CUAWORLD-RT: CAN VISION-LANGUAGE AGENTS THINK AND ACT IN REAL TIME?

Anonymous authors
Paper under double-blind review

ABSTRACT

AI agents have made substantial progress in domains where they have a non-trivial amount of time to think between actions, such as mathematics, software engineering, and performing tasks on desktop applications. However, it is unclear how such models fare in settings that demand *real-time interaction*, such as playing a video game or operating a physical device. In these settings, the environment state may evolve as the agent is reasoning, or even as the underlying model is performing a forward pass. To gain insight into the real-time interaction capabilities of vision-language agents, we construct CUAWorld-Realtime (RT). Our key idea is to construct *visual minigames* that demand a portfolio of capabilities required for interactive real-time settings. We build an agentic pipeline for automatically generating such environments based on a taxonomy of six capabilities required for real-world and real-time interaction. CUAWORLD-RT comprises 100 minigame families and 1,000 configurations that vary across task difficulty and the action interface. A key feature of CUAWorld-RT is that it allows for controlled comparisons across different capabilities, such as real-time interaction, temporal reasoning and exploration of the action interface. Using CUAWORLD-RT, we evaluate frontier and open-weight models. Even GPT-6 Astra solves only 12.5% of real-time tasks at the highest difficulty, while being extremely slow. We also find that models' strong coding capabilities can partially offset their weak real-time capabilities, although their performance remains significantly low. We will publicly release CUAWORLD-RT and our environment generation framework to spur further research on AI agents in real-time interactive settings.[1]

1 INTRODUCTION

Vision-language agents, such as computer-use agents (CUAs), have made rapid progress in operating digital software through screenshots, mouse actions, and keyboard input (Qin et al., 2025). They have shown super-human performance across various challenging long-horizon tasks (Yuan et al., 2026; Aggarwal et al., 2026; Jang et al., 2026). However, current agents are still extremely slow, and it remains unclear if they can operate in real-time environments such as playing digital games or operating physical interfaces in the real world (Black et al., 2025). For example, current agents often take more than ten seconds to choose a single action. In a video game or on a physical interface, the environment keeps changing during this time, so the action may no longer be valid or may even be harmful when it executes, and the agent cannot react to events that occurred in the meantime. Beyond real-time interaction, such settings also require capabilities such as 3D spatial understanding and temporal reasoning (Yang et al., 2025a; Liu et al., 2024), exploring unfamiliar controls, and planning (Aggarwal & Welleck, 2025; Shridhar et al., 2020). However, it remains unclear which of these capabilities limit performance during real-time interaction.

Existing evaluations give sharply different answers, where computer-use benchmarks suggest that frontier CUAs can perform well in long-horizon tasks (Yuan et al., 2026; Aggarwal et al., 2026; Jang et al., 2026), while the same models make little progress in interactive environments such as video games (Zhang et al., 2026; Ying et al., 2026; Ouyang et al., 2026). Since each game combines many of these capabilities in a single fixed task, such results cannot show which capability

[1] Code and data are available at `https://anonymous.4open.science/r/cuaworld-rt-iclr-FDB0`.

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

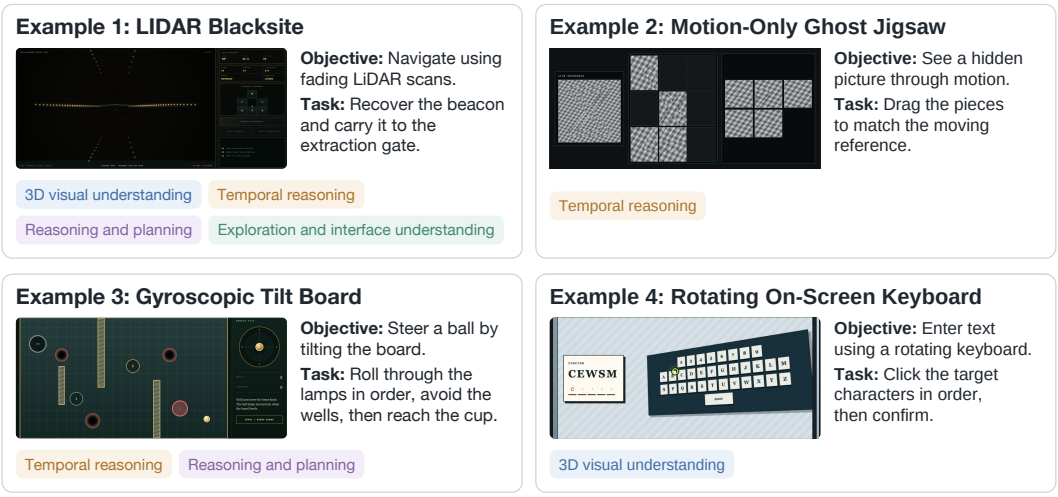


Figure 1: **Examples from CUAWORLD-RT, a benchmark for studying real-time interaction.** Its 100 visual minigame environments cover different combinations of capabilities, with controlled variations in difficulty and interaction.

is missing. To address this, we develop a scalable method for programmatically generating interactive real-time visual minigames with controlled capability requirements, which we call CUAWORLD-RT (Real-Time). These minigames retain the visual, temporal, and action demands of real-time and physical interfaces while avoiding the complexity of complete games. Our key idea is that since each minigame is implemented as a program, we can control the clock and adjust difficulty, interaction, and required capabilities, isolating which capabilities limit an agent acting in real time.

We first conduct an extensive survey of prior work from the embodied literature (Savva et al., 2019; Srivastava et al., 2021; Yang et al., 2025c), video games (Zhang et al., 2026), and computer use (Cheng et al., 2024; Xie et al., 2024) to identify six fundamental capabilities necessary for succeeding in such environments. Using this taxonomy, we select minigames whose solutions require identifiable combinations of these capabilities and that collectively provide broad coverage. We further require each minigame to have a traceable seed source, distinct visual and interaction requirements, an objectively gradable outcome, and meaningful mouse or keyboard interactions.

At the core of our generation method is a creation–audit pipeline (Aggarwal et al., 2026) that turns each selected minigame into an executable program. For each minigame, a creator agent writes the complete implementation, constructs an executable oracle solution using the minigame's internal state, and provides visual and temporal evidence that the minigame behaves as intended and is solvable. Each program defines a minigame family with 10 controlled configurations per seed: five task-specific difficulty levels and two interaction modes that realize the same intended task through simplified or full controls. Independent audit agents verify the program, evidence, oracle solution, and controlled settings, and return feedback until they satisfy the intended functional, visual, temporal, interaction, and difficulty requirements. Additional review agents then annotate the capabilities required by each configuration, allowing us to measure how performance changes across capability requirements, difficulty, interaction, and real-time interaction.

Using this pipeline, we construct CUAWORLD-RT, a benchmark of 100 minigame families and 1,000 controlled configurations spanning six capabilities. Current agents perform poorly on CUAWORLD-RT, with even GPT-6 Astra solving only 12.5% of real-time tasks at the highest difficulty. Agents struggle with 3D spatial understanding, temporal reasoning, and exploration of the interface. Pausing the environment while the agent responds improves the pass rate on real-time configurations from 8.8% to 32.5%, showing that slow responses are a key source of failure. Similarly, simplifying the controls or providing interface rules upfront improves performance, indicating that agents also struggle to act through unfamiliar interfaces. Together, these results suggest that current agents fail in real time both because they lack key capabilities and because they respond too slowly to act on what they observe.

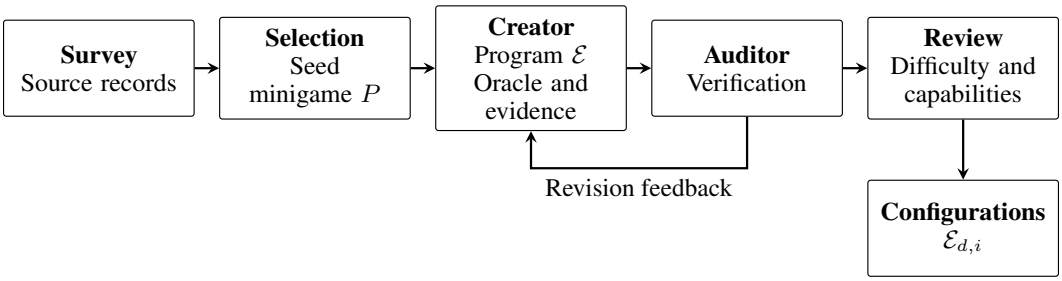


Figure 2: **Overview of our construction method.** A creator agent implements selected minigames with controllable difficulty and interaction, constructs oracle solutions, and records evidence of their behavior. An independent auditor checks the implementation and evidence, returning issues for revision until the minigame passes the audit. Review agents calibrate difficulty and annotate capability requirements for controlled model evaluation.

In summary, our contributions are three-fold. First, we develop a scalable method for programmatically constructing interactive visual minigames with executable oracle solutions and control over real-time interaction, several capability requirements, difficulty, and interaction. Second, we introduce CUAWORLD-RT, a benchmark of 100 minigame families and 1,000 configurations spanning six capabilities. Third, through extensive evaluations, we show that current state-of-the-art agents achieve low success rates while incurring high inference costs and long completion times, identifying temporal reasoning, interface exploration and interaction, and real-time interaction as key limitations.

2 METHODOLOGY

To identify which capabilities limit an agent acting in real time, we need tasks in which the required capabilities, the action interface, and the flow of time can each be controlled while the underlying task stays fixed. We therefore implement each selected seed minigame P as an executable environment $\mathcal{E}$ whose program defines its clock, internal state, and controls. Each environment has ten configurations $\mathcal{E}_{d,i}$, where $d \in \{1, \ldots, 5\}$ specifies the difficulty and $i \in \{\text{simplified}, \text{full}\}$ specifies the interaction mode, and each configuration can be run live or paused (Section 2.4). Since a broken or unsolvable minigame would be counted as an agent failure, every configuration must also be solvable, objectively gradable, and consistent with the other configurations of its minigame. We construct these environments with separate agent roles: survey agents collect candidate minigames, creator agents implement them, audit agents verify them, and review agents calibrate difficulty and annotate capabilities. Figure 2 summarizes the construction and review procedure.

2.1 TASK DISCOVERY AND SELECTION

We prompt survey agents to search prior work, public demonstrations, visual puzzles, online videos, and social media for challenges that satisfy four criteria: they can be formulated as self-contained minigames, require visual understanding and grounding, require actions through a visual interface, and have objectively verifiable outcomes. These include tasks that involve moving objects, hidden information, physical manipulation, or changes that occur between actions. For every source, the survey agent records its provenance, objective, visual structure, required interactions, temporal behavior, failure and recovery process, and information needed to verify success.

We manually review the first 15 minigames and use them as quality standards for subsequent agent iterations. We select candidates with a traceable source, distinct visual and interaction requirements, an objectively gradable outcome, and meaningful mouse or keyboard actions. We also require the selected set to cover different combinations of the six capabilities studied in our evaluation. We exclude tasks dominated by OCR, static image classification, arithmetic, or question answering, since these tasks do not require the forms of visual interaction that we seek to measure.

2.2 CREATOR–AUDITOR CONSTRUCTION

To construct $\mathcal{E}$ from P, we adapt the creator–auditor procedure from Gym-Anything (Aggarwal et al., 2026). The creator writes a program that implements the minigame, supports changes to difficulty and interaction, and exposes a common interface for screenshot observations and mouse and keyboard actions.

The creator implements the generator, interface, dynamics, action handling, grader, and verifier for $\mathcal{E}$. For example, Rotating On-Screen Keyboard (Figure 3) allows the code to be typed in simplified interaction and requires clicks on the rotating keys in full interaction. The creator also constructs an oracle solver that uses the privileged state of each object inside the environment and the codebase of the environment to determine a valid solution and executes that solution through the same action space available during evaluation. A successful oracle execution therefore demonstrates that the generated task is reachable and that its action, grading, and verification paths agree. The creator records the implementation, oracle trajectory, command outputs, and visual and temporal evidence needed to inspect these claims (Appendix B.1).

Each implementation is then audited by a fresh audit agent that cannot modify the program. The audit agent inspects the code and recorded evidence, runs the environment when necessary, and checks generation, solvability, rendering, action handling, grading, failure, and recovery. It also verifies that difficulty configurations create meaningful changes and that different variants preserve the same generated problem. The audit agent returns unsupported claims and issues to the creator for revision, and the loop repeats until an audit passes.

2.3 CONTROLLED TASK VARIATIONS

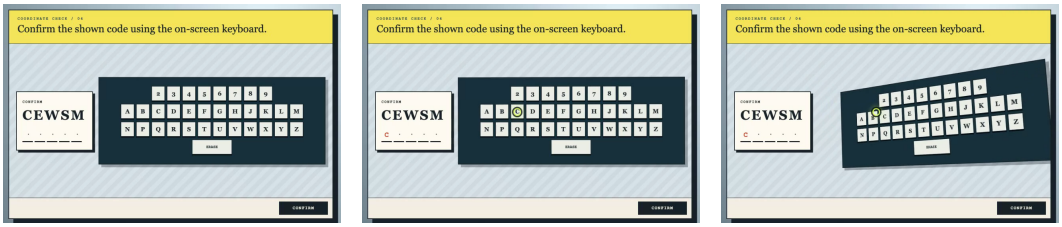
We vary d, i, or the real-time setting while holding the others and the random seed fixed to measure the effect of each.

A difficulty-review agent first reads the generator, visible interface, grader, verifier, and oracle solver to determine what the original minigame requires. It compares this configuration with an approved set of reference minigames and assigns it to one of five levels. The creator coding agent preserves the original configuration at that level and constructs the remaining levels around it by changing the amount of information, number of dependencies, speed, precision, dynamics, or simultaneous control required for success. A difficulty-review agent then checks each adjacent pair and rejects any spurious difficulty increase, such as requesting repeated actions or adding ambiguity. The agent also ensures that the same types of perception, memory, decisions, and control are required across difficulty levels. For example, the levels of Rotating On-Screen Keyboard (Figure 3) vary the code length, the number and size of keys, and the keyboard motion.

The creator coding agent also implements simplified and full interaction for every difficulty level. Simplified interaction exposes proxy controls. For example, instead of dragging the mouse in a 3D scene to navigate, the model could use three buttons for rotation about different axes. Full interaction requires the direct mouse or keyboard manipulation appropriate to the minigame. For a fixed seed and difficulty, the audit agent verifies that both modes generate the same minigame and differ only in the controls used to act on it. Appendices B.2 and B.3 give examples of both variations.

2.4 EVALUATING REAL-TIME INTERACTION

We evaluate each configuration $\mathcal{E}_{d,i}$ in two settings to study how model response time affects real-time interaction. During *live* execution, the task continues to change from observation through action execution at the game's natural speed. In Rotating On-Screen Keyboard, for example, a key can move between observation and action, so the model must account for its position when the click executes. To minimize the delay after a model issues an action, we adapt Gym-Anything (Aggarwal et al., 2026) to send requests directly to the sandbox, with median request-to-response times below 50 ms in our timing tests (Appendix E.1). During *paused* execution, the task advances through an observation window and then freezes on the final frame until the action is executed. Both settings use the same minigame, seed, difficulty, interaction mode, and verifier. Appendix E describes the evaluation protocol.

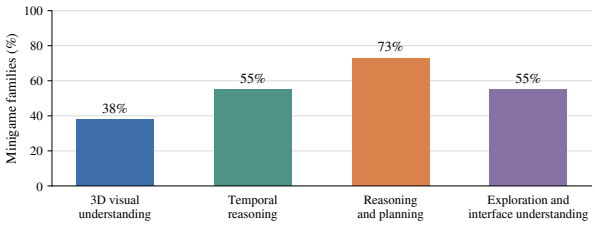


(a) The code and keyboard before the first action.

(b) The first key is selected.

(c) The keyboard moves before the next click.

Figure 3: **Rotating On-Screen Keyboard.** The model must enter a code by clicking keys that move between observation and action.



(a) Capability coverage.

Statistic	Count
Minigame families	100
Difficulty levels	5
Interaction modes	2
Difficulty–interaction configurations	1,000
Configurations requiring real-time operation	246
Configurations not requiring real-time operation	754

(b) Benchmark statistics.

Figure 4: **Overview of CUAWORLD-RT. (a)** Percentage of minigame families requiring each capability. **(b)** Number of minigame families and configurations.

2.5 ADDITIONAL CAPABILITIES REVIEW

To relate agent failures to the capabilities a minigame requires, a capability-review agent labels each configuration $\mathcal{E}_{d,i}$ for 3D visual understanding, temporal reasoning, reasoning and planning, and exploration and interface understanding (see Appendix D for definitions). It reviews the game's program, interface, verifier, oracle solution, and recorded behavior to identify what a solution using screenshots and mouse or keyboard actions requires. Each label is supported by code or behavioral evidence.

We consider a configuration real-time when the agent must use recent visual evidence to choose an action for the state in which it will execute, and changes during the observation-to-action delay can reduce that action's value. A real-time review agent checks this requirement separately for each difficulty and interaction configuration using the implementation and recorded behavior. Appendix H gives the full definition and review prompts.

3 CUAWORLD-RT

We use our method to construct CUAWORLD-RT, comprising 100 interactive visual minigame environments. Each environment has five difficulty levels and two interaction modes, giving 1,000 configurations $\mathcal{E}_{d,i}$. We evaluate models on these configurations using the two real-time settings defined in Section 2.4.

3.1 CAPABILITY COVERAGE

To balance coverage of the four task-dependent capabilities, we select 100 of the 200 candidate minigame families by minimizing the standard deviation of the proportions requiring each capability. Of the selected families, 38% require 3D visual understanding, 55% require temporal reasoning, 73% require reasoning and planning, and 55% require exploration and interface understanding (Figure 4). Of the resulting 1,000 difficulty–interaction configurations, 246 (24.6%) require real-time operation. Table 6 lists all 100 minigames and their capability labels.

Table 1: **Model performance on CUAWORLD-RT with full interaction in the live setting.** Even the strongest model performs poorly on tasks requiring real-time interaction.

	All difficulties			Difficulty 5		
Model	**Real-time**	**Non-real-time**	**Average**	**Real-time**	**Non-real-time**	**Average**
Qwen3.5-9B	1.5	0.5	0.8	3.1	0.0	1.0
Qwen3.8-27B	3.0	4.3	4.0	0.0	1.5	1.0
GPT-5.6 Luna (xHigh)	**12.1**	**33.4**	**27.8**	0.0	17.6	12.0
GPT-6 Astra (Low)	—	—	—	**12.5**	**66.2**	**49.0**

Table 2: **Effect of coding on real-time performance.** Astra benefits most from coding on tasks requiring real-time interaction.

	Real-time		Non-real-time	
Model	**Coding**	**Non-coding**	**Coding**	**Non-coding**
GPT-6 Astra (Low)	34.4	12.5	72.1	66.2
GPT-5.6 Luna (xHigh)	0.0	0.0	22.1	17.6

Table 3: **Efficiency on the real-time configurations at difficulty 5.** All models are slower and less successful than the human author.

Agent	**Pass rate ↑ (%)**	**Avg. time ↓ (min)**	**Avg. cost ↓ ($)**
Qwen3.5-9B	3.1	81.5	0.91
Qwen3.8-27B	0.0	144.8	10.78
GPT-5.6 Luna	0.0	25.2	0.36
GPT-6 Astra	12.5	11.7	6.96
Human	78.1	3.62	—

3.2 Evaluation Protocol

We evaluate models by pass rate, time to solve, and inference cost, and compare these metrics across difficulty levels, interaction modes, capability requirements, and the real-time settings defined in Section 2.4. Models receive task instructions and interact through screenshots, mouse actions, and keyboard input, without access to program internals or oracle solutions. Task-specific verifiers determine whether each attempt succeeds. Appendix F describes the models, prompts, budgets, and cost computation.

4 Results and Analysis

4.1 Experimental Setup

We evaluate Qwen3.5-9B, Qwen3.8-27B, GPT-5.6 Luna with xHigh reasoning effort, and GPT-6 Astra with low reasoning effort. Our primary evaluations use full interaction in the live setting across all five difficulty levels, with Astra evaluated at difficulty 5. Following prior work on computer-use agents (Xie et al., 2024; Aggarwal & Welleck, 2025), models observe the environment only through screenshots and act only through mouse and keyboard actions. Models cannot run code or access the page source or program internals. In the coding experiment, we additionally allow models to run code in an isolated sandbox, where they can request screenshots and execute the same mouse and keyboard actions through an API. Interaction with the environment remains the same. Further experiments vary difficulty, interaction mode, the real-time setting, and whether interface rules are provided.

4.2 Models Remain Poor Interactive Agents

Table 1 compares the pass rates of four models, separately for configurations that require real-time interaction and those that do not. Across all difficulty levels, GPT-5.6 Luna achieves a pass rate of

27.8%, compared with 4.0% for Qwen3.8-27B and 0.8% for Qwen3.5-9B. At the highest difficulty, Astra solves 49.0% of the tasks and Luna solves 12.0%, while both Qwen models achieve 1.0%. These results show that current models struggle to solve most tasks, with particularly low performance from the open-weight models.

Performance is lowest on configurations that require real-time interaction. At difficulty 5, Astra solves 66.2% of the non-real-time configurations but only 12.5% of the real-time configurations, and no other model solves more than 3.1% of the real-time configurations. Luna shows the same gap across all difficulty levels, solving 12.1% of the real-time configurations compared with 33.4% of the others, and it solves none of the real-time configurations at difficulty 5. Qwen3.5-9B and Qwen3.8-27B remain below 5% in both groups, so the real-time requirement makes little difference for models that already fail most tasks. Thus, even the strongest model, which solves two-thirds of the non-real-time configurations at difficulty 5, fails on most configurations in which the environment changes while it responds. In the following sections, we examine the effect of real-time operation and then investigate the capabilities and interaction requirements that limit performance.

4.3 Real-Time Interaction Remains Difficult

We next examine whether the effect of pausing depends on a task's real-time requirement (Figure 5). We compare live and paused execution with full interaction, averaging across difficulty levels 1, 3, and 5. On configurations requiring real-time interaction, pausing increases Luna's pass rate from 8.8% to 32.5%. On configurations without this requirement, pass rate also increases from 37.1% to 44.9%. The improvement in both groups shows that the benefit of pausing extends beyond configurations requiring real-time interaction. Even with the environment paused, Luna solves few real-time configurations, suggesting that response speed is only one source of difficulty.

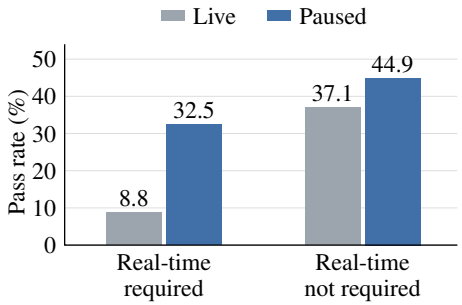


Figure 5: **Effect of pausing on models.** Pausing improves performance in both groups, with lower pass rates on tasks requiring real-time interaction.

4.4 Strong Coding Capabilities Can Partially Offset Weak Real-Time Capabilities

We next examine whether models can use code to overcome some of the difficulties of real-time interaction. The action space remains the same: keyboard and mouse inputs. Models receive API access to request screenshots and execute actions, allowing them to write scripts that automate these interactions (Appendix F.5). Table 2 shows results for GPT-6 Astra and GPT-5.6 Luna at the highest difficulty with full interaction under live execution. For Astra, coding increases the real-time pass rate from 12.5% to 34.4%, a nearly threefold increase, compared with a gain of 5.9 percentage points on non-real-time tasks. Luna shows smaller gains, with no real-time tasks solved in either condition.

A qualitative analysis of the trajectories shows that Astra uses image-processing scripts to process screenshots and execute actions between model responses, partially compensating for its slow reaction time. For instance, in Rotating On-Screen Keyboard, GPT-6 Astra uses image template matching to locate and click the requested keys from screenshots, producing a reaction time of less than 200 ms. In Clockwork Clutch Safe, where moving gears must be aligned, it tracks gear markers and times clutch releases using estimates of their motion (see Appendix G for more examples). However, we do not see any successful use of code scripts by GPT-5.6-Luna, likely because of its weaker coding capabilities. We posit that similar techniques could help models compensate for slow reactions in other real-time domains, including video games and physical interfaces. Future work could explore how models can construct and adapt such scripts across environments. Nonetheless, performance remains below human levels, and such scripts may be difficult to construct when the relevant visual information cannot be extracted reliably or cannot be prepared for ahead of time.

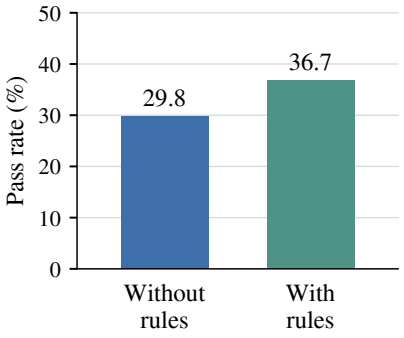


Figure 7: **Effect of providing interface rules.** Luna solves more tasks when given the rules.

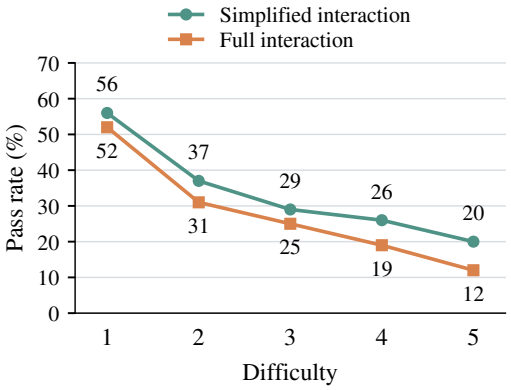


Figure 8: **Effect of difficulty and interaction.** Luna's pass rate decreases with difficulty and increases with simplified interaction.

4.5 Models Are Slow and Inefficient

Table 3 reports the pass rate, time, and inference cost on the 32 real-time configurations at difficulty 5 with full interaction in the live setting. Astra achieves the highest pass rate among the models, 12.5%, but still takes 11.7 minutes and costs $6.96 per attempt on average. Luna takes 25.2 minutes at a cost of $0.36 and solves none of these configurations. The Qwen models take 81.5 and 144.8 minutes per attempt while solving at most 3.1% of them. As a case study, we compare these results with those of a human author who, like the evaluated models, sees each minigame for the first time. The human pass rate is 78.1%, with an average of 3.62 minutes per task (Appendix F.7). The long completion times highlight the importance of improving model speed alongside pass rate for real-time applications (Abhyankar et al., 2026; Black et al., 2025).

4.6 Performance Depends on Capabilities

Figure 6 compares Luna's performance on configurations that require each capability with those that do not, using simplified interaction and paused execution across all five difficulty levels. Pass rates are lower on configurations requiring 3D visual understanding, temporal reasoning, or exploration and interface understanding, while configurations requiring reasoning and planning have a higher pass rate (54.1% versus 46.8%). The largest gap is for exploration and interface understanding, with pass rates of 42.2% and 56.9% with and without this requirement, compared with gaps of 9.1 and 7.4 percentage points for 3D visual understanding and temporal reasoning. When we separate configurations by their real-time requirement, the exploration gap persists within both groups, while the temporal-reasoning gap largely disappears among non-real-time configurations (54.5% versus 54.3%). Since capability requirements overlap, we further examine interface exploration through the controlled experiment in Section 4.7.

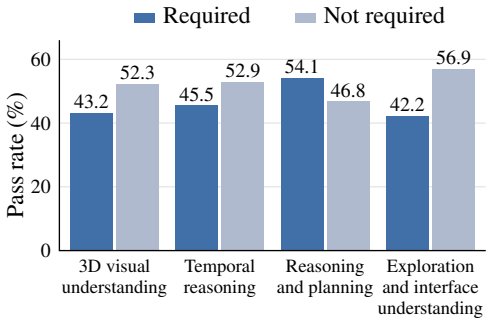


Figure 6: **Performance across capability requirements.** With simplified interaction and paused execution, Luna performs worse on configurations requiring 3D visual understanding, temporal reasoning, or exploration and interface understanding, with the largest gap for exploration.

4.7 Providing Interface Rules Improves Performance

Figure 7 compares Luna's performance with and without rules that it would otherwise need to discover through interaction. We append an environment-specific rulebook describing the controls, action effects, and world rules to Luna's task prompt (Appendix F.6), and evaluate with full interaction

under live execution on environments requiring exploration and interface understanding. Across the five difficulty levels, providing these rules increases pass rate from 29.8% to 36.7%. The model must still solve the generated minigame and carry out its solution through the same controls. This improvement suggests that Luna can use interface knowledge that it struggles to acquire through interaction, identifying interface exploration as a source of difficulty.

4.8 Performance Decreases with Difficulty and Improves with Simplified Interaction

An advantage of our method is that we can vary a minigame's difficulty to study how model performance changes as its demands increase. Figure 8 reports Luna's performance across the five difficulty levels under live execution. Averaged across both interaction modes, pass rate decreases from 54.0% at difficulty 1 to 16.0% at difficulty 5. Simplified interaction improves performance at every difficulty level, increasing the overall pass rate from 27.8% to 33.6%. Because the underlying minigames are unchanged, this improvement shows that some failures arise from using the full controls, even when Luna can solve the task through simplified interaction. The decline under both interaction modes shows that the challenges introduced at higher difficulty levels persist when the controls are simplified.

5 Related Work

Benchmarks for interactive visual agents. Prior benchmarks evaluate agents on complete tasks in software (Xie et al., 2024; Aggarwal & Welleck, 2025; Aggarwal et al., 2026), video games (Zhang et al., 2026; Ouyang et al., 2026), and embodied simulators (Savva et al., 2019; Yang et al., 2025c). These tasks combine many capabilities, making it difficult to identify which limits performance. CUAWorld-RT enables controlled comparisons by varying difficulty, interaction, and the real-time setting within each visual minigame.

Capability-specific evaluation of VLMs. Capability-specific benchmarks test 3D spatial understanding (Yang et al., 2025a), temporal reasoning (Liu et al., 2024), and visual reasoning (Chia et al., 2024) through questions about images or videos. Interactive benchmarks also study visual reasoning (Mayer et al., 2025; Wang et al., 2026b) or relate performance to capability requirements (Wu et al., 2023; Li et al., 2024). We complement these evaluations with configuration-level capability labels and controlled changes to difficulty, interface, and real-time interaction.

Real-time interaction. Robotics methods account for inference delay through asynchronous execution or prediction of future states (Black et al., 2025; Shukor et al., 2025; Tang et al., 2025). Computer-use benchmarks measure latency (Abhyankar et al., 2026), while game benchmarks compare paused and real-time play (Zhang et al., 2026; Park et al., 2025). We compare live and paused execution on the same minigame and seed, separately for configurations with and without real-time requirements, to examine how response speed affects performance.

6 Conclusion

In this work, we introduce CUAWorld-RT, a benchmark of 100 minigame families and 1,000 configurations for studying the capabilities required for real-time interaction. We construct these minigames using a scalable method with controlled capability requirements and executable oracle solutions. Our evaluations show that current VLMs achieve low success rates and require substantial time even on tasks they solve. We identify temporal reasoning, interface exploration, 3D spatial understanding, and interaction with the interface as key weaknesses, with slow responses further limiting performance in real-time environments. These findings highlight that progress in real-time interaction depends on improving both models' ability to solve tasks and their ability to execute solutions accurately and much faster.

AI Use Statement

We used AI assistance for general software development, creating figures, research on generating synthetic environments, polishing the writing across the manuscript, and verifying related work.

REFERENCES

Reyna Abhyankar, Qi Qi, and Yiying Zhang. OSWorld-Human: Benchmarking the Efficiency of Computer-Use Agents, 2026. URL https://arxiv.org/abs/2506.16042.

Pranjal Aggarwal and Sean Welleck. Programming with Pixels: Can Computer-Use Agents do Software Engineering?, 2025. URL https://arxiv.org/abs/2502.18525.

Pranjal Aggarwal, Graham Neubig, and Sean Welleck. Gym-Anything: Turn any Software into an Agent Environment, 2026. URL https://arxiv.org/abs/2604.06126.

ARC Prize Foundation. ARC-AGI-3: A new challenge for frontier agentic intelligence, 2026.

Ruairidh M. Battleday, Kai Sandbrink, Jimi Cullen-Drohan, Zihan Yan, Timothy Muller, Clare Maguire, Ales Kubicek, Fraser Greenlee-Scott, Sukrit Sumant, Tri Dao, Jürgen Schmidhuber, Michal Valko, Joshua Tenenbaum, Thomas L. Griffiths, Zeb Kurth-Nelson, and James C. R. Whittington. DiG-bench: Discovery in games, 2026. URL https://arxiv.org/abs/2608.12593.

Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-Time Execution of Action Chunking Flow Policies, 2025. URL https://arxiv.org/abs/2506.07339.

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, 2024.

Yuan Cao, Dezhi Ran, Mengzhou Wu, Yuzhe Guo, Xin Chen, Ang Li, Gang Cao, Gong Zhi, Hao Yu, Linyi Li, Wei Yang, and Tao Xie. Gui-genesis: Automated synthesis of efficient environments with verifiable rewards for gui agent post-training, 2026. URL https://arxiv.org/abs/2602.14093.

Jiangjie Chen, Qianyu He, Siyu Yuan, Aili Chen, Zhicheng Cai, Weinan Dai, Hongli Yu, Qiying Yu, Xuefeng Li, Jiaze Chen, Hao Zhou, and Mingxuan Wang. Enigmata: Scaling logical reasoning in large language models with synthetic verifiable puzzles, 2025. URL https://arxiv.org/abs/2505.19914.

Yuxi Chen, Haoyu Zhai, Chenkai Wang, Rui Yang, Lingming Zhang, Gang Wang, and Huan Zhang. Captcha solving for native gui agents: Automated reasoning-action data generation and self-corrective training. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2603.23559.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing GUI Grounding for Advanced Visual GUI Agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9313–9332, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.505. URL https://aclanthology.org/2024.acl-long.505/.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. In *International Conference on Learning Representations (ICLR)*, 2019.

Yew Ken Chia, Vernon Toh Yan Han, Deepanway Ghosal, Lidong Bing, and Soujanya Poria. Puzzlevqa: Diagnosing multimodal reasoning challenges of language models with abstract visual patterns, 2024. URL https://arxiv.org/abs/2403.13315.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning, 2019. URL https://arxiv.org/abs/1912.01588.

Mariya Davydova, Daniel Jeffries, Patrick Barker, Arturo Márquez Flores, and Sinéad Ryan. OSUniverse: Benchmark for multimodal GUI-navigation AI agents, 2025.

Gelei Deng, Haoran Ou, Yi Liu, Jie Zhang, Tianwei Zhang, and Yang Liu. Oedipus: Llm-enchanced reasoning captcha solver, 2024. URL https://arxiv.org/abs/2405.07496.

Esther Derman, Gal Dalal, and Shie Mannor. Acting in delayed environments with non-stationary markov policies. In *International Conference on Learning Representations (ICLR)*, 2021. arXiv:2101.11992.

Maxence Faldor, Jenny Zhang, Antoine Cully, and Jeff Clune. Omni-epic: Open-endedness via models of human notions of interestingness with environments programmed in code, 2024. URL https://arxiv.org/abs/2405.15568.

Xingyu Fu, Yushi Hu, Bangzheng Li, Yu Feng, Haoyu Wang, Xudong Lin, Dan Roth, Noah A. Smith, Wei-Chiu Ma, and Ranjay Krishna. Blink: Multimodal large language models can see but not perceive, 2024. URL https://arxiv.org/abs/2404.12390.

Deepanway Ghosal, Vernon Toh Yan Han, Chia Yew Ken, and Soujanya Poria. Are language models puzzle prodigies? algorithmic puzzles unveil serious challenges in multimodal reasoning, 2024. URL https://arxiv.org/abs/2403.03864.

Lanxiang Hu, Mingjia Huo, Yuxuan Zhang, Haoyang Yu, Eric P. Xing, Ion Stoica, Tajana Rosing, Haojian Jin, and Hao Zhang. lmgame-bench: How good are LLMs at playing games?, 2025a.

Mengkang Hu, Pu Zhao, Can Xu, Qingfeng Sun, Jianguang Lou, Qingwei Lin, Ping Luo, and Saravan Rajmohan. Agentgen: Enhancing planning abilities for large language model based agent via environment and task generation. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2025b.

Lawrence Keunho Jang, Jing Yu Koh, Daniel Fried, and Ruslan Salakhutdinov. Odysseys: Benchmarking web agents on realistic long horizon tasks, 2026. URL https://arxiv.org/abs/2604.24964.

Mengdi Jia, Zekun Qi, Shaochen Zhang, Wenyao Zhang, Xinqiang Yu, Jiawei He, He Wang, and Li Yi. Omnispatial: Towards comprehensive spatial reasoning benchmark for vision language models, 2025. URL https://arxiv.org/abs/2506.03135.

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. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. arXiv:2401.13649.

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, 2025. URL https://arxiv.org/abs/2504.07981.

Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Li Erran Li, Ruohan Zhang, Weiyu Liu, Percy Liang, Li Fei-Fei, Jiayuan Mao, and Jiajun Wu. Embodied agent interface: Benchmarking LLMs for embodied decision making. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2024. arXiv:2410.07166.

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, 2026.

Yuanxin Liu, Shicheng Li, Yi Liu, Yuxiang Wang, Shuhuai Ren, Lei Li, Sishuo Chen, Xu Sun, and Lu Hou. TempCompass: Do Video LLMs Really Understand Videos? In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 8731–8772, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.517. URL https://aclanthology.org/2024.findings-acl.517/.

Yaxin Luo, Zhaoyi Li, Jiacheng Liu, Jiacheng Cui, Xiaohan Zhao, and Zhiqiang Shen. Open captchaworld: A comprehensive web-based platform for testing and benchmarking multimodal llm agents, 2025. URL https://arxiv.org/abs/2505.24878.

Wufei Ma, Haoyu Chen, Guofeng Zhang, Yu-Cheng Chou, Jieneng Chen, Celso M de Melo, and Alan Yuille. 3dsrbench: A comprehensive 3d spatial reasoning benchmark, 2024. URL https://arxiv.org/abs/2412.07825.

Julius Mayer, Mohamad Ballout, Serwan Jassim, Farbod Nosrat Nezami, and Elia Bruni. iVISPAR – an interactive visual-spatial reasoning benchmark for VLMs, 2025.

Mingyu Ouyang, Siyuan Hu, Kevin Qinghong Lin, Hwee Tou Ng, and Mike Zheng Shou. GameWorld: Towards Standardized and Verifiable Evaluation of Multimodal Game Agents, 2026. URL https://arxiv.org/abs/2604.07429.

Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wolczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, Jakob Nicolaus Foerster, Jack Parker-Holder, and Tim Rocktäschel. BALROG: Benchmarking agentic LLM and VLM reasoning on games. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2411.13543.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2025.

Dongmin Park, Minkyu Kim, Beongjun Choi, Junhyuck Kim, Keon Lee, Jonghyun Lee, Inkyu Park, Byeong-Uk Lee, Jaeyoung Hwang, Jaewoo Ahn, et al. Orak: A foundational benchmark for training and evaluating LLM agents on diverse video games, 2025.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. UI-TARS: Pioneering Automated GUI Interaction with Native Agents, 2025. URL https://arxiv.org/abs/2501.12326.

Simon Ramstedt and Christopher Pal. Real-time reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. arXiv:1911.04448.

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, 2024.

Yufan Ren, Konstantinos Tertikas, Shalini Maiti, Junlin Han, Tong Zhang, Sabine Süsstrunk, and Filippos Kokkinos. VGRP-Bench: Visual grid reasoning puzzle benchmark for large vision-language models, 2025. URL https://arxiv.org/abs/2503.23064.

Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Küttler, Edward Grefenstette, and Tim Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A Platform for Embodied AI Research, 2019. URL https://arxiv.org/abs/1904.01201.

Ziyao Shangguan, Chuhan Li, Yuxuan Ding, Yanan Zheng, Yilun Zhao, Tesca Fitzgerald, and Arman Cohan. Tomato: Assessing visual temporal reasoning capabilities in multimodal foundation models, 2024. URL https://arxiv.org/abs/2410.23266.

Jiajun Shi, Jian Yang, Jiaheng Liu, Xingyuan Bu, Jiangjie Chen, Junting Zhou, Kaijing Ma, Zhoufutu Wen, Bingli Wang, Yancheng He, Liang Song, Hualei Zhu, Shilong Li, Xingjian Wang, Wei Zhang, Ruibin Yuan, Yifan Yao, Wenjun Yang, Yunli Wang, Siyuan Fang, Siyu Yuan, Qianyu He, Xiangru Tang, Yingshui Tan, Wangchunshu Zhou, Zhaoxiang Zhang, Zhoujun Li, Wenhao Huang, and Ge Zhang. Korgym: A dynamic game platform for llm reasoning evaluation, 2025. URL https://arxiv.org/abs/2505.14552.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Remi Cadene. Smolvla: A vision-language-action model for affordable and efficient robotics, 2025. URL https://arxiv.org/abs/2506.01844.

Yueqi Song, Tianyue Ou, Yibo Kong, Zecheng Li, Graham Neubig, and Xiang Yue. Visualpuzzles: Decoupling multimodal reasoning evaluation from domain knowledge, 2025. URL https://arxiv.org/abs/2504.10342.

Sanjana Srivastava, Chengshu Li, Michael Lingelbach, Roberto Martín-Martín, Fei Xia, Kent Vainio, Zheng Lian, Cem Gokmen, Shyamal Buch, C. Karen Liu, Silvio Savarese, Hyowon Gweon, Jiajun Wu, and Li Fei-Fei. BEHAVIOR: Benchmark for Everyday Household Activities in Virtual, Interactive, and Ecological Environments, 2021. URL https://arxiv.org/abs/2108.03332.

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Jiaming Tang, Yufei Sun, Yilong Zhao, Shang Yang, Yujun Lin, Zhuoyang Zhang, James Hou, Yao Lu, Zhijian Liu, and Song Han. Vlash: Real-time vlas via future-state-aware asynchronous inference, 2025. URL https://arxiv.org/abs/2512.01031.

Xiwen Teoh, Yun Lin, Siqi Li, Ruofan Liu, Avi Sollomoni, Yaniv Harel, and Jin Song Dong. Are CAPTCHAs still bot-hard? generalized visual CAPTCHA solving with agentic vision language model. In *34th USENIX Security Symposium (USENIX Security 25)*, 2025.

Bowen Wang, Dunjie Lu, Junli Wang, Tianyi Bai, Shixuan Liu, Zhipeng Zhang, Haiquan Wang, Hao Hu, Tianbao Xie, Shuai Bai, Dayiheng Liu, Que Shen, Junyang Lin, and Tao Yu. Cua-gym: Scaling verifiable training environments and tasks for computer-use agents, 2026a. URL https://arxiv.org/abs/2605.25624.

Junyu Wang, Changjia Zhu, Yuanbo Zhou, Lingyao Li, Xu He, Mingkui Wei, and Junjie Xiong. Cognition: From evaluation to defense against multimodal llm captcha solvers, 2025a. URL https://arxiv.org/abs/2512.02318. USENIX Security 2026.

Qineng Wang, Baiqiao Yin, Pingyue Zhang, Jianshu Zhang, Kangrui Wang, Zihan Wang, Jieyu Zhang, Keshigeyan Chandrasegaran, Han Liu, Ranjay Krishna, Saining Xie, Jiajun Wu, Li Fei-Fei, and Manling Li. Mindcube: Spatial mental modeling from limited views, 2025b. URL https://arxiv.org/abs/2506.21458.

Xinyu Wang, Bohan Zhuang, and Qi Wu. Are large vision language models good game players? In *International Conference on Learning Representations (ICLR)*, 2025c. arXiv:2503.02358.

Zirui Wang, Junyi Zhang, Jiaxin Ge, Long Lian, Letian Fu, Lisa Dunlap, Ken Goldberg, XuDong Wang, Ion Stoica, David M. Chan, Sewon Min, and Joseph E. Gonzalez. Visgym: Diverse, customizable, scalable environments for multimodal agents, 2026b. URL https://arxiv.org/abs/2601.16973.

Yule Wen, Yixin Ye, Yanzhe Zhang, Diyi Yang, and Hao Zhu. Real-time reasoning agents in evolving environments. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2511.04898.

Yue Wu, Xuan Tang, Tom M. Mitchell, and Yuanzhi Li. Smartplay: A benchmark for llms as intelligent agents, 2023.

Zonglin Wu, Yule Xue, Yaoyao Feng, Xiaolong Wang, and Yiren Song. Mca-bench: A multimodal benchmark for evaluating captcha robustness against vlm-based attacks, 2025. URL https://arxiv.org/abs/2506.05982.

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, 2024. URL https://arxiv.org/abs/2404.07972.

Zelai Xu, Zhexuan Xu, Xiangmin Yi, Huining Yuan, Mo Guang, Kaiwen Long, Xinlei Chen, Yi Wu, Chao Yu, and Yu Wang. VS-Bench: Evaluating VLMs for strategic abilities in multi-agent environments. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026. arXiv:2506.02387.

Jihan Yang, Shusheng Yang, Anjali W. Gupta, Rilyn Han, Li Fei-Fei, and Saining Xie. Thinking in Space: How Multimodal Large Language Models See, Remember, and Recall Spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10632–10643, June 2025a.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents, 2025b. URL https://arxiv.org/abs/2504.21798.

Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, Heng Ji, Huan Zhang, and Tong Zhang. EmbodiedBench: Comprehensive Benchmarking Multi-modal Large Language Models for Vision-Driven Embodied Agents, 2025c. URL https://arxiv.org/abs/2502.09560.

Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, Chris Callison-Burch, Mark Yatskar, Aniruddha Kembhavi, and Christopher Clark. Holodeck: Language guided generation of 3d embodied ai environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Lance Ying, Ryan Truong, Prafull Sharma, Kaiya Ivy Zhao, Nathan Cloos, Kelsey R. Allen, Thomas L. Griffiths, Katherine M. Collins, José Hernández-Orallo, Phillip Isola, Samuel J. Gershman, and Joshua B. Tenenbaum. AI Gamestore: Scalable, Open-Ended Evaluation of Machine General Intelligence with Human Games, 2026. URL https://arxiv.org/abs/2602.17594.

Mengqi Yuan, Zilong Zhou, Xinzhuang Xiong, et al. Osworld 2.0: Benchmarking computer use agents on long-horizon real-world tasks, 2026.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, Cong Wei, Botao Yu, Ruibin Yuan, Renliang Sun, Ming Yin, Boyuan Zheng, Zhenzhu Yang, Yibo Liu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. MMMU: A Massive Multi-discipline Multimodal Understanding and Reasoning Benchmark for Expert AGI. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9556–9567, June 2024.

Xiang Yue, Tianyu Zheng, Yuansheng Ni, Yubo Wang, Kai Zhang, Shengbang Tong, Yuxuan Sun, Botao Yu, Ge Zhang, Huan Sun, Yu Su, Wenhu Chen, and Graham Neubig. MMMU-Pro: A More Robust Multi-discipline Multimodal Understanding Benchmark. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15134–15186, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.736. URL https://aclanthology.org/2025.acl-long.736/.

Alex L. Zhang, Thomas L. Griffiths, Karthik R. Narasimhan, and Ofir Press. VideoGameBench: Can Vision-Language Models complete popular video games?, 2026. URL https://arxiv.org/abs/2505.18134.

Jianrui Zhang, Mu Cai, and Yong Jae Lee. Vinoground: Scrutinizing lmms over dense temporal reasoning with short videos, 2024. URL https://arxiv.org/abs/2410.02763.

Jiayi Zhang, Yiran Peng, Fanqi Kong, Cheng Yang, Yifan Wu, Zhaoyang Yu, Jinyu Xiang, Jianhao Ruan, Jinlin Wang, Maojia Song, HongZhang Liu, Xiangru Tang, Bang Liu, Chenglin Wu, and Yuyu Luo. Autoenv: Automated environments for measuring cross-environment agent learning, 2025. URL https://arxiv.org/abs/2511.19304.

Henry Hengyuan Zhao, Kaiming Yang, Wendi Yu, Difei Gao, and Mike Zheng Shou. WorldGUI: An interactive benchmark for desktop GUI automation from any starting point, 2025.

Xiangxi Zheng, Linjie Li, Zhengyuan Yang, Ping Yu, Alex Jinpeng Wang, Rui Yan, Yuan Yao, and Lijuan Wang. V-MAGE: A game evaluation framework for assessing vision-centric capabilities in multimodal large language models, 2025a.

Xinyue Zheng, Haowei Lin, Kaichen He, Zihao Wang, Zilong Zheng, and Yitao Liang. MCU: An evaluation framework for open-ended game agents. In *International Conference on Machine Learning (ICML)*, 2025b. arXiv:2310.08367.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2307.13854.

APPENDIX CONTENTS

A Limitations **16**

B Minigame Construction **17**

B.1 Creator and Audit Agents 17

B.2 Difficulty Levels 17

B.3 Interaction Modes 18

B.4 Real-Time Revisions 18

C Minigame List **18**

D Capability Annotation Criteria **18**

E Real-Time Evaluation Protocol **20**

E.1 Action and Screenshot Latency 20

F Evaluation Details **20**

F.1 Models and Harnesses 20

F.2 Action Space and Prompt 21

F.3 Budgets and Outcomes 21

F.4 Time and Cost 21

F.5 Coding Protocol 21

F.6 Interface Rulebooks 21

F.7 Human Baseline 21

G Examples of Model-Written Controllers **23**

H Real-Time Definition and Review Prompts **23**

H.1 Definition 23

H.2 Review Prompts 24

I Extended Related Work **32**

A LIMITATIONS

We focus on evaluating existing agents, although our programmatic task generators and automatic verifiers also provide a basis for training. Studying whether training on these minigames improves real-time interaction is an important direction for future work.

Our minigames capture specific demands of real-time environments to enable controlled comparisons across capabilities, difficulty, and interaction. Whether improvements on these tasks transfer to complete games and physical interfaces remains an open question.

Finally, CUAWORLD-RT covers a selected set of capabilities and task mechanics. Our creation–audit framework provides a way to extend this coverage with additional minigames and capability combinations.

B MINIGAME CONSTRUCTION

This appendix details the construction procedure summarized in Section 2. All agents in the construction pipeline use GPT-5.6.

B.1 CREATOR AND AUDIT AGENTS

The creator agent receives one selected minigame together with its survey records and source links. It also inspects previously approved minigames as working examples of the environment structure. The creator implements the environment specification, task, generator, browser interface, server grader, exported verifier, oracle solver, difficulty and interaction controls, real-time settings, and tests. It may create or edit only the files of its own minigame and cannot modify the shared runtime, evaluator, or clock. The creator runs all ten difficulty and interaction configurations in the browser in the live and paused settings and checks generation, grading, result export, verification, failure, and retry. It records screenshots, recordings, and command outputs as evidence, together with a solution video in which the oracle solver completes the task in the browser.

A fresh audit agent then inspects the environment and its evidence without modifying any files. It checks the following items and ends its report with a verdict of `PASS` or `REVISION_REQUIRED`.

(a) **Difficulty baseline.** The generated task is assigned to the appropriate level by comparison with approved reference minigames, without relying on the environment name or the creator's description.

(b) **Difficulty variations.** Levels 1 to 5 increase in what the agent must perceive, decide, remember, or control. Adding repetitive steps is not sufficient.

(c) **Interaction.** For the same seed and difficulty, simplified and full interaction use the same minigame, information, goal, timing, physics, tolerances, and success condition. Full interaction requires the intended mouse or keyboard operations on the minigame.

(d) **Real time.** In live execution, the environment continues while the model produces an action. In paused execution, it remains frozen until the action arrives, and the action then executes with the intended behavior.

(e) **End-to-end behavior.** All configurations load, remain playable, produce the correct grading result, export the result, handle failure, and allow another attempt.

(f) **Evidence.** The audit agent ignores comments and written claims that the running environment does not support. Screenshots, recordings, frame sequences, exported results, and browser behavior serve as evidence, and missing visual evidence is a severe issue.

The audit agent also checks that the creator changed only files within its permitted scope. The creator revises the implementation in response to the report, and a new audit agent repeats the audit.

B.2 DIFFICULTY LEVELS

The five difficulty levels, from very easy (level 1) to very hard (level 5), are defined in the same way for every minigame in CUAWORLD-RT. A level describes the exact configuration the agent receives, and the original configuration of a minigame can belong to any level. A difficulty-review agent assigns levels by comparison with approved reference minigames, including Isometric Voxel Extraction Mine (level 1), Cursor-Controlled Constellation Hunt (level 2), Gyroscopic Tilt Board (level 3), Blind Dice Courier (level 4), and Exact-Change Candy Cascade (level 5).

To construct the other levels, the creator changes parameters that alter the problem the agent must solve. Depending on the minigame, these include the information available to the agent, visual ambiguity, the number of variables handled together, dependencies between decisions, planning depth, motion speed and timing windows, action precision, and recovery options. More steps, rounds, or waiting do not by themselves make a configuration harder. For every pair of adjacent levels, the creator must state what changes in the agent's decision or control problem. Table 4 shows the five levels of Rotating On-Screen Keyboard.

Table 4: **Difficulty levels of Rotating On-Screen Keyboard.** The keyboard rotates in three dimensions with irregularly timed motion.

Level	Code length	Keys	Keyboard motion
1	3	12 large keys	Gentle, begins after the second character
2	4	20 keys	Begins after the first character
3	5	26 keys	Faster, begins after the first character
4	5	32 keys	Never repeats the same turn within an attempt
5	8	32 smaller keys	Fast, sharply irregular, starts immediately

Table 5: **Examples of simplified and full interaction.**

Minigame	Simplified	Full
Rotating On-Screen Keyboard	Type the code on the physical keyboard.	Click each on-screen key while it moves.
Clockwork Clutch Safe	Click a labeled release or engage button beneath each seal.	Drag each seal's clutch lever between its coupled and free positions.
Parallel Grillmaster	Select a food, then click buttons to start cooking and move it to the tray.	Drag each food from the order to the grill and then to the serving tray.
LIDAR Blacksite	Hold labeled movement and turning buttons, and use buttons to scan, pick up, and verify.	Hold movement keys, drag the view to turn, click the scene to scan or pick up the beacon, and walk through the exit gate.

B.3 Interaction Modes

Section 2.3 defines the two interaction modes. For a fixed seed and difficulty, both modes generate the same world, information, goal, and action effects. The grader rejects actions submitted through the controls of the other mode. Table 5 shows examples.

B.4 Real-Time Revisions

We also reviewed the minigames with real-time configurations for two kinds of mechanisms: those that let an agent stop the relevant dynamics without losing progress, such as a brake that halts moving parts, and repeating motion that permits a precomputed solution. We revised 19 minigames to remove these mechanisms, and CUAWorld-RT uses the revised versions.

C Minigame List

Table 6 lists the 100 minigames in CUAWorld-RT and the capabilities each requires.

D Capability Annotation Criteria

CUAWorld-RT studies six capabilities. Review agents annotate the four capabilities below. The remaining two, interaction and real-time interaction, are varied through the interaction mode and the real-time setting. Appendix H defines when a configuration requires real-time interaction.

The authors define the following criteria for capability annotation. Review agents apply them to the requirements of a solution using screenshots and visible mouse or keyboard controls.

Visual understanding. We classify visual understanding as 2D or 3D according to the spatial understanding required to solve the task. Decorative perspective alone is insufficient for a 3D label.

Temporal reasoning. A configuration requires this capability when every general screenshot-only solution must interpret change, motion, order, or duration, remember relevant earlier state that is no longer visible, or control when and for how long an action remains active. The temporal relationship must affect action choice or execution. A simple visible sequence, idle animation, or automatic transition to a static result does not establish this requirement.

Table 6: **Minigames in CUAWORLD-RT.** Capability labels: 3D visual understanding (3D), temporal reasoning (T), reasoning and planning (R), and exploration and interface understanding (E).

Minigame	3D	T	R	E	Minigame	3D	T	R	E
After Hours at the Reliquary		●	●	●	Horizon Relay		●		
Anamorphic Registration Press				●	Impossible Ecology		●	●	●
Anthill Front		●	●	●	Impossible Panorama	●	●		●
Apothecary Dead Reckoning		●	●	●	Isometric Voxel Extraction Mine	●		●	●
Ballast Lantern		●	●		Lampwrights Program	●		●	
Bandaged Rose Window			●	●	Lantern Loft	●		●	
Blind Corridor Oscilloscope		●	●	●	Lanternfin Dive	●	●	●	●
Blind Dice Courier	●	●	●		Lanternwing Roundup	●	●	●	
Bomb Manual From Hell			●		Letter Rapids		●		
Branch Repair	●		●	●	LIDAR Blacksite	●	●	●	●
Bureaucratic Signature Trap				●	Live Control-Flow Wiring Lab		●	●	●
Cell Gatekeeper			●		Living Scaffold			●	●
Charter of the Nine Cantons		●	●	●	Magnetic-Stripe Purgatory		●		●
Circle Limit Twist			●	●	Motion-Only Ghost Jigsaw		●		
Clockbeat Catacomb		●	●		Museum of Lost Gestures		●		
Clockwork Clutch Safe		●	●		Occlusion Shell Swindle		●		
Clockwork Doppelgänger Customs		●	●	●	Parallax / Inertial Jigsaw Alignment	●			●
Cloudpost Circuit	●	●	●	●	Parallax Orchard	●			●
Cloudstep Caddie	●		●		Parallel Grillmaster		●		
Collision Chimes		●	●		Pearl Lattice	●		●	
Concertina Courier		●	●		Photograph Eats the Room	●		●	●
Consent Gauntlet		●	●		Pocket Locksmith	●		●	●
Consequences Boss		●			Polarized Palimpsest		●		●
Coordinates by Another Name				●	Polycube Parcel	●		●	●
Courtesy Junction		●	●		Polyrhythm Customs		●		
Crackglaze Crossing		●	●	●	Popup Exorcist				●
CRAFTCHA: Alchemy Bench		●	●		Rayglass Vault			●	●
Crash-Deadline Hovercar		●	●		Recursive Dollhouse Smuggling	●		●	
Crater Walker	●		●	●	Reflow Vitrine			●	●
Cursor-Controlled Constellation Hunt				●	Reload Interruption		●		
Dead Man's Switch		●	●		Reveal to Identify			●	●
Downsky Causeway	●	●	●		Ribbon Switchboard				●
Dual-Projection Sculpture Rig	●	●	●	●	Rising Causeway	●	●	●	
Elastic Membrane Sorter		●	●		Rorschach / Subjective Prompt With A Fixed Rubric			●	●
Elbow Engine		●	●		Rotate The Wrong Thing Upright	●		●	●
Ember Anvil	●		●	●	Rotating On-Screen Keyboard	●			
Facet Lantern	●		●	●	Shadow Crime Lab	●		●	●
Firewatch Fold		●	●	●	Slot-Reel Character Capture		●		
First Change Memory		●		●	Surveyors Toybox	●		●	
Five-Second Rule		●			The Flat Prisoner	●	●	●	●
Five-System Verification Reactor		●	●		The Last Carbon Isles				
Flip-Gate Cascade			●	●	The Unlabeled Drawer			●	●
Fluke Census		●	●	●	The Unwatched Wing	●	●	●	●
Forced-Perspective Moving Day	●		●		Three-Camera Claw Machine	●	●	●	
Four-Pane Pilgrimage			●	●	Tiny FPS Customs	●		●	●
Four-Tab Robot Handshake		●			Tomographic Baggage Surgery	●	●	●	●
Funeral With No Instructions			●	●	Top-Face Dice Arithmetic	●	●	●	
Gyroscopic Tilt Board		●	●		Two-Lamp Dyeworks			●	●
Hearthlift Courier	●		●	●	Wrong Number		●		●
Hologram Silhouette Foundry	●		●		Zero-G Cable Autopsy	●		●	

Reasoning and planning. Solving requires inferring constraints or choosing actions whose consequences affect later decisions. The annotation identifies the relevant inference or action dependency. Basic visual recognition alone does not establish this requirement.

Exploration and interface understanding. The agent must interact to reveal relevant information or learn how the interface behaves before solving the task. Routine feedback, an automatic transition that supplies the next required information, or following explicit controls does not establish this requirement.

Review agents inspect the implementation and recorded behavior for each difficulty and interaction mode. Each annotation is supported by code or behavioral evidence. Private state used by an oracle provides evidence about the implementation and is excluded from the information available to the evaluated agent.

E Real-Time Evaluation Protocol

The shared runtime controls the execution schedule, and the task clock begins when the environment starts. In live execution, an observation request returns a current frame while the task clock continues. Paused execution advances the task through an observation window and returns a chronological sequence of frames. The clock then stops on the final frame until the action executes and resumes for the next observation window.

The observation window and number of frames are set per minigame. For 39 minigames whose state does not change between actions, the window is 0 ms with one frame. For the others, the window ranges from 80 to 1,200 ms with one to eight frames.

The runtime records when each frame is captured and each action is executed. We also record model response time and the latency of the observation and action pathways. These measurements allow the evaluation protocol and the timing of individual actions to be inspected independently of the task's final score.

E.1 Action and Screenshot Latency

The action client runs on the same machine as its sandbox and uses a persistent direct HTTP connection. An action request can also return the subsequent screenshot. We measure latency from the start of the request until its complete response, including the image, has arrived. Model inference and subsequent image loading and decoding are outside this measurement.

In isolated timing tests with one client, we measured 150 sequential requests per condition after five warm-up requests. Median latencies were 39.2 ms for Rotating On-Screen Keyboard, 42.5 ms for an idle LIDAR Blacksite scene, and 46.8 ms while issuing LIDAR scan clicks. The corresponding 95th percentiles were 52.1, 55.3, and 70.7 ms. These measurements characterize the tested request path and include slower responses beyond the 50 ms target. They do not establish an upper bound for every request or for concurrent evaluations.

F Evaluation Details

This appendix details the evaluation protocol summarized in Section 3.2.

F.1 Models and Harnesses

We evaluate GPT-6 Astra with low reasoning effort and GPT-5.6 Luna with xHigh reasoning effort through the Codex CLI (version 0.153.3). Each GPT model receives a single `computer` tool. We disable the shell, code execution, web search, browser, and built-in computer-use features. Qwen3.5-9B and Qwen3.8-27B are served through the Tinker API with temperature 1.0, top-p 0.95, top-k 20, and at most 2,048 output tokens per response. They use their native computer-use function format with coordinates on a 1000×1000 grid, and their context contains the last 100 turns with at most 20 screenshots.

Table 7: **Prices used to compute cost**, in USD per million tokens.

Model	Input	Cached input	Output
Qwen3.5-9B	0.66	0.132	1.995
Qwen3.8-27B	2.48	0.496	7.46
GPT-5.6 Luna	0.20	0.02	1.20
GPT-6 Astra	10.00	1.00	50.00

F.2 Action Space and Prompt

For the GPT models, each call to the `computer` tool contains an ordered list of mouse and keyboard actions and returns a screenshot after the actions complete. The actions include moving, clicking, dragging, scrolling, typing, pressing keys, holding and releasing keys and mouse buttons, and waiting. The model instructions and the text after each task description both instruct all models to use only the visible controls of the task webpage, without writing code or inspecting the page. Figure 9 shows the complete prompt for Rotating On-Screen Keyboard in the live setting. In the paused setting, the paragraph following `Temporal mode` is replaced by the text in Figure 10.

F.3 Budgets and Outcomes

Each attempt allows at most 100 steps. A step is one request that contains one or more actions in order. Requests that only return a screenshot do not consume steps. Each Codex session is limited to 3,600 seconds, and attempts that reach this limit count as failures. We exclude and rerun attempts that end with a provider or infrastructure error. The task verifier determines whether each attempt succeeds.

F.4 Time and Cost

Time is the wall-clock duration of an attempt from the start to the end of its episode. Cost is the API-equivalent cost of the recorded token usage,

$$\text{cost} = \frac{(n_{\text{input}} - n_{\text{cached}})\, p_{\text{input}} + n_{\text{cached}}\, p_{\text{cached}} + n_{\text{output}}\, p_{\text{output}}}{10^6},$$

where reasoning tokens are counted once as output tokens. Table 7 lists the prices. For GPT-6 Astra, requests with more than 272K input tokens are charged twice the input and cached-input prices and 1.5 times the output price.

F.5 Coding Protocol

In the coding experiment, the model runs in the Codex CLI with its standard tools inside an isolated sandbox. It receives an authenticated HTTP endpoint that accepts the same actions and returns screenshots, together with a short Python client. The model can write programs that process screenshots and issue actions through this endpoint. Access to the task machine, page source, and browser developer tools remains prohibited.

F.6 Interface Rulebooks

For the experiment in Section 4.7, each rulebook describes a minigame's controls, action effects, fixed world rules, feedback, submission, and recovery. It excludes generated answers, target coordinates, hidden state, and seed-specific layouts or solutions. The rulebook is appended to the task prompt, and all other settings remain unchanged.

F.7 Human Baseline

One of the authors played the 32 real-time configurations at difficulty 5 with full interaction in the live setting, seeing each minigame for the first time. The author solved 25 of the 32 configurations (78.1%). The average time per configuration, measured with a standard clock, was 3.62 minutes.

```
Solve the task using your own visual reasoning and the computer tool. Choose literal
mouse/keyboard inputs or request a screenshot with actions=[]. Writing or executing code,
scripts, loops, OCR, automated image processing, and background automation is prohibited.
Use only visible controls of the task webpage. Do not open a terminal, external
application, browser developer tools, console, inspector, source, DOM, or network panel. Do
not navigate, reload, change URLs, inspect hidden state, or switch to unrelated tabs.

Use the computer tool with {"actions": [...]} for all observations and inputs. Every call
returns the screenshot after its inputs complete. Start with actions=[].

Coordinates use the returned 1920x1080 screenshot at its original
resolution: x increases rightward and y downward from [0, 0]. Coordinates
are passed directly to the task display.

Native env.step input objects (use the fields needed for your action):
- Mouse point fields: move, left_click, right_click, middle_click, double_click,
triple_click. Each takes [x,y],
  e.g. {"mouse":{"move":[100,200]}}.
- Mouse drag fields: left_click_drag, right_click_drag. Each takes [[x1,y1],[x2,y2]].
- "mouse": {"buttons": {...}} accepts boolean states: left_down, left_up, right_down,
right_up, middle_down, middle_up.
- "mouse": {"scroll": n} sends n wheel notches, positive down, negative up.
  These are NOT pixels. To scroll at a point, move the pointer first.
- "keyboard" accepts text, keys, keys_down, keys_up. text is a string. keys,
  keys_down and keys_up accept a key name or a list of key names, e.g. "w",
  "shift", "ctrl", "left", "space", "Return". keys taps a chord. keys_down
  presses without releasing. keys_up releases. A separate wait after a tap
  does not hold the key. There is no duration field on keyboard or mouse.
- {"action":"wait", "time":0.2} waits in seconds ("seconds" is also accepted).
- {"action":"screenshot"} or [] observes without spending the step budget.
  A screenshot entry inside a batch does not yield an intermediate frame.

Keep distinct inputs in separate list entries when order matters. The runtime
processes mouse before keyboard within a combined object, so use keys_down,
mouse input, keys_up entries for modifier-held gestures. To draw a curved path,
send mouse down, a sequence of mouse moves (and waits if needed), then mouse up.
Inputs can stay held across requests and screenshots. Do not replace a held
input with repeated taps or type repeated letters to simulate movement.

Temporal mode: live
The task keeps running while you reason, deliver inputs, and wait.
Each response returns one instantaneous screenshot after the action batch
finishes. Held keys/buttons remain down across responses until you release them.
A batch containing down, wait, up can express a timed hold in live mode.
execute_at_s is not available in this mode.


You have at most 100 steps. Each non-observation request, including an ordered action list,
consumes one step. Screenshot-only requests do not consume steps. Inputs in a batch execute
in order. Release held inputs before finishing. Stop when done is true.

Task:
Confirm the shown code using the on-screen keyboard.

Solve the task using your own visual reasoning and the computer tool. Choose literal
mouse/keyboard inputs or request a screenshot with actions=[]. Writing or executing code,
scripts, loops, OCR, automated image processing, and background automation is prohibited.
Use only visible controls of the task webpage. Do not open a terminal, external
application, browser developer tools, console, inspector, source, DOM, or network panel. Do
not navigate, reload, change URLs, inspect hidden state, or switch to unrelated tabs.

Your answer only counts once it is submitted: after completing the task, press the visible
submit/confirm/verify control and check the UI acknowledges it before you declare the task
finished.
```

Figure 9: **Prompt for Rotating On-Screen Keyboard in the live setting.**

```
The task clock is paused while you reason and while inputs are delivered.
Each screenshot request or step response advances one configured observation
window, returning its frames in chronological order. The task then freezes on
the final frame, and the next input acts on that state. A wait sleeps in wall
time but does not advance the paused task clock. To hold a key or mouse button
while the task advances, send down in one request and up in a later request.
Additional screenshot requests advance additional
windows while the input stays held. Down, wait, up in a single batch does NOT
advance paused task time between down and up.
```

Figure 10: **Timing instructions in the paused setting.**

G Examples of Model-Written Controllers

We examine saved trajectories from Astra's coding evaluation at difficulty 5 with full interaction in the live setting. In Rotating On-Screen Keyboard, the model uses perspective transforms and image templates to locate requested keys, converts their positions to screen coordinates, and clicks them. In Wrong Number, it fits curves to waveform pixels from screenshots, estimates the phase error, and adjusts a slider. Both attempts pass the task verifier.

Image-based control also appears in failed attempts. In Gyroscopic Tilt Board, Astra detects the ball and direction indicator, estimates velocity, and moves the tilt control along a sequence of waypoints, but the attempt fails. These examples show how scripts can process observations and issue actions between model responses. Successful script execution alone does not establish that the task was solved.

H Real-Time Definition and Review Prompts

H.1 Definition

The real-time label is assigned to a fixed difficulty and interaction configuration. Let H_t denote the visible interaction history through task time t, I_t the agent's own prior actions and their task times, and W_t^w the visible window from $t - w$ to t. Let u denote the terminal task outcome, with less task-active continuation time after delivery of the next action used only to break ties between equal outcomes. Expectations use visible task information and average over future randomness. Private generator or page state is unavailable to the policy.

A grouped control mode D contains every occurrence, across seeds, rounds, and attempts, of the same outcome-affecting control mode. Let $C_\Delta(D)$ contain the histories in D from which success remains possible after a delay Δ with no new action. Only histories that become unsalvageable after this delay are excluded. Let $A^*_\Delta(H_t)$ be the set of actions maximizing expected u when delivered at $t + \Delta$ and followed by optimal continuation. Let H_t^τ be the history after τ with no new action, and $Q(H_t, a)$ the expected u when a is the next action and is followed by optimal continuation.

A configuration satisfies the real-time definition if and only if there exist a grouped control mode D, a delay $\Delta > 0$, and a bounded visual window w such that $C_\Delta(D)$ is nonempty and all three conditions hold:

(i) No policy using only the agent's own action history and task times selects an optimal delayed action for every viable history:

$$\nexists q : I_t \mapsto a \quad \text{such that} \quad \forall H_t \in C_\Delta(D),\ q(I_t) \in A^*_\Delta(H_t).$$

(ii) A policy using that history and the recent visual window can select an optimal delayed action for every viable history:

$$\exists g : (I_t, W_t^w) \mapsto a \quad \text{such that} \quad \forall H_t \in C_\Delta(D),\ g(I_t, W_t^w) \in A^*_\Delta(H_t).$$

(iii) For some $H_t \in C_\Delta(D)$, action a, and $0 < \tau < \Delta$, the environment evolves autonomously to H_t^τ, success remains possible, and the action loses value or advantage over no action:

$$Q(H_t^\tau, a) < Q(H_t, a) \qquad \text{or}$$
$$Q(H_t^\tau, a) - Q(H_t^\tau, \text{no-op}) < Q(H_t, a) - Q(H_t, \text{no-op}).$$

An unavailable or ineffective action has the value of no action. The strict loss in (iii) must arise from non-clock task-state evolution. Less remaining deadline, later completion alone, a fixed wait that enables an action, and a fixed animation whose result remains available do not satisfy this condition.

All three conditions must be witnessed by the same visible autonomous process and the same D, Δ, and w. The evidence within W_t^w must support prediction of the process's effect at $t + \Delta$ and determine the delayed-optimal action in (i) and (ii). The evidence can occur across frames or in an accumulated trace, chart, or trajectory within one screenshot. Unrelated animation and future randomness independent of the visible window cannot supply this evidence.

The witness delay Δ is chosen on the time scale of the task behavior. It begins within an active control opportunity and may extend beyond the point where that opportunity degrades or closes. It is independent of the current model provider's latency. The configured observation window and frame count bound w. A stationary policy over the visibly evidenced dynamic state is allowed.

Pre-run exception. For the configuration-level label, a configuration is classified as not real-time if at least 50% of its generated instances admit a successful pre-run solution. After preparation, such a solution starts the autonomous outcome phase and sends no further outcome-affecting actions until that phase ends. Final submission or certification is administrative and is excluded from this action count. Reviewers establish this threshold from source evidence or a reproducible seed probe. If the threshold is unresolved and could change a positive mathematical classification, the reviewer returns `unresolved`. Failure of any mathematical condition already establishes a negative classification.

H.2 Review Prompts

The following are the exact prompt templates used to review the 490 configurations added to the 510 existing configuration labels. The first template assigns one configuration to a reviewer. The second assigns the requested configurations of one minigame to a reviewer, with a separate judgment for each configuration. Both use the same mathematical definition and pre-run exception. Assignment fields are supplied by the launcher. The original template text, including filenames and output instructions, is reproduced unchanged.

H.2.1 One Configuration per Review

```
# Independent real-time classification prompt

Prompt version: `selected-100-pending-v1`

Reuses the historical real-time pilot prompt, adding the settled pre-run
threshold and explicit report destination.

Frozen definition source:
`weird_captcha_gym/docs/controllability/real-time.md`

Frozen definition SHA-256:
`885cd87d0c59d5017b673a2b8696f251e31eedf0c8e388e32d47101625bcaf09`

Use the configuration and SOURCE_ROOT, CASE_INDEX, and OUTPUT_PATH values
supplied by the launcher. Read source only from that frozen SOURCE_ROOT,
not the current checkout. Paths below are relative to SOURCE_ROOT.

---

You are an independent reviewer classifying exactly one Weird CUA Bench
configuration.

## Configuration

- Public environment name: `{public_environment_name}`
- Environment ID: `{environment_id}`
- Difficulty: `{difficulty}`
- Interaction mode: `{interaction_mode}`
```

````
Decide whether this exact configuration is real-time under the frozen
mathematical definition below.

Do not use an existing classification, capability annotation, audit
result, result matrix, or another reviewer's answer. Do not infer the
answer from the environment name or task description.

Read `AGENTS.md` and
`weird_captcha_gym/docs/controllability/real-time.md` completely. Then
read the complete relevant implementation:

1. the exact difficulty and interaction profile in `controls.json`;
2. the generator;
3. the visible browser implementation and relevant CSS;
4. the grader and verifier; and
5. the solver, when present.

Study a normal screenshot-only solution using visible task controls.
Private state and scripted solvers are implementation evidence, not
information available to the policy.

## Frozen mathematical definition

```text
H_t              visible interaction history through task time t
I_t              the agent's own prior actions and task times in H_t
W_t^w            the visible window from t-w through t
u                 terminal task outcome, with less task-active
continuation
                  time after delivery of the next action used only to
                  break
                  ties between equal outcomes
D                 one grouped control mode
C_Delta(D)        histories in D from which success remains possible
after Delta
                  of no new action
A*_Delta(H_t)     actions maximising expected u when delivered at
t+Delta,
                  followed by optimal continuation
H_t^tau           history tau later if the agent supplies no new action
Q(H_t, a)         expected u if a is the next action, followed by optimal
continuation
```

Expectations use only visible task information and average over future
randomness; private generator or page state is not available to the
policy. For a fixed difficulty and interaction configuration, `D`
contains every occurrence across seeds, rounds, and attempts of the same
outcome-affecting control mode. It is not a hand-picked early interval.
`C_Delta(D)` removes only histories whose outcome is already
unsalvageable after the delay.

A fixed difficulty and interaction configuration of `E` is real-time iff
there are a grouped control mode `D`, a delay `Delta > 0`, and a bounded
visual window `w` such that `C_Delta(D)` is nonempty and:

```text
(i)    there is no q : I_t -> a
       with q(I_t) in A*_Delta(H_t) for every H_t in C_Delta(D)

(ii)   there is a g : (I_t, W_t^w) -> a
       with g(I_t, W_t^w) in A*_Delta(H_t)
       for every H_t in C_Delta(D)
````
````

```
(iii)  for some H_t in C_Delta(D), action a, and 0 < tau < Delta,
       the environment changes autonomously from H_t to H_tˆtau,
       success remains possible from H_tˆtau, and

       Q(H_tˆtau, a) < Q(H_t, a)

       or

       Q(H_tˆtau, a) - Q(H_tˆtau, no-op)
         < Q(H_t, a) - Q(H_t, no-op).
```

An unavailable or ineffective action has the value of `no-op`. The strict loss in clause (iii) must be caused by non-clock task-state evolution, not merely by less remaining deadline or later completion. An action becoming enabled or more effective after a fixed wait does not satisfy the clause. A fixed animation that later enables controls or permanently archives its result is not an expiring opportunity.

The three clauses must be witnessed by the same phenomenon. The autonomous evolution in clause (iii) must be visibly evidenced within `W_tˆw`, and that evidence must be what makes the delayed-optimal action depend on observation in clauses (i) and (ii). The window must support prediction of the evolution's effect at `t+Delta`; unrelated idle animation or future randomness independent of the window cannot supply clause (iii).

The evidence may appear across consecutive frames or as an accumulated visible trace, chart, or trajectory in one screenshot. A single screenshot is allowed when it visibly contains the relevant temporal evidence. This does not admit a static puzzle next to an unrelated animation.

`Delta` is a reported witness horizon on the time scale of the task behavior, not the current provider's response latency. It must begin inside a nontrivial active control opportunity and may extend across the point at which that opportunity gets worse or closes. The configured `observation_window_ms` and `frames_per_observation` bound `w`; they do not force every environment to need every captured frame.

The conjunction has these required consequences:

- Clause (i) rules out a fixed schedule or a policy driven only by the agent's own action clock.
- Clause (ii) requires a bounded recent visual window to be sufficient. This excludes unobservable randomness and information that must be recalled only from outside the window.
- Clause (iii) requires a useful action opportunity to get worse inside the delay because the task state changes. This excludes fixed waits, minimum-duration holds solvable from the agent's own clock, and information that remains actionable indefinitely.
- A stationary visual policy is allowed. It can map the currently evidenced changing state to an action without using absolute task time.
- All three clauses must use one common `D`, `Delta`, `w`, and visible autonomous process.

Pre-run threshold

For the configuration-level catalog label, a configuration is Not real-time when
at least 50% of its generated instances admit a successful pre-run solution:
after preparation, the solution starts the autonomous outcome phase and sends

```
no more outcome-affecting actions until that phase ends. Final submission
or
certification is administrative and does not count as an action during
the run.

Record whether the threshold is met, with source-structural evidence or
an actual reproducible seed probe. A failed existing solver does not
prove that no pre-run solution exists. Do not invent a percentage. If the
threshold cannot be determined and it could change a mathematical Yes
into No, return unresolved. A configuration that already fails the
mathematical definition is No regardless of an unresolved pre-run rate.

## Review procedure

1. State the controls available in this exact difficulty and interaction
mode.
2. Describe every period in which the task changes without a new agent
action.
3. State whether an outcome-affecting control remains available during
each such period.
4. Enumerate the legitimate grouped control modes `D`.
5. Try to construct an observation-free policy `q` using only `I_t`.
6. Try to construct a bounded-window policy `g` using only `I_t` and
`W_t^w`.
7. Check whether the same visible autonomous process causes an available
action to become unavailable or strictly less effective within some
`Delta`.
8. For `yes`, give one concrete common witness: `D`, `Delta`, `w`, the
visible process, why `q` fails, why `g` succeeds, and the expiring or
degrading action.
9. For `no`, identify the failed clause and explain why no legitimate
choice of `D`, `Delta`, and `w` repairs it, or establish that the pre-run
threshold overrides a mathematical Yes.
10. Return `unresolved` instead of guessing when the complete source does
not determine the answer.

Return exactly one JSON object:

```json
{
  "case_index": "{case_index}",
  "public_environment_name": "{public_environment_name}",
  "environment_id": "{environment_id}",
  "difficulty": "{difficulty}",
  "interaction_mode": "{interaction_mode}",
  "label": "yes | no | unresolved",
  "controls_available": [],
  "autonomous_processes": [],
  "grouped_control_modes": [],
  "delta_ms": null,
  "window_ms": null,
  "clause_i": {"holds": true, "reason": ""},
  "clause_ii": {"holds": true, "reason": ""},
  "clause_iii": {"holds": true, "reason": ""},
  "common_visible_process": "",
  "pre_run": {
    "threshold": 0.5,
    "at_least_half": null,
    "rate": null,
    "basis": "source_structural | seeded_probe | unresolved |
    not_applicable",
    "reason": "",
    "seed_probe": null
  },
  "files_read_completely": [],
```
```

````
  "source_evidence": [
    {"path": "", "lines": "", "claim": ""}
  ],
  "uncertainties": []
}
```

Write the JSON object, with numeric case_index and difficulty, to the
assigned OUTPUT_PATH using apply_patch. This one report is the only
permitted persistent output. Do not edit puzzle sources, definitions,
historical audits, or another report. Do not inspect other first-pass
reviews, existing classification/capability files, evaluation results, or
previous conversations. The audit manifest is parent-only; the launcher
gives you your exact assignment.

You may run read-only source/math probes when needed, with
PYTHONDONTWRITEBYTECODE=1 and without modifying source. For a seed probe,
include seeds, numerator/denominator, and reproducible probe code in
pre_run.seed_probe. No browser or desktop access, external
websites/services, nested agents, or empirical CUA evaluation. A
source-review report may be unresolved; do not guess to finish.

files_read_completely and source_evidence must use repository-relative
paths. Read each complete implementation file needed by the assigned
mechanic, including shared helpers actually used. If a shared file is
long, read it in successive ranges to EOF; keyword searches do not
substitute for full reads. Record concrete code evidence for the exact
difficulty and interaction. Do not make changes to obtain a preferred
label.
````

H.2.2 CONFIGURATIONS GROUPED BY MINIGAME

````
# Per-game real-time classification prompt

Prompt version: `selected-100-per-game-v2`

Preserves the first-phase mathematical definition, pre-run threshold, and
evidence requirements. Changes only the assignment unit: one reviewer
reads one game once and judges its requested difficulty/interaction
configurations together.

Frozen definition source:
`weird_captcha_gym/docs/controllability/real-time.md`

Frozen definition SHA-256:
`885cd87d0c59d5017b673a2b8696f251e31eedf0c8e388e32d47101625bcaf09`

Use the game assignment and SOURCE_ROOT, CASE_INDEX, CONFIGURATIONS, and
OUTPUT_PATH values supplied by the launcher. Read source only from that
frozen SOURCE_ROOT, not the current checkout. Paths below are relative to
SOURCE_ROOT.

---

You are an independent reviewer classifying one Weird CUA Bench game
across its difficulty and interaction configurations in a single review.

## Assignment
````
````

````
The launcher supplies the exact public environment name, environment ID,
game CASE_INDEX, and CONFIGURATIONS list. Each configuration has its own
numeric case_index, difficulty, and interaction_mode. Normally the list
contains all ten configurations: five difficulties crossed with Full and
Simplified. A partially completed game may supply only its unfinished
configurations. Read all ten active profiles together to understand their
differences, but return judgments only for the supplied CONFIGURATIONS.
Do not inspect or infer the already completed judgments.

Read each shared implementation file once, then apply the definition
separately to every requested configuration. Explicitly check
mode-specific inputs and level-dependent behavior; do not copy one answer
across the game without checking those differences. These are multiple
configuration judgments from one fresh game-level reviewer, not
independent reviewer contexts per configuration.

Do not use an existing classification, capability annotation, audit
result, result matrix, or another reviewer's answer. Do not infer the
answer from the environment name or task description.

Read `AGENTS.md` and
`weird_captcha_gym/docs/controllability/real-time.md` completely. Then
read the complete relevant implementation:

1. the exact difficulty and interaction profile in `controls.json`;
2. the generator;
3. the visible browser implementation and relevant CSS;
4. the grader and verifier; and
5. the solver, when present.

Study a normal screenshot-only solution using visible task controls.
Private state and scripted solvers are implementation evidence, not
information available to the policy.

## Frozen mathematical definition

```text
H_t              visible interaction history through task time t
I_t              the agent's own prior actions and task times in H_t
W_t^w            the visible window from t-w through t
u                 terminal task outcome, with less task-active
continuation
                  time after delivery of the next action used only to
                  break
                  ties between equal outcomes
D                 one grouped control mode
C_Delta(D)        histories in D from which success remains possible
after Delta
                  of no new action
A*_Delta(H_t)     actions maximising expected u when delivered at
t+Delta,
                  followed by optimal continuation
H_t^tau           history tau later if the agent supplies no new action
Q(H_t, a)         expected u if a is the next action, followed by optimal
continuation
```

Expectations use only visible task information and average over future
randomness; private generator or page state is not available to the
policy. For a fixed difficulty and interaction configuration, `D`
contains every occurrence across seeds, rounds, and attempts of the same
outcome-affecting control mode. It is not a hand-picked early interval.
`C_Delta(D)` removes only histories whose outcome is already
unsalvageable after the delay.
````

````
A fixed difficulty and interaction configuration of `E` is real-time iff
there are a grouped control mode `D`, a delay `Delta > 0`, and a bounded
visual window `w` such that `C_Delta(D)` is nonempty and:

```text
(i)    there is no q : I_t -> a
       with q(I_t) in A*_Delta(H_t) for every H_t in C_Delta(D)

(ii)   there is a g : (I_t, W_t^w) -> a
       with g(I_t, W_t^w) in A*_Delta(H_t)
       for every H_t in C_Delta(D)

(iii)  for some H_t in C_Delta(D), action a, and 0 < tau < Delta,
       the environment changes autonomously from H_t to H_t^tau,
       success remains possible from H_t^tau, and

       Q(H_t^tau, a) < Q(H_t, a)

       or

       Q(H_t^tau, a) - Q(H_t^tau, no-op)
         < Q(H_t, a) - Q(H_t, no-op).
```

An unavailable or ineffective action has the value of `no-op`. The strict
loss in clause (iii) must be caused by non-clock task-state evolution,
not merely by less remaining deadline or later completion. An action
becoming enabled or more effective after a fixed wait does not satisfy
the clause. A fixed animation that later enables controls or permanently
archives its result is not an expiring opportunity.

The three clauses must be witnessed by the same phenomenon. The
autonomous evolution in clause (iii) must be visibly evidenced within
`W_t^w`, and that evidence must be what makes the delayed-optimal action
depend on observation in clauses (i) and (ii). The window must support
prediction of the evolution's effect at `t+Delta`; unrelated idle
animation or future randomness independent of the window cannot supply
clause (iii).

The evidence may appear across consecutive frames or as an accumulated
visible trace, chart, or trajectory in one screenshot. A single
screenshot is allowed when it visibly contains the relevant temporal
evidence. This does not admit a static puzzle next to an unrelated
animation.

`Delta` is a reported witness horizon on the time scale of the task
behavior, not the current provider's response latency. It must begin
inside a nontrivial active control opportunity and may extend across the
point at which that opportunity gets worse or closes. The configured
`observation_window_ms` and `frames_per_observation` bound `w`; they do
not force every environment to need every captured frame.

The conjunction has these required consequences:

- Clause (i) rules out a fixed schedule or a policy driven only by the
agent's own action clock.
- Clause (ii) requires a bounded recent visual window to be sufficient.
This excludes unobservable randomness and information that must be
recalled only from outside the window.
- Clause (iii) requires a useful action opportunity to get worse inside
the delay because the task state changes. This excludes fixed waits,
minimum-duration holds solvable from the agent's own clock, and
information that remains actionable indefinitely.
- A stationary visual policy is allowed. It can map the currently
evidenced changing state to an action without using absolute task time.
````

```
- All three clauses must use one common `D`, `Delta`, `w`, and visible
autonomous process.

## Pre-run threshold

For the configuration-level catalog label, a configuration is Not
real-time when
at least 50% of its generated instances admit a successful pre-run
solution:
after preparation, the solution starts the autonomous outcome phase and
sends
no more outcome-affecting actions until that phase ends. Final submission
or
certification is administrative and does not count as an action during
the run.

Record whether the threshold is met, with source-structural evidence or
an actual reproducible seed probe. A failed existing solver does not
prove that no pre-run solution exists. Do not invent a percentage. If the
threshold cannot be determined and it could change a mathematical Yes
into No, return unresolved. A configuration that already fails the
mathematical definition is No regardless of an unresolved pre-run rate.

## Review procedure

1. State the controls available in this exact difficulty and interaction
mode.
2. Describe every period in which the task changes without a new agent
action.
3. State whether an outcome-affecting control remains available during
each such period.
4. Enumerate the legitimate grouped control modes `D`.
5. Try to construct an observation-free policy `q` using only `I_t`.
6. Try to construct a bounded-window policy `g` using only `I_t` and
`W_t^w`.
7. Check whether the same visible autonomous process causes an available
action to become unavailable or strictly less effective within some
`Delta`.
8. For `yes`, give one concrete common witness: `D`, `Delta`, `w`, the
visible process, why `q` fails, why `g` succeeds, and the expiring or
degrading action.
9. For `no`, identify the failed clause and explain why no legitimate
choice of `D`, `Delta`, and `w` repairs it, or establish that the pre-run
threshold overrides a mathematical Yes.
10. Return `unresolved` instead of guessing when the complete source does
not determine the answer.

Return exactly one game-level JSON object with numeric game case_index,
environment_id, public_environment_name, and configuration_reports.
configuration_reports is an array containing exactly one object for each
supplied configuration, with no duplicates or omitted or extra
configurations. Each array member uses this schema (case_index is the
configuration case_index, not the game case_index):

```json
{
  "case_index": "{case_index}",
  "public_environment_name": "{public_environment_name}",
  "environment_id": "{environment_id}",
  "difficulty": "{difficulty}",
  "interaction_mode": "{interaction_mode}",
  "label": "yes | no | unresolved",
  "controls_available": [],
  "autonomous_processes": [],
  "grouped_control_modes": [],
```
```

```
  "delta_ms": null,
  "window_ms": null,
  "clause_i": {"holds": true, "reason": ""},
  "clause_ii": {"holds": true, "reason": ""},
  "clause_iii": {"holds": true, "reason": ""},
  "common_visible_process": "",
  "pre_run": {
    "threshold": 0.5,
    "at_least_half": null,
    "rate": null,
    "basis": "source_structural | seeded_probe | unresolved |
    not_applicable",
    "reason": "",
    "seed_probe": null
  },
  "files_read_completely": [],
  "source_evidence": [
    {"path": "", "lines": "", "claim": ""}
  ],
  "uncertainties": []
}
```

```
Write the single game-level JSON object to the assigned OUTPUT_PATH using
apply_patch. Every configuration_reports member must contain numeric
case_index and difficulty. The game-level object must identify the
assigned game and contain the complete requested configuration set. This
one game report is the only permitted persistent output. Do not edit
puzzle sources, definitions, historical audits, or another report. Do not
inspect other first-pass reviews, existing classification/capability
files, evaluation results, or previous conversations. The audit manifest
is parent-only; the launcher gives you your exact assignment.

You may run read-only source/math probes when needed, with
PYTHONDONTWRITEBYTECODE=1 and without modifying source. For a seed probe,
include seeds, numerator/denominator, and reproducible probe code in
pre_run.seed_probe. No browser or desktop access, external
websites/services, nested agents, or empirical CUA evaluation. A
source-review report may be unresolved; do not guess to finish.

files_read_completely and source_evidence must use repository-relative
paths. Read each complete implementation file needed by the assigned
mechanic, including shared helpers actually used. If a shared file is
long, read it in successive ranges to EOF; keyword searches do not
substitute for full reads. Record concrete code evidence for the exact
difficulty and interaction. Do not make changes to obtain a preferred
label.

Before finishing, load your JSON and check the game identity, exact
requested configuration coverage, numeric IDs, label/clause consistency,
and all cited file paths and line bounds. Shared evidence may be reused
where it applies, but every configuration must remain independently
understandable. A structural check is not evidence that the substantive
classification is correct. Do not read other reviewers' reports or the
audit manifest.
```

I Extended Related Work

This appendix expands on the related work discussion in Section 5.

Benchmarks for interactive visual agents. Existing benchmarks for vision-language agents span web, desktop, mobile, game, and embodied environments. Web benchmarks such as WebArena and

VisualWebArena provide self-hosted websites with realistic tasks (Zhou et al., 2024; Koh et al., 2024), while desktop benchmarks such as OSWorld and Windows Agent Arena evaluate open-ended tasks across operating system applications (Xie et al., 2024; Bonatti et al., 2024). OSWorld 2.0 further extends this setting to 108 long-horizon workflows, where the best agent completes only 20.6% of tasks (Yuan et al., 2026). On mobile, AndroidWorld provides tasks across 20 Android apps (Rawles et al., 2024), and Programming with Pixels and Gym-Anything extend computer-use evaluation to software engineering and to 200 professional software applications (Aggarwal & Welleck, 2025; Aggarwal et al., 2026). Game benchmarks evaluate the same models on complete games, ranging from classic video games (Zhang et al., 2026) and reinforcement learning environments such as NetHack (Paglieri et al., 2025) to platformers, puzzle games, and browser games (Hu et al., 2025a; Zheng et al., 2025a; Ying et al., 2026; Ouyang et al., 2026). ARC-AGI-3 evaluates exploration and goal inference in turn-based abstract games, where humans solve all environments while frontier models solve less than 1% (ARC Prize Foundation, 2026). Embodied benchmarks similarly evaluate agents in simulated homes and physical environments (Savva et al., 2019; Shridhar et al., 2020; Srivastava et al., 2021; Yang et al., 2025c). However, since each task in these benchmarks requires many capabilities at once, low performance does not identify which capability limits the agent.

Several works attempt to attribute performance to individual capabilities. SmartPlay maps each of its six games to a subset of nine agent capabilities (Wu et al., 2023), LVLM-Playground evaluates perception, question answering, rule following, and end-to-end play separately on the same board games (Wang et al., 2025c), and Embodied Agent Interface decomposes embodied decision making into separate modules, arguing that final success rates cannot pinpoint which ability is missing (Li et al., 2024). VS-Bench similarly reports perception, strategic reasoning, and decision making separately (Xu et al., 2026). Other benchmarks vary conditions of the same task, such as difficulty tiers (Davydova et al., 2025; Zheng et al., 2025b), initial states (Zhao et al., 2025), or interface changes between actions (Liu et al., 2026). However, these works either operate through text or turn-based interfaces, or vary a single condition without controlling the capabilities a task requires. In contrast, CUAWORLD-RT annotates the capabilities required by each configuration, and varies difficulty, interaction interface, and real-time setting on the same minigame.

Capability-specific evaluation of VLMs. A large body of work evaluates individual capabilities of VLMs through question answering. For 3D spatial understanding, VSI-Bench, MindCube, 3DSR-Bench, and OmniSpatial evaluate spatial relations, viewpoint changes, and mental models of scenes (Yang et al., 2025a; Wang et al., 2025b; Ma et al., 2024; Jia et al., 2025), while BLINK shows that perception tasks that humans solve easily remain difficult for multimodal models (Fu et al., 2024). For temporal reasoning, TempCompass, TOMATO, and Vinoground evaluate speed, direction, and event order in videos (Liu et al., 2024; Shangguan et al., 2024; Zhang et al., 2024), with TOMATO showing that many questions in earlier video benchmarks can be answered from a single frame. For GUI grounding, SeeClick and ScreenSpot-Pro evaluate whether models can locate screen elements from instructions (Cheng et al., 2024; Li et al., 2025). Finally, PuzzleVQA, AlgoPuzzleVQA, and VisualPuzzles evaluate abstract and algorithmic reasoning over images (Chia et al., 2024; Ghosal et al., 2024; Song et al., 2025), and MMMU and MMMU-Pro evaluate multidisciplinary reasoning over images and text (Yue et al., 2024; 2025). However, in all of these benchmarks the model answers a question about a static input, without having to act on its understanding. Closer to our work, iVISPAR evaluates the spatial reasoning of VLM agents on a sliding-tile puzzle presented in 3D, 2D, and text (Mayer et al., 2025), and VisGym provides 17 interactive environments with configurable difficulty and input representation (Wang et al., 2026b). CUAWORLD-RT complements these works by covering temporal reasoning, exploration and interface understanding, and real-time interaction, and by annotating the capabilities required by each configuration.

Real-time interaction. Acting in dynamic environments requires accounting for the delay between observing the environment and executing an action. In robotics, real-time action chunking and asynchronous inference execute previously planned actions while the next chunk is computed (Black et al., 2025; Shukor et al., 2025), and VLASH conditions the policy on the state predicted at execution time (Tang et al., 2025). Reinforcement learning has similarly studied decision making under action and observation delays (Ramstedt & Pal, 2019; Derman et al., 2021). For computer-use agents, OSWorld-Human measures end-to-end latency and finds that agents take 2.7 to 4.3 times more steps than necessary (Abhyankar et al., 2026). Game benchmarks have taken different approaches to latency. GameWorld pauses the game during inference and reports a separate real-time study

with two models (Ouyang et al., 2026), VideoGameBench runs games in real time and additionally provides a Lite version that pauses the emulator while the agent is thinking (Zhang et al., 2026), and Orak compares paused and real-time play on StarCraft II (Park et al., 2025). Real-Time Reasoning Gym advances text-based games after a fixed number of generated tokens, comparing reactive and planning agents under increasing time pressure (Wen et al., 2026). Our work complements these efforts by evaluating the same minigame and seed in the live and paused settings, and by labeling which configurations require real-time interaction.

Automated environment construction. A growing body of work uses coding agents and LLMs to construct environments for agents. Gym-Anything uses a creation agent and an independent audit agent to convert 200 software applications into computer-use environments (Aggarwal et al., 2026). CUA-Gym separates an agent that constructs environment states from one that writes reward functions (Wang et al., 2026a), and GUI-GENESIS converts real applications into lightweight web environments with executable rewards (Cao et al., 2026). AutoEnv generates 36 text-based environments with 358 levels (Zhang et al., 2025), while AgentGen and OMNI-EPIC generate environments together with tasks of varying difficulty (Hu et al., 2025b; Faldor et al., 2024). Similar pipelines construct software engineering tasks from code repositories (Pan et al., 2025; Yang et al., 2025b) and 3D scenes for embodied agents (Yang et al., 2024). Procedural generation further provides task instances with adjustable complexity, including Procgen games (Cobbe et al., 2019), MiniHack and BabyAI (Samvelyan et al., 2021; Chevalier-Boisvert et al., 2019), and verifiable reasoning problems in Reasoning Gym and Enigmata (Stojanovski et al., 2025; Chen et al., 2025). However, these approaches construct software, text, or game environments, and do not produce interactive visual minigames whose difficulty and interaction interface can be varied while preserving the underlying task. We extend the creator–auditor approach to such minigames, where each minigame has five difficulty levels, two interaction modes, and an oracle solution.

CAPTCHAs and visual puzzles. Many of the seed minigames in CUAWORLD-RT are derived from CAPTCHAs and puzzle games. CAPTCHA research has studied whether automated solvers can pass challenges designed to distinguish humans from bots. Recent works solve reasoning and visual CAPTCHAs with LLMs and VLMs (Deng et al., 2024; Teoh et al., 2025), analyze which types of challenges remain difficult for these models (Wang et al., 2025a; Wu et al., 2025), and train GUI agents on interactive CAPTCHAs (Chen et al., 2026). Open CaptchaWorld evaluates web agents on 20 types of interactive CAPTCHAs, where the best agent achieves 40.0% compared to 93.3% for humans (Luo et al., 2025). Beyond CAPTCHAs, KORGym, DiG-bench, and VGRP-Bench evaluate multi-turn reasoning in text, game, and grid puzzles (Shi et al., 2025; Battleday et al., 2026; Ren et al., 2025). However, these works primarily target security or reasoning in isolation. In contrast, CUAWORLD-RT uses such challenges as seed minigames, and adds controlled capability requirements, difficulty levels, interaction interfaces, and real-time settings.