every cell measures the loop and the binder rather than perception. The binder is the resident normalized cross-correlation with the parameters of Appendix C.8; the host binder is the NumPy implementation used in §**??**. Recorded intervals on this track are $E \to O$ 30.0 ms, $O \to B$ 5–7 ms, $B \to D$ 5,207–5,208 ms, and $D \to A$ 20.0 ms in every 5.2 s cell; the GPU was shared with a running model study, which the wall-clock intervals include. The device timestamp of a commit lies about 1 s after the harness's D because the runner renders the frames that fell due during the sleep after the planner returns; the harness does not charge that rendering to the controller, and the device does. Every residency check (the commit kernel ordered after the last admitted frame) passed in 100/100 trials of every resident cell.

Task and cell	Controller	SR	Mean error (pixels)	Labels
Notification, Live, window 1, 2, 4 s	resident late binding	0/100 each	—	late, outdated
Notification, Live, window 8, 16 s	resident late binding	100/100 each	—	—
Notification, Paused and Replay, all windows	resident late binding	100/100 each	—	—
Balloon Drift, 4 / 8 / 16 pixels/s	frozen coordinate	0/100 each	21 / 42 / 84	outdated
Balloon Drift, 4 / 8 / 16 pixels/s	resident late binding	100/100 each	1.0	—
Balloon Drift, 4 / 8 / 16 pixels/s	restart on update	0/100 each	—	late, outdated
Balloon Drift, 0 pixels/s and irrelevant motion	frozen; late binding	100/100 each	0.0	—
Balloon Jump, 96 pixels at 2 s (before D)	frozen; late binding	0/100; 100/100	96; 0.0	outdated; —
Balloon Jump, 96 pixels at 14 s (after A)	frozen; late binding	100/100; 100/100	0.0; 0.0	—
Balloon Select, 8 pixels/s	frozen; late binding	0/100; 99/100	42; 1.4	outdated; 1 outdated
Drift 200 pixels/s, 1.0 s deliberation, 1.3 s window	frozen	0/100	212	outdated
Drift 200 pixels/s, 1.0 s deliberation, 1.3 s window	host late binding	9/100	12.9	91 invalidated
Drift 200 pixels/s, 1.0 s deliberation, 1.3 s window	resident late binding	35/100	12.6	65 invalidated
Drift 100 pixels/s, 1.0 s deliberation, 1.3 s window	host; resident late binding	100/100; 100/100	5.9; 6.0	—

Table 9: The CUDA-resident control core on the software track, 100 held-out seeds (1000–1099) per cell, Live unless stated. The planner is a color detector with a fixed delay after B, not a model. Median binder time on the device is 1.2 ms per binding (about 40 μs of overhead when no binding is made). The 200 pixels/s rows sit on the hit boundary (§D.1): the two binders differ by about 1 ms of the 64 ms the half-width admits. Outdated is *outdated at commitment*; invalidated is *invalidated after commitment*.

D.2 THE DISTRIBUTIONAL GYM AND THE IN-PROCESS PLANNERS

Manifests. An episode is one row of a frozen manifest: environment seed, stimulus seed, task family, payload, deadline W_i, target speed and direction, update time, route condition, and the expected event and observation hashes. The file is hashed before any controller runs; every trace records the hash and is verified against the row after the trial. Deadlines are drawn log-uniform on $[1, 16]$s (an empirical window sample may replace the law); speeds for the drifting balloon log-uniform on $[1, 16]$ pixels/s at the default and on $[50, 400]$ pixels/s for the ceiling cells of Table 10, whose deadline is 1 s so that the target stays on the 640×480 canvas. Ten episodes per family in this pilot (seeds 5,000,000–5,000,009); Live, Replay, and Paused conditions run on each. Each family is reported separately; families are never pooled.

Scores. Live success is scored at receipt A as in §**??**. For the declared deadline law P_W, define

$$\mathrm{SR}_{F_W} = \frac{1}{n} \sum_{i=1}^{n} C_i R_i \, P_W\left(W > t_A^{(i)} - t_E^{(i)}\right), \tag{1}$$

where C_i marks a semantically correct received action and R_i marks receipt. This is the mean over Live trials of $C_i R_i \, P_W(W > E \to A)$, with P_W the manifest's deadline law; it is the expected success on a fresh draw of deadlines and is smooth in the agent's latency, where the observed success moves in steps of $1/n$. For the served agents of §5.2 the same quantity is computed from their Live traces at the 16 s window (Agent S3, 100 seeds: 96 clicks inside the target, $E \to A$ median 19.5 s, range 11.7–42.7 s; EvoCUA-8B native grammar, 10 seeds: 6 clicks inside, median 9.1 s), a trial contributing $P_W(W > E \to A)$ when its click landed inside the target and 0 otherwise; these are derived scores, not new runs. Wilson intervals of the observed success at $n = 10$ are $[0.72, 1.00]$ for 10/10, $[0.24, 0.76]$ for 5/10, and $[0.00, 0.28]$ for 0/10. Paired differences against the color-detector baseline, McNemar counts, retention (Live success among Replay-solvable episodes), and the per-family curves (W_{50}, W_{90}, v_{50}) are in `results/D1_*_pilot.md`.

The detector at D. The core's admission kernel computes the color detector's bounding box on every admitted frame (it already served the kind check of §**??**). Under late binding a rule chooses the committed coordinate from three candidates: the detector's center on the newest frame, the NCC-tracked template, and the original proposal. Rule 1 (detector-first) takes the detector when it sees the target, else the tracked template when it matches, else the proposal; rule 2 (NCC-first) takes the tracked template when it matches, else the detector, else the proposal. The binding taken (`detected_at_D`, `tracked_at_D`, or `state_at_B`), the match score, and the displacement from the proposal are recorded on every trace, and the residency audit is unchanged (no host copy in the window). "Planner at D" asks the same model again on the newest frame at commit and uses its answer as the coordinate.

Runtime. The three model planners are TensorRT Edge-LLM 0.10.0 engines (FP16 LLM and visual engines built from the checkpoints' ONNX exports) loaded once in the controller's process and called once per decision; the frame reaches the model as a PNG file, so this path removes the serving process and the socket but not the host copy of the frame (the audit counts it as a planner host step). Engine load is 3 s for the 0.8 B and 60 s for the 27 B model, once per process. The 0.8 B model answers a 640×480 frame in 240–260 ms; the 27 B in 2–3 s for `wait` and 5.3–6.4 s for a 20-token click. Two local patches to the export path (an FP16 cast where a traced activation arrives as float32, on the generic LLM path and in the Qwen2.5-VL visual encoder) are kept with the code.

The software track's frame path. On the 320×240 Notification canvas a published frame is admitted within 5 ms. On the 640×480 balloon canvas each frame's render, publication, and upload costs about 22 ms against the 16.7 ms period, so admissions lag publication by a median of 94 ms (maximum 187 ms) and the first frame showing the target is admitted about 180 ms after publication; this is the $O \to B$ column of Table 10. The core's decision and binding are unaffected.

A as an independent clock. One could ask whether the receipt A carries information that the decision D and the effect it causes (the next event on the E stream, e_4 in Figure 5) do not already carry, that is, whether one interval $D \to E$ would serve in place of $D \to A$ and $A \to E$. The two halves have different answers. The first half is fixed at 20 ms in every row of Table 10, so within the gym A is D plus a constant and nothing in these tables depends on stamping it; the only evidence that this half matters on its own is the delivery-delay intervention of §**??**, where the same order committed at the same D is valid or invalid according to a delay drawn on 0–40 ms. The second half is what the pending-effect task measures. Between A and the effect, 0.5–8 s here, the frames are identical to the frames before A: the card is present and not done. A controller whose state has only two values, not yet acted and effect seen, has no representation of this interval and re-decides on every frame inside it, which is the 17.9 duplicates per episode and 0/10 of the memoryless rows. The pending record is the third state, acted but not yet shown, and it is anchored at A rather than at D (a click may be dropped in transit) or at the effect (by then the duplicates have been sent). The ack-aware row locates the interval's end: it retires the record on an acknowledgement that arrives before the effect, and re-clicks on the six episodes whose delay exceeds its 3 s retry, so the state must persist until the effect itself is observed. The 0/10 to 10/10 contrast therefore measures whether the controller represents the interval $A \to E$ at all. It does not show that anchoring the record at A rather than at D matters: with delivery fixed at 20 ms and no dropped click in these runs, a record anchored at D would retire at the same effect and produce the same counts. Separating the two anchors requires variable or dropped delivery with the same memory rule under each anchor, which we have not run.

What is excluded. Cells in which the planner's duration is an inserted delay (the emulated deliberation of Table 9) are not part of these tables; every row is a real planner at its own speed. UI-TARS-1.5-7B in-process rows were not completed for this version: its visual encoder's ONNX export does not build as a TensorRT engine (a half-by-float matrix product in the Qwen2.5-VL visual path), so its in-process cells ran without a visual engine and every decision failed; they are excluded rather than reported as agent failures. The physical capture and input path (Appendix B) is not measured in this paper.

Controller	Runtime	Planner	Detector	Memory	Task	$E \to O$	$O \to B$	$B \to D$	$D \to A$	$E \to A$ p50/p95	SR_{F_W} 1–16	SR_{F_W} 1–64	W_{50} / v_{50}	Hits	$\bar{G}$
Chronotope	C++ process	Qwen3.5-0.8B	detector-first	clock table (rules off)	card	30	2	209	20	262/580	1.00	1.00	-	10/10	0.0
Chronotope	C++ process	Qwen3.5-0.8B	none	clock table (rules off)	card	30	2	208	20	260/584	0.00	0.00	-	0/10	-110.0
Chronotope	C++ process	Qwen3.5-0.8B	NCC template	clock table (rules off)	card	30	2	210	20	262/593	0.00	0.00	-	0/10	-110.0
Chronotope	core	Qwen3.5-0.8B	detector-first	clock table (rules off)	card	30	1	259	20	312/5684	0.94	0.96	-	10/10	0.0
Chronotope	core	Qwen3.5-0.8B	none	clock table (rules off)	card	30	1	255	20	308/3015	0.00	0.00	-	0/10	-110.0
Chronotope	core	Qwen3.5-0.8B	NCC template	clock table (rules off)	card	30	1	254	20	306/2945	0.00	0.00	-	0/10	-110.0
Chronotope	core	Qwen3.5-0.8B	NCC-first	clock table (rules off)	card	30	1	261	20	313/2887	0.76	0.77	0.08	8/10	-18.5
Chronotope	core	Qwen3.5-0.8B	Qwen3.5-0.8B at D	clock table (rules off)	card	30	2	553	20	615/3291	0.00	0.00	-	0/10	-140.5
Chronotope	core	Qwen3.8-27B	detector-first	clock table (rules off)	card	30	1	5477	20	5528/68161	0.34	0.53	6.18	5/10	-200.0
Chronotope	core	Qwen3.8-27B	none	clock table (rules off)	card	30	1	5561	20	5612/67893	0.34	0.53	6.18	5/10	-200.7
Chronotope	core	Qwen3.8-27B	NCC template	clock table (rules off)	card	30	1	5490	20	5541/69536	0.34	0.53	6.18	5/10	-200.7
Chronotope	core	Qwen3.8-27B	NCC-first	clock table (rules off)	card	30	1	5465	20	5516/73731	0.35	0.53	6.18	5/10	-200.0
Chronotope	core	Qwen3.8-27B	Qwen3.8-27B at D	clock table (rules off)	card	30	1	10543	20	10594/66160	0.18	0.42	8.91	3/10	-280.4
Chronotope	core	color detector	none	clock table (rules off)	card	30	5	7	20	62/63	1.00	1.00	-	10/10	0.0
Chronotope	core	color detector	NCC template	clock table (rules off)	card	30	5	8	20	63/64	1.00	1.00	-	10/10	0.0
Chronotope	core	Qwen3.5-0.8B	detector-first	clock table (rules off)	card_wide	30	4	561	20	619/5618	0.80	0.80	0.19	8/10	-80.0
Chronotope	core	Qwen3.5-0.8B	NCC template	clock table (rules off)	card_wide	30	1	565	20	619/2932	0.00	0.00	-	0/10	-141.8
Chronotope	core	color detector	none	clock table (rules off)	card_wide	30	3	4	20	59/62	1.00	1.00	-	10/10	0.0
Chronotope	core	color detector	NCC template	clock table (rules off)	card_wide	30	5	8	20	63/64	1.00	1.00	-	10/10	0.0
Chronotope	C++ process	Qwen3.5-0.8B	detector-first	clock table (rules off)	balloon 1–16	30	244	300	20	626/1370	-	-	-	10/10	-4.3
Chronotope	C++ process	Qwen3.5-0.8B	none	clock table (rules off)	balloon 1–16	30	214	264	20	528/1064	-	-	-	0/10	-152.9
Chronotope	C++ process	Qwen3.5-0.8B	NCC template	clock table (rules off)	balloon 1–16	30	214	251	20	516/983	-	-	-	0/10	-152.8
Chronotope	core	Qwen3.5-0.8B	detector-first	clock table (rules off)	balloon 1–16	30	170	374	20	594/3220	-	-	-	10/10	-4.0
Chronotope	core	Qwen3.5-0.8B	none	clock table (rules off)	balloon 1–16	30	168	374	20	586/3179	-	-	-	0/10	-80.2
Chronotope	core	Qwen3.5-0.8B	NCC template	clock table (rules off)	balloon 1–16	30	171	374	20	594/5796	-	-	-	0/10	-83.1
Chronotope	core	Qwen3.5-0.8B	NCC-first	clock table (rules off)	balloon 1–16	30	167	374	20	591/3149	-	-	27.49	9/10	-5.3
Chronotope	core	Qwen3.5-0.8B	Qwen3.5-0.8B at D	clock table (rules off)	balloon 1–16	30	168	747	20	961/3554	-	-	-	0/10	-70.1
Chronotope	core	Qwen3.8-27B	detector-first	clock table (rules off)	balloon 1–16	30	169	5331	20	5550/66811	-	-	15.60	9/10	-83.3
Chronotope	core	Qwen3.8-27B	none	clock table (rules off)	balloon 1–16	30	167	5289	20	5506/64870	-	-	2.41	4/10	-110.5
Chronotope	core	Qwen3.8-27B	NCC template	clock table (rules off)	balloon 1–16	30	166	5291	20	5508/74394	-	-	15.60	9/10	-82.8
Chronotope	core	Qwen3.8-27B	NCC-first	clock table (rules off)	balloon 1–16	30	167	5326	20	5542/64149	-	-	15.60	9/10	-82.8
Chronotope	core	Qwen3.8-27B	Qwen3.8-27B at D	clock table (rules off)	balloon 1–16	30	167	12669	20	12886/74924	-	-	2.19	4/10	-183.0
Chronotope	core	color detector	detector-first	clock table (rules off)	balloon 1–16	30	178	5	20	233/243	-	-	-	10/10	-4.0
Chronotope	core	color detector	none	clock table (rules off)	balloon 1–16	30	292	7	20	347/362	-	-	-	10/10	-4.1
Chronotope	core	color detector	NCC template	clock table (rules off)	balloon 1–16	30	297	16	20	363/373	-	-	-	10/10	-4.2
Chronotope	core	color detector	detector-first	clock table (rules off)	balloon 50–400	30	175	5	20	229/232	-	-	169.54	5/10	-11.4
Chronotope	core	color detector	none	clock table (rules off)	balloon 50–400	30	176	0	20	227/236	-	-	-	0/10	-40.9
Chronotope	core	color detector	NCC template	clock table (rules off)	balloon 50–400	30	183	5	20	242/282	-	-	169.54	5/10	-12.0
Chronotope	C++ process	color detector	NCC template	clock table (rules off)	pending	30	6	7	20	63/64	-	-	-	0/10 (dup 17.90/ep)	-4.1
Chronotope	C++ process	color detector	NCC template	clock table + pending	pending	30	5	7	20	62/63	-	-	-	10/10 (dup 0.00/ep)	-4.1
Chronotope	C++ process	color detector	NCC template	clock table + pending + ack	pending	30	5	7	20	62/64	-	-	-	4/10 (dup 0.70/ep)	-4.1
Chronotope	C++ process	color detector	NCC template	clock table + pending + stale 1 s	pending	30	5	7	20	62/64	-	-	-	10/10 (dup 0.00/ep)	-4.1
Chronotope	C++ process	Qwen3.5-0.8B	detector-first	clock table (rules off)	pending	30	2	207	20	259/571	-	-	-	0/10 (dup 16.20/ep)	-4.4
Chronotope	C++ process	Qwen3.5-0.8B	detector-first	clock table + pending	pending	30	2	211	20	263/576	-	-	-	10/10 (dup 0.00/ep)	-4.4
Chronotope	C++ process	Qwen3.5-0.8B	detector-first	clock table + pending + ack	pending	30	2	207	20	260/578	-	-	-	4/10 (dup 0.70/ep)	-4.4
Chronotope	C++ process	Qwen3.5-0.8B	detector-first	clock table + pending + stale 1 s	pending	30	2	206	20	259/574	-	-	-	10/10 (dup 0.00/ep)	-4.4
Chronotope	C++ process	Qwen3.5-0.8B	detector-first	clock table (rules off)	static	30	209	256	20	528/23943	-	-	-	9/10	-80.0
Chronotope	C++ process	Qwen3.5-0.8B	none	clock table (rules off)	static	30	199	248	20	498/23875	-	-	-	0/10	-209.6
Chronotope	C++ process	Qwen3.5-0.8B	NCC template	clock table (rules off)	static	30	213	252	20	512/24317	-	-	-	0/10	-209.6
Chronotope	core	Qwen3.5-0.8B	detector-first	clock table (rules off)	static	30	171	376	20	597/3659	-	-	-	10/10	0.0
Chronotope	core	Qwen3.5-0.8B	none	clock table (rules off)	static	30	168	376	20	592/3704	-	-	-	0/10	-68.7
Chronotope	core	Qwen3.5-0.8B	NCC template	clock table (rules off)	static	30	168	375	20	592/3179	-	-	-	0/10	-68.7
Chronotope	core	Qwen3.5-0.8B	NCC-first	clock table (rules off)	static	30	169	384	20	602/3082	-	-	-	8/10	-4.0
Chronotope	core	Qwen3.5-0.8B	Qwen3.5-0.8B at D	clock table (rules off)	static	30	167	751	20	969/3570	-	-	-	0/10	-68.7
Chronotope	core	Qwen3.8-27B	detector-first	clock table (rules off)	static	30	167	5329	20	5549/64335	-	-	-	10/10	-80.0
Chronotope	core	Qwen3.8-27B	none	clock table (rules off)	static	30	166	5299	20	5522/70905	-	-	-	10/10	-81.4
Chronotope	core	Qwen3.8-27B	NCC template	clock table (rules off)	static	30	168	5303	20	5521/66758	-	-	-	10/10	-81.4
Chronotope	core	Qwen3.8-27B	NCC-first	clock table (rules off)	static	30	167	5298	20	5519/66796	-	-	-	10/10	-81.4
Chronotope	core	Qwen3.8-27B	Qwen3.8-27B at D	clock table (rules off)	static	30	167	10686	20	10902/74594	-	-	-	10/10	-81.4
Chronotope	core	color detector	none	clock table (rules off)	static	30	285	7	20	340/360	-	-	-	10/10	0.0
Chronotope	core	color detector	NCC template	clock table (rules off)	static	30	287	16	20	353/370	-	-	-	10/10	0.0

Table 10: Every (controller, runtime, planner, detector, memory) row of the distributional gym pilot on every task family, ten frozen episodes each, Live. Runtime: "core" = the CUDA/C++ control core alone; "core+Edge-LLM (Py binding)" = the model called through Edge-LLM's Python binding with the frame as a PNG; "C++ process" = the agent as one process (`chronotope --serve`) with the model in-process on the device frame; "vLLM" = a served model. Detector: "detector-first" is detector, then tracker, and "NCC-first" is tracker, then detector, in the terms of §D.2. Median milliseconds; SR_{F_W} under the two declared deadline laws (card task); W_{50} or v_{50} from the logistic fit on the drawn deadlines or speeds (law-free); Hits at the manifest's drawn values (pending task: duplicates per episode in parentheses); $\bar{G}$ the mean return. "card" is Notification with the drawn deadline, "balloon 1–16" and "balloon 50–400" Balloon Drift with the speed drawn on those ranges, "static" the 0 pixels/s control, "pending" the pending-effect task with the drawn effect delay.

D.3 Notification and Player with the Model

The 27B model runs the sponsor task on a synthetic video player whose content pans before the sponsor starts, across the same windows as the notification rendering (Table 2; Paused gate 10/10; 100 held-out seeds per cell; 0 invalid trials). Median Live deliberation is 5.24 s, and the transition sits where the notification rendering places it, between the 4 s and 8 s windows; every Live failure at 2 and 4 s carries *late* and *outdated at commitment* (100/100 discordant Replay–Live pairs, exact $p = 1.6 \times 10^{-30}$). At the 1 s window Replay also fails, 0/100, with the label *outdated at commitment* on a committed `wait`: the panning content triggers a decision before the sponsor appears, that decision commits `wait` at 1.85 s after the sponsor has appeared, and at 1 s the sponsor is gone before a click decision can begin (three `wait` decisions, no commit). At 2 s and more the click decision begins at 1.85 s and commits at 7.1 s, so first receipt is 6.7 s after onset against 5.2 s on the notification rendering. The paired static-player run (the `player_static` rendering: the same player with nothing moving before the sponsor; 100 seeds, the same seeds; Paused gate 10/10) has one decision per Live trial, beginning at 0.53 s, against two (three at 1 s) beginning at 0.05 s on the animated player. Success is identical in all 15 cells; the Live return is -6.19 and -6.20s at 8 and 16 s against -7.72 s animated (paired $\Delta\bar{G} = +1.54$ and $+1.52$s), and Replay at 1 s is 100/100 against 0/100 (100 discordant pairs, exact $p = 1.6 \times 10^{-30}$). The cost of the wasted decision is therefore 1.5 s per trial, visible to the return and to Replay and invisible to the success rate.

D.4 Reversal Position and Label

the Trading task's simulator can place a signal reversal anywhere on the timeline. Figure 15 and Table 11 keep the signal, the frames, the decision, and the committed order fixed and move only the reversal. If the world changes before the agent commits, and the agent had already seen the change, the label is *outdated at commitment*; if the world changes between commitment and arrival, the label is *invalidated after commitment*; if it changes after arrival, nothing is wrong. The same order earns -1, -1, and $+1$. Lateness is not involved in any of the three; the window is open throughout.

Reversal	Relative to	Paused	Replay	Live	Live label
none	—	100/100	100/100	100/100	—
600 ms	before D, observed	100/100	100/100	0/100	outdated at commitment
750 ms	in $(D, A]$	100/100	100/100	0/100	invalidated after commitment
900 ms	after A	100/100	100/100	100/100	—

Table 11: the Trading task oracle results, 100 seeds per cell, utilities $(+1, -1, 0)$ for a valid order, an invalid order, and no order. Every Live outcome matches the closed-form prediction from the event times.

D.5 Notification Return

An oracle sweep on the player (100 seeds per cell; compute time 0.5–5.2 s; seek delay 50 ms) validates the return definition $G_A = -T_{\text{sponsor}}$: in Paused and Replay the return equals the seek delay whatever the compute time, because the sponsor waits; in Live every extra second of compute is a second more of sponsor watched, and once compute exceeds the sponsor length the skip never lands. The return therefore separates a late but valid skip from one that never arrives, which a success count cannot.

Controller	Updates the current action?	New model call?	Same B, D, A?
Initial coordinate	no	no	yes
Next-decision buffer	no	no	yes
Restart-on-update	through a new decision	yes	no
Late binding	the coordinate only	no	yes

Table 12: The four controllers of Balloon Drift; all begin from the same model call, and restart adds replacement calls. Initial coordinate is the same decision's proposal scored at the same A as the late-bound click (the same-decision counterfactual); next-decision buffering leaves the current action identical to it.

D.6 WHICH REMEDY IN WHICH INFORMATION REGIME

Regime	Cell	Prepared	Late binding	Predicted	Re-decision
Known before E	fixed-layout notification, 1 s window	100/100	0/100	0/100	0/100
Revealed before D	seeded notification, 8 s window	0/100	100/100	—	—
Revealed before D	code replaced at 2 s	0/100	100/100	—	—
Continues after D	balloon 8 pixels/s, 1.5 s route	0/100	0/100	100/100	0/100

Table 13: Scripted controllers on the simulated clock, 100 seeds per cell, 5.0 s of compute for every deliberating binding. *Prepared* constructs the action from the rendering's fixed geometry before E and releases it at the first frame showing the event ($E \to A$ 50 ms); it has nothing correct to prepare once the event reveals the content. Prediction alone survives change that continues through a slow route. With the 27B model in the same regimes (100 seeds per cell): on the fixed-layout notification at 1 s and 2 s windows the prepared action scores 100/100 against 0/100 for the deliberating model, and on the seeded notification at 8 s the prepared action scores 0/100 against 100/100 for late binding.

D.7 SLOW TARGETS AND THE RESTART THRESHOLD

Table 14 extends Figure 16 to 1 and 2 pixels/s. A target that moves less than its own half-width during deliberation is still hit by the first-frame coordinate, so late binding changes nothing there. Restart-on-update is a different story. At 2 pixels/s the movement detector fires, the controller discards a click that would have hit, calls the model again, and the target moves during every replacement call; 0/100 where doing nothing scores 100/100. A remedy triggered by motion can be worse than no remedy when the motion is harmless.

Raising restart's movement threshold moves this boundary and nothing else. With $\tau = 12$ pixels the detector no longer fires at 2 pixels/s (travel $\approx$10 pixels), and restart then scores 100/100 there because it has become the same controller as the initial coordinate; at 4 pixels/s it fires under both thresholds and scores 0/100 under both, after 2.0 replacement calls and a first commit at 17.4 s. Over the six restart cells, restart succeeds exactly where the initial coordinate succeeds and fails exactly where its detector fires, because under continuous drift the target moves again during every replacement call. The threshold chooses where the failure begins; it cannot make re-decision outrun the target.

Speed (pixels/s)	Travel in 5.3 s (pixels)	Initial	Restart ($\tau = 6$)	Restart ($\tau = 12$)	Late bind	Initial error (pixels)
1	≈ 5	100/100	100/100	100/100	100/100	6.0
2	≈ 11	100/100	0/100	100/100	100/100	10.9
4	≈ 21	0/100	0/100	0/100	100/100	20.9

Table 14: Balloon Drift results at slow speeds, 100 seeds per cell, same 27 B model and 24 pixels balloon as Figure 16. The balloon's half-width is 12 pixels. τ is restart's movement threshold in pixels; the main-text results use $\tau = 6$.

D.8 BALLOON SELECT COUNTS

Table 15 gives the Balloon Select counts summarized in §**??**; Figure 17 shows one of the two trials in which the stale coordinate landed on a distractor.

Controller	Named balloon chosen at B	Hit named at A	Hit a distractor	Hit nothing	First commit (s)
Initial coordinate	100/100	0/100	2/100	98/100	5.1
Late bind	100/100	99/100	0/100	1/100	5.1
Restart	—	0/100	0/100	100/100	16.9

Table 15: Balloon Select results for the 27B model, 100 seeds. The model chose the correct balloon in every trial; only its coordinate was out of date, and the binder recovered 99 of its 100 correct choices (99 discordant same-decision pairs, all favoring late binding; exact McNemar $p < 10^{-29}$). Restart's choice at B is not defined for a restarted decision. Paused calibration: the named balloon was clicked in 100/100 of 100 disjoint seeds.

D.9 Jump Position and Label

Figure 19 gives the agent counts behind §5.4. The oracle sweep of Figure 18 moves only the jump across B, D, and A with a scripted clicker (500 ms of compute, 60 ms of delivery) whose four bindings share B, D, and A exactly, 100 seeds per cell; every cell is 0/100 or 100/100.

D.10 Receipt-State Prediction on a Drifting Target

The experiments with a real model update the coordinate from the newest pre-commit frame and do not predict where the target will be at A. An oracle sweep on the moving-target task characterizes that fourth intervention (Figure 20): a scripted clicker with 300 ms of compute and 100 ms of delivery commits the same decision under four bindings that share B, D, and A exactly. Aiming at the state seen when deliberation began misses by the travel during the whole $B \to A$ interval; the newest state at D removes the deliberation part but not the delivery part, which at 120 pixels/s is 12 pixels of horizontal travel, enough to miss a 6 pixels target; a velocity extrapolator that is told the true delivery delay lands within 1 pixels; the oracle lands exactly. The predictor's advantage is conditional on knowing the delay. When its delay estimate is wrong by Δ, its error is $v\,|\Delta|$: at 120 pixels/s a 20 ms misestimate costs 2.4 pixels and still hits, a 40 ms misestimate costs 4.8 pixels and misses. Prediction therefore repairs post-commit change only to the extent that $D \to A$ is known, which is the operational reason §**??** insists on measuring it.

D.11 Which Label a Closing Window Produces

The three timing labels partition every oracle failure on the windowed-target task by where the closing falls relative to O, D, and A. Figure 21 (bottom) sweeps a scripted clicker's compute and delivery times against a 400 ms window. A decision that commits before the window closes but arrives after it is *invalidated after commitment*, and the boundary between that region and success is a diagonal because compute and delivery trade one for one; a decision that commits after the closing but before any frame depicting the closing could reach the controller is *missed evidence*; a decision that commits after such a frame has already been published is *outdated at commitment*. *Late* accompanies all three. The last two rows do not depend on delivery, because their failure is already decided at D. Every cell matches the closed-form prediction written independently of the harness.

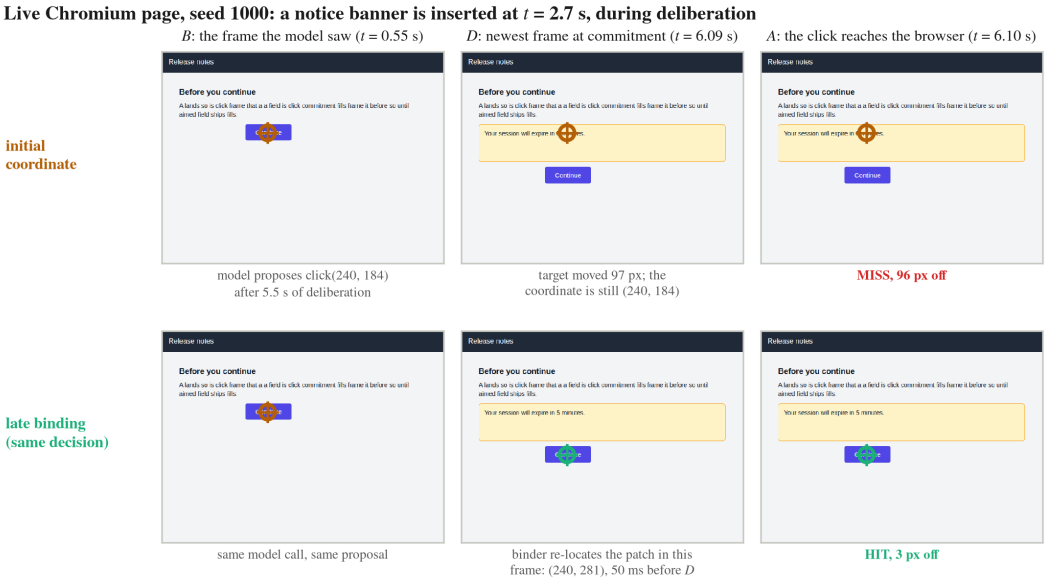


Figure 11: Live Chromium page, seed 1000: a notice banner is inserted at 2.7 s, during a 5.5 s deliberation, and displaces the Continue button by 97 pixels. Top: the proposed coordinate (240, 184) lands on the banner. Bottom: the same decision with late binding; the binder re-locates the button 50 ms before D and the click lands 3 pixels from its center. Frames are the 15 Hz screenshots taken through the observation boundary (§5.4).

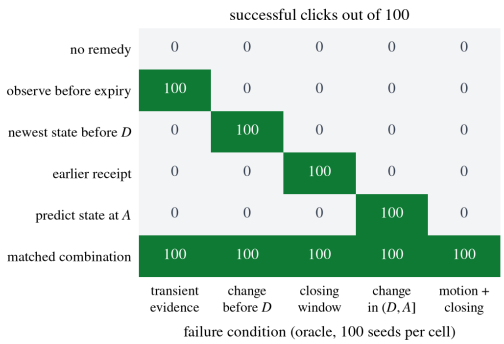


Figure 12: Interventions are selective. Rows are single interventions; columns are the failure conditions they were designed for; each cell counts successful clicks in 100 seeded Live episodes with a scripted controller whose only imperfection is the timing of its path. Each intervention repairs its own condition and none of the others; the combined condition needs both matching interventions. Static targets score 100/100 under every row (negative control, not shown).

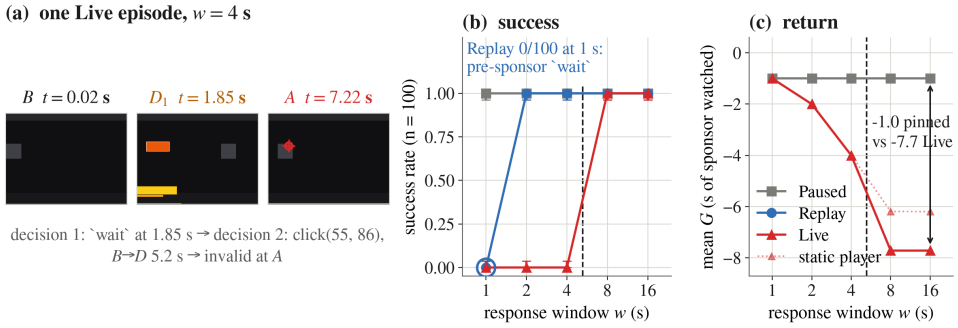


Figure 13: the Notification task on the video player, 27B model, 100 seeds per cell. (a) One recorded Live episode at the 4 s window (seed 200). The frame at B shows panning content and no sponsor; the first decision commits `wait` at 1.85 s (D_1), when the sponsor banner and its unlabeled skip button are already on screen; the second decision commits a click 5.2 s later, and at A (7.22 s) the sponsor has been gone since 4.5 s. (b) Success at receipt versus response window, Wilson 95% intervals; dashed line, the median deliberation of the click decision. Paused and Replay coincide except at 1 s, where Replay also fails: the sponsor leaves before the first `wait` returns. (c) Mean return, seconds of sponsor watched: pinned, the seek delay only; Live, the wasted decision plus the deliberation; dotted, Live on the paired static player, 1.5 s better.

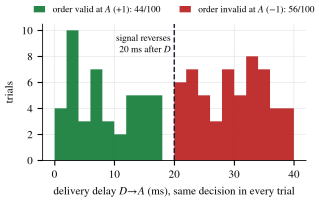


Figure 14: A delivery-delay intervention (§**??**): the Trading task, scripted controller, 100 seeds. Every trial commits the same valid order at the same D; the signal reverses 20 ms later; only the delivery delay varies (uniform 0–40 ms). Orders received before the reversal earn $+1$ (green), after it -1 (red); every trial falls where its measured delay predicts.

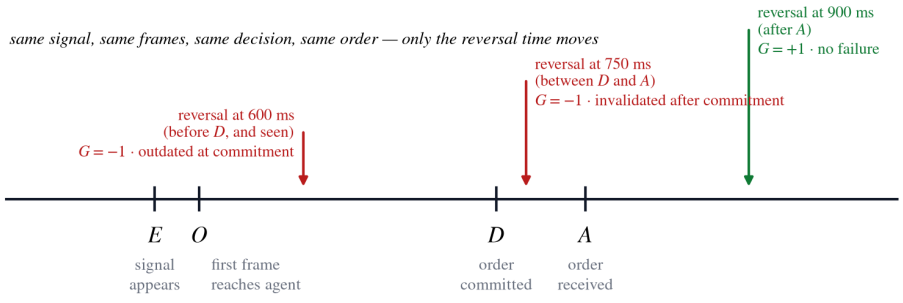


Figure 15: the Trading task reversal regimes on the four-event timeline (oracle controller, D at 730 ms, A at 790 ms). Three copies of the same trial differ only in when the signal reverses.

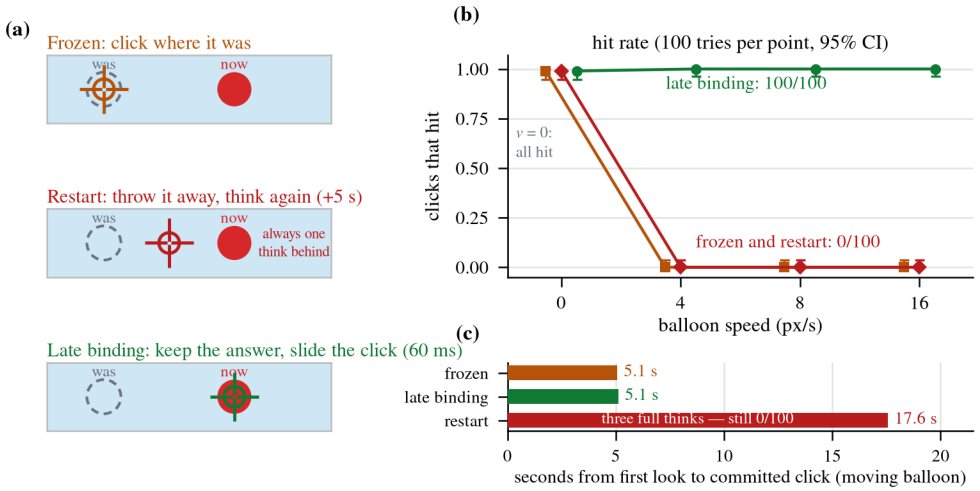


Figure 16: Balloon Drift controller comparison. (a) Initial coordinate commits the point proposed at O; restart begins another model call after movement; late binding keeps the decision and updates only (x, y) before D. (b) Hit rates over 100 trajectories per speed with 95% intervals; at each nonzero speed all 100 same-decision pairs favor late binding (exact McNemar, $p < 10^{-29}$), and a 100/100 cell bounds the true rate only to $[0.96, 1.00]$. (c) Under continuous drift, restart first commits after 17.6 s, compared with 5.1 s for late binding.

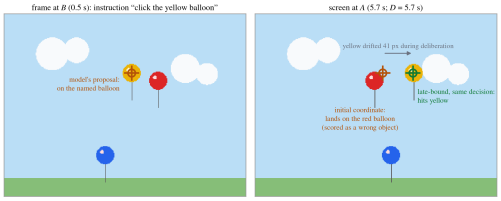


Figure 17: Balloon Select, seed 1010. Left: the frame at B; the model was told to click the yellow balloon and proposed a point on it. Right: the screen at A, 5.2 s later. The yellow balloon has drifted 41 pixels and the red distractor has crossed into its old position, so the initial coordinate lands on red and an outcome-only scorer would call it a wrong-object error. The late-bound click from the same decision hits yellow.

	jump before D		jump $(D, A]$	jump after A	
state at B (frozen)	miss	miss	miss	hit	hit
predicted at A (extrapolate)	miss	miss	miss	hit	hit
newest at D (late bind)	hit	hit	miss	hit	hit
true at A (oracle)	hit	hit	hit	hit	hit
	100 ms	300 ms	560 ms	900 ms	1400 ms
	label: outdated at commitment		*label: invalidated after commit*	*label: no label*	

Figure 18: Four bindings of the same committed click (shared B, D, A; scripted clicker; 100 seeds per cell) against the jump time; every cell is 0/100 or 100/100.

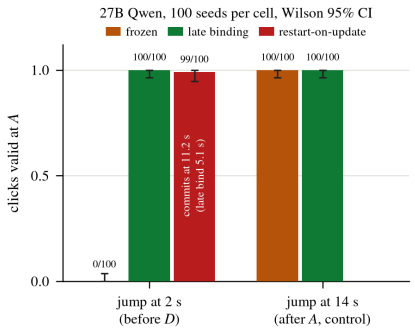
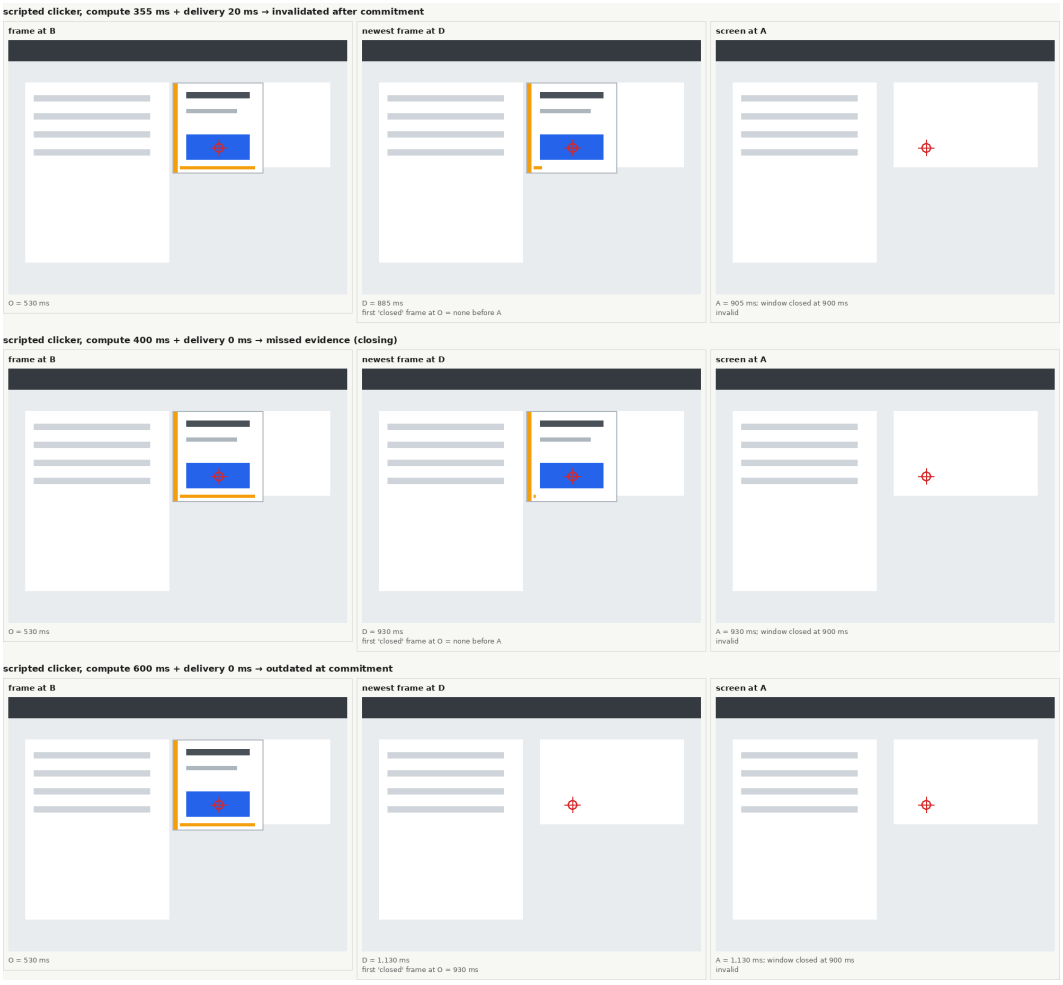


Figure 19: Balloon Jump, 27B model, 100 seeds per cell, success at receipt with Wilson 95% intervals. A jump before D defeats the initial coordinate but not the late-bound click of the same decision; restart-on-update also succeeds after a one-time jump but first commits at 11.2 s instead of 5.1 s. A jump after A affects no controller.



Figure 20: Four bindings of the committed coordinate on a balloon drifting at 120 pixels/s (and 20 pixels/s upward) with B, D, A matched (scripted clicker, 300 ms compute, 100 ms delivery). Left: the frame every binding saw at B. Then the screen at A with each binding's click: the state at B misses by 52 pixels, the newest state at D by 17 pixels, the predicted state at A by 1 pixels, the true state at A by 0 pixels.



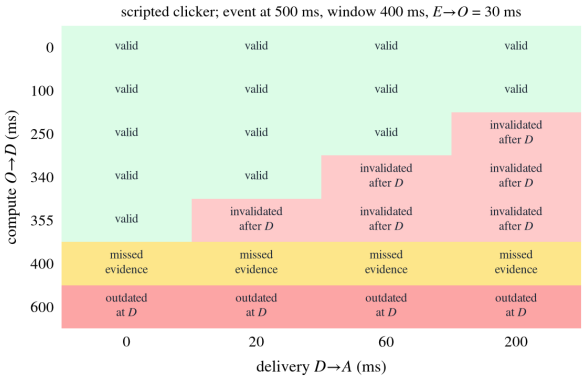


Figure 21: Top: the same closing, three labels. Each row is one Live cell of the scripted clicker with the window closing at 900 ms, shown as the frame at B, the newest frame at D, and the screen at A. Top: 355 ms of compute and 20 ms of delivery; the card is present at D and gone at A (*invalidated after commitment*). Middle: 400 ms of compute; the card was already gone at D, but no frame showing that had reached the controller (*missed evidence*). Bottom: 600 ms of compute; a frame showing the empty desktop was available before D and the fixed-input decision committed regardless (*outdated at commitment*). For this controller the three classes are exhaustive and exclusive. **Bottom:** Which label a closing window produces. A scripted clicker on the windowed target (event at 500 ms, 400 ms window, 60 Hz frames, 30 ms publication delay), swept over compute and delivery. Green cells arrive in time. Along the diagonal boundary compute and delivery trade one for one; below it the failure is already decided at D, so delivery no longer matters.

E AGENT QUALIFICATION

Every agent configuration is admitted to a timing study only after clicking the target under Paused execution in at least 8 of 10 calibration seeds on the rendering used in that study. Table 16 lists every configuration tried. The 27 B model passed on every rendering; the smaller general model answered `wait` on about half the frames that showed the target and was 13–25 pixels off when it clicked; the GUI-specialized model emits absolute pixels whatever the prompt asks (the mirror image of the Qwen family, which emits a 0–1000 grid whatever the prompt asks), so it was re-run in its own action grammar as the one declared alternative, and then passed on the balloon but read the synthetic desktop notification as a page still loading and waited. With a 64-token budget its native grammar was truncated before the action in every answer; the budget is therefore part of the controller specification. A timing comparison on an agent that cannot do the task while the world is paused would attribute its Live failures to time, which the protocol forbids; those configurations are reported here and used for no claim.

UI-TARS on the balloon. UI-TARS-1.5-7B was the first second model family to pass a gate, in its native grammar on the balloon; it was run on the Balloon Drift subset at 100 held-out seeds (1000–1099) with the initial coordinate and with late binding at 0 and 8 pixels/s (median $B \to D$ 6.0 s). On the static target both controllers score 70/100 with identical clicks; the 30 misses are its own aiming error (median 4.5 pixels, 90th percentile 16 pixels, on a 12 pixels half-width), and the binder moved no static click (100/100 zero translation). At 8 pixels/s the initial coordinate scores 0/100 (69 *outdated at commitment*, 31 aiming errors) and late binding 69/100; all 69 discordant same-decision pairs favor late binding (exact McNemar $p = 3.4 \times 10^{-21}$), and the frozen counterfactual agrees with an independent frozen run on every seed. Late binding therefore returns this agent to within one trial of its static-target accuracy; its ten-seed pilot (0/10 versus 6/10) is superseded, and its 9/10 gate result is a qualification check, not an estimate of that accuracy. It passes no windowed-target check, so it contributes no Paused–Replay–Live row.

Agent	Rendering	Grammar	Tokens	Paused gate	Failure mode	$B \to D$ (s)
Qwen3.8-27B	windowed target	JSON, 0–1000 grid	64	10/10; 100/100	—	5.1–5.3
Qwen3.8-27B NVFP4	windowed target	JSON, 0–1000 grid	64	10/10; 100/100	—	2.15–2.16
Qwen3.8-27B	balloon (Balloon)	JSON, 0–1000 grid	64	10/10; 99–100/100	—	5.1
Qwen3-VL-8B-Instruct	windowed target	JSON, 0–1000 grid	64	1/10	waits; 13–25 pixels off	1.7
Qwen3-VL-8B-Instruct	balloon	JSON, 0–1000 grid	64	1/10	same	1.7
UI-TARS-1.5-7B	windowed target	JSON, pixels	64	6/10	waits on 4/10	2.0
UI-TARS-1.5-7B	balloon	JSON, 0–1000 grid	64	0/10	answers pixels (≈67 pixels off)	2.0
UI-TARS-1.5-7B	windowed target	native	64	0/10	truncated before the action	—
UI-TARS-1.5-7B	windowed target	native	256	3/10	"page still loading", waits	6.2–7.1
UI-TARS-1.5-7B	balloon	native	256	9/10	aiming error 6.4 pixels mean	6.2
Gemma-4-31B NVFP4	windowed target	JSON, 0–1000 grid	64	10/10; 100/100	—	3.4
Agent S3 (EvoCUA-8B + UI-TARS-1.5-7B)	windowed target	native (pyautogui code)	64	10/10; 96/100	4 aiming errors	19.5
EvoCUA-8B	windowed target	native (tool call)	4096	0/10	answers `wait` on every first frame	7.6–9.9
EvoCUA-32B	windowed target	native (tool call)	4096	3/10	x within 2–7 pixels; y collapsed to mid-height	—

Table 16: Paused competence check for every agent configuration tried (threshold 0.80 over 10 calibration seeds; the 27B configuration was also calibrated on 100 seeds wherever the main text reports 100-seed cells). Only a configuration that passes contributes a timing claim on that rendering. Because each agent's Live transition sits at its own deliberation, agent ordering can differ between Paused and Live: Agent S3 is within four trials of the 27B model under Paused and 0/100 against 100/100 at the 8 s window under Live. The EvoCUA rows are the models in their own grammar and harness semantics (a first-step `wait` is never re-invoked on a static frame); they are reported as not passing in this harness, not as incompetence.