PLAYSUITE: A LARGE-SCALE BENCHMARK FOR INTERACTIVE VISUAL INTELLIGENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advances in multimodal foundation models yield strong performance on static perception and reasoning benchmarks, yet largely overlook a central aspect of intelligence: acting competently in dynamic environments over extended time horizons. We introduce PLAYSUITE, a large-scale benchmark for evaluating interactive visual intelligence across more than 5K open-source video games curated from *PyWeek* and *itch.io*. Spanning diverse genres and engines, including Pygame, HTML5, Godot, and Unity, these independent games are largely out-of-distribution for current models, reducing the likelihood that success can be achieved by retrieving memorized walkthroughs or web-scale training artifacts. To enable scalable evaluation across heterogeneous titles, we develop a unified closed-loop interaction framework optimized for HPC clusters alongside a Video-LLM-as-a-judge protocol that maps observable gameplay milestones to standardized progress levels. We evaluate fourteen recent open models spanning vision-language models, computer-use agents, and vision-language-action models. Our results reveal a considerable perception-action gap: despite strong reasoning capabilities, current models struggle to maintain sustained progress and exhibit recurring failures in spatial grounding, action execution, and self-correction. PLAYSUITE provides a reproducible and extensible testbed for measuring progress from visual perception to goal-directed interaction, and a foundation for developing models that can act, adapt, and generalize in dynamic visual environments.

1 INTRODUCTION

Recent advances in multimodal foundation models have rapidly expanded their capabilities beyond static visual understanding toward multimodal reasoning and interaction. Modern models can operate graphical user interfaces across desktop, mobile, and web platforms and control embodied agents over extended tasks (Bai et al., 2025b; Zhou et al., 2026; Qin et al., 2025; Physical Intelligence et al., 2025). Yet many multimodal benchmarks still emphasize perception and reasoning over static inputs (Yue et al., 2024; Liu et al., 2024). Real-world intelligence (Lake et al., 2017), by contrast, is inherently interactive and temporal: actions alter future observations, decisions must adapt to feedback, and agents must recover from mistakes over time. Recent agent benchmarks (Xie et al., 2024b; Zhou et al., 2024; Zheng et al., 2025; Zhang et al., 2025a) increasingly evaluate such closed-loop capabilities, but doing so across large and diverse sets of environments remains challenging. This motivates a central question: *how well do current multimodal models translate their perceptual and reasoning capabilities into goal-directed interaction?*

Games provide a particularly rich testbed for this question. They combine aspects of computer use with spatial reasoning, temporal dynamics, and goal-directed behavior in interactive environments. Recent frontier systems have demonstrated sustained interaction in complex games from visual observations and natural-language goals (Tan et al., 2024; Bolton et al., 2025; Anthropic, 2026; OpenAI, 2026), but these demonstrations remain concentrated in a small number of environments. Existing game benchmarks similarly span only a limited number of distinct games (Guss et al., 2019; Fan et al., 2022; Zheng et al., 2025; Paglieri et al., 2025; Park et al., 2026; Zhang et al., 2025a; Wang et al., 2026), leaving it unclear how broadly these capabilities generalize. We introduce PLAYSUITE, a large-scale benchmark for interactive visual intelligence across 5,734 distinct video games, representing more than two orders of magnitude more game environments (> 100x) than prior multi-game benchmarks. These environments are drawn from a larger pool of approximately

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.

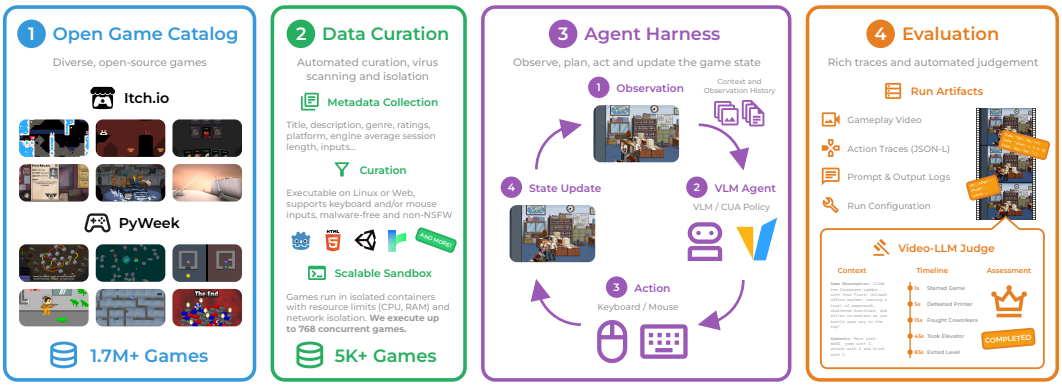


Figure 1: **The PLAYSUITE benchmark.** Our open-source benchmark for interactive visual intelligence comprises four stages: (1) Open Game Catalog: A diverse library of 1.7M video games sourced from *itch.io* and *PyWeek*. (2) Data Curation: An automated filtering and sandboxing pipeline that distills the catalog into approximately 17.5K executable, safe, and reproducible environments, of which 5,734 are used in our evaluation. (3) Interactive Agent Harness: A high-performance, closed-loop system where foundation models interact via standard keyboard/mouse interfaces for optimized large-scale HPC evaluation. (4) Evaluation Artifacts: A comprehensive suite of gameplay videos, action traces, and a Video-LLM-as-a-judge mechanism to quantify goal-directed performance.

17.5K launchable games validated by our pipeline, providing substantial room for future expansion. An illustration of PLAYSUITE is presented in Figure 1.

We construct PLAYSUITE from games hosted on PyWeek and Itch.io. PyWeek contributes lightweight Python-based games, while itch.io substantially expands the visual, mechanical, and engine diversity of the benchmark with games developed using Unity, Godot, GameMaker, RPG Maker, and other frameworks. Together, these ecosystems provide a broad distribution of independently developed, human-designed environments, ranging from simple 2D games to visually complex 3D and physics-driven settings. Evaluating agents across thousands of heterogeneous games introduces two key challenges: scalable interaction and comparable evaluation. Different engines expose disparate execution and control interfaces, motivating a common interaction layer that enables agents to operate across games using standard keyboard and mouse actions. At the same time, games do not share a common reward or scoring interface. We therefore introduce a progress-based Video-LLM-as-a-judge methodology (Zheng et al., 2023; Lin et al., 2023), which extracts observable gameplay milestones and maps them to progress levels. Together, these components enable scalable and comparable evaluation across thousands of diverse game environments.

Using this new infrastructure, we benchmark fourteen recent open models spanning general-purpose vision-language models (VLMs), computer-use agents (CUAs), and vision-language-action models (VLAs). Despite strong visual reasoning capabilities, current models achieve meaningful gameplay progress on only a minority of games. Qualitative inspection of gameplay traces further identifies recurring difficulties in spatial grounding, control selection, action execution, and adaptation over time, highlighting the gap between interpreting an environment and acting effectively within it.

Nearly a decade after OpenAI Universe (OpenAI, 2016), PLAYSUITE revisits the vision of broad-environment evaluation in the setting of pretrained multimodal agents that can be deployed across heterogeneous games without environment-specific training. We release PLAYSUITE as an open and extensible benchmark, including environment wrappers, evaluation protocols, and high-performance cluster tooling[1]. In summary, our contributions are threefold: (1) a large-scale benchmark spanning over 5K heterogeneous interactive visual environments; (2) a unified, HPC-optimized infrastructure for closed-loop evaluation across diverse game engines; and (3) a scalable milestone-based Video-LLM evaluation protocol for measuring gameplay progress without environment-specific rewards. The benchmark is designed to grow as new environments and increasingly capable interactive models emerge.

[1]Our code is available at https://anonymous.4open.science/r/PlaySuite-301D/.

Table 1: **Related work.** Comparison of PLAYSUITE against existing video game benchmarks.

Benchmark	Domain	# Games	Models				Evaluation		
			LLM	VLM	VLA	CUA	Metric	HPC	Dynamic
GameBench (Costarelli et al., 2024)	Text	9	✓	✗	✗	✗	Game score	✗	✗
GameArena (Hu et al., 2025)	Text	6	✓	✗	✗	✗	Win rate	✗	✗
Balrog (Paglieri et al., 2025)	2D	6	✓	✓	✗	✗	Task completion	✗	✗
ING-VP (Zhang et al., 2024)	2D	6	✗	✓	✗	✗	Task completion	✗	✗
V-MAGE (Zheng et al., 2025)	2D/3D	5	✗	✓	✗	✗	ELO ranking	✗	✗
VS-Bench (Xu et al., 2026b)	2D/3D	10	✗	✓	✗	✗	Game score	✗	✗
Orak (Park et al., 2026)	2D/3D	12	✓	✓	✗	✗	Game score	✗	✗
VisGym (Wang et al., 2026)	2D/3D	17	✗	✓	✗	✗	Success rate	✗	✗
VideoGameBench (Zhang et al., 2025a)	2D/3D	23	✗	✓	✗	✗	Progress tracking	✗	✓
PLAYSUITE	2D/3D	**5,734**	✗	✓	✓	✓	Automated judge	✓	✓

2 RELATED WORK

Foundation models in interactive environments. LLMs have shifted from passive information retrieval toward active agency in closed-loop systems (Wang et al., 2024). Early work focused of reasoning-chain planning, decomposing natural language goals into executable sub-steps (Huang et al., 2022; Xie et al., 2023), but requiring recursive feedback to adapt to environment shifts (Yao et al., 2023; Wang et al., 2023). With the emergence of multimodal foundation models, research has shifted toward unified perception-action policies (Mokaria et al., 2026). General VLMs (Bai et al., 2025b; Gemma Team, 2025) ground linguistic instructions in raw visual pixels, while specialized GUI agents (Qin et al., 2025; Lin et al., 2025; Wu et al., 2025) predict control tokens directly. Despite these advancements, models still struggle with the fundamental challenges of interactive intelligence: translating granular perception into effective action, maintaining long-horizon planning, and exhibiting robust spatial reasoning in dynamic environments. This persistent gap highlights the need for an open-source, large-scale, and heterogeneous benchmark to evaluate and refine these capabilities across interactive scenarios, moving beyond narrowly defined domains toward truly general-purpose agency.

Game-based benchmarks. To address performance saturation and data contamination in traditional evaluations, the field has pivoted toward dynamic, interactive suites that require real-time decision-making (Wong et al., 2025; Chen et al., 2025). Digital navigation benchmarks (He et al., 2024; Zhang et al., 2025b; Zhou et al., 2024; Yao et al., 2022) assess tool-use and GUI automation, while embodied AI frameworks focus on atomic physical interactions (Yang et al., 2025). Games have emerged as a primary testbed due to their inherent complexity and high-frequency temporal dynamics. Early benchmarks were restricted to 2D or grid-based environments (Costarelli et al., 2024; Nasir et al., 2024; Hafner, 2022; Küttler et al., 2020; Wang et al., 2025a; Chollet et al., 2025). While these provided essential testbeds for logic and planning, they failed to capture the spatial nuances of the real world (Zhang et al., 2024; Wang et al., 2025a). Recent 3D environments require agents to process perspective, depth, and occlusion (Qiao et al., 2023; Hu et al., 2025; Xie et al., 2024a), but remain constrained to single titles (Fan et al., 2022; Guss et al., 2019), or small sets of games requiring bespoke per-title engineering and hand-crafted rewards (Zheng et al., 2025; Paglieri et al., 2025; Park et al., 2026; Ying et al., 2026). Modern modular harnesses decouple perception and reasoning to tackle diverse games without domain-specific engineering (Zhang et al., 2025c;a; Lu et al., 2025; ARC Prize Foundation, 2026), often adopting the LLM/VLMs-as-a-judge paradigm to assess agentic competence (Zheng et al., 2023; Pan et al., 2024; Lin et al., 2023). PLAYSUITE synthesizes these advancements into a scalable paradigm, replacing bespoke engineering with a unified, judge-based assessment across a large variety of game mechanics. A comprehensive comparison of the existing video games benchmarks against PLAYSUITE is provided in Table 1, highlighting our contributions in scale, infrastructure, and dynamic evaluation.

3 PLAYSUITE CONSTRUCTION

PLAYSUITE is a large-scale benchmark designed to evaluate general-purpose interactive agents across a vast distribution of game environments. Unlike existing benchmarks that rely on domain-



Figure 2: **Examples from the PLAYSUITE game corpus.** PLAYSUITE contains a broad collection of independently developed games with varied visual styles, mechanics, camera perspectives, and controls. These examples illustrate the range of settings agents must handle, from 2D platform and puzzle games to racing, exploration, shooting, and 3D navigation environments.

specific simulators or restricted game lists, PLAYSUITE provides a unified interaction framework that standardizes communication across heterogeneous engines. Each game is executed in an isolated sandbox using a virtual display, enabling efficient and scalable execution across GPU clusters without requiring physical displays. To ensure the benchmark captures a broad spectrum of common-sense interactive intelligence across different tasks, we curated a library of 5,734 open-source games from PyWeek and Itch.io. These platforms offer a unique diversity of 2D and 3D dynamics that are designed for human players to learn intuitively within seconds, making them ideal for testing the zero-shot performance of multimodal foundation models, see examples in Figure 2.

Filtering and quality control. From an initial crawl, we selected games with a community rating $> 3.0/5.0$ and at least 5 ratings to ensure playability and coherent goal design. We filtered for platform compatibility, retaining only Linux and Web-based games, and strictly excluded NSFW content and games with broken assets through a multi-stage automated and manual verification process. To ensure secure execution at scale, we implemented a custom safety triage pipeline that assigns a risk score to each game based on structural and metadata signals. High-risk archives were quarantined, while all others were executed in a network-isolated sandbox; see Appendix A.1.

Taxonomy and categorization. The collection spans nine distinct genres: `Action`, `Adventure`, `Puzzle`, `Shooter`, `Simulation`, `Survival`, `Strategy`, `Fighting`, `Racing`. Our taxonomy was derived from a cross-analysis of prior work on game-based AI (Lee et al., 2014; Hu et al., 2024) and frequent tags used by the developer community. This ensures that the benchmark covers diverse forms of interaction and decision-making. For games from *itch.io*, we used existing metadata tags. For *PyWeek* games, which lack native tagging, we employed a frontier LLM to categorize each title based on its source code documentation and README files. Further details are provided in Appendix A.2. This diverse distribution ensures that models cannot succeed through narrow heuristics tailored to a single genre. Genre counts are shown in Appendix Figure 8.

Evaluation pipeline. We formalize the evaluation as a sequential decision-making process where a model functions as an agent interacting with a game environment $\mathcal{E}$. At each time step $t \in [0, T]$, the model receives an input $\mathcal{I}_t = \langle p, o_t, h_t \rangle$, where p is a unified prompt template with game-specific metadata, $o_t \in \mathbb{R}^{H \times W \times 3}$ is the raw pixel frame, and $h_t = \{(o_{t-k}, a_{t-k}), \ldots, (o_{t-1}, a_{t-1})\}$ is an optional temporal history of previous observations and actions. Crucially, models are not provided with solution strategies; they must autonomously infer mechanics and objectives from visual feedback and metadata. The model generates a structured output $\mathcal{O}_t = \langle r_t, a_t \rangle$, consisting of a reasoning trace r_t and an action a_t. This defines a policy $\pi(\mathcal{O}_t \mid p, o_t, h_t)$ that must generalize across the transition dynamics of thousands of environments. By requiring the agent to bridge the gap between seeing the current state o_t and doing the correct action a_t without bespoke tuning, this protocol shifts the focus from static recognition to functional, interactive intelligence under varied control dynamics.

Milestone-based evaluation. A key challenge in evaluating model performance across thousands of heterogeneous environments is the lack of a standardized reward signal. We address this with a scalable Video-LLM-as-a-judge framework that measures progress from observable gameplay events. The judge reports timestamped evidence of events such as score increases, entry into a new area, checkpoint or objective completion, and game completion. A fixed rule then maps these observations to five ordered progress levels: `None` (0), `Minimal` (1), `Partial` (2), `Substantial` (3), and `Completed` (4). Partial progress requires either reaching a new area or level, or increasing the score; Substantial progress requires completing an intermediate objective or observing both forms of

progress; and Completed requires explicit visual evidence of game completion. Minimal progress is assigned when player control is evident through at least two distinct interactions.

Our evaluation setup uses Qwen3.5-9B with the milestone prompt, structured JSON decoding, and video sampled at 1 fps. We report the mean Progress Score across valid evaluations and the fraction of runs reaching at least Partial progress. The mean Progress Score summarizes ordinal outcomes across the benchmark and should not be interpreted as the fraction of a game completed. The complete prompt, schema, and scoring implementation are provided in Appendix A.7. To complement automated evaluation, we provide an inspection toolkit that overlays gameplay videos with the agent's generated responses and actions together with the judge's timestamped assessments. This allows us to audit evaluations, inspect model and judge behavior, and provide human feedback or corrections starting from the automated analysis rather than reviewing trajectories from scratch.

Judge validation. We validate Qwen3.5-9B against six human annotators on 60 gameplay videos, comprising 45 *itch.io* and 15 *PyWeek* runs spanning 14 model presets and both history conditions. Human-only Krippendorff's α is 0.624, while including the judge as a seventh rater yields $\alpha = 0.605$. In pairwise comparisons, the judge's progress rating is within one level of the human rating in 90.4% of cases, compared with 93.0% for human–human pairs. We additionally evaluate Qwen3.5-27B, Gemma-4-31B-it, and Molmo2-8B. Qwen3.5-27B achieves the highest agreement with humans at 91.6% within one level, compared with 90.4% for Qwen3.5-9B. We therefore use Qwen3.5-9B as the default judge, providing comparable agreement while remaining more practical for evaluation at scale. See Appendix A.7.3 for full results.

4 PLAYSUITE EXPERIMENTAL ARCHITECTURE

To maintain a scalable and engine-agnostic evaluation across the environments, we developed a standardized experimental architecture. This setup ensures that model success is a product of general interactive intelligence rather than per-game optimization.

Input and output configuration. For models evaluated through the standard prompt, we use shared system and user templates that insert the game title, taxonomic genre, available developer-provided metadata or instructions, the current visual observation, and optional textual history of previous interactions. Unlike benchmarks that provide predefined game-specific control mappings (Zheng et al., 2025; Park et al., 2026; Zhang et al., 2025a), PLAYSUITE does not assume prior knowledge of a game's mechanics. When developer documentation specifies the controls, the prompt instructs the model to follow them. When controls are absent, the model must infer them through visual context and exploration, aided by fallback keyboard controls and a set of decision heuristics for choosing between keyboard and mouse actions. The prompt also defines the supported action grammar and instructs the model to change strategy when previous actions produce no visible progress. This setup approximates how a player approaches an unfamiliar game with limited documentation while ensuring that the model's actions remain executable by the evaluation harness.

The action grammar includes key presses, simultaneous key combinations, timed holds, absolute and relative mouse movement, left/right clicks, scrolling, and repeated clicks. Dragging is unsupported, so we exclude games whose controls mention dragging. The standard template requires to generate a structured response with Observation, Reasoning, and Action fields. This includes a reasoning trace, which incorporates both a high-level summary of the current observation and the underlying logic for the subsequent move, followed by the next predicted action. The harness parses the action for execution and retains the complete raw response, including the observation and reasoning fields when present. Models with native computer-use or visual-action interfaces retain their model-specific prompts, output formats, and coordinate conventions. Model-specific adapters translate their predicted actions into the shared keyboard and mouse interface. More details about the prompt templates, model assignments, and parsers can be found in Appendix A.5.

Temporal context and history. To facilitate temporal reasoning and long-horizon planning, we incorporate a rolling history of previous interactions, $h_t = \{(o_{t-k}, a_{t-k}), \ldots, (o_{t-1}, a_{t-1})\}$. This history is intended to help the model capture causal relationships between actions and environmental transitions. We empirically evaluated history lengths of $k \in \{0, 3, 5, 10\}$ to determine the optimal balance between temporal awareness and context window efficiency.

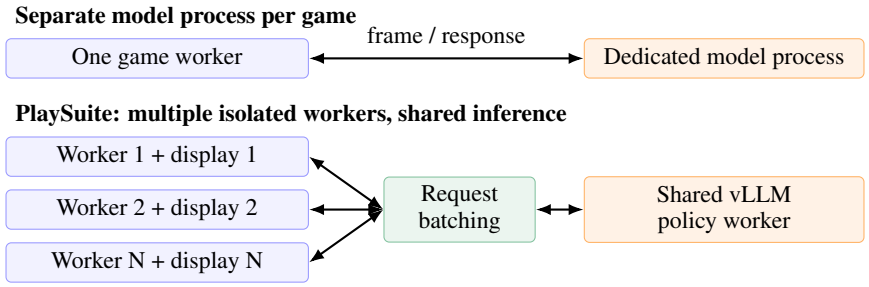


Figure 3: **Batched multi-worker execution.** Comparison between a sequential one-game-per-model-process evaluation pattern and the PLAYSUITE batched architecture. Multiple sandboxed game workers run on one node, each with an isolated virtual display, while a shared vLLM-backed inference worker batches ready frame-and-prompt requests on the GPU.

Baseline models. We benchmark 14 models spanning three broad families of interactive vision models: general-purpose VLMs, CUAs, and VLAs. Our evaluation includes VLMs such as *Gemma-4-26B-A4B-it*, *Gemma-4-E4B-it*, *Qwen3-Omni-30B-A3B-Instruct*, *Qwen3-VL-8B-Instruct*, *Qwen2.5-VL-7B-Instruct*, *Gemma-3-12B-it*, *Qwen2.5-Omni-7B*, and *InternVL2.5-8B*, alongside models explicitly designed for computer interaction, including *EvoCUA-8B*, *UI-TARS-1.5-7B*, *OpenCUA-7B*, *GUI-Owl-1.5-8B-Instruct*, *OS-Atlas-Pro-7B*, and *ShowUI-2B*. All models are run with a temperature 0.5 and a maximum output length of 512 tokens. By including these varied architectures, we provide a broad comparison of how different modeling priors handle the zero-shot challenges of interactive game environments. We restrict evaluation to open-weight models, thereby excluding proprietary systems such as Cradle (Tan et al., 2024) and SIMA 2 (Bolton et al., 2025). Appendix A.4 lists the exact model identifiers, prompt templates, and adapters.

Execution and interface setup. The evaluation pipeline is designed for high-throughput, display-independent execution on HPC infrastructure. Across both subsets, game progression is suspended during model inference. This decouples game time from model-serving latency and ensures that each selected action is applied to the state observed by the model. To accommodate heterogeneous game engines, the pipeline supports two execution formats, detailed in Appendix A.6. In both, model-specific adapters map generated actions to a shared action representation, after which a unified virtual input driver emits either low-level `pygame` events or system-level keyboard and mouse events.

Batched multi-worker execution. Figure 3 illustrates how PLAYSUITE parallelizes closed-loop evaluation on HPC infrastructure. A naive closed-loop evaluation launches one game and one model process per node, duplicating model weights, under-utilizing the GPU during game rendering and environment startup. In contrast, we launch many independent game workers on a single HPC node. Each worker is assigned a unique virtual display, an isolated output directory, and a per-worker sandbox scratch space. Workers capture frames, construct model prompts, inject parsed keyboard or mouse actions, and write the resulting video and action trace.

Model inference is managed by a shared vLLM worker. Lightweight game workers send frame-and-prompt requests to this server over a Unix socket so that game processes can remain network-isolated. The inference worker holds a single copy of the model in GPU memory, batches ready requests, and returns raw model outputs to the workers, where model-specific parsers translate text into the common action interface. This design amortizes model loading across games, improves GPU utilization, and separates CPU-bound game execution from GPU-bound inference. In a controlled evaluation with Qwen3-VL-8B on one H100, eight concurrent workers increase throughput from 6.7 to 42.4 evaluations per GPU-hour. Additional measurements of launch, gameplay, and judging throughput are provided in Appendix A.8.

5 PLAYSUITE BENCHMARK RESULTS

We evaluate fourteen open models on PLAYSUITE across 5,630 *itch.io* games and 104 *PyWeek* games. We report the mean Progress Score on the five-level scale from *None* (0) to *Completed* (4), together with the percentage of runs reaching at least *Partial* progress.

Table 2: **Main benchmark results across PLAYSUITE .** Mean Progress Score under the history ($k = 3$) and no-history conditions, together with the percentage of no-history runs reaching at least *Partial* progress, across 5,630 *itch.io* games and 104 *PyWeek* games. Progress is measured on the five-level ordinal scale from *None* (0) to *Completed* (4). Best results are shown in bold.

	itch.io			*PyWeek*		
Model	**History**	**No History**	**Partial+**	**History**	**No History**	**Partial+**
Vision-Language Models (VLMs)						
Qwen3-Omni-30B-A3B	0.62	**0.84**	**19.5%**	0.89	**1.26**	**37.0%**
Qwen2.5-VL-7B	0.41	0.82	17.7%	0.67	1.18	31.7%
Gemma-4-26B-A4B-it	**1.02**	0.71	17.3%	**1.37**	0.96	24.8%
Qwen3-VL-8B	0.64	0.70	15.5%	0.73	1.07	28.3%
Qwen2.5-Omni-7B	0.27	0.67	13.2%	0.53	0.95	24.5%
Gemma-4-E4B-it	0.77	0.66	13.8%	0.94	0.96	25.7%
InternVL2.5-8B	0.26	0.54	10.0%	0.44	0.91	27.7%
Gemma-3-12B-it	0.47	0.49	10.1%	0.67	0.75	22.8%
Computer-Use Agents (CUAs)						
UI-TARS-1.5-7B	0.65	0.71	15.1%	0.81	1.10	27.6%
EvoCUA-8B	0.81	0.67	15.1%	1.10	0.99	25.7%
OpenCUA-7B	0.63	0.59	12.0%	0.85	0.97	29.3%
GUI-Owl-1.5-8B	0.56	0.56	12.1%	0.89	0.79	21.0%
Vision-Language-Action Models (VLAs)						
OS-Atlas-Pro-7B	0.36	0.51	10.8%	0.50	0.63	15.0%
ShowUI-2B	0.34	0.43	8.4%	0.47	0.72	21.2%

Overall performance. Table 2 shows that current multimodal agents make meaningful progress on only a minority of games in PLAYSUITE . In the no-history condition, Qwen3-Omni attains the highest mean Progress Score on both corpora, with 0.84 on *itch.io* and 1.26 on *PyWeek*, while reaching at least *Partial* progress in 19.5% and 37.0% of runs, respectively. With interaction history, Gemma-4-26B achieves the highest mean score on both corpora, reaching 1.02 on *itch.io* and 1.37 on *PyWeek*. Performance also does not separate cleanly by model family: CUAs remain competitive with similarly sized VLMs, while the evaluated VLAs generally trail the strongest VLM and CUA models.

Qualitative failure modes. Inspection of individual trajectories reveals that low progress can arise from failures at multiple stages of the perception–action loop. Models may misread on-screen text or entities, make inaccurate clicks, or fall into repetitive action loops. Other observed failures include using controls unsupported or unmapped by the game, instruction-following failures, passive waiting without strategic cause, context pollution from interaction history, and state mismanagement in which previously observed information is contradicted. Figure 4 illustrates a few failure modes within a single trajectory.

Table 3: **Progress by genre on *itch.io*.** Mean Progress Score and Partial+ rate in the no-history condition for genres with at least 60 games. PyWeek and history results are reported in Appendix A.9.1.

Genre	N	Mean	Partial+
Simulation	506	0.893	27.1%
Strategy	223	0.858	25.2%
Shooter	205	0.771	20.4%
Survival	171	0.770	21.7%
Racing	72	0.705	18.9%
Action	958	0.620	14.4%
Puzzle	1,271	0.606	10.0%
Adventure	2,184	0.544	9.5%

Performance across game genres. Mean progress varies substantially across game genres. As shown in Table 3, among genres containing at least 60 games, models achieve the highest mean progress on Simulation and Strategy games, while Adventure and Puzzle games yield the lowest scores. Despite these differences, the relative ordering of models is largely consistent across genres. The correlation between each genre-specific model ranking and the overall ranking ranges from Spearman $\rho = 0.88$ to 0.99, indicating that models that perform well overall generally also perform well across individual

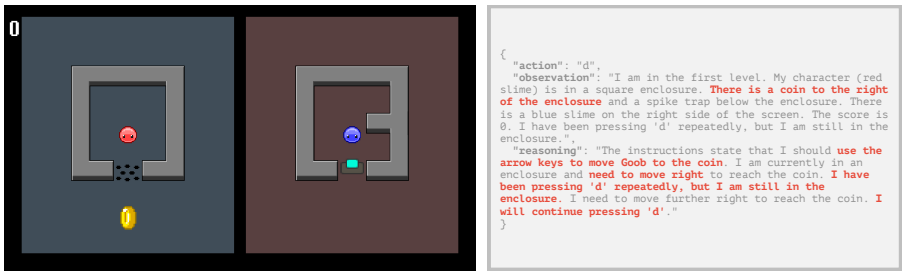


Figure 4: **Failure mechanisms for Gemma-3 on Goob.** The model correctly perceives the position of the spike and blue slime, but incorrectly places the coin on the right side of the screen, producing a perception error. Although it recognizes that the instructions specify the arrow keys for movement, it repeatedly presses "d" despite receiving no progress, illustrating failures in control selection, instruction following, and recovery from repetitive actions.

genres. Per-model breakdowns, PyWeek results, and performance with history are provided in Appendix A.9.1.

Effect of action frequency. To test whether performance is limited by the 0.33,Hz interaction rate, we repeat the *PyWeek* evaluation at 3,Hz. Mean Progress Score increases from 0.862 to 0.926, indicating a modest benefit from more frequent control. This improvement occurs with and without history, with no clear evidence that action frequency changes the effect of history. See Appendix A.9.2.

How many games are needed? To assess how benchmark size affects model comparisons, we repeatedly sample games from the full *itch.io* corpus and compare the resulting no-history model ranking with the full-corpus ranking. With 50 games, the median rank correlation is 0.846, rising to 0.937 with 200 games, 0.969 with 500 games, and 0.987 with 2,000 games. Fine-grained ordering is considerably less stable: the highest-ranked model is recovered in 49.0% of 50-game samples, 55.5% at 200 games, 70.0% at 500 games, and 87.5% at 2,000 games, while the full-corpus top-three set is recovered in only 26.0% of 200-game samples and 45.5% even at 2,000 games. These results show that relatively small samples recover the broad model ordering, whereas substantially larger samples are required to reliably distinguish closely performing models. Accordingly, we use a 200-game *itch.io* subset as a practical scale for the following additional controlled analyses.

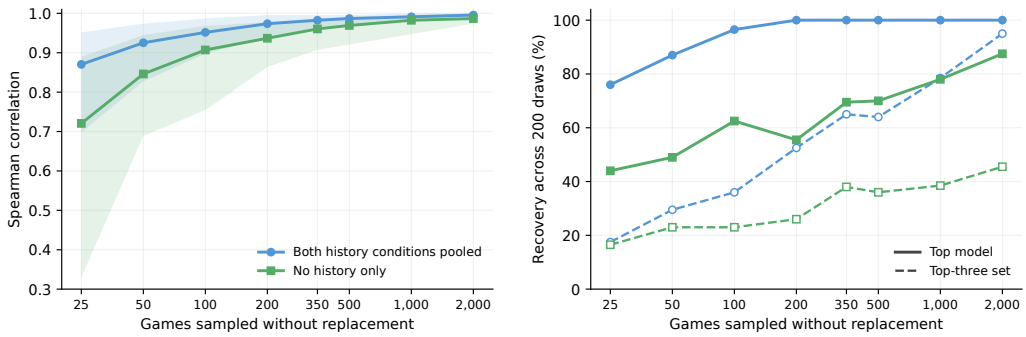


Figure 5: **Leaderboard stability as a function of sampled *itch.io* games.** Curves show how model rankings from randomly sampled game subsets compare with the full *itch.io* corpus ranking. Broad model ordering stabilizes with a few hundred games, whereas substantially larger samples are needed for fine-grained comparisons among the highest-ranked models.

Impact of prompting strategies. We test whether explicit strategy traces or few-shot examples improve the eight VLMs evaluated with a shared prompt format, allowing prompt changes to be compared without model-specific interface differences. Neither modification improves average performance, with few-shot prompting reducing progress on both subsets. See Appendix A.9.3.

Impact of temporal history. To investigate the role of temporal context, we evaluate model performance across history lengths $k \in \{0, 3, 5, 10\}$ on *PyWeek* and the *itch.io* 200 subset. As illustrated in Figure 6, our results reveal a counterintuitive trend: no history yields the highest average progress on both corpora. Increasing history from three to five interactions provides no clear benefit, while extending it to ten interactions reduces progress further. Individual models can nevertheless benefit from history, as shown in Table 2. Full results for both subsets are provided in Appendix A.9.4.

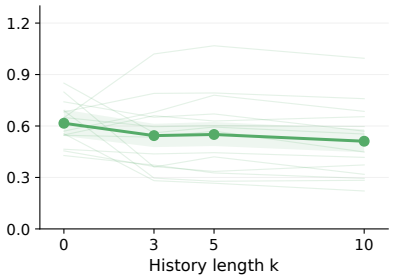


Figure 6: **Impact of history length on model performance** Mean Progress Score across history lengths on the 200-game itch subset. Faint lines show individual models and the bold line shows the mean across models.

Cost-performance analysis Evaluation costs vary substantially across models due to model size, serving configuration, and generation length. Higher progress can come at sharply increasing cost: Qwen3-Omni improves mean progress over Qwen2.5-VL, but requires 105.5 versus 59.9 H100 GPU-hours per 1,000 games. Figure 16 in Appendix A.9.5 illustrates this cost–performance trade-off.

6 Conclusion

We introduce PlaySuite , a large-scale benchmark for evaluating interactive visual intelligence in open-source video games. By combining a heterogeneous game corpus, a unified closed-loop interaction harness, scalable cluster execution, and a Video-LLM-as-a-judge evaluation protocol, PlaySuite enables systematic measurement of model behavior in settings where agents must translate perception and reasoning into interaction. Strikingly, capabilities that appear strong in static visual reasoning and computer-use settings do not reliably transfer to gameplay: across fourteen open VLMs, CUAs, and VLAs, meaningful progress remains limited to a minority of games, with recurring failures in spatial grounding, control selection, action execution, passive waiting, and the effective use of temporal context.

Recent frontier systems are beginning to demonstrate substantial competence in individual games, making it increasingly important to evaluate whether such capabilities generalize across broad distributions of interactive environments. PlaySuite provides a framework for studying this question at scale: our pipeline currently validates approximately 17.5K executable, safe, and reproducible game environments, of which 5,734 are used in the present evaluation. In this sense, PlaySuite revisits the broad-environment vision of OpenAI Universe (OpenAI, 2016) at a time when evaluating generalization across thousands of interactive environments is becoming practically meaningful. As models become increasingly capable of interacting with and generating functional interactive environments, scalable evaluation over large and evolving environment distributions will become increasingly important.

Limitations. Our evaluation focuses on zero-shot inference and therefore measures out-of-the-box generalization rather than maximum performance after fine-tuning or RL. The closed-loop protocol removes model-inference latency from game dynamics, so PlaySuite does not measure real-time reaction speed. Our milestone-based evaluation captures observable progress at scale but does not directly measure strategic efficiency, reasoning quality, or mastery of underlying game mechanics. Finally, because the benchmark uses publicly available games, source code, and metadata, data contamination cannot be ruled out; incorporating newly released games can reduce, but not eliminate, this risk.

AI use statement

We used generative AI tools for software development and debugging, experimental infrastructure design, literature search, brainstorming, and drafting and refining scientific text, tables, and figures. AI assistance was particularly useful in developing and troubleshooting the heterogeneous game-evaluation infrastructure, including virtual-display and browser instrumentation, game-process control, action parsing, and scalable execution. All AI-assisted code, methodological suggestions, literature references, and writing were reviewed, tested, or verified by the authors. Generative models used within the benchmark itself, including automated categorization and Video-LLM evaluation, are

separately described and validated in the paper. We take responsibility for the final content of this work.

REFERENCES

Anthropic. Claude fable 5 and claude mythos 5. Anthropic, June 2026. June 9, 2026.

ARC Prize Foundation. ARC-AGI-3: A new challenge for frontier agentic intelligence. *arXiv preprint arXiv:2603.24621*, 2026.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025a.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2.5-VL technical report. *arXiv preprint arXiv:2502.13923*, 2025b.

Adrian Bolton, Alexander Lerchner, Alexandra Cordell, Alexandre Moufarek, Andrew Bolt, Andrew Lampinen, Anna Mitenkova, Arne Olav Hallingstad, Bojan Vujatovic, Bonnie Li, et al. Sima 2: A generalist embodied agent for virtual worlds. *arXiv preprint arXiv:2512.04797*, 2025.

Simin Chen, Yiming Chen, Zexin Li, Yifan Jiang, Zhongwei Wan, Yixin He, Dezhi Ran, Tianle Gu, Haizhou Li, Tao Xie, et al. Recent advances in large language model benchmarks against data contamination: From static to dynamic evaluation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2025.

Zhe Chen, Weiyun Wang, Yue Cao, Yangzhou Liu, Zhangwei Gao, Erfei Cui, Jinguo Zhu, Shenglong Ye, Hao Tian, Zhaoyang Liu, et al. Expanding performance boundaries of open-source multimodal models with model, data, and test-time scaling. *arXiv preprint arXiv:2412.05271*, 2024.

Francois Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers, and Henry Pinkard. ARC AGI 2: A new challenge for frontier ai reasoning systems. *arXiv preprint arXiv:2505.11831*, 2025.

Anthony Costarelli, Mat Allen, Roman Hauksson, Grace Sodunke, Suhas Hariharan, Carlson Cheng, Wenjie Li, Joshua M Clymer, and Arjun Yadav. Gamebench: Evaluating strategic reasoning abilities of LLM agents. In *Language Gamification Workshop*, 2024.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. MineDojo: Building open-ended embodied agents with internet-scale knowledge. In *Advances in Neural Information Processing Systems*, volume 35, pp. 18343–18362, 2022.

Gemma Team. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Gemma Team. Gemma 4 technical report, 2026. URL https://arxiv.org/abs/2607.02770.

William H Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. MineRL: a large-scale dataset of minecraft demonstrations. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 2442–2448, 2019.

Danijar Hafner. Benchmarking the spectrum of agent capabilities. In *International Conference on Learning Representations*, 2022.

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

Lanxiang Hu, Qiyu Li, Anze Xie, Nan Jiang, Ion Stoica, Haojian Jin, and Hao Zhang. GameArena: Evaluating LLM reasoning through live computer games. In *International Conference on Learning Representations*, 2025.

Sihao Hu, Tiansheng Huang, Gaowen Liu, Ramana Rao Kompella, Fatih Ilhan, Selim Furkan Tekin, Yichang Xu, Zachary Yahn, and Ling Liu. A survey on large language model-based game agents. *arXiv preprint arXiv:2404.02039*, 2024.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pp. 9118–9147, 2022.

Itch.io. A simple way to find, download and distribute indie games online. https://itch.io. Accessed: 2026-04-28.

Heinrich Küttler, Nantas Nardelli, Alexander Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. In *Advances in Neural Information Processing Systems*, volume 33, pp. 7671–7684, 2020.

Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40:e253, 2017.

Jin Ha Lee, Natascha Karlova, Rachel Ivy Clarke, Katherine Thornton, and Andrew Perti. Facet analysis of video game genres. In *Proceedings of the iConference 2014*, pp. 125–139, 2014. URL https://api.semanticscholar.org/CorpusID:46322261.

Haowei Lin, Zihao Wang, Jianzhu Ma, and Yitao Liang. Mcu: A task-centric framework for open-ended agent evaluation in minecraft. In *Second Agent Learning in Open-Endedness Workshop*, 2023.

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

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, et al. Mmbench: Is your multi-modal model an all-around player? In *European conference on computer vision*, pp. 216–233. Springer, 2024.

Wenxuan Lu, Jiangyang He, Zhanqiu Zhang, Yiwen Guo, and Tianning Zang. Cultivating game sense for yourself: Making VLMs gaming experts. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2025.

Neel Mokaria, Rishie Raj, Dheeraj Baiju, Xiaoqian Shen, Shraman Pramanick, Kevin Qinghong Lin, Arda Senocak, Mike Zheng Shou, Philip Torr, Mohamed Elhoseiny, et al. A survey on foundations and frontiers of multimodal agentic frameworks: Techniques and applications. *arXiv preprint arXiv:2608.20379*, 2026.

Muhammad Umair Nasir, Steven James, and Julian Togelius. GameTraversalBenchmark: Evaluating planning abilities of large language models through traversing 2d game maps. In *Advances in Neural Information Processing Systems*, volume 37, pp. 31813–31827, 2024.

OpenAI. Universe. OpenAI, December 2016. URL https://openai.com/index/universe/.

OpenAI. GPT-6 Astra: A new generation of intelligence. OpenAI, 2026.

Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wolczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, et al. BALROG: Benchmarking agentic LLM and VLM reasoning on games. In *International Conference on Learning Representations*, 2025.

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. Autonomous evaluation and refinement of digital agents. In *First Conference on Language Modeling*, 2024.

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. In *International Conference on Learning Representations*, 2026.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025. URL https://arxiv.org/abs/2504.16054.

PyWeek. Python game programming challenge. https://pyweek.org. Accessed: 2026-04-28.

Dan Qiao, Chenfei Wu, Yaobo Liang, Juntao Li, and Nan Duan. GameEval: Evaluating LLMs on conversational games. *arXiv preprint arXiv:2308.10032*, 2023.

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

SteamDB. Database of everything on steam. https://steamdb.info/. Accessed: 2026-04-28.

Weihao Tan, Wentao Zhang, Xinrun Xu, Haochong Xia, Ziluo Ding, Boyu Li, Bohan Zhou, Junpeng Yue, Jiechuan Jiang, Yewen Li, et al. Cradle: Empowering foundation agents towards general computer control. *arXiv preprint arXiv:2403.03186*, 2024.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

Xinyu Wang, Bohan Zhuang, and Qi Wu. Are large vision language models good game players? In *International Conference on Learning Representations*, 2025a.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Boyuan Zheng, LI PEIHANG, Fangyu Lei, Ruisheng Cao, Yeqiao Fu, Dongchan Shin, Martin Shin, Hu Jiarui, Yuyan Wang, Jixuan Chen, Yuxiao Ye, Danyang Zhang, Yipu Wang, Heng Wang, Diyi Yang, Victor Zhong, Y.Charles, Zhilin Yang, and Tao Yu. OpenCUA: Open foundations for computer-use agents. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, pp. 139756–139806. Curran Associates, Inc., 2025b. doi: 10.52202/085713-4669. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/cc7ae529e945226b0d52ea4ac478c4f3-Paper-Conference.pdf.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: interactive planning with large language models enables open-world multi-task agents. In *Advances in Neural Information Processing Systems*, pp. 34153–34189, 2023.

Zirui Wang, Junyi Zhang, Jiaxin Ge, Long Lian, Letian Fu, Lisa Dunlap, Ken Goldberg, XuDong Wang, Ion Stoica, David M Chan, et al. VisGym: Diverse, customizable, scalable environments for multimodal agents. *arXiv preprint arXiv:2601.16973*, 2026.

Annie Wong, Thomas Bäck, Aske Plaat, Niki van Stein, and Anna V Kononova. Reasoning capabilities of large language models on dynamic tasks. *arXiv preprint arXiv:2505.10543*, 2025.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. OS-ATLAS: Foundation action model for generalist gui agents. In *International Conference on Learning Representations*, 2025.

Junlin Xie, Ruifei Zhang, Zhihong Chen, Xiang Wan, and Guanbin Li. WhodunitBench: Evaluating large multimodal agents via murder mystery games. In *Advances in Neural Information Processing Systems*, volume 37, pp. 86655–86687, 2024a.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024b.

Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. Translating natural language to planning goals with large-language models. *arXiv preprint arXiv:2302.05128*, 2023.

Haiyang Xu, Xi Zhang, Haowei Liu, Junyang Wang, Zhaozai Zhu, Shengjie Zhou, Xuhao Hu, Feiyu Gao, Junjie Cao, Zihua Wang, Zhiyuan Chen, Jitong Liao, Qi Zheng, Jiahui Zeng, Ze Xu, Shuai Bai, Junyang Lin, Jingren Zhou, and Ming Yan. Mobile-agent-v3.5: Multi-platform fundamental gui agents, 2026a. URL https://arxiv.org/abs/2602.16855.

Jin Xu, Zhifang Guo, Jinzheng He, Hangrui Hu, Ting He, Shuai Bai, Keqin Chen, Jialin Wang, Yang Fan, Kai Dang, Bin Zhang, Xiong Wang, Yunfei Chu, and Junyang Lin. Qwen2.5-omni technical report, 2025a. URL https://arxiv.org/abs/2503.20215.

Jin Xu, Zhifang Guo, Hangrui Hu, Yunfei Chu, Xiong Wang, Jinzheng He, Yuxuan Wang, Xian Shi, Ting He, Xinfa Zhu, Yuanjun Lv, Yongqi Wang, Dake Guo, He Wang, Linhan Ma, Pei Zhang, Xinyu Zhang, Hongkun Hao, Zishan Guo, Baosong Yang, Bin Zhang, Ziyang Ma, Xipin Wei, Shuai Bai, Keqin Chen, Xuejing Liu, Peng Wang, Mingkun Yang, Dayiheng Liu, Xingzhang Ren, Bo Zheng, Rui Men, Fan Zhou, Bowen Yu, Jianxin Yang, Le Yu, Jingren Zhou, and Junyang Lin. Qwen3-omni technical report, 2025b. URL https://arxiv.org/abs/2509.17765.

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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2026b.

Taofeng Xue, Chong Peng, Mianqiu Huang, Linsen Guo, Tiancheng Han, Haozhe Wang, Jianing Wang, Xiaocheng Zhang, Xin Yang, Dengchang Zhao, Jinrui Ding, Xiandi Ma, Yuchen Xie, Peng Pei, Xunliang Cai, and Xipeng Qiu. Evocua: Evolving computer use agents via learning from scalable synthetic experience, 2026. URL https://arxiv.org/abs/2601.15876.

Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, et al. EmbodiedBench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. In *International Conference on Machine Learning*, pp. 70576–70631, 2025.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, volume 35, pp. 20744–20757, 2022.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

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, et al. Ai gamestore: Scalable, open-ended evaluation of machine general intelligence with human games. *arXiv preprint arXiv:2602.17594*, 2026.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. 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*, pp. 9556–9567, 2024.

Alex L Zhang, Thomas L Griffiths, Karthik R Narasimhan, and Ofir Press. VideoGameBench: Can vision-language models complete popular video games? *arXiv preprint arXiv:2505.18134*, 2025a.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pp. 1–20, 2025b.

Haoran Zhang, Hangyu Guo, Shuyue Guo, Meng Cao, Wenhao Huang, Jiaheng Liu, and Ge Zhang. ING-VP: Mllms cannot play easy vision-based games yet. *arXiv preprint arXiv:2410.06555*, 2024.

Yuxuan Zhang, Haoyang Yu, Lanxiang Hu, Haojian Jin, and Hao Zhang. General modular harness for LLM agents in multi-turn gaming environments. *arXiv preprint arXiv:2507.11633*, 2025c.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623, 2023.

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. *arXiv preprint arXiv:2504.06148*, 2025.

Hanzhang Zhou, Panrong Tong, Xu Zhang, Quyu Kong, Chenglin Cai, Tianyu Xia, Gongjie Zhang, Jianan Zhang, Long Li, Long Chen, et al. Qwen-ui-agent technical report: Toward next-generation real-world centric foundation gui agents. *arXiv preprint arXiv:2607.28227*, 2026.

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

A APPENDIX

A.1 DATA CURATION METHODOLOGY AND TECHNICAL VALIDATION

The transition from the raw data pool to the final PLAYSUITE benchmark involved a rigorous multi-stage pipeline focused on quality, technical viability, and safety. We initially considered a pool of 1460 *PyWeek* games and 1.7M *itch.io* games. Across both platforms, we applied a quality gate by selecting only games with a community rating $> 3.0/5.0$. For the *itch.io* corpus specifically, we applied additional constraints, including only free-to-play and fully-released titles with at least 5 individual ratings, verified compatibility for Linux or Web-based execution, support for keyboard and/or mouse input. We additionally executed each candidate game to confirm it launches without errors. Following this initial filtering, we applied dataset-specific refinement layers. The *PyWeek* dataset underwent three further rounds of technical pruning. We excluded titles requiring deprecated Python versions (e.g., Python 2.x), leaving 376 video games. We subsequently restricted the selection to games developed using the `Pygame` library and we performed automated dry-runs to identify and remove games containing internal runtime bugs or broken assets. This resulted in a final, stable set of 104 high-fidelity Python games.

The *itch.io* dataset underwent a dual-layered security and content audit to mitigate risks associated with executing third-party binaries. First, we implemented an automated content filtering to exclude games containing NSFW content, via tag-based matching. We also conducted an additional targeted analysis with the specific intent of identifying games where an AI agent could plausibly encounter situations matching particular risk categories. This allowed us to categorize potential safety-critical interactions within the benchmark, as reported in Table 4. Candidate games were identified via tag-based matching and validated using *gpt-5.4-nano-2026-03-17*. Second, we deployed a custom safety triage scanner to evaluate structural game archive signals (e.g., password protection, nested binaries) and metadata risks. Archives were assigned an integer risk score to gate our execution policy. High-risk games (score ≥ 4) were quarantined for manual review, while games with a score of 3, often indicating download failures or unexplained binaries, were subject to extra logging. Minor signals (scores 1–2) and clean archives (score 0) were executed within our network-isolated sandbox. Established games with high ratings (rating count > 50) were categorized as "trusted" (score < 0). Following these filters, the final *itch.io* collection was pruned to 5,630 games, ensuring a secure and computationally reliable distribution for large-scale evaluation. Following technical validation, we excluded games that could not be evaluated reliably. Table 5 summarizes these exclusion reasons.

Table 4: Distribution of harmful content categories identified in the *itch.io* dataset.

Harmful Categories	**Game Count**
Firearms Individual Violence	13
Scheming Manipulation	35
Value Misalignment	91
Inconsistent Behavior	24
Explosives Violence	3
Economic Waste	12
Data Exfiltration	4
Economic Redistribution	2
Explosives Creation	4
Infrastructure Damage	4
Firearms Mass Violence	2
Bias Stereotypes	4
Accidents	2

A.2 GAME GENRE TAXONOMY AND CATEGORIZATION

The taxonomy used in PLAYSUITE was developed through a multi-stage process of data-driven analysis and alignment with established game-AI literature. Our goal was to create a classification system that reflects both the computational challenges (e.g., reactive control vs. long-term reasoning) and the distribution of mechanics present in the source platforms.

Table 5: **Additional technical exclusions from the evaluation set.** Number of games excluded and their identified reason.

Reason	Games
Construct runtime freeze failure	225
Dragging is the only listed control	72
Requires multiple players	25
Not an interactive game	18
Gamepad required	2

We began by analyzing the native metadata from the *itch.io* crawl. After initial filtering, we identified 17 primary category tags: `Action`, `Platformer`, `Adventure`, `Puzzle`, `Shooter`, `Simulation`, `Game Roly Playing`, `Survival`, `Strategy`, `Visual Novel`, `Interactive Fiction`, `Educational`, `Fighting`, `Racing`, `Card Game`, `Rythm`, `Sports`. However, we observed that individual titles were frequently mapped to multiple tags (e.g., a `Platformer` being tagged as both `Action` and `Adventure`). To resolve these ambiguities and ensure the utility for evaluating specific AI capabilities, we cross-referenced these tags with established gameplay taxonomies. Specifically, our categorization is grounded in recent literature on LLM-based game agents, which distinguishes genres based on the core cognitive challenges and agent capabilities required, such as reasoning-heavy vs. reaction-based dynamics (Hu et al., 2024; Lee et al., 2014).

By reconciling these academic perspectives with industry-standard tagging systems like *SteamDB* (SteamDB), we consolidated the library into 9 distinct genres. Crucially, we formulated detailed qualitative descriptions for each category, following (Lee et al., 2014). These descriptions serve a dual purpose: providing a rigorous definition for the taxonomy and acting as the semantic grounding for our automated classification of untagged games.

- `Action`: Games with a heavy emphasis on a series of actions performed by the player in order to meet a certain set of objectives. Examples are Metal Gear Solid, Rhythm Doctor, Super Mario Bros and Space Invaders.
- `Adventure`: Games which are set in a world for the player to explore and complete a certain set of objectives through a series of actions. Examples include The Legend of Zelda and Prince of Persia.
- `Puzzle`: Games with an objective of figuring out the solution by solving enigmas, navigating, and manipulating and reconfiguring objects. Examples include Tetris and Minesweeper.
- `Shooter`: Games involving shooting at, and often destroying, a series of opponents or objects. Examples include Doom and Duck Hunt.
- `Simulation`: Games recreating an experience of a real world activity in the game world. Examples include SimCity, The FIFA series, Trauma Center.
- `Survival`: Survival games are action-based, open-world experiences where players start with minimal resources in a hostile environment, tasked with staying alive as long as possible. Examples include Minecraft and The Forest.
- `Strategy`: Games characterized by players' strategic decisions and interventions to bring the desired outcome. Examples include StarCraft and the Total War series.
- `Fighting`: Games where the player controls a game character to engage in a combat against an opponent. Examples include Street Fighter and Mortal Kombat.
- `Racing`: Games involving driving various types of vehicles as the main action, with an objective of winning a race against an opponent. Examples include Mario Kart and Gran Turismo.

Following our taxonomy, we mapped the game library to these genres based on platform-specific metadata.

For the *itch.io* dataset, we performed a rule-based merge to map the 17 original categories into our 9 canonical genres, as shown in Table 6. When a game carried multiple tags, we resolved them to

Table 6: Mapping of original *itch.io* metadata tags to the 9 canonical PLAYSUITE genres.

Original *itch.io* genres	**PLAYSUITE genre**
Puzzle, Educational	Puzzle
Adventure, Visual Novel, Interactive Fiction, Role Playing	Adventure
Action, Platformer, Rhythm	Action
Simulation, Sports	Simulation
Strategy, Card Game	Strategy
Shooter	Shooter
Survival	Survival
Racing	Racing
Fighting	Fighting

a single canonical genre by selecting the more specific PLAYSUITE category, where specificity is measured as the inverse number of *itch.io* tags mapped onto each PLAYSUITE genre (e.g., `Racing`, mapped from a single *itch.io* tag, is more specific than `Action`, which is mapped from three). If a game had no genre metadata, we assigned one using *gpt-5.4-nano-2026-03-17*, providing the model with the game title, tags and description.

For the *PyWeek* dataset, which lacks a native tagging system, we implemented an automated classification pipeline using Qwen2.5-14B-Instruct. To ensure consistency and high-fidelity mapping, we provided the model with the exact category descriptions and representative examples detailed in our taxonomy. For each game, the model analyzed the provided game description, README file and diary entries to infer the primary genre. By using the qualitative definitions of our taxonomy as the direct prompt, we ensured that the *PyWeek* genre categorization remained semantically aligned with the *itch.io* set, creating a unified distribution across the entire benchmark. Specifically, the textual definitions for the 9 categories provided above were used verbatim to populate the `<categories_str>` placeholder in our classification prompt. The specific prompt template used for the automated categorization process is detailed in Figure 7. The resulting genre distributions for the evaluated *itch.io* and *PyWeek* subsets are shown in Figure 8.

A.3 LICENSING AND TERMS OF USE

PLAYSUITE respects the intellectual property rights and distribution terms set forth by the original hosting platforms and developers.

In accordance with the official *PyWeek* rules (PyWeek), developers retain all copyrights to their entries. However, by submitting an entry, authors grant a transferable, irrevocable license to redistribute, copy, and run the entry without modification, and to distribute unmodified screenshots, provided no fee is charged. We distribute the *PyWeek* subset of PLAYSUITE under these terms, ensuring that the original game files remain unmodified and that our benchmark is provided as a free resource for the research community. For games that included independent license files within their source code, those specific terms are respected as an alternative to the standard *PyWeek* clause.

Games sourced from *itch.io* were selected from publicly available, free-to-play titles. In accordance with the *itch.io* Terms of Service (Itch.io), developers retain ownership of the content they upload. Our evaluation framework interacts with these games as a "user" of the service, utilizing the files as provided by the developers for the purpose of non-commercial research and benchmarking. We do not modify the game binaries or assets. To ensure compliance with developer intent, we have restricted our corpus to titles that are explicitly marked for public distribution and free access. Furthermore, we provide metadata and direct links to the original game pages to ensure proper attribution to the original creators.

A.4 MODEL CONFIGURATIONS

Table 7 summarizes the 14 evaluated baselines, their model families, parameter scales, and checkpoint identifiers.

Prompt for PyWeek Categorization:

You are an expert in videogame genre classification. Your task is to assign each game to EXACTLY ONE of the genres in the following list:
`<categories_str>`
In the list above, each entry corresponds to one genre. For each entry, the name of the genre is before the colon and the genre description is after the colon. The description also includes examples of games in that genre.

Analyze the game information below and determine the most appropriate genre.
Game Name: `<game_name>`
Game Description: `<game_description>` `<readme_file>`
Developer Diary Entries: `<diary_entries>`
Based on the game name, description, and developer notes above, determine the PRIMARY genre.

IMPORTANT RULES:
1. Respond with ONLY the genre name from the list above
2. Choose the single BEST matching genre
3. If unsure, make your best guess based on the description
4. Remember that a video game genre is defined by similar gameplay characteristics and how the player interacts with the environment, rather than by the setting or story of the game or its medium of play.
5. In your reasoning, include two to three reasons why your selection genre is the best match for the game. Please indicate what information you used to infer these reasons. If you have low confidence in the category selection, please also add a sentence or two explaining why you are unsure.
6. Return ONLY a JSON object with this format:
{ "categories": ["Category"],
"confidence": "high|medium|low",
"reasoning": "Brief explanation" }

Example output: {"categories": ["Action"], "confidence": "high", "reasoning": "The game involves jumping between platforms and combat mechanics. As mentioned in the read me, the objective is to take the player from starting point A to end point B. Diary entry 2 shows that there are X levels in the game."}

Your JSON response:

Figure 7: Prompt used for the automated categorization of PyWeek games.

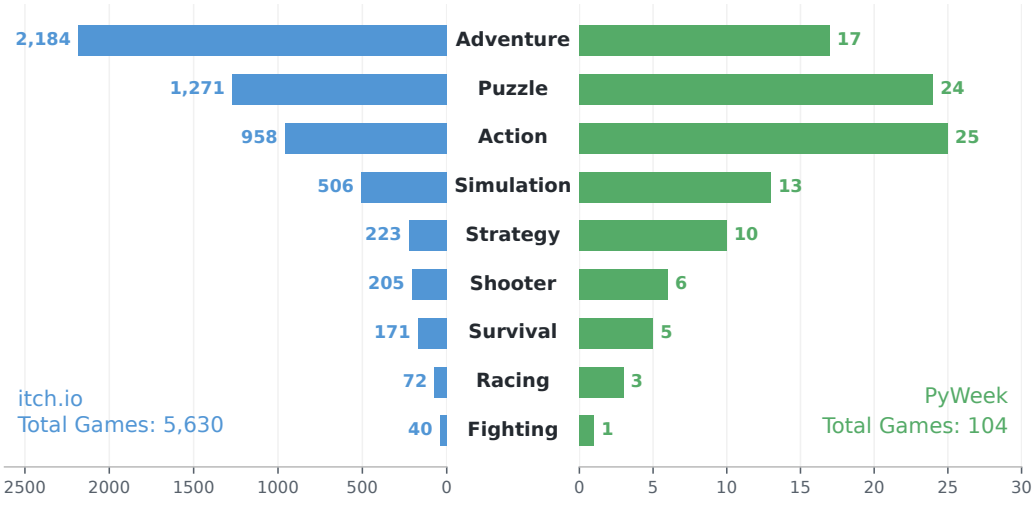


Figure 8: **Genre distribution of the PLAYSUITE evaluation set.** Distribution of the 5,630 evaluated *itch.io* games and 104 *PyWeek* games across the nine canonical PLAYSUITE genres.

Table 7: **Baseline models.** Model family, parameter scale, Hugging Face checkpoint, and reference for each evaluated model.

Model	Family	Parameters	Hugging Face identifier	Reference
Qwen3-Omni-30B-A3B-Instruct	VLM	30B (3B active)	`Qwen/Qwen3-Omni-30B-A3B-Instruct`	(Xu et al., 2025b)
Qwen2.5-VL-7B-Instruct	VLM	7B	`Qwen/Qwen2.5-VL-7B-Instruct`	(Bai et al., 2025b)
Gemma-4-26B-A4B-it	VLM	26B (4B active)	`google/gemma-4-26B-A4B-it`	(Gemma Team, 2026)
Qwen3-VL-8B-Instruct	VLM	8B	`Qwen/Qwen3-VL-8B-Instruct`	(Bai et al., 2025a)
Qwen2.5-Omni-7B	VLM	7B	`Qwen/Qwen2.5-Omni-7B`	(Xu et al., 2025a)
Gemma-4-E4B-it	VLM	8B (4.5B eff.)	`google/gemma-4-E4B-it`	(Gemma Team, 2026)
InternVL2.5-8B	VLM	8B	`OpenGVLab/InternVL2_5-8B`	(Chen et al., 2024)
Gemma-3-12B-it	VLM	12B	`google/gemma-3-12b-it`	(Gemma Team, 2025)
UI-TARS-1.5-7B	CUA	7B	`ByteDance-Seed/UI-TARS-1.5-7B`	(Qin et al., 2025)
EvoCUA-8B	CUA	8B	`meituan/EvoCUA-8B-20260105`	(Xue et al., 2026)
OpenCUA-7B	CUA	7B	`xlangai/OpenCUA-7B`	(Wang et al., 2025b)
GUI-Owl-1.5-8B-Instruct	CUA	8B	`mPLUG/GUI-Owl-1.5-8B-Instruct`	(Xu et al., 2026a)
OS-Atlas-Pro-7B	VLA	7B	`OS-Copilot/OS-Atlas-Pro-7B`	(Wu et al., 2025)
ShowUI-2B	VLA	2B	`showlab/ShowUI-2B`	(Lin et al., 2025)

Model-specific prompting and output parsing are part of the evaluated configuration. Table 8 summarizes the parser and prompt template used for each baseline.

Table 8: **Model interfaces.** Output parser and prompt template used for each evaluated model.

Model	Parser	Prompt template
Gemma-4-26B-A4B-it	qwen3	standard
EvoCUA-8B	computer_use	computer_use
Qwen3-Omni-30B-A3B	qwen3	standard
Gemma-4-E4B-it	qwen3	standard
UI-TARS-1.5-7B	uitars	uitars
Qwen3-VL-8B	qwen3	standard
Qwen2.5-VL-7B	standard	standard
OpenCUA-7B	opencua	opencua
GUI-Owl-1.5-8B	computer_use	computer_use
Gemma-3-12B-it	standard	standard
Qwen2.5-Omni-7B	standard	standard
OS-Atlas-Pro-7B	osatlas	osatlas
InternVL2.5-8B	qwen3	standard
ShowUI-2B	showui	showui

A.5 Prompt Templates

General-purpose VLMs use a shared prompt consisting of a system instruction defining the agent's role and a user template containing game metadata, available controls, interaction history, and the required action format. The complete VLM prompt is shown in Figure 19.

Specialized CUAs and VLAs retain their model-specific prompting and output formats, with only minor adaptations to the PLAYSUITE interaction setting. The computer-use prompt is shown in Listing 20; model-specific prompt and parser assignments are summarized in Table 8.

Prompting ablations involving explicit strategy traces and few-shot examples are reported in Appendix A.9.3.

A.6 EXECUTION DETAILS

PyWeek. For Python-based games, we instrument the game process directly by setting `SDL_VIDEODRIVER=dummy` and overriding the internal display update cycle, allowing raw RGB observations to be captured directly from the game framebuffer. A virtual clock controls game progression and allows us to set the interaction cadence explicitly. We evaluate at interaction rates of 3 Hz and 0.33 Hz. Each run is capped at 10,000 game frames, corresponding to approximately 333 seconds (~ 5.5 minutes) of gameplay at 30 fps.

itch.io. For standalone and browser-based games, we use Xvfb to provide an off-screen display for frame capture. Browser based games are controlled by freezing browser clocks, while native game processes are paused directly. We evaluate at 0.33 Hz, with each run spanning 300 seconds of game time and approximately 100 interaction steps. Inference pauses are removed from the judged video. We exclude 225 Construct 3 games whose game loops rely on Web Workers that cannot be reliably paused by our browser clock interception mechanism.

Action interface. The shared action grammar supports single keys, chords, timed holds, mouse motion, left/right clicks, wheel events, and multiple clicks. Timed holds in the generalist VLM prompt are capped at two seconds.

A.7 VIDEO-LLM-AS-A-JUDGE ADDITIONAL DETAILS

A.7.1 JUDGE CONFIGURATION

All reported benchmark results use `Qwen/Qwen3.5-9B` as the Video-LLM judge. Gameplay videos are sampled at 1 fps, with individual frames constrained to a maximum resolution of 512×288 pixels to balance visual detail with computational throughput. Decoding uses constrained JSON generation, temperature 0, and seed 41. The judge is served with batch size 8, up to 8 concurrent sequences, a 65,536-token context window, a 16,384-token generation limit, and up to 32,768 batched tokens. For *itch.io*, the judge additionally receives game context that is separate from the information provided to the playing agent.

A.7.2 JUDGE MILESTONE PROMPT AND PROGRESS RULES

To ground the judge in each environment, the input prompt includes game-specific metadata. For *itch.io*, this comprises the title, author, genre, description, community tags, and controls; for *PyWeek*, it includes the title, author, genre, and project README. Figure 9 shows the milestone prompt and output schema; for readability, the game-specific context is replaced by a descriptive placeholder.

IMPORTANT:

- Return **ONLY valid JSON** matching the schema. No markdown fences, no prose.
- Every field asks whether something is **visible at least once anywhere in the video**. Watching more footage can only ADD events, never remove them.
- Base every answer strictly on visible evidence. When uncertain, answer false / 0 — never guess.

Task: Report which of the following concrete events are visible in this gameplay video. Do NOT rate quality, skill, or overall progress — only report observable events.

Field definitions:

- "menu_or_title_seen": a menu, title screen, or splash screen is visible at any point.
- "left_menu_into_gameplay": the video shows a transition from a menu or title screen into live gameplay (the reverse does not count).

- "player_control_evident": the game state visibly responds to player inputs at least twice (character moves purposefully, UI reacts to clicks/keys, selections change). Autonomous animation does not count.
- "distinct_interactions": integer count of DISTINCT visible interaction events (movement burst, click with UI response, attack, jump, item pickup...). Repeats of the identical action in place count once. Cap at 20. Estimate quickly — do NOT enumerate every interaction one by one; a rough count is fine.
- "score_or_resource_increased": a numeric score, resource count, currency, progress bar, or similar UI indicator visibly increases.
- "new_area_or_level_reached": the player visibly enters an area, room, screen, or level not seen earlier in the video (scrolling within one screen does not count).
- "checkpoint_or_objective_completed": an explicit checkpoint, mission complete, level clear, or objective-done indicator appears.
- "win_or_completion_screen": an explicit victory / you-win / run-complete screen, or credits reached by finishing the game. A game-over, "you lose", death, or failure screen is NOT a completion screen — count those under "death_or_game_over_count" instead.
- "death_or_game_over_count": integer count of visible deaths, game-over screens, or run-ending failures.
- "stuck_or_looping": the majority of the video shows the same action or scene repeating with no visible change.
- "milestones": one timestamped entry per TRUE boolean above (and per death), naming the field and what is visible on screen at that moment.

Output JSON schema:

```
{
  "gameplay_happened": true|false,
  "menu_or_title_seen": true|false,
  "left_menu_into_gameplay": true|false,
  "player_control_evident": true|false,
  "distinct_interactions": number,
  "score_or_resource_increased": true|false,
  "new_area_or_level_reached": true|false,
  "checkpoint_or_objective_completed": true|false,
  "win_or_completion_screen": true|false,
  "death_or_game_over_count": number,
  "stuck_or_looping": true|false,
  "milestones": [
    {"t_sec": number, "field": string, "on_screen": string}
  ]
}
```

Game Context

[Per-game context from the page snapshot or documentation.]

Figure 9: Milestone prompt; game context varies by video.

Rather than directly assigning a progress score, the judge extracts observable gameplay milestones together with timestamped evidence. These include player control, score increases, entry into new areas or levels, completion of intermediate objectives, and game completion. The resulting structured output is mapped to the five progress levels using the fixed rules in Table 9. Rules are evaluated in descending priority and do not require lower-level conditions to hold.

A.7.3 Human and cross-judge validation

We validate the judge on 60 gameplay videos, comprising 45 *itch.io* and 15 *PyWeek* runs spanning the evaluated model configurations and both history conditions. Six human annotators independently label each video while blinded to model and condition.

Human-only Krippendorff's α is 0.624, using squared distance between the five progress levels, and human annotators agree within one progress level in 93.0% of pairwise comparisons. Qwen3.5-9B agrees within one level with individual human annotations in 90.4% of comparisons; including it as a seventh rater yields $\alpha = 0.605$.

Table 9: **Deterministic mapping from milestones to progress.** Evaluate rows from top to bottom and return the first matching level. Conditions do not require the lower-level milestones to hold. Field names match the judge output.

Score	Level	Condition
4	Completed	`win_or_completion_screen`
3	Substantial	`checkpoint_or_objective_completed` **or** (`new_area_or_level_reached` **and** `score_or_resource_increased`)
2	Partial	`new_area_or_level_reached` **or** `score_or_resource_increased`
1	Minimal	`player_control_evident` **and** `distinct_interactions` ≥ 2
0	None	No condition above is satisfied.

To assess sensitivity to judge choice, we additionally evaluate Qwen3.5-27B, Gemma-4-31B-it, and Molmo2-8B on the same 60 videos. Table 10 shows that both Qwen configurations track human progress similarly, while the other judges exhibit greater variation.

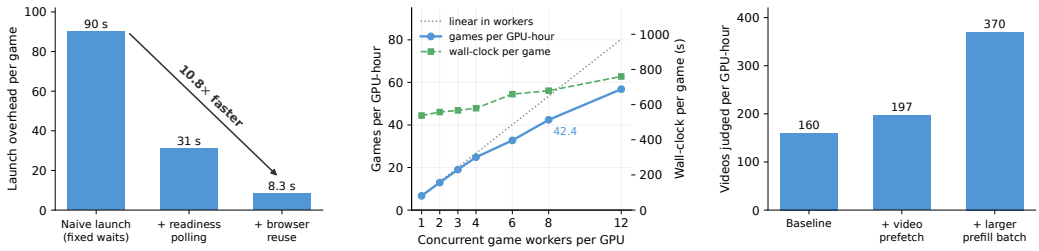
Table 10: **Judge–human agreement on the validation set.** Agreement between each Video-LLM judge and the six human annotators, averaged over available comparisons. Milestone agreement is computed over the nine Boolean milestone fields.

Judge	Milestone agreement	Within one
Qwen3.5-9B	84.4%	90.4%
Qwen3.5-27B	83.9%	91.6%
Gemma-4-31B-it	82.5%	77.3%
Molmo2-8B	77.9%	81.2%

A.8 Benchmark execution efficiency

We characterize evaluation throughput using Qwen3-VL-8B on a single H100. The initial launcher relied on fixed waits for the virtual display, browser, and game to become ready. Replacing these waits with readiness checks allows execution to proceed as soon as each component is available, while keeping the browser and virtual display alive amortizes their initialization across games. Together, these changes reduce fixed launch overhead from approximately 90 s to 8.3 s per game. Sharing one vLLM server across concurrent game workers further increases throughput from 6.7 evaluations per GPU-hour with one worker to 42.4 with eight workers. Although throughput continues to increase beyond eight workers, higher concurrency increases CPU utilization and per-game wall-clock time, bringing some runs close to the harness safety timeout; we therefore use eight workers for the full benchmark. Figure 10 summarizes these measurements.

We similarly optimize judging by overlapping video loading and decoding with model inference and by increasing the number of prompt tokens that vLLM can process in each batch. The latter is particularly important because each judge request contains tens of thousands of video tokens, making prompt prefill a substantial part of inference. On a controlled set of 300-s videos, these optimizations increase steady-state Qwen3.5-9B throughput from 160 to 370 videos per GPU-hour without changing the resulting milestone levels on comparable runs. When engine startup and other end-to-end overhead are included, the measured throughput is approximately 277 full-length *itch.io* videos per GPU-hour.



(a) Launch overhead (b) Concurrency on one H100 (c) Judging throughput

Figure 10: **Measured execution efficiency.** (a) Fixed overhead per game before the first captured frame, from 50-game launch tests. The naive launcher waits fixed intervals for the virtual display and the game page; readiness polling replaces these waits, and browser reuse keeps the browser and display running between games. (b) Qwen3-VL-8B on one H100, with concurrent game workers sharing a single vLLM server and the same games evaluated at each concurrency level. Throughput increases with worker count. (c) Increasing Qwen3.5-9B judge throughput through video prefetching and a larger batched-token budget (`max_num_batched_tokens`=32,768) on 300s videos.

A.9 Additional benchmark analyses

A.9.1 Performance across game genres

Figures 11 and 12 show the per-model genre breakdowns in the no-history condition. On *itch.io*, the aggregate pattern reported in the main text is broadly reflected across individual models, with higher progress on Simulation and Strategy and lower progress on Adventure and Puzzle. The *PyWeek* results show greater variation across models, though this may partly reflect the small number of games in several genres. Table 11 additionally reports genre-level performance under both history conditions. Adding three interactions of history ($k = 3$) reduces mean progress across every reported genre on both subsets, consistent with the overall history results.

Table 11: **Genre-level progress under both history conditions.** Mean Progress Score for history lengths $k = 0$ and $k = 3$ on *itch.io* and *PyWeek*. N denotes the number of games.

	itch.io			*PyWeek*		
Genre	N	$k = 0$	$k = 3$	N	$k = 0$	$k = 3$
Simulation	506	0.893	0.800	13	1.293	1.173
Strategy	223	0.858	0.704	10	1.054	0.702
Shooter	205	0.771	0.682	6	0.763	0.720
Survival	171	0.770	0.644	5	1.174	0.857
Racing	72	0.705	0.654	3	0.976	0.714
Action	958	0.620	0.591	25	0.817	0.699
Puzzle	1271	0.606	0.546	24	0.893	0.771
Adventure	2184	0.544	0.452	17	0.883	0.664

A.9.2 Effect of action frequency

Figure 13 shows the effect of increasing the *PyWeek* interaction rate from 0.33 Hz to 3 Hz for each model. Mean progress increases in both conditions: from 0.942 to 1.036 without history and from 0.778 to 0.823 with $k = 3$ history. The improvement is therefore 0.049 points larger without history.

Increasing the action rate does not provide clear evidence of an interaction with history. The mean gap between the no- history and history conditions increases from 0.165 points at 0.33 Hz to 0.214 points at 3 Hz, but a game-level bootstrap gives a 95% interval of $[-0.027, 0.129]$ for this change. Moreover, per-model history effects at the two action rates are strongly correlated ($r = 0.96$), indicating that models' relative sensitivity to history remains similar across interaction rates.



Figure 11: **Per-model progress by genre on *itch.io*.** Mean Progress Score in the no-history condition at 0.33 Hz for each model and genre. Columns include genres with at least 60 games; n denotes the number of games. Models are ordered by overall no-history performance.

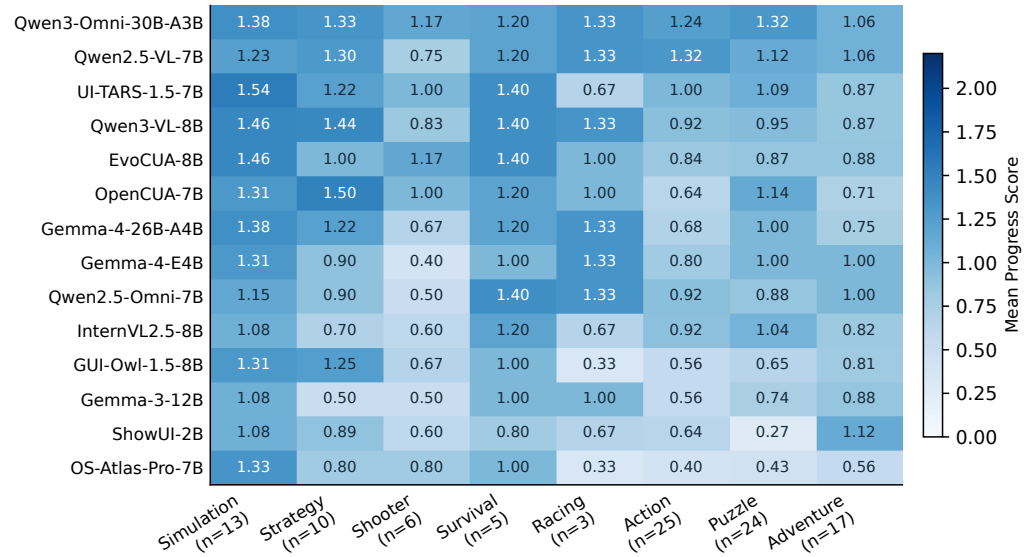


Figure 12: **Per-model progress by genre on *PyWeek*.** Mean Progress Score in the no-history condition at 0.33 Hz for each model and genre. n denotes the number of games. Models are ordered by overall no-history performance. Per-genre counts are small, so these results are primarily descriptive.

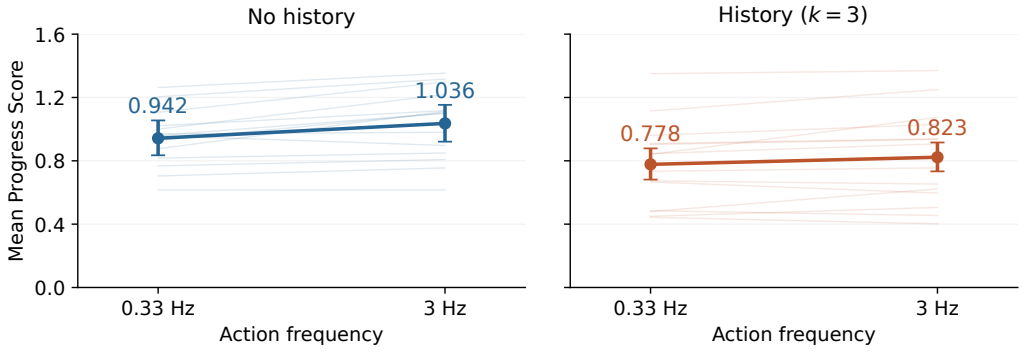


Figure 13: **Effect of action frequency on *PyWeek*.** Mean Progress Score at 0.33 Hz and 3 Hz under the no-history and $k = 3$ history conditions. Faint lines show individual model means and darker lines the mean over matched game–model pairs. Error bars denote 95% game-bootstrap intervals.

A.9.3 Impact of prompting strategies

We evaluate two modifications to the standard prompt for the eight VLMs: an explicit "Strategy" field requiring the model to state its overarching goal, and three few-shot examples demonstrating the expected response format. We restrict these ablations to the eight VLMs using the standard prompt template, since CUAs and VLAs rely on model-specific response formats and parsers that could confound prompt effects with parsing failures. Figure 14 reports the resulting mean Progress Scores on *PyWeek* and the 200-game *itch.io* subset. Neither modification improves average performance. Adding an explicit strategy reduces mean progress from 0.785 to 0.747 on *PyWeek*, while performance on *itch.io* remains nearly unchanged (0.538 to 0.535). Few-shot examples reduce performance on both corpora, from 0.789 to 0.720 on *PyWeek* and from 0.538 to 0.504 on *itch.io*. Overall, adding prompt structure does not improve gameplay performance under our evaluation setup.

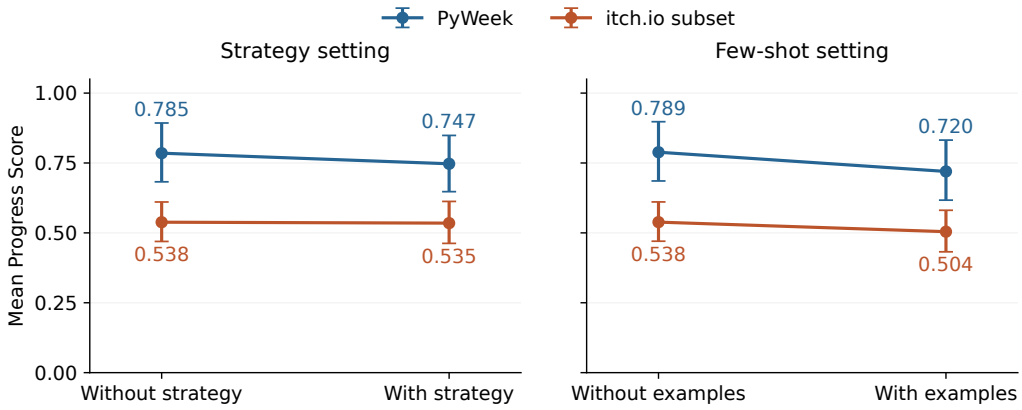


Figure 14: **Effect of prompt additions.** Mean Progress Score with and without an explicit Strategy field (left) or three few-shot examples (right), for eight VLMs at $k = 3$. Each comparison uses matched game–model cells. Error bars show 95% game-bootstrap intervals for the means.

A.9.4 Impact of temporal history

Figure 15 reports performance across history lengths $k \in \{0, 3, 5, 10\}$ for both subsets. On *PyWeek*, mean Progress Score decreases from 0.946 at $k = 0$ to 0.778 at $k = 3$, remains similar at $k = 5$ (0.793), and falls to 0.740 at $k = 10$. The *itch.io* subset shows the same pattern, decreasing from 0.616 at $k = 0$ to 0.544 at $k = 3$, 0.551 at $k = 5$, and 0.511 at $k = 10$. Thus, additional history does not improve average performance on either corpus, although individual models can differ in their response to temporal context.

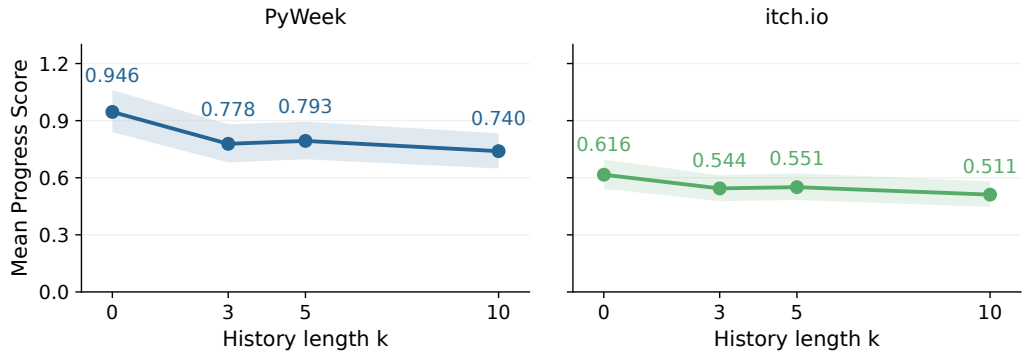


Figure 15: **Impact of temporal history across both corpora.** Mean Progress Score for history lengths $k \in \{0, 3, 5, 10\}$ on *PyWeek* and the 200-game *itch.io* subset. No history achieves the highest mean performance on both corpora, while longer histories provide no consistent improvement.

A.9.5 Cost–performance analysis

We measure realized evaluation cost for the no-history *itch.io* campaign as allocated H100 GPU-hours per 1,000 scored games, including startup, idle allocation, retries, and failed or interrupted jobs. Judging cost is accounted for separately and excluded here. Figure 16 shows substantial variation in both cost and performance across models. Qwen2.5-VL achieves a mean Progress Score of 0.822 at 59.9 GPU-hours per 1,000 games, while Qwen3-Omni reaches 0.841 at 105.5 GPU-hours. For Qwen3-Omni, both GPUs used per replica are included in the reported cost. Costs therefore reflect the realized compute required by each serving configuration.

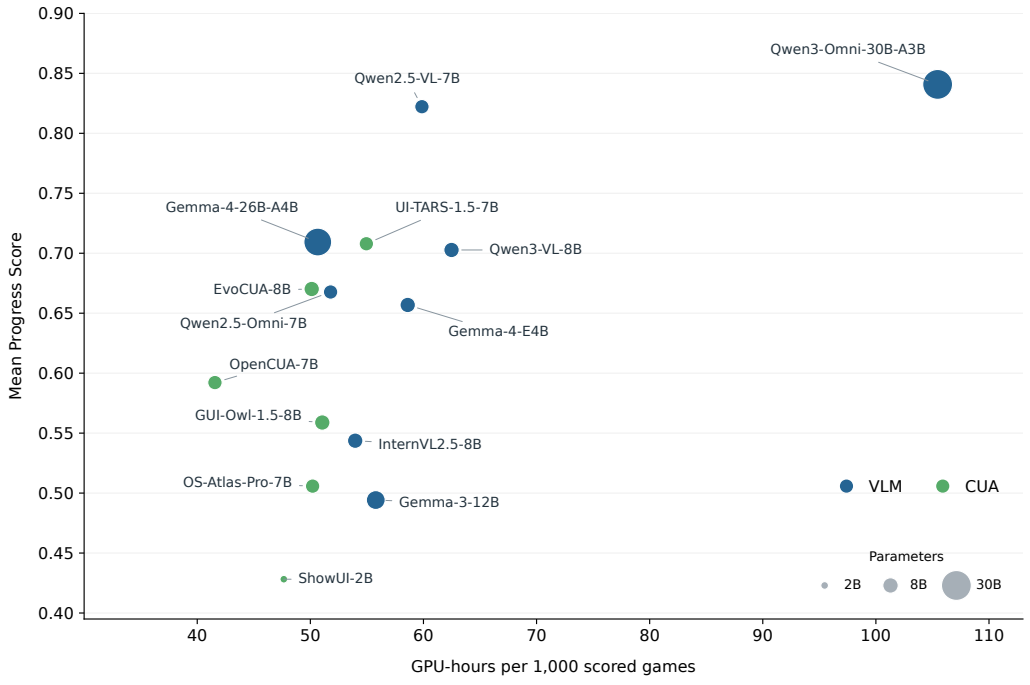
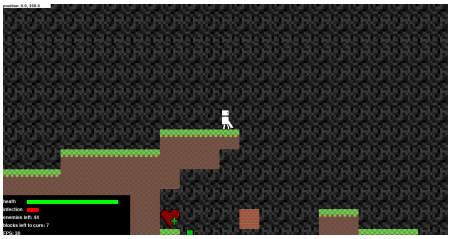


Figure 16: **Cost–performance trade-off on *itch.io*.** Mean Progress Score against H100 GPU-hours per 1,000 scored games in the no-history setting. Marker color denotes model family and marker size denotes parameter count.

A.10 FAILURE MODES

In this section we provide another example of the failure mechanisms discussed in the main text.



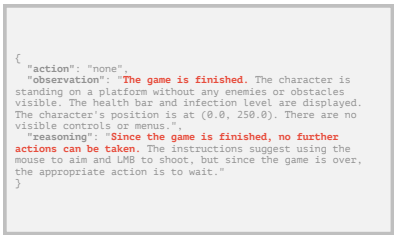


Figure 17: Example of a *passive wait* failure mechanism. Qwen2.5-VL successfully starts the game, but since the initial game screen shows no visible enemies or apparent obstacles, it assumes the game has ended and waits, even if no on-screen information confirms that the game has been completed.

A.11 ADDITIONAL TABLES

You are an expert video game AI agent. Your goal is to play the provided game to maximize progress, score, or reach a win state.

INSTRUCTIONS:

1. Analyze Controls: Read the "Game Description" and "Game Instructions". Ignore HTML tags like <br>.
   - Extract specific mappings (action: key). Keep this in memory but when prompt return only the key.
   - CRITICAL: If the instructions mentions "LMB" (Left Mouse Button), you must output a mouse action requiring coordinates ("CLICK: x, y").
   - If no controls are listed, infer them (e.g., Platformers usually use Arrows/WASD + Space).
2. Analyze Visuals: Look at the current frame. Identify the player character, enemies, obstacles, and the UI (health bar, score).
3. Detect Loops (Crucial): Compare the current frame to the history (if available).
   - If the visual state is identical to previous frames, you are STUCK.
   - If you are STUCK, you MUST change your strategy. Try a different key or direction.
4. Exploration: If the objective is unclear, interact with objects or move towards unexplored areas of the screen.

Figure 18: Shared VLM system prompt.

You are playing an itch.io game called "`<$game_name$>`".`<$category_suffix$>`

`<$game_description$><$readme_section$>`

`<$history_section$>`

=== CONTROL MAPPING RULES (STRICT) ===

STEP 1: CHECK INSTRUCTIONS

- The instructions above is the TRUTH.
- If the instruction lists keys (e.g., "jump : w"), use EXACTLY those keys.
- If the instruction lists "LMB" you must output "CLICK: x, y".

STEP 2: FALLBACK DEFAULTS

- ONLY use these if the instruction are missing or does not mention the action:
  - Movement: a (left), d (right), w (up), s (down)
  - Actions: space (jump/confirm), enter (start/confirm), escape (menu/back), click: x, y (shoot)
  - Other: z, x, c, q, e, r, f, 1, 2, 3, 4, 5

=== HOW TO HANDLE MOUSE / LMB ===

- MENUS: If you see text like "Start", "Start Game", "Play", "New Game", "Proceed" or similar output: CLICK: x, y (coordinates of the button).
- LMB: If the instructions says "Shoot: LMB", look for an enemy or target on screen. Output: CLICK: x, y (coordinates of the enemy).
- AIMING: If the instructions says "Aim with mouse", look at where you want to shoot and output: CLICK: x, y.

=== HOW TO HANDLE GAME OVER / END STATES ===

- If the frame shows a game-over, death, victory, or level-complete screen (e.g. text like "Game Over", "You Died", "Retry", "Resume", "Restart", "Continue", "Next Level", "Play Again", "Try Again"): CLICK: x, y (coordinates of the button).
- If it is not clickable (no visible button, just text), use the fallback keys most likely to confirm/continue: enter, space, escape (try enter first unless the instructions say otherwise).

=== ACTION FORMAT ===

- Single key: space
- Key chord (keys pressed together): shift+a
- Timed hold (movement, charging): hold:left:0.8 or hold:right+space:0.5 (seconds, max 2)
- Wait: wait - take no action (use SPARINGLY)
- Mouse click: CLICK: x, y
- Right click (if the game uses it, e.g. RTS commands, alt fire): rightclick:x,y
- Double click (open/activate, some UIs need it): click:x,y click:x,y
- Mouse wheel (zoom / weapon switch / scroll lists): scroll:up or scroll:down (add :N for N notches, e.g. scroll:down:5)
- Mouse move only (look around / aim, no click): move:x,y or relative move:+dx,-dy (mouse-look games; smooth sweep, add :SECONDS for a slower pan)

=== DECISION RULES ===

1. MENU SCREEN. If you see a game menu screenshot, click on text buttons ('Start', 'Start Game', 'Play', 'New Game', 'Proceed'): Use "CLICK: x, y".
2. GAME SCREEN:
   - Prefer keys defined in the instructions.
   - Try to move, jump, shoot, or interact based on what you see and on the instructions provided before.
   - If stuck (no changes in history), switch actions immediately (e.g., moving, jump, or shoot).
3. Loop Prevention: NEVER repeat the same action more than 2 times unless you see clear progress.

You should think step by step and respond with the following format.

Response:

Observation: ... (Describe what you see in the last frame, the character's current position, nearby obstacles and enemies, and if this is a menu or game.)

Reasoning: ... (Based on the current observation, think step by step and explain how you choose the action.)

Action: ... (One action line only: a key ("space"), a chord ("shift+a"), a timed hold ("hold:left:0.8"), "wait", or "CLICK: x, y". Do NOT add any other words.)

Your response:

Figure 19: Shared VLM user prompt template, including game-context placeholders, action instructions, and the response format.

Examples:
- Observation: The screen displays the main menu of a game with "Start" and "Quit" options. There is no gameplay visible yet, indicating the game is not actively running. The player must begin the game to proceed.
  Reasoning: Since this is a menu screen, the correct action is to click the "Start" button to begin the game. According to the control rules, for menu buttons like "Start", I must output the coordinates of the button.
  Action: CLICK: 100, 500
- Observation: The player character is standing in a dark, enclosed space, without visible enemies or obstacles yet.
  Reasoning: Since the game has just started, the next logical step is to explore the area and interact with objects to progress. The instructions state that I need to use 'x' to interact with the environment.
  Action: x
- Observation: The player character is still at position (370.0, 700.0) on the same grassy platform. The environment has not changed, with dark matter and enemies visible on the right. The player is stuck and stagnant.
  Reasoning: Repeated attempts to move right ('d') have not resulted in any progress, as the player remains in the same position. This indicates the action is ineffective. I must change strategy. I should try jumping.
  Action: w

Your response:

Figure 21: Few-shot examples demonstrating structured reasoning and control-token selection for generalist VLM evaluation complementing the user prompt template shown in Figure 19.

Please generate the next move according to the UI screenshot, instruction and previous actions.

Instruction: You are playing an itch.io game called "<$game_name$>".<$category_suffix$> Play the game to make as much progress as possible (start the game from any menu, then control the character).

<$game_description$><$readme_section$>

Game-specific guidance:

- If the screen shows a menu, click the button that starts the game ("Start", "Play", "New Game" or similar) with left_click.
- During gameplay, PREFER keyboard actions via the 'key' action: keys listed in the game instructions above take priority; otherwise try ["a"]/["d"]/["w"]/["s"] to move, ["space"] to jump or act, ["enter"] to confirm, ["escape"] for menus.
- Use left_click during gameplay only for aiming or shooting when the instructions mention the mouse.
- If the screen shows a game-over, death, victory, or level-complete screen (text like "Game Over", "You Died", "Retry", "Resume", "Restart", "Continue", "Next Level", "Play Again", "Try Again"), left_click the continue/retry button; if there is no visible button (just text), use the 'key' action with ["enter"] first, then ["space"] or ["escape"].
- Never use 'terminate'; if nothing needs doing, use 'wait'.
- Do not repeat an action that produced no visible change; try a different key or click target.

Previous actions:

<$history_section$>

Figure 20: Computer-use template used by EvoCUA and GUI-Owl.

You should think step by step and respond with the following format.

Response:

Observation: ... (Describe what you see in the last frame, the character's current position, nearby obstacles and enemies, and if this is a menu or game.)

Strategy: ... (Describe your overarching strategy at this point of the game.)

Reasoning: ... (Based on the current observation, think step by step and explain how you choose the action.)

Action: ... (One action line only: a key (`"space"`), a chord (`"shift+a"`), a timed hold (`"hold:left:0.8"`), `"wait"`, or `"CLICK: x, y"`. Do NOT add any other words.)

Your response:

Figure 22: **Strategy setting.** Response-format excerpt for the eight VLMs using the shared prompt. The variant adds the `Strategy` field between `Observation` and `Reasoning` in the user template (Figure 19); the shared system instructions (Figure 18) are unchanged.