IPR-1: Interactive Physical Reasoner

Supplementary Material

In this supplementary, we further provide the additional contents as follows:

1. Further Discussion

Recent progress in world models and interactive agents has produced systems that can predict future states, learn latent dynamics, and act across large numbers of games. While we share certain design choices with these systems—such as learning latent dynamics, adopting multimodal interfaces, and scaling across diverse environments—our motivation is fundamentally different. Rather than optimizing for task performance within a single domain, we aim to approximate cross-domain invariants: the physical and causal structures that remain stable across heterogeneous worlds. Below, we situate our paradigm relative to representative systems, organized by methodological families.

World-model-centric approaches. Early world-model work and the Dreamer series [9–13] show that learning a latent dynamics model and updating a policy from imagined rollouts can master diverse control tasks from pixels. Dreamer progressively strengthens this paradigm: DreamerV1/V2 introduce latent RSSM dynamics with imagination-based actor–critic updates; DreamerV3 demonstrates that a single configuration can reliably solve over 150 tasks across Atari, continuous control, and Minecraft; Dreamer4 further improves robustness and exploration in harder, long-horizon domains. Collectively, these results establish that prediction in latent space is a powerful tool for efficient RL and long-horizon control.

The Genie family [4, 6, 18] takes a complementary step by treating the world model itself as a *generative environment*. Genie learns a latent action interface from Internet videos and uses it to drive a video world model that converts text or frame prompts into interactive, playable environments. Subsequent versions (Genie-2/3) extend this idea to longer, higher-resolution, and partially 3D worlds with persistent object state and richer user interaction, suggesting that latent world models can serve as general-purpose sandboxes for training and evaluating agents rather than only internal simulators.

V-JEPA and V-JEPA 2 [2, 5] further push prediction into the feature space: instead of reconstructing pixels, they learn joint-embedding predictive encoders on Internet-scale video. V-JEPA 2-AC augments this with a latent action-conditioned head trained on a small amount of robot interaction data, showing that purely self-supervised video pretraining can be post-hoc adapted into an actionable world model capable of zero-shot manipulation without per-task finetuning. This line of work highlights that high-quality dynamics for physical reasoning do not require pixel-level supervision.

SIMA and SIMA-2 [23, 24] focus on building scalable, instructable multiworld agents in 3D games. SIMA trains a vision–language–action system that follows free-form language instructions across many commercial titles via keyboard-and-mouse control, demonstrating that a single agent can generalise across heterogeneous game interfaces and tasks. SIMA-2 upgrades this framework with a stronger backbone and richer virtual worlds, improving instruction following and in-context learning of new tasks. However, both SIMA variants largely treat the environment as a black box: they rely on language-driven policy learning rather than explicit latent dynamics for imagination or planning.

VLM/VLA-centric agents. A parallel line of work builds *vision–language(-action)* agents that treat the game as a black-box interface and learn to map instructions and screen pixels directly to high-level actions. Early VLA-based game agents [7, 21, 25] explore this direction by combining pretrained VLMs with keyboard–mouse or GUI control, often wrapping the environment through OS- or browser-level APIs. These systems demonstrate that a single pretrained backbone can drive diverse games and applications with minimal task-specific finetuning, but typically rely on scripted tools, slow deliberation, or narrow benchmarks.

Game-TARS [26] pushes this paradigm to scale. It trains a generalist game agent with a unified, human-aligned keyboard–mouse action space, pretraining on hundreds of billions of multimodal tokens collected from OS, web, and simulation games. This large-scale pretraining, together with continual-loss scheduling and sparse-thinking strategies, yields strong performance across open-world Minecraft, web-based 3D games, and FPS benchmarks, often surpassing general-purpose VLMs of comparable size. The key insight is that anchoring the action space to a human-native interface enables broad reuse of trajectories and supports scalable cross-domain training.

Lumine [22] provides an open recipe for building real-time generalist agents in 3D open worlds. Powered by a VLM backbone, Lumine processes raw pixels at low fre-

quency while emitting precise 30 Hz keyboard–mouse actions, and adaptively invokes heavier reasoning only when necessary. Trained primarily in a single but rich title (Genshin Impact), it completes hours-long storylines, handles diverse tasks such as exploration, combat, and puzzle solving, and zero-shot transfers to other games with different graphics and interaction dynamics. This line of work underscores that strong semantic reasoning, combined with human-like interaction loops, can already produce impressive in-domain and cross-game competence.

$\pi^{\star}_{0.6}$ [14] is an RL-enhanced large model trained with preference optimization and long-horizon interactive rollouts, producing a general policy that exhibits strong cross-environment competence in web tasks, games, and interactive reasoning. The model benefits heavily from scale, both data scale and model capacity, and demonstrates that sufficiently large policies can generalize to unseen tasks with minimal task-specific engineering. However, $\pi^{\star}_{0.6}$ does not expose an explicit latent dynamics model, nor does it articulate how prediction or physical regularities structure the policy; its improvements originate primarily from reinforcement tuning on massive interaction data rather than structured cross-domain abstractions.

Our interactive physical reasoner. In contrast to the above lines of work, our motivation is explicitly *cross-domain*. Games differ dramatically in appearance, controls, and reward structures, yet we observe that many of them instantiate a small set of shared *physical and causal mechanisms*: gravity, collisions, momentum exchange, and contact-driven state changes. Crucially, these mechanisms tend to be expressed not in pixels but in the *action space*: actions are the agent's only means to interactively induce physical effects, and different domains often implement similar effects (jump, move, dash, interact) even under mismatched key layouts and visuals.

This suggests that a domain-invariant interface should be built not from raw controls but from a latent action space that captures *what the action does to the world*. Inspired by the intuition-based action extraction in Genie, we learn such a space—*PhysCode*—by encoding visual cues around hand–object–scene interactions and letting a VQ codebook automatically cluster domains whose actions induce similar physical outcomes. PhysCode, therefore, materializes the shared causal structure across heterogeneous games.

With a unified action space in hand, the next question is how to model dynamics. We evaluate both pixel-space and latent-space prediction in the ablation study 4.1, and consistent with V-JEPA–style findings, latent dynamics are substantially more efficient and more stable. However, a pure world model—even with accurate latent rollouts—remains confined to intuitive physics and short-horizon prediction; it lacks the high-level reasoning and cross-domain abstraction needed for complex tasks.

To close this gap, we bring in a pretrained VLM that already exhibits some cross-domain generalization in games, as evidenced by works like Game-TARS, but Game-TARS typically relies on costly human prompts and annotations. Instead, we aim to let the agent learn *directly* from interactive environments, using prediction inside the loop of action selection. Our IPR framework couples the two components through PhysCode: the VLM observes the current visual context and task description, then proposes candidate latent actions; the world model performs short-horizon rollouts in PhysCode space to forecast their physical consequences; and a GRPO-style objective reinforces VLM policies whose imagined futures are safe, physically consistent, and task-aligned. In this way, prediction is no longer just an exploration aid—it becomes an in-the-loop imagination mechanism that continuously sharpens the VLM's physical and causal reasoning across domains.

2. Benchmark Details

2.1. Game Sources

Retro games. We curate 863 open-source retro titles via STABLE-RETRO [19], covering NES, SNES, GENESIS, SMS consoles, *etc*. These environments provide frame-perfect emulation with discrete controller actions (D-pad directions, up to four face buttons, and start/select), and span a wide range of genres including *platformers*, *shooters*, *sports*, *racing*, *etc*. We focus on titles where motion and interaction are governed by clear physical rules (jumping under gravity, rigid-body collisions, projectile–enemy interactions, kinematics, *etc*.). For each game, we annotate the dominant *physical* mechanism (*e.g*., platformer gravity, rigid-body contact, projectile motion, kinematic logic) and *causal* structure (*e.g*., resource-accumulation objectives, score-based progression, survive-as-long-as-possible tasks, shortest-time-to-goal objectives, or unlocking mechanisms to obtain rewards). This diversity encourages agents to capture shared physical–causal regularities rather than overfit to title-specific sprites, textures, or control layouts.

HTML games. We additionally include 134 HTML/Canvas games collected from public web repositories, comprising both license-free and permissively licensed titles. Compared to retro consoles, these games rely heavily on mouse and mixed mouse+keyboard interaction (click, drag, hold, scroll), often with modern 2D physics engines (*e.g*., Box2D-style rigid-body dynamics). We instrument a Chromium-based browser with a lightweight JavaScript/Playwright wrapper to (i) capture rendered canvas frames at a fixed frame rate and (ii) log low-level input events (mouse position, button state, and keyboard presses) together with timestamps. When available, we also

Figure 1. **Overview of our 1,000 games,** containing old-fashioned retro games, HTML/canvas games, and modern commercial games.

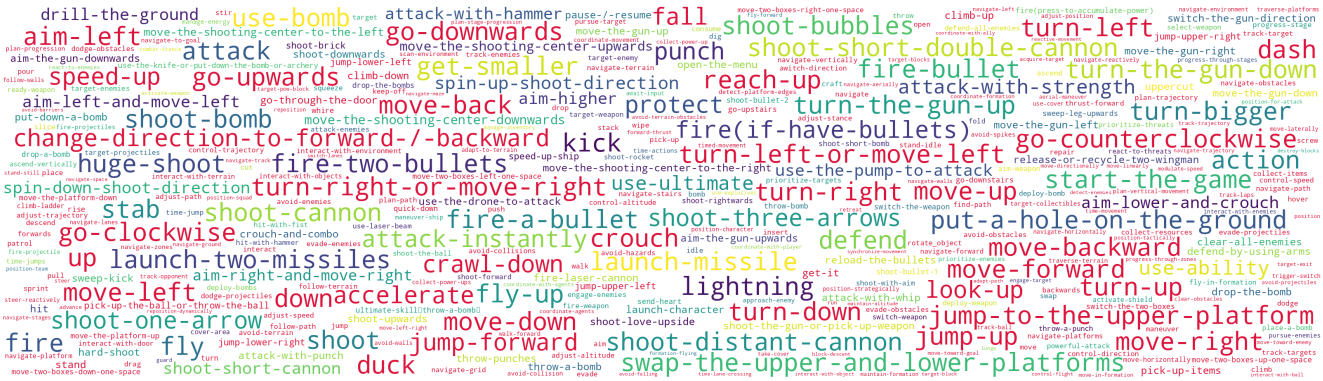

Figure 2. **Word cloud of action semantics across thousands of game worlds.** These shared semantics provide the structural foundation for cross-domain transfer. Actions highlighted in red represent those shared with general robotic operations, while the size of each word reflects its frequency in our data recipe.

read a small set of game variables exposed in JavaScript (*e.g.*, score, level, remaining lives) as auxiliary state. To unify control across heterogeneous HTML titles, we define a hybrid action space consisting of: (1) a discrete keyboard state vector (one-hot over pressed keys); (2) a continuous mouse position represented by normalized coordinates $(x, y) \in [0, 1]^2$, automatically scaled to each canvas size; and (3) mouse interaction flags, including left/right click, hold (for continuous dragging), and scroll. This representation covers the vast majority of HTML/Canvas interaction patterns while remaining compatible with the discrete and low-dimensional interfaces used by other environments in our benchmark.

Commercial games. Finally, we include 3 lightweight commercial games using properly licensed PC builds: PORTAL 2, ASTRONEER, and MINECRAFT. We respect copyright and treat these games as black-box applications, and do not modify their binaries or access internal source code. Instead, the agent interacts through a virtual desktop: we capture RGB frames from a virtual display, and wrap human-like input via an emulator layer that maps keyboard events to discrete action indices and converts mouse movement/clicks into the same grid-based macro-actions used for HTML games. For real-time titles, we run the game in a frame-stepped mode: the environment advances one step only after an action is issued, and remains effectively paused while the agent performs reasoning, so decision latency does not affect in-game timing. To expose the higher-level semantic state of the game, each commercial game is paired with a pre-defined list of target objects and goals. We record rich-perspective gameplay videos for each title and uniformly sample $\sim$150 frames per game. On these frames, we manually annotate bounding boxes for all target categories (*e.g.*, cubes, buttons, and portals in PORTAL 2), and use them to train a lightweight YOLOv11 [15] detector. At run time, this detector provides object-centric cues on top of raw RGB, which we use to enrich prompts and

evaluation without changing the underlying game binaries. Across retro, HTML, and commercial games, we use a unified logging interface to record consistent (x_t, a_t, r_t, x_{t+1}) trajectories (r_t from the extracted state and rules), enabling joint training and evaluation under a shared interaction format.

2.2. Data Collection and Preprocessing

Human gameplay recording. For each game, we collect **4 minutes** of human interaction data from **at least two independent players** to balance individual skill biases. If the two players' performance exhibits a large score gap (typically $> 1.5\times$ difference), we recruit a **third annotator** to provide additional trajectories, ensuring stable coverage of feasible strategies and reducing overfitting to a single playstyle. Each trajectory is stored as a sequence of stepwise tuples

$$(x_t, a_t, r_t, x_{t+1}), \tag{1}$$

where x_t is the rendered frame, a_t the human action, r_t the instantaneous game reward (if available), and x_{t+1} the next frame.

Frame rates and control logging. For **retro** titles (NES, SNES, Genesis, SMS), we adopt the **native system frame rate** provided by the emulator and record the console's discrete button events. For **HTML/Canvas** and **commercial games**, we instead capture frames at a unified **60 FPS**, together with full logs of keyboard events, mouse deltas, and console states, so that all sources can be brought to a common temporal resolution.

Semantic action and physics/causality annotation. During recording, annotators additionally provide **lightweight semantic tags** for each short action segment. These tags describe both *what* characters are doing: they include action semantics (*e.g.*, *jump*, *dodge*, *charge*, *aim*, *grab*), local physical principles (*e.g.*, *gravity-driven fall*, *sliding under friction*, *momentum carry-over*), and simple causal relations (*e.g.*, *hit switch* $\rightarrow$ *open door*, *push object* $\rightarrow$ *block hazard*), as well as short goal/instruction snippets describing the intended skill or sub-task. These semantics are later used in Sec. **??** in the main paper as grounding signals when inducing semantics-aware actions.

Data preprocessing. To ensure uniform sequence quality across heterogeneous sources, we apply a series of preprocessing steps. First, we **normalize time intervals**: retro games provide fixed-step transitions through the emulator, while HTML and AAA titles may exhibit variable render intervals. We resample all trajectories to an aligned 60 Hz timeline and interpolate missing states when necessary, so that downstream models can assume a fixed time step. Second, we **remove non-interactive segments** such as cutscenes, loading screens, menus, and extended full-idle periods. These segments are automatically detected using simple motion statistics and input-entropy thresholds over recent frames and key/mouse events. Third, we **rebalance idle or no-op periods**: players often hold still or wait for environmental cycles, which would otherwise dominate the dataset. We therefore downsample long idle windows (*e.g.*, keeping only 1 out of every k idle frames) while explicitly preserving the beginning and end of each idle episode to maintain temporal context. Finally, we apply **action smoothing and deduplication**: for mouse movement and other analog-like controls, we smooth out small jitter to avoid spurious micro-movements; for discrete actions, we collapse repeated no-ops or very short flicks that do not meaningfully change the game state.

2.3. Evaluation Metrics

Survival. measures the average number of environment steps an agent survives before an irreversible failure (*e.g.*, losing all lives, falling into a death pit, running out of health, or entering a terminal game-over state). It captures the agent's ability to avoid risk, prevent collisions, and keep the episode alive.

Game horizons vary widely: in some titles, a random policy dies within tens of steps, while in others it can wander for thousands. To obtain comparable scores, we derive a per-game *step scale* from the order of magnitude of a random policy's lifetime. For each game m, we estimate $L^{(m)}_{\text{rand}} = \mathbb{E}[\text{steps}_{\text{rand}}]$, and define

$$H^{(m)} = 10^{\lfloor \log_{10} L^{(m)}_{\text{rand}} \rfloor + 1}. \tag{2}$$

Given an agent with average lifetime $L^{(m)}_{\text{agent}} = \mathbb{E}[\text{steps}_{\text{agent}}]$, the normalized score is

$$\text{SurvivalScore}^{(m)} = \frac{L^{(m)}_{\text{agent}}}{H^{(m)}}. \tag{3}$$

Here $H^{(m)}$ acts as a game-dependent "step unit" (*e.g.*, 100, 1000, 10000), keeping survival values in a comparable range without directly normalizing by the exact random baseline.

Curiosity. is designed to capture how broadly an agent explores the environment, beyond merely staying alive. We measure exploration as the *area* of the state space that an agent visits, computed in a learned representation space using MAGNIPY. Concretely, we subsample frames from each evaluation episode and embed them with a pretrained vision encoder CLIP to obtain feature vectors $\{f_t\}$.

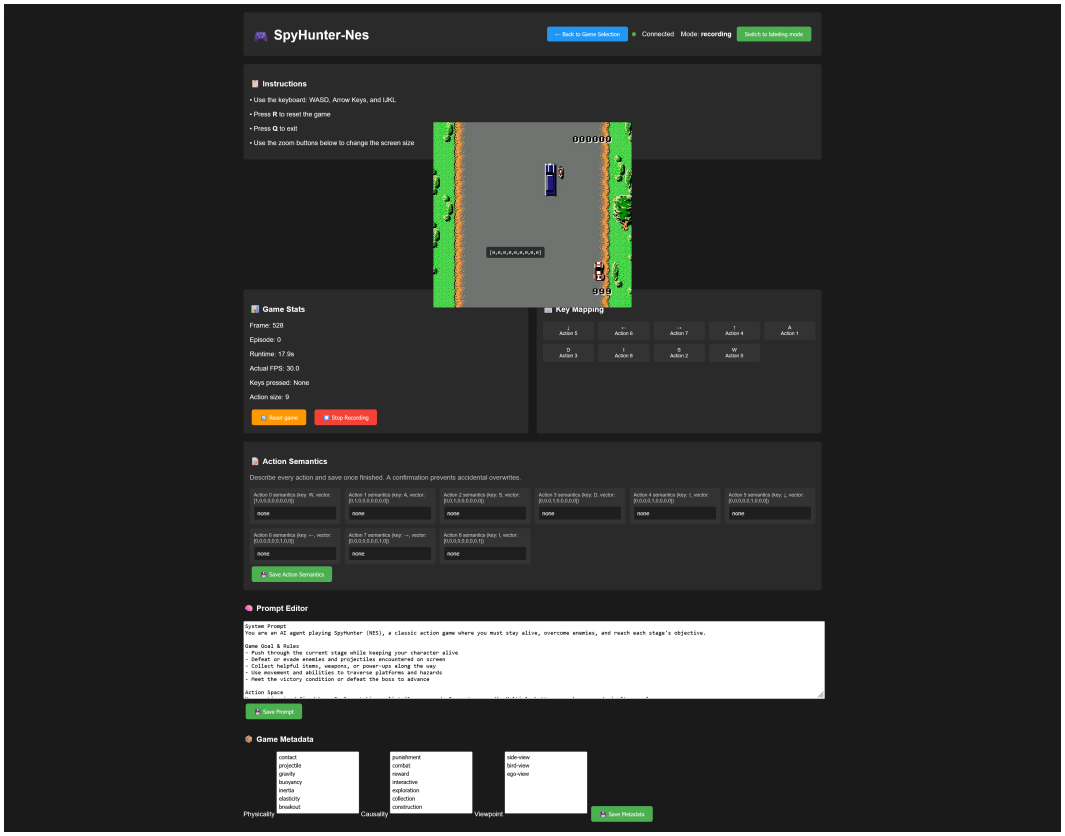

Figure 3. **Overview of our game-recording website tools.**

We then apply MAGNIPY [1, 16] to these features to estimate the volume of the region covered by the agent's trajectory: MAGNIPY treats each feature as a point in the embedding space and approximates the union of local neighborhoods around these points, yielding a scalar coverage score that increases when the agent visits new, diverse states and saturates when it revisits already explored regions. We compute this coverage per episode and average across episodes for each game.

👑**Utility.** 👑 measures progress toward explicit task goals, such as maximizing score, winning matches, or completing puzzles. Because different games expose different reward signals, we unify them into a scalar *game score* before normalization. In practice, our score types include: (i) raw in-game numerical scores for arcade-style titles; (ii) binary or fractional success indicators for win/loss and puzzle-completion tasks; (iii) time- or step-based objectives where finishing earlier yields a higher score (we invert and rescale time so that "faster is better"); and (iv) resource-based objectives (*e.g.*, items collected, checkpoints reached) that reflect underlying causal goals such as *collect resources to unlock new areas* or *clear all enemies to progress*. To compare utility across games, we report a human-normalized score (HNS). For each game, we measure the average score of a random policy, $\text{score}_{\text{rand}}$, and the average score of human players, $\text{score}_{\text{human}}$. Given an agent with average score $\text{score}_{\text{agent}}$, we define

$$\text{UtilityScore} = \frac{\text{score}_{\text{agent}} - \text{score}_{\text{rand}}}{\text{score}_{\text{human}} - \text{score}_{\text{rand}} + \varepsilon}, \tag{4}$$

optionally clipped to a reasonable range for robustness. A value of 0 indicates random-level performance, 1 roughly corresponds to human-level performance, and values above 1 reflect super-human success on the game's causal objectives.

3. Implementation Details

3.1. PhysCode

Inputs and temporal windowing. Given a gameplay video with per-step controls, we construct short clips of length $T{=}8$. For each time index t, we form a triplet $(x_t, x_{t+\Delta}, y_t)$, where x_t and $x_{t+\Delta}$ are RGB frames and y_t is a lightweight textual description of the executed control (*e.g.*, "move right and jump"). We extract three cues: (i) DINOv3 appearance features

$f_t, f_{t+\Delta} = \phi_{\text{DINO}}(x_t), \phi_{\text{DINO}}(x_{t+\Delta})$ from the final patch tokens (global-pooled to a single 1024-d vector), (ii) dense optical flow $u_t = \text{Flow}(x_t, x_{t+\Delta})$ computed by a FlowNet-style network and downsampled to match the DINOv3 patch grid, and (iii) semantic embeddings $e_t = \phi_{\text{T5}}(y_t)$ from a frozen T5 encoder (we use the [CLS] token as a 768-d vector). For efficiency, we precompute f_t, $f_{t+\Delta}$ and u_t offline and only store the compact intermediate representations.

Gated modality fusion. To form a physics-centric token at time t, we first project each modality to a shared d-dimensional space (d=512 by default):

$$\tilde{f}_t = W_f f_t, \quad \tilde{u}_t = W_u \,\text{Pool}(u_t), \quad \tilde{e}_t = W_e e_t, \tag{5}$$

where $\text{Pool}(\cdot)$ is a spatial average pooling over the flow field. A small gating MLP $g(\cdot)$ outputs unnormalized gates $(\alpha_f, \alpha_u, \alpha_e)$ conditioned on the concatenation $[\tilde{f}_t; \tilde{u}_t; \tilde{e}_t]$. We then form normalized gates via a softmax:

$$w_m = \frac{\exp(\alpha_m)}{\sum_{m' \in \{f,u,e\}} \exp(\alpha_{m'})}, \quad m \in \{f, u, e\}, \tag{6}$$

and obtain the fused representation

$$h_t = w_f \tilde{f}_t + w_u \tilde{u}_t + w_e \tilde{e}_t. \tag{7}$$

To avoid over-reliance on privileged motion cues, we apply *flow dropout* with probability p=0.5: when dropped, the flow feature $\tilde{u}_t$ is replaced by zero and the gates are renormalized over $\{f, e\}$. We further add an ℓ_1 penalty $\lambda_{\text{gate}} \sum_t \sum_m |w_m - \bar{w}_m|$ to discourage degenerate single-modality solutions, where $\bar{w}_m$ is a uniform prior.

Spatio-temporal encoder and codebook. A spatio-temporal encoder E_ψ maps the fused sequence $\{h_{t-k}\}_{k=0}^{T-1}$ to a continuous latent $z_t \in \mathbb{R}^d$. We instantiate E_ψ as a lightweight 6-layer Transformer with hidden size d=512, 8 attention heads, and a temporal positional embedding; only the last token (corresponding to t) is used for code assignment. We maintain a VQ codebook $\mathcal{C} = \{c_k\}_{k=1}^{K}$ with K=256 codes of dimension d, updated with EMA. The continuous latent z_t is quantized to the nearest entry

$$a_t = \arg\min_{k \in \{1,\ldots,K\}} \|z_t - c_k\|_2^2, \qquad \hat{z}_t \equiv c_{a_t}, \tag{8}$$

and an *action* is defined as a short sequence $a_t^{\text{LAT}} = \langle c_{t,1:L} \rangle$ by taking a sliding window of L consecutive indices (we use L=4 by default). The sequence representation is obtained by averaging pooling the corresponding embeddings $\{c_{c_{t,\ell}}\}_{\ell=1}^{L}$.

Training objective and prediction head. Given $(f_t, f_{t+\Delta}, h_t)$, the decoder D_ψ predicts the future feature $\hat{f}_{t+\Delta} = D_\psi(f_t, \hat{z}_t)$. We use a standard VQ-VAE loss with a feature-prediction target:

$$\mathcal{L}_{\text{LA}} = \left\|\hat{f}_{t+\Delta} - f_{t+\Delta}\right\|_2^2 + \beta \left\|\text{sg}[z_t] - \hat{z}_t\right\|_2^2 + \gamma \left\|z_t - \text{sg}[\hat{z}_t]\right\|_2^2, \tag{9}$$

where $\text{sg}[\cdot]$ denotes stop-gradient and (β, γ) are codebook and commitment weights (set to 0.25 and 0.25 respectively). We additionally attach a light temporal head that predicts $\Delta f_{t+\Delta} = f_{t+\Delta} - f_t$ from $\hat{z}_t$, encouraging codes to align with dynamical changes rather than static appearance.

Optimization and data usage. PhysCode is pretrained on the entire 1,000+ game corpus. We randomly sample 4-minute human trajectories per title and extract $(x_t, x_{t+\Delta}, y_t)$ with $\Delta \in \{1, 2, 4\}$, balancing short- and medium-term dynamics. We train for 500k steps with AdamW (learning rate 1×10^{-4}, weight decay 0.05, cosine schedule with 5k warmup), batch size 1024 clips, and gradient-norm clipping at 1.0. DINOv3, FlowNet, and T5 encoders are frozen; only the fusion module, Transformer, codebook, and decoder are learned. We found that smaller K (*e.g.*, 128) collapses dynamics from distinct physics into shared codes, while much larger codebooks ($K \geq 2048$) hurt sample efficiency and lead to under-used codes.

Inference-time usage. At test time, optical flow is not available. We therefore disable the flow gate by fixing w_u=0 and renormalizing over $\{f, e\}$, and reuse the same E_ψ and codebook to obtain a_t from appearance+semantics only. The resulting discrete tokens form a temporally predictive vocabulary that (i) clusters trajectories with matched physics (*e.g.*, gravity+contact) and (ii) stays separable under physics shifts, and are used as the shared action interface for both the world model and the VLM in IPR.

3.2. Interactive Physical Reasoner

We detail the three stages of IPR and the training protocol used in our experiments.

Stage 1: PhysCode pretraining. IPR builds on the PhysCode vocabulary described in Sec. 3.1. We first pretrain PhysCode on human gameplay across all 1,000+ games. Environments are sampled uniformly over titles and replay segments, and we enforce a balanced mixture over physical mechanisms (gravity, projectile, contact, etc.) to avoid overfitting to a single physics family. The resulting codebook and encoder are frozen for all subsequent stages.

Stage 2: Latent-conditioned world model with a critic. Given fixed PhysCode indices, we replace raw controls by

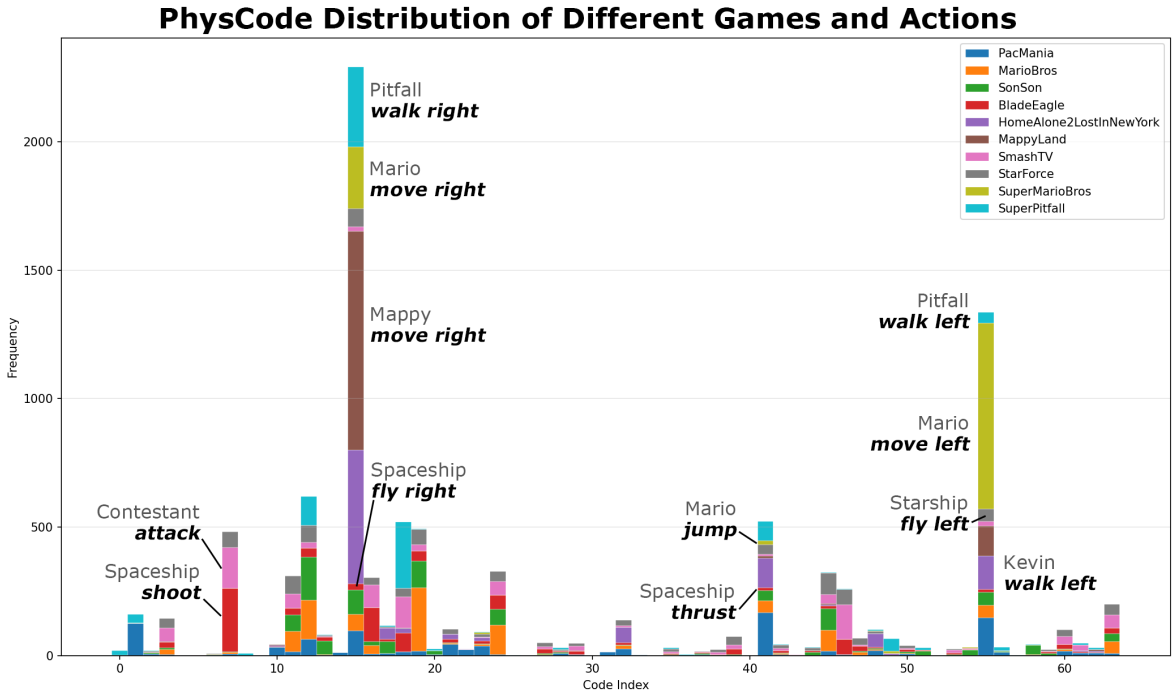

Figure 4. **Distribution of *PhysCode* in different game domains.** Some action codes share across games, typically `move right`, `jump`, while others are separated according to different physical domains.

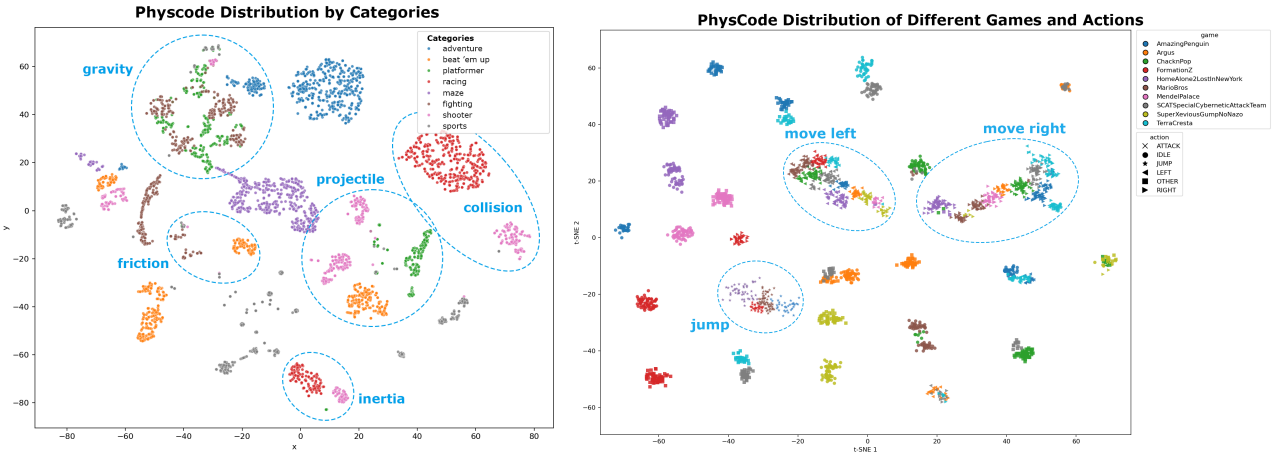

Figure 5. **Qualitative analysis of PhysCode latent space.** The visualization demonstrates that the learned representations effectively cluster by both high-level physical concepts (left) and fine-grained action-game pairs (right).

latent action tokens. For each transition $(f_t, a_t, r_t, f_{t+\Delta})$ we embed a_t into an action embedding $e_t^a \in \mathbb{R}^d$, concatenate with the current feature, and feed into a feature-level predictor P_θ:

$$(\hat{f}_{t+\Delta}, V_\theta(f_t, a_t)) = P_\theta\big([f_t; e_t^a]\big), \tag{10}$$

where V_θ is a scalar value head sharing all but the last layer with the feature predictor. We implement P_θ as an 8-layer Transformer over short latent trajectories (length H) to capture multi-step interactions; during training, we unroll on real data segments of length H=5.

World model training is split into two phases: (i) a pure prediction phase with

$$\mathcal{L}_{\text{pred}} = \big\|\hat{f}_{t+\Delta} - f_{t+\Delta}\big\|_1, \tag{11}$$

using only pre-collected trajectories and no reward infor-

mation, followed by (ii) a value-learning phase, where we freeze the dynamics layers and only train the critic head using a TD-style loss

$$\mathcal{L}_{\text{value}} = \ell_Q\Big(V_\theta(f_t, a_t),\ r_t + \gamma \max_{a'} V_{\theta^-}(f_{t+\Delta}, a')\Big), \tag{12}$$

with a slowly updated target network θ^-. This separation stabilizes learning: the dynamics focus on physics-consistent feature evolution, while the critic adapts to level-specific reward scales.

Stage 3: VLM alignment to PhysCode. We adopt Qwen3-VL-8B as the backbone and extend its tokenizer with K special PhysCode tokens $\{\langle \texttt{PC\_k} \rangle\}_{k=1}^{K}$. Each PhysCode index a_t is mapped to its corresponding token, enabling the VLM to produce latent actions as part of its normal autoregressive decoding.

We first perform a perception–action alignment stage on 10k human frame–action pairs. For each pair $(x_t, a_t^{\text{LAT}}, g)$, where g is a textual goal or instruction, we format the input as an interleaved image–text prompt and the target as the PhysCode sequence:

$$\texttt{[IMG(}x_t\texttt{)]}\ \ \texttt{``Goal:}\ \ g\texttt{''} \rightarrow \langle \texttt{PC}_{c_{t,1}} \rangle \ldots \langle \texttt{PC}_{c_{t,L}} \rangle.$$

We train with a standard teacher-forced cross-entropy loss only on the PhysCode tokens, keeping most of the language parameters close to their initialization via a small learning rate and weight decay. This stage teaches the VLM to (i) parse visual context, (ii) understand goals, and (iii) output correctly structured PhysCode sequences.

Stage 4: Prediction-reinforced GRPO. After alignment, we place the world model in the loop and train the VLM with GRPO using imagined rollouts. At each real environment step:

1. Encode the current frame x_t to f_t using the same DINOv3 encoder as in PhysCode.
2. Condition Qwen3-VL on x_t and the current task prompt g, and *sample* B candidate latent action sequences $\{a_t^{(b)}\}_{b=1}^{B}$ (we use B=8, temperature 0.7, and top-p=0.9).
3. For each candidate $a_t^{(b)}$, unroll the world model for H steps in feature space, obtaining a predicted return $\hat{R}_t^{(b)}$ from the critic head (discount factor γ=0.99).
4. Normalize returns within the candidate set to compute advantages $A^{(b)} = (\hat{R}^{(b)} - \bar{R})/(\sigma_R + \varepsilon)$.
5. Update the VLM with the GRPO objective

$$\mathcal{L}_{\text{GRPO}} = -\frac{1}{B}\sum_{b=1}^{B} A^{(b)} \log \pi_\phi(a_t^{(b)} \mid x_t, g) + \beta_{\text{KL}}\, \text{KL}\big(\pi_\phi(\cdot \mid x_t, g) \,\|\, \pi_{\phi_0}(\cdot \mid x_t, g)\big), \tag{13}$$

where π_{ϕ_0} is the initial aligned VLM and β_{KL} controls a conservative trust region.

We interleave real environment interaction and imagination-based updates in a 1:k ratio (one real step followed by k=4 imagination-only updates sampled from a replay buffer of recent contexts), which significantly improves data efficiency.

Inference and control routing. At test time, the world model remains in the loop but no longer updates. Given (x_t, g), the VLM proposes B candidate PhysCode sequences as during training; the world model scores them via short-horizon imagination, and we execute the highest-scoring candidate. A lightweight router T_{env} maps the selected PhysCode sequence to environment-specific controls (keyboard/mouse macros or gamepad buttons) using a per-game lookup table learned from human trajectories and short calibration episodes. This keeps the reasoning and prediction in a unified latent space while adapting only a small mapping layer to each new game.

Overall, these stages realize IPR as a *prediction-reinforced reasoning* loop: PhysCode provides a physics-organized latent action interface, the world model supplies imagination and value estimates in this interface, and the VLM is continually refined to prefer actions whose imagined consequences lead to safer survival, broader exploration, and higher utility.

3.3. Exp 1: PhysCode Validation Setup

Data and model implementation. To investigate how different action spaces influence the learning of shared physical dynamics across heterogeneous environments and their ability to generalize to unseen games, we curate a representative subset of 200 games from our benchmark. For each game, we collect a dataset of 1 million frames paired with ground-truth actions generated by a random policy. We employ **V-JEPA 2-AC** [2] and **GenieRedux** [20] as the backbone world models. For V-JEPA 2-AC, we first train a ViT-L image encoder from scratch on the combined 200-game dataset (refer to Table 3 in the main paper for encoder ablations). We then post-train the predictive components conditioned on three inputs: (i) the previous frame's image latent z_{t-1}, encoded from the raw pixel frame; (ii) the previous action a_{t-1}, the representation of which varies by experimental setting (detailed below); and (iii) an auxiliary state vector s_{t-1}, which is set to a zero vector for all experiments in this section.

Action conditioning variants. The core variable in this experiment is the representation of the action input a_{t-1}. We compare four distinct configurations:

1. **Keyboard (raw shared).** We train a single model jointly across all games using raw control inputs. We determine the maximum button configuration size within this

200-game domain ($D_{\max} = 12$) and pad the multi-hot vectors of simpler controllers with zeros to match this dimension. This represents a naive union of hardware interfaces, where all inputs are normalized to a fixed $1 \times D_{\max}$ vector.

2. **Language (semantic shared).** We construct a unified semantic action space to resolve the aliasing of raw keys (*e.g.*, key `A` may trigger `Jump` in one game but `Attack` in another). We manually annotate the function of every button in every game using natural language and create a superset of all unique semantics, resulting in a global semantic vector of size $D_{\text{sem}} = 173$ (covering actions such as `move left`, `jump`, `shoot`). We generate a static mapping matrix for each game that projects its raw multi-hot vector into this sparse, 173-dimensional global vector.
3. **PhysCode (Ours).** We use the discretized latent codes derived from our proposed method. As described in Sec. **??** of the main paper, raw actions are replaced by quantized indices $a_t \in \{1, \ldots, K\}$ from the learned codebook (we set $K = 256$). These indices are projected via a learnable embedding layer before being fed into the world model. Unlike language, this aligns actions based on physical dynamics (*e.g.*, momentum, contact) rather than human-defined semantics.
4. **Ad-hoc (Single-game expert).** We train a separate world model for each game individually. The input a_{t-1} is the raw game-specific multi-hot vector with dimension $1 \times D_{\text{game}}$. This serves as an oracle upper bound for intra-game prediction quality but lacks any cross-game generalization capabilities.

Evaluation protocols. We evaluate these representations across three regimes, corresponding to the results reported in Table 1 of the main paper:

- **Confusion test (joint training).** We train a single model on the union of all 200 games and evaluate it on the training set (Table 1a). This measures the model's ability to handle conflicting control schemes (interface aliasing) without performance degradation.
- **Leave-n-out transfer.** We then evaluate the same model from the joint training phase on a separate, held-out set of 10 unseen games that were not part of the training data (Table 1b). This protocol tests true zero-shot generalization to entirely new environments using the shared action interface.
- **Physics-conditioned transfer.** To disentangle semantic generalization from physical grounding, we categorize games into four dominant mechanisms (*e.g.*, *Gravity*, *Inertia*). We train specialized models on subsets of 20 games sharing a single mechanism (using the Language-aligned model as a baseline) and evaluate them on held-out games that either match or mismatch the training physics (Tab. 1c in the main paper). This verifies whether the action space captures reusable physical laws or merely memorizes semantic bindings.

3.4. Exp 2: One Model for All Games

RL. To instantiate a unified, multi-task model for both PPO and DQN algorithms, we employ a dynamic parameterization scheme. This is achieved by integrating *task embeddings* with a *hypernetwork* architecture. The core idea is to condition the parameters of the policy and value functions directly on the task identity, enabling a single model to specialize its behavior across different tasks.

The training procedure for a given task is as follows:

1. **Data collection**: Agent interacts with the environment to collect trajectory data $\tau = (s_t, a_t, r_t, s_{t+1})$.
2. **Task conditioning**: The current task ID z is mapped to a continuous vector representation $\mathbf{e}_z$ (the task embedding).
3. **Parameter generation**: The task embedding $\mathbf{e}_z$ is fed into a hypernetwork h_ϕ, which outputs the parameters θ_z for the target network:
   - For DQN: θ_z defines the weights of the Q-network.
   - For PPO: θ_z defines the weights of the actor $\pi(a|s; \theta_z)$ and critic $V(s; \theta_z)$ heads.
4. **Loss computation & optimization**: The agent's loss (*e.g.*, TD-error for DQN, clipped surrogate objective for PPO) is computed using the generated parameters θ_z. Gradients are backpropagated through both the primary loss and the hypernetwork h_ϕ to update the shared parameters ϕ.

VLM. We evaluate several strong vision–language policies as prompt-only baselines: **GPT-5** and **GPT-4o** [17] (closed-source, accessed through their official APIs), and two high-capacity open-source models, **Qwen3-VL-30B-A3B** and **Qwen2.5-VL-72B** [3]. All models are used in a purely zero-shot manner without any task-specific fine-tuning.

Following the interaction format defined in *videogamebench* [27], each query consists of a structured prompt with four components: (i) *Game overview* describing the environment type (NES/SNES/Genesis/HTML), the available control interface, and major causal rules (*e.g.*, hazards, damage, reward triggers) to facilitate understanding the target; (ii) *Human-annotated action space*, where we provide the discrete actions extracted from human gameplay or emulator documentation, normalized to a canonical textual form; (iii) *Task and goals*, summarizing human-labeled objectives (survival, avoiding collisions, collecting items, defeating enemies, reaching exits); (iv) *Step context*, including the current frame, a brief history of recent actions, and (when available) high-level semantics such as "the platform collapses after stepping on it" or "the projectile follows a parabolic trajectory".

This format allows each VLM to reason with explicit physics- and causality-related cues instead of relying solely on one-frame appearance.

Required output structure. Each model is instructed to always return three fields: `THOUGHT` (free-form situational analysis), `MEMORY` (persistent long-horizon notes), and `ACTION` (the chosen control from the provided action space). We parse only the `ACTION` field and execute the corresponding environment action verbatim. The remaining fields are logged for qualitative analysis and do not affect control.

Inference loop. At every environment step t, the current frame x_t, game description, and the last L steps of history are inserted into the template. The model generates autoregressively, and the final `ACTION: [XXX]` token is mapped directly to the environment's action interface. All baselines use identical prompting templates to ensure fairness across models.

Example Prompt for VLM Baselines

(1) System Prompt. You are an AI agent playing *BillAndTedsExcellentGameBoyAdventure* (Game Boy), a classic action game where you must stay alive, overcome enemies, and reach each stage's objective.

(2) Game Goal & Rules.
- Push through the current stage while keeping your character alive.
- Defeat or evade enemies and projectiles encountered on screen.
- Collect helpful items, weapons, or power-ups along the way.
- Use movement and abilities to traverse platforms and hazards.
- Meet the victory condition or defeat the boss to advance.

(3) Action Space.
Your action is defined by a 9-element binary list (1 = pressed, 0 = not pressed). Multiple buttons may be pressed simultaneously.

Index	Button	Meaning
0	B	Attack with weapons or sprint when held
1	–	Unused slot – keep at 0
2	SELECT	Open sub-menus or cycle through inventory/options
3	START	Pause the game or open the main menu
4	UP	Move up, climb, or aim upward
5	DOWN	Move down, crouch, or drop through platforms
6	LEFT	Move or face left
7	RIGHT	Move or face right
8	A	Jump or confirm actions

(4) Action Combination Examples.
- Move right: `[0,0,0,0,0,0,0,1,0]`
- Move left: `[0,0,0,0,0,0,1,0,0]`
- Jump in place: `[1,0,0,0,0,0,0,0,0]`
- Jump while moving right: `[1,0,0,0,0,0,0,1,0]`
- Trigger a special ability: `[0,0,0,0,0,0,0,0,1]`
- Climb or enter upward path: `[0,0,0,0,1,0,0,0,0]`

(5) Output Format.
You *must* respond with **only** one valid JSON object in the exact format below. Do not include any other text, explanations, or markdown formatting. "thought": "reasoning about the current game state, strategy, and why you choose this action.", "action": "press_key", "action_input": [1,0,0,0,0,0,0,0,0], "memory": "note about your current status."

(6) Critical Directives.
- **Fixed Length**: The array length must be exactly 9.
- **Binary Elements**: Elements must be either 0 or 1.
- **Concurrency**: Multiple `1`s are allowed.
- **Think Then Act**: Analyze internally, and output the JSON format above.

(7) User Prompt.
Analyze the current gameplay frame and output the JSON format above.

World model. We implement a latent-level video predictor based on V-JEPA2 / Genie-style architectures. Given (z_t, a_t), the model predicts $(\hat{z}_{t+1}, \hat{r}_t)$ using masked temporal transformers. It performs 5–10 step rollouts for imagined optimization.

Genie. Our Genie implementation includes the following key enhancements over the baseline GENIEREDUX:

- **Increased visual fidelity:** The original model operated on low-resolution (64×64) inputs and reconstructions, which we identified as a source of significant information loss due to aggressive downsampling. To mitigate this, we increased the input and output spatial resolution to 224×224, thereby preserving finer-grained visual details crucial for complex environments.
- **Multi-action embedding:** The baseline GENIEREDUX was limited to a small, fixed set of five semantically-aligned, one-hot encoded actions. To support a broader and more flexible action space, we designed a novel action processing module. This module takes a multi-discrete action vector (*e.g.*, $[0, 1, 0, 0, 0, 1, 0, 1, 0]$), identifies the indices of activated actions, performs embedding lookups for each active index, and aggregates the resulting embeddings via mean pooling to produce a unified action representation for the world model.
- **Semantic action space alignment:** We extended the action space into a larger, semantically structured space. Furthermore, we performed cross-game semantic alignment on this space, enabling the model to interpret and utilize actions consistently across different tasks and environments.

DreamerV3. We adopt the official DreamerV3 architecture [12], utilizing a shared Recurrent State Space Model (RSSM) [8] backbone with categorical latent states to capture universal physical dynamics. To handle the distinct objectives and reward scales across 1,000+ games (*e.g.*, sparse survival signals v.s. dense score accumulation), we employ a *multi-head* architecture: while the visual encoder and recurrent dynamics model are shared across all titles, we instantiate separate Actor and Critic heads for each game. We rely on symlog predictions to normalize reward magnitudes and train the shared backbone jointly on all environments.

V-JEPA 2. Following the V-JEPA 2-AC formulation [2], we employ a non-generative world model that predicts in the representation space rather than reconstructing pixels. The model is trained in two phases: First, a ViT-L video encoder is pre-trained on our dataset of game frames using the self-supervised masked modeling objective, learning to predict latent representations of masked regions. Second, we freeze the encoder and post-train a latent Action-Conditioned (AC) predictor on offline trajectories collected by a random policy. This predictor learns to autoregressively forecast the latent representations of future frames conditioned on the context and action sequence, capturing physical dynamics in the abstract feature space.

IL. We include Behavior Cloning (BC) on human frame–action pairs and SFT on VLM, where VLM predicts latent actions from visual tokens.

ACT-BC. We implement a standard behavior-cloning visuomotor transformer following the ACT paradigm. Each training sample consists of $(x_{t-H:t}, a_t)$, where $x_{t-H:t}$ are the last $H{=}4$ RGB frames and a_t is either the discrete action vector or PhysCode latent action. Frames are resized to $128{\times}128$ and encoded by a lightweight 3-layer ConvNet, whose output tokens are fed into a 12-layer transformer. The model predicts a_t with a cross-entropy loss and is trained jointly on all games without any domain-specific parameters, following the "single policy for all games" setting used in ACT.

Qwen3-VL-8B-BC. We also evaluate a large-model BC baseline using Qwen3-VL-8B. At each timestep, we construct a simple prompt containing: (1) the current frame x_t (encoded by the model's native vision encoder), and (2) a fixed instruction template, like prompts in the VLM part. To avoid generating free-form language, we disable chain-of-thought decoding and restrict the output vocabulary to the action only. BC supervision is applied using next-token prediction: the ground-truth action is appended after a `<action>` tag, and the model is trained to reproduce it exactly. We do not use memory tokens, history text, or reasoning steps—Qwen3-VL operates purely as a frame-to-action predictor under teacher forcing.

Training. Both ACT-BC and Qwen3-VL-8B are trained on the same human trajectories used throughout the paper. We use AdamW with a learning rate of $2{\times}10^{-4}$ for ACT and $1{\times}10^{-5}$ for Qwen3-VL fine-tuning, batch size 64, and train for 300k steps. As shown in Sec. 3.6, low-quality BC supervision may override pretrained priors and degrade long-horizon performance, making these IL baselines strong short-horizon solvers but weak in cross-domain reasoning. Table 1 reports the absolute performance across selected games, where the significant variance in raw scores validates the necessity of our HNS normalization and demonstrates IPR's superior performance over GPT-5.

Table 1. Absolute performance on a representative subset of diverse games.

Game	Human	Random	PPO	DQN	DreamerV3	V-JEPA2	Genie	ACT-BC	Qwen3-VL-BC	Qwen3-VL-30B-A3B	GPT-4o	GPT-5	IPR
Shooter													
AfterBurn. (Gen)	365710	8680	403000	93000	0	0	0	0	0	367500	**492760**	136890	**238000**
SuperThun. (Gen)	12450	58	8132	108	**12826**	3567	3672	3567	3567	105	110	105	**43020**
GunNac (Nes)	23000	2860	21600	600	1360	200	400	200	**24200**	900	5100	37100	**38000**
SpaceMeg. (Snes)	0110	10	2680	**15120**	0	0	0	0	0	1210	13710	31340	**32110**
Sagaia (Gen)	57500	17100	800	400	0	0	**25300**	0	800	23300	14500	17500	**22400**
UzuKeobuk. (Gen)	3600	800	4800	**6980**	800	800	800	5600	800	5200	3600	800	**11250**
AirDiver (Gen)	2000	200	**11000**	0	0	0	0	0	0	100	500	1000	**10000**
SpaceHarr. (Sms)	10552	927	5867	5867	427	427	587	6927	**7585**	2436	1367	3479	**5500**
UchuuNoK. (Snes)	5000	2000	**12300**	1700	0	0	0	0	0	5000	1900	5000	**5500**
T2TheArca. (Sms)	6000	750	**5125**	0	0	0	0	0	0	1400	1000	**5150**	5075
DynamiteD. (Sms)	7200	0	**1800**	0	0	0	**1800**	0	0	700	1000	1000	**5000**
SpaceHarr. (Nes)	5000	37	**4817**	1136	0	0	0	0	0	4657	1814	240	**4650**
SpartanX2 (Nes)	8000	200	**7700**	5200	0	0	0	0	0	2100	3700	**3900**	3400
GalagaDem. (Nes)	9000	1790	9180	**9300**	0	0	0	0	0	1930	1840	3080	**3200**
ChoujikuuYou. ()	250	90	150	10	300	350	200	100	**440**	60	290	300	**2200**
AstroWarr. (Sms)	6900	2000	1100	300	100	100	100	**5000**	200	800	700	700	**2100**
Gynoug (Gen)	8000	1000	100	**2800**	0	0	200	800	1200	400	400	1600	**1900**
DefenderI. (Nes)	600	600	**1400**	800	0	0	600	0	0	0	0	1200	**1800**
BomberRai. (Sms)	4300	500	**2900**	500	0	0	0	400	100	200	400	200	**1200**
TerraCres. (Nes)	4700	10	300	0	0	0	300	300	**400**	70	50	1000	**1200**
GradiusI. (Snes)	3800	0	**1000**	0	0	0	0	0	0	400	200	900	**1200**
GrindStor. (Gen)	17850	0	**18520**	6600	0	0	0	0	0	6430	560	200	**630**
TransBot (Sms)	70	492	**360**	120	120	128	136	146	234	116	124	112	**600**
ThunderFo. (Gen)	170	80	25	20	5	5	60	0	120	210	**290**	275	**550**
Axelay (Snes)	367	127	**476**	335	206	288	416	302	331	289	338	394	**490**
Caliber50 (Gen)	500	100	**2600**	0	0	0	0	0	0	300	400	400	**450**
TwinCobra. (Gen)	970	187	145	66	20	20	**271**	161	200	119	203	152	**430**
DariusFo. (Snes)	50	300	270	**340**	0	0	140	290	280	240	100	240	**400**
Gaiares (Gen)	930	100	100	0	0	100	100	0	120	180	**200**	380	**400**
DynamiteD. (Gen)	1930	10	0	0	0	0	0	450	**1080**	80	60	300	**370**
AeroFigh. (Snes)	31	10	7	**8**	0	0	3	0	0	0	0	200	**300**
ChouFuyuu. (Nes)	800	100	**400**	0	100	0	200	0	0	100	0	**300**	**300**
VRTrooper. (Gen)	300	50	800	0	0	0	900	**1050**	0	0	250	270	**300**
SuperRTy. (Snes)	124	0	18	0	0	0	**26**	0	6	0	0	10	**280**
PowerStri. (Sms)	250	71	204	102	90	30	34	30	**221**	98	157	167	**247**
Satellite. (Sms)	10	60	120	0	100	0	0	0	**200**	120	120	170	**240**
ThunderFo. (Gen)	120	10	0	0	0	0	0	10	100	0	**110**	100	**200**
ThunderFo. (Gen)	166	200	20	2	0	0	0	0	0	**182**	84	180	**200**
PowerStri. (Sms)	152	52	109	23	168	115	72	115	154	66	**173**	153	**191**
Sagaia (Sms)	261	10	144	**156**	0	0	0	32	39	14	90	98	**148**
StarForce (Nes)	85	12	11	8	31	5	5	**42**	41	11	24	37	**130**
Exerion (Nes)	51	1	22	4	0	0	**28**	0	11	2	3	8	**100**
TwinCobra (Nes)	769	56	**100**	20	40	10	60	60	40	0	0	50	**100**
SonicWin. (Snes)	38	19	**22**	12	2	4	0	4	0	6	4	8	**60**
ThunderS. (Snes)	84	0	**20**	0	0	0	0	8	2	0	0	12	**52**
AbadoxThe. (Nes)	530	10	**450**	0	0	0	0	0	140	0	0	20	**50**
GradiusII (Nes)	27	32	6	0	**24**	20	0	0	10	21	21	**46**	**46**
Airstrike. (Gen)	80	20	20	40	0	0	0	0	**80**	20	20	20	**40**
DariusTw. (Snes)	26	35	**50**	3	8	6	9	8	13	14	11	20	**38**
Gradius (Nes)	1180	1	10	0	0	0	0	200	**380**	4	5	10	**30**
ThunderBl. (Sms)	300	0	**50**	0	0	0	0	0	20	0	0	**10**	**10**
StarSoldi. (Nes)	52	2	2	**13**	0	0	0	7	2	0	0	2	**8**
CosmicEps. (Nes)	240	0	**260**	0	0	0	100	0	0	0	0	**10**	5
Magmax (Nes)	23	1	5	1	0	0	0	0	**6**	2	1	**2**	1
Platformer													
Battletoa. (Gen)	16000	500	**5000**	0	0	0	0	0	0	**5000**	**5000**	**15500**	**15500**
ZoolNinja. (Gen)	15672	20	0	**2175**	0	0	0	0	0	722	322	**13247**	12522
SuperPitf. (Nes)	10000	600	**6500**	6000	4500	6000	6000	6000	4500	6000	6000	**6000**	**6000**
DragonsL. (Snes)	5000	1200	**31200**	6600	0	0	0	0	0	1800	2000	**5200**	**5200**
Battletoa. (Gen)	1000	100	2000	1000	0	0	0	0	0	**4000**	2000	2000	**4000**
ZoolNinj. (Snes)	3200	900	**9900**	3400	0	0	0	0	0	1600	3000	3100	**3300**
BombJack (GB)	7000	800	1200	0	200	400	0	0	1600	1400	**3400**	**2000**	**2000**

Continued on next page

Table 1 — *continued*

Game	Human	Random	PPO	DQN	DreamerV3	V-JEPA2	Genie	ACT-BC	Qwen3-VL-BC	Qwen3-VL-30B-A3B	GPT-4o	GPT-5	IPR
Joust (Nes)	2750	500	**6750**	3750	0	0	0	0	0	2750	2500	**2000**	**2000**
Adventur. (Snes)	2250	150	1850	1250	0	0	0	0	0	950	**1900**	1650	**1950**
3NinjasKi. (Gen)	3630	50	50	0	0	0	0	0	**110**	60	100	110	**1800**
DennisTh. (Snes)	1500	0	**2500**	0	0	0	0	0	0	500	500	1000	**1500**
KidKlown. (Snes)	1200	20	**16610**	9450	0	0	0	0	0	1120	20	**1210**	**1210**
AttackOfTh. (GB)	100	0	**100**	0	0	0	0	0	0	25	20	50	**600**
BoobyBoys (GB)	25	12	**13**	0	0	0	0	5	10	5	5	9	**400**
MoeroTwin. (Nes)	2000	100	**1400**	200	0	0	0	500	200	0	0	200	**400**
BlockKuzus. (GB)	650	150	150	100	100	100	100	100	250	100	**500**	**300**	**300**
Sansuu5An. (Nes)	440	20	110	0	0	0	**160**	0	100	100	100	180	**200**
HomeAlone. (Nes)	370	20	100	40	40	40	40	**120**	**120**	**120**	80	80	**160**
TwinBee (Nes)	60	2	4	0	0	0	0	**5**	3	0	0	**100**	**100**
MappyLand (Nes)	2	20	11	0	0	0	0	**506**	10	9	6	4	**100**
CircusCha. (Nes)	20	0	0	0	0	0	0	0	0	**100**	0	20	**100**
ComicalMa. (Sms)	7000	20	**100**	0	0	0	0	0	0	0	**100**	**100**	**100**
Frogger (Gen)	180	10	**40**	30	0	0	0	0	0	30	20	**70**	60
MarioBros (Nes)	30	1	**50**	0	0	0	0	40	0	10	5	10	**40**
SuperTro. (Snes)	30012	8	**30130**	26132	0	0	0	0	0	45	8	10	**31**
Tick (Gen)	180	0	0	0	0	0	0	0	0	0	**80**	**30**	**30**
TwinBee3P. (Nes)	105	10	**25**	5	20	5	5	20	10	10	4	**20**	15
AmazingPen. (GB)	99	3	20	**25**	0	0	0	0	1	10	10	**20**	**20**
ChubbyChe. (Nes)	53	3	0	0	0	0	0	0	**20**	0	0	**13**	10
DarkCastl. (Gen)	25	3	0	0	0	0	**12**	0	0	0	2	5	**10**
CircusCap. (Nes)	30	0	0	**10**	**10**	2	0	0	**10**	0	0	**5**	**5**
DonDokoDo. (Nes)	38	0	5	0	0	0	0	0	**13**	0	0	**2**	1
Fighting													
StreetsOf. (Gen)	1000	980	**2670**	1400	0	0	0	0	0	820	260	1430	**6580**
StreetSma. (Gen)	4000	800	**11200**	500	0	0	0	0	0	1000	2500	3500	**3900**
Cliffhang. (Gen)	2500	500	**6500**	1500	0	0	0	0	0	1500	1000	**3000**	**3000**
CaptainAm. (Gen)	3000	90	**6579**	812	0	0	0	0	0	2686	1512	**2736**	2424
IronSword. (Nes)	2030	648	1892	**2488**	0	0	0	0	0	1644	1300	2248	**2396**
StreetsOf. (Gen)	1010	20	810	590	0	0	0	0	0	**1560**	1230	1520	**1660**
WorldHero. (Gen)	3000	100	3000	0	0	0	0	0	0	**4100**	500	1000	**1600**
FistOfThe. (Nes)	400	0	**13900**	1100	0	0	0	0	0	40	50	100	**1200**
ArtOfFigh. (Gen)	8000	0	**7900**	1600	0	0	0	0	0	1600	100	500	**700**
CaptainPl. (Gen)	900	50	700	0	0	0	50	0	**950**	250	200	250	**430**
ToxicCrus. (Gen)	464	35	251	197	0	0	0	**920**	36	208	26	18	**318**
CaptainPl. (Nes)	1200	0	300	0	0	0	0	**400**	**400**	0	0	100	**200**
BlaZeonT. (Snes)	800	0	**1300**	200	0	0	0	0	0	40	40	100	**110**
Action													
SuperSmas. (Gen)	70040	15070	**60080**	46030	0	0	0	0	0	33570	32540	60060	**61510**
Cameltry (Snes)	45310	630	35860	**43130**	0	0	0	0	0	35720	23310	21750	**29560**
SuperSma. (Snes)	8000	105	5954	**6005**	0	0	0	0	0	5206	5357	4604	**15520**
Growl (Gen)	10050	500	**25000**	4000	0	0	0	0	0	17500	4000	**14500**	14000
AcceleBr. (Snes)	12000	200	**18800**	17800	0	0	0	0	0	13500	11500	**12500**	12000
NoahsArk (Nes)	113250	400	**34500**	1800	0	0	0	0	0	11250	0	**11750**	11350
FlyingDra. (Nes)	11000	4500	**30000**	23500	0	0	0	0	0	11000	9000	10500	**11000**
MagicSwo. (Snes)	15200	720	1860	**3130**	0	0	0	0	2920	50	1060	6710	**10960**
AdvancedB. (Gen)	6000	50	**5600**	3200	0	0	0	0	0	1200	900	6900	**7100**
MaouRenji. (Gen)	9020	300	**9350**	0	0	0	0	0	0	3900	3150	5300	**5750**
SteelEmpi. (Gen)	7000	40	**9100**	8300	0	0	0	0	0	6900	210	500	**5600**
Predator2 (Sms)	5990	910	5990	**8815**	0	0	0	0	0	5900	3575	1040	**5420**
ExMutants (Gen)	2310	0	**10000**	0	0	0	0	0	0	100	100	100	**5000**
BubbleBob. (Nes)	4000	0	**3300**	0	0	0	0	0	0	2000	0	**5100**	5000
RangerX (Gen)	5000	0	**13600**	0	0	0	0	0	0	2000	3900	3900	**4300**
RainbowIs. (Sms)	5000	0	2500	2000	0	0	0	0	0	**4200**	1000	3800	**4000**
Predator2 (Gen)	1000	400	**3600**	1400	0	0	0	0	0	900	0	1900	**4000**
BodyCount (Gen)	3800	730	**3490**	2098	0	0	0	0	0	0	3200	2800	**3850**
Trog (Nes)	3000	1260	2245	570	0	0	0	0	0	**2920**	2045	2520	**3535**
RenAndSti. (Gen)	3000	1750	**3500**	2250	0	0	0	0	0	750	750	3250	**3500**
KaiteTsu. (Snes)	45650	850	**10450**	4800	1000	500	2050	2800	3800	1400	1050	1050	**3200**
DragonThe. (Gen)	7010	200	**14600**	2400	0	0	0	0	0	2000	2500	2500	**3000**
AtomicRob. (Gen)	3000	500	**9700**	2000	0	0	0	0	0	1000	3000	**3000**	**3000**
Millipede (Nes)	4020	546	**9032**	2006	0	0	0	0	0	2359	1500	2000	**2922**

Continued on next page

Table 1 — *continued*

Game	Human	Random	PPO	DQN	DreamerV3	V-JEPA2	Genie	ACT-BC	Qwen3-VL-BC	Qwen3-VL-30B-A3B	GPT-4o	GPT-5	IPR
MidnightR. (Gen)	3000	800	**3000**	2800	0	0	0	0	0	600	1800	1600	**2600**
Universal. (Gen)	4100	50	**4751**	0	0	0	0	0	0	500	751	2403	**2500**
Xenon2Meg. (Gen)	2000	0	**4000**	1750	0	0	0	0	0	1600	0	**2720**	2500
Punisher (Nes)	6010	1410	**6310**	4710	0	0	0	0	0	3440	2230	2230	**2365**
FZSenkiAx. (Gen)	3000	90	**710**	230	0	0	0	0	0	20	10	20	**2300**
RainbowIs. (Nes)	4000	100	**2000**	**2000**	0	0	0	0	0	1500	1000	**2200**	**2200**
BattleZe. (Snes)	9000	200	**10100**	2100	0	0	0	0	0	2400	2100	800	**2100**
SuperFant. (Gen)	100	700	100	1000	0	1000	0	1000	1000	1300	**2000**	1600	**2000**
Wolfchild (Gen)	3900	200	**4600**	1500	100	50	0	0	0	700	600	800	**2000**
AtomicRun. (Gen)	6150	1100	400	450	0	0	0	1850	**4150**	600	1800	**1850**	**1850**
LastStarf. (Nes)	1000	50	**8940**	4890	0	0	0	0	0	620	100	1550	**1750**
SuperXevi. (Nes)	1090	20	**590**	10	130	100	60	240	170	80	90	1440	**1530**
SonSon (Nes)	2120	40	30	460	10	10	610	160	**760**	220	190	170	**1510**
Trojan (Nes)	11050	50	**800**	400	0	0	0	100	400	400	500	1000	**1500**
SolDeace (Gen)	1844	221	30	200	510	200	60	0	360	520	**730**	1100	**1450**
KungFuKid (Sms)	5000	200	**16600**	400	0	0	0	0	0	3200	400	1200	**1400**
Legendary. (Nes)	8100	100	**2700**	900	900	900	900	900	900	900	900	900	**1300**
Granada (Gen)	1000	0	**2590**	1600	0	0	0	0	0	1120	1100	1200	**1280**
Renegade (Sms)	5000	50	**3100**	400	0	0	0	0	0	1800	1000	**1350**	1200
AeroStar (GB)	1600	200	1800	**2300**	0	0	0	0	0	0	200	**1400**	1200
EarthDef. (Snes)	14700	200	**6600**	700	200	200	1400	400	1000	200	1200	200	**1200**
SameSameS. (Gen)	1661	348	90	**1145**	37	54	71	18	240	152	149	175	**1140**
ArrowFlas. (Gen)	696	84	136	12	0	12	156	24	**424**	24	24	24	**1000**
BubbleBob. (Sms)	1987	2	**345**	0	0	0	0	0	0	0	0	25	**1000**
MegaSWIV (Gen)	10000	10	0	350	0	0	0	1300	**3050**	200	300	500	**1000**
Renegade (Nes)	2000	50	**1800**	100	0	0	0	0	0	1350	100	300	**1000**
8Eyes (Nes)	2650	250	500	**950**	0	0	0	0	**950**	600	500	**1000**	950
Parodius (Snes)	7700	0	100	**600**	0	0	0	0	0	500	400	700	**900**
KingOfThe. (Gen)	400	400	**1400**	100	0	0	0	0	0	100	200	100	**800**
UNSquadr. (Snes)	1000	50	**900**	800	0	0	0	0	0	400	100	**800**	**800**
Phalanx (Snes)	320	30	152	350	5	5	58	**373**	66	5	5	10	**700**
YoukaiDou. (Nes)	2000	100	**1500**	1000	0	0	0	0	0	**1500**	500	**1000**	700
Elemental. (Gen)	2360	60	90	**180**	30	10	120	0	70	60	60	310	**660**
ShaqFu (Gen)	4960	10	200	**240**	0	0	0	0	0	**240**	100	400	**640**
Phelios (Gen)	2400	90	**20800**	0	0	0	0	0	0	5900	700	400	**600**
RType (Sms)	2000	0	800	1200	0	0	0	600	**1400**	0	600	**600**	**600**
TigerHeli (Nes)	160	32	15	20	47	20	10	0	**59**	42	37	32	**590**
SCATSpeci. (Nes)	289	92	10	5	34	10	0	116	84	123	**127**	103	**570**
Dangerous. (Gen)	210	60	235	245	0	0	0	0	60	**260**	195	265	**520**
LifeForce (Nes)	174	45	12	0	0	0	0	**60**	24	10	15	12	**510**
Viewpoint (Gen)	2150	200	450	**1500**	0	0	950	90	650	0	0	200	**450**
FantasyZo. (Nes)	18	68	**37**	2	6	2	1	2	1	3	6	18	**400**
TaskForce. (Gen)	3220	10	**2580**	80	0	0	80	80	80	2	80	100	**400**
ElevatorA. (Nes)	500	10	**300**	0	0	0	0	0	0	0	**300**	300	**400**
AlphaMiss. (Nes)	329	75	12	154	51	20	75	**283**	152	12	24	48	**360**
BWings (Nes)	5100	100	**700**	300	100	200	100	400	400	100	300	200	**300**
CrackDown (Gen)	100	0	**200**	**200**	0	0	0	0	0	**200**	0	**300**	**300**
CosmoGan. (Snes)	34290	50	1130	530	0	0	3910	60	**4480**	0	180	110	**230**
ZeroWing (Gen)	1114	20	20	30	10	10	10	**40**	10	30	10	10	**200**
Argus (Nes)	1470	60	140	30	80	0	0	0	**200**	30	110	130	**200**
CityConne. (Nes)	208	14	38	**179**	24	21	23	24	116	39	19	57	**200**
BackToThe. (Gen)	600	50	**200**	0	0	0	0	0	0	100	**200**	**200**	**200**
MendelPal. (Nes)	520	60	**530**	140	0	0	100	0	90	0	10	110	**200**
PitfallTh. (Gen)	200	25	**1625**	50	0	0	0	0	0	125	50	125	**175**
Daisenpuu. (Gen)	1336	28	61	**64**	0	0	60	16	0	30	30	45	**160**
Formation. (Nes)	11	1	**5**	3	3	1	0	0	1	1	0	1	**100**
InsectorX (Nes)	33	10	**31**	5	1	30	20	30	12	2	2	2	**100**
BlockKuz. (Snes)	50	50	50	**5800**	50	50	50	50	0	50	50	**100**	**100**
Seicross (Nes)	21	12	0	**26**	2	18	18	18	21	0	3	3	**90**
BattleUnit. (GB)	44	38	**42**	22	8	12	10	12	22	10	22	16	**60**
SkyKid (Nes)	101	3	**73**	8	3	5	6	6	32	9	6	6	**40**
DevilishT. (Gen)	140	20	40	40	40	40	50	50	**70**	40	40	**60**	40
Hellfire (Gen)	460	20	**75**	15	30	15	30	30	60	30	30	**30**	**30**
RamboFirs. (Sms)	130	0	**70**	20	0	0	0	0	10	15	20	25	**30**

Continued on next page

Table 1 — *continued*

Game	Human	Random	PPO	DQN	DreamerV3	V-JEPA2	Genie	ACT-BC	Qwen3-VL-BC	Qwen3-VL-30B-A3B	GPT-4o	GPT-5	IPR
1943 (Nes)	5	10	**55**	9	6	6	7	6	7	7	5	7	**20**
IndianaJo. (Gen)	495	0	0	0	0	0	15	0	**215**	10	15	10	**15**
Tetris (GB)	20000195	8	0	0	14	1	**15**	0	2	4	0	3	**14**
Stinger (Nes)	82	2	**20**	2	0	0	0	0	17	0	5	**12**	10
Californi. (Gen)	4250	0	10	**180**	10	10	10	10	**180**	10	10	**10**	**10**
Dimensio. (Snes)	25	0	6	2	6	4	**12**	10	**12**	7	11	**10**	9
Adventure. (Nes)	2200	0	0	**10**	0	0	0	0	**10**	0	0	**10**	**10**
TetrisAt. (Snes)	535	0	**68**	0	6	0	0	0	3	0	0	**10**	**10**
FinalBubb. (Sms)	2782	0	**105**	0	4	0	0	0	0	0	0	**15**	10
Spriggan. (Snes)	57	5	9	4	0	0	0	0	**26**	4	3	**8**	4
DigDugIIT. (Nes)	10	0	0	0	0	0	0	0	0	**1**	**1**	**4**	1
SpankysQ. (Snes)	7	1	1	0	0	0	0	0	**2**	0	0	**2**	1
Parodius (Nes)	49	0	**15**	4	0	0	0	0	6	0	1	**2**	1

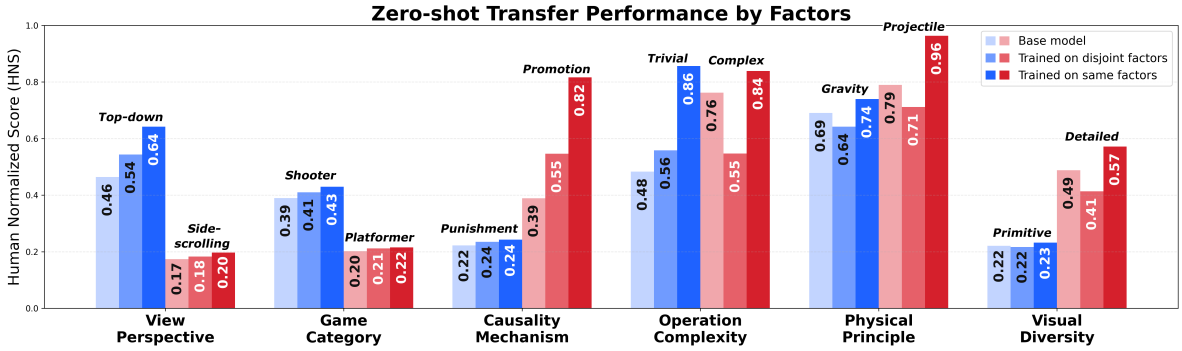

Figure 6. **Comparative analysis of factor-wise zero-shot transfer performance.** Disjoint training (lighter bars) facilitates generalization on structural factors (e.g., View Perspective) via disentanglement, while it incurs negative transfer on dynamic factors (e.g., Physical Principle) due to cognitive interference.

3.5. Exp 3: Zero-shot Transfer to Unseen Games

We study Game-to-Unseen (G2U) transfer by splitting the full game pool into a training pool and a held-out target set $\mathcal{T}_U$. The games in $\mathcal{T}_U$ (20 titles by default) are excluded from *all* training stages, including PhysCode pretraining, world-model learning, and IPR optimization.

From the remaining games, we construct stratified subsets $\{S_N\}$ of increasing size N (from tens to several hundred games). Each S_N approximately matches the full pool in terms of annotated physical and causal mechanisms (*e.g.*, gravity-driven platformers, projectile shooters, frictional top-down motion, rigid-contact puzzles), control interfaces (NES/SNES/Genesis/SMS/HTML), and visual/genre styles. This stratification keeps domain bias roughly fixed so that variations in performance mainly reflect the amount and diversity of interactive experience.

For each subset size N, we train a full IPR pipeline under a fixed configuration. PhysCode is pretrained on S_N with the same VQ-VAE setup as in the main paper (codebook size K=512, identical encoder/decoder, flow dropout p=0.5) and never sees data from $\mathcal{T}_U$. The world model is then trained on trajectories from S_N with a fixed architecture and rollout horizon, without any game-specific tuning. Finally, we attach the learned PhysCode interface and world model to a Qwen3-VL-8B backbone and optimize IPR with GRPO on multi-step latent rollouts. Learning rate, batch size, rollout length, GRPO sampling temperature, and other optimization hyperparameters are kept identical across all N.

Zero-shot evaluation is performed *only* on the unseen titles in $\mathcal{T}_U$. For a given N, we freeze PhysCode, the world model, and the IPR policy, and directly deploy the agent to these games without fine-tuning, reward re-scaling, or game-specific calibration. We reuse the same PhysCode-to-environment router as in the seen games, and adopt a fixed decoding temperature and sampling scheme at inference. For each unseen game, we roll out full episodes up to the native time limit or termination, and compute the three metrics defined in the main paper: 🍖 (normalized survival time), 🧭 (normalized exploration coverage), and 👑 (normalized task reward). Reported scores are averaged over all episodes and all 50 games in $\mathcal{T}_U$.

Under this protocol, performance on all three objectives improves monotonically with N, with larger relative gains at small N and continued, though diminishing, improvements as more diverse games are added. This supports our claim that G2U behavior is driven by exposure to varied physical and causal environments rather than game-specific tuning.

Table 2. Latent-action v.s. pixel-based prediction.

Agent Type	L1 ↓	MSE ↓
Pixel-based	0.0259	0.5622
Latent-based	**0.0195**	**0.3821**

3.6. Exp 4: Prediction-Based Interactive Reasoner

We ablate four components on a shared Qwen3-VL-8B backbone with the PhysCode interface: world-model prediction, GRPO-based group-wise optimization, PPO-based optimization, and behavior cloning (BC) on 10k human frame–action pairs. We construct a training game set with 200 games, containing all kinds of games. And we then construct one validation game set with 20 never-trained games, balanced by difficulty and novelty. Adding WM prediction and GRPO on top of the pretrained VLM consistently improves survival, curiosity, and utility, indicating that imagination-guided updates strengthen long-horizon reasoning. In contrast, inserting a low-quality BC stage before RL hurts performance: the model overfits to suboptimal demonstrations, its original pretrained reasoning is partially overwritten, and even after GRPO or PPO on optimizations, it underperforms the no-BC variant.

4. Additional Ablation Study

4.1. Value Prediction in Latent v.s. Pixel Space

To justify our design choice of performing imagination and planning within a compact latent space, we investigate the fidelity of reward prediction when operating on learned representations versus raw sensory inputs. We frame this as a value estimation task using Temporal Difference (TD) learning on offline datasets. Specifically, we curate a dataset of trajectories generated by a random policy and train a value function to predict the expected return (TD target) from a given state-action pair.

We compare two distinct architectures: (1) Pixel-based Predictor, a convolutional network that takes the raw RGB frame x_t and action a_t as input to directly regress the value; and (2) Latent-based Predictor, a lightweight MLP that operates on the frozen visual embedding z_t (extracted via the V-JEPA ViT-L encoder used in our main pipeline) concatenated with a_t.

We evaluate both models on a held-out test set using Mean Squared Error (MSE) and L1 loss against the computed TD targets. As shown in Table 2, the latent-based model significantly outperforms the pixel-based baseline.

This substantial performance gap highlights the difficulty of extracting sparse reward signals directly from high-dimensional pixel space, which is often dominated by high-frequency noise, shifting textures, and task-irrelevant background details. The pixel-based model must simultaneously learn to parse visual geometry and estimate value, leading

Table 3. Ablation on PhysCode codebook size K. Code usage is the fraction of codes visited on a held-out split; action separation measures the alignment between codes and human actions (higher is better).

K	Code usage (%)	Action separation
32	3.1	0.006
64	3.1	0.006
128	1.6	0.006
256	**10.5**	**0.063**
512	0.4	0.011
1024	0.5	0.021

to slower convergence and overfitting to visual nuisance. In contrast, the V-JEPA latent space—pretrained to capture structural and dynamical consistency—effectively filters out these distractions. It provides a compact abstraction of the physical state, allowing the value head to focus entirely on causal associations between states and rewards. This result empirically validates our architecture: performing reasoning and imagination in a semantic latent space is not only computationally efficient but also yields more accurate physical and value predictions than operating in raw pixels.

4.2. PhysCode Codebook Size

We ablate the PhysCode codebook size with $K \in \{32, 64, 128, 256, 512, 1024\}$. Our goal is twofold: (i) the codebook should be *compact enough* so that codes are effectively used, rather than wasted on rare patterns, and (ii) it should be *expressive enough* to separate distinct control behaviors and their induced physics.

For each K, we pretrain PhysCode under the same protocol and evaluate two properties: (1) *code usage*, computed as $U = \frac{\#\text{main codes}}{K}$ on a held-out split, where *main codes* are those whose empirical usage exceeds 0.1% of all assignments, reflecting how effectively the codebook capacity is utilized; and (2) *action separation*, measuring the alignment between codes and human action labels (*e.g.*, normalized mutual information between a_t^{LAT} and discrete key configurations).

We observe a clear trade-off as K varies. Very small codebooks ($K \leq 128$) over-compress behavior: only a few codes are actually used, and they mix heterogeneous key patterns, leading to poor action separation. Extremely large codebooks ($K \geq 512$) suffer from the opposite issue: capacity is badly under-utilized (less than 1% of codes are active), and similar behaviors get fragmented across many rarely visited entries. $K = 256$ strikes a favorable balance, achieving the highest code usage and a substantially stronger alignment with human actions, which in turn yields the best downstream performance. We therefore adopt $K = 256$ as the default codebook size in all main experiments.

Table 4. Ablation on ViT-L encoder configurations for the world model.

Encoder Setup	Cosine ↑	MSE ↓	L1 ↓
Pretrained	0.8888	0.2223	0.3262
Fine-tuned	0.8924	0.2150	0.2705
Trained from Scratch	**0.9891**	**0.0216**	**0.0758**

4.3. Latent Encoder Pretraining

Our IPR's world model requires an effective visual encoder to ground its predictions in the visual dynamics of the environment. While V-JEPA 2 offers a powerful foundation pretrained on millions of hours of internet videos [2], our work operates in the visually distinct domain of games. This introduces a potential domain gap, where features learned from real-world videos may not be optimal for capturing the specific appearance of game environments.

We compare three ViT-L encoder configurations, all followed by a V-JEPA 2-AC-style predictor trained on our game data: (1) a frozen, off-the-shelf pretrained encoder, (2) the same pretrained encoder fine-tuned on our game trajectories, and (3) a ViT-L encoder trained from scratch using only our game dataset.

The results in Table 4 demonstrate a clear performance hierarchy. The frozen pretrained encoder performed the worst, confirming a significant domain gap. Fine-tuning offered a moderate improvement, but the best predictive accuracy was unequivocally achieved by the encoder trained from scratch on in-domain data. This indicates that for specialized visual domains such as retro games, a tailored feature extractor is more effective than adapting a general-purpose one, as it avoids the potentially confounding inductive biases from out-of-domain pretraining.

4.4. Transfer Factors across Categories

To further validate the robustness of our representations across diverse environments, we conduct a fine-grained factor-wise transfer analysis as illustrated in Fig. 6, which reveals two distinct generalization regimes contingent on the nature of the underlying factors. For structural attributes such as *view perspective*, *causality mechanism*, and *game category*, our results show that training on disjoint factor sets facilitates positive transfer, as varying these surface-level configurations forces the model to disentangle intrinsic task dynamics from visual layouts. Conversely, factors tied to latent mechanics, specifically *physical principles* and *operation complexity*, often exhibit negative transfer where strong priors learned from source environments conflict with target mechanics. This phenomenon mirrors human **cognitive interference**, suggesting that while structural diversity promotes invariant representations, mechanical discrepancies remain a primary bottleneck for zero-shot adaptation.

5. Case Study

5.1. Experience-based Methods

Experience-based agents such as ACT and Qwen-BC show clear strengths in learning from human demonstrations, but also exhibit systematic limitations. ACT (Figure 7) benefits strongly from expert trajectories: it can extract effective strategies for difficult segments and achieves high scores on tasks with stable, low-variance dynamics. However, its imitation-heavy nature makes it prone to inheriting suboptimal human behavior and to degrading under environmental noise. Similarly, Qwen-BC (Figure 8) excels at reproducing high-difficulty actions with high fidelity and maintains very stable action sequences, yet its generalization is weak. When facing novel situations or imbalanced action distributions, the policy often collapses into passive idling or repetitive single-action loops. These behaviors collectively show that experience-based policies are powerful in familiar, low-variance regimes but struggle to extrapolate, exposing the limits of purely demonstration-driven learning in interactive environments.

5.2. RL-based Methods

Reinforcement learning agents, such as PPO (Figure 9) and DQN (Figure 10), demonstrate distinct ability to master complex motor control and identify key environmental interactions. These models excel at discovering effective key combinations for simultaneous maneuvering and attacking, as well as exploiting environmental features like cover to advance game progression. However, their reliance on scalar reward signals leads to significant brittleness. As observed in the case studies, both PPO and DQN are prone to exploiting poorly shaped rewards, leading to degenerate strategies like "dying on purpose" or repetitive movement loops to farm points. Furthermore, as exploration rates decay, these agents frequently suffer from policy collapse, halting necessary exploration and failing repeatedly at identical game states due to a lack of semantic understanding.

5.3. Prediction-based Methods

World model approaches, represented by Dreamer (Figure 11), Genie (Figure 13), and V-JEPA (Figure 12), exhibit strong temporal stability and high action efficiency. These agents are characterized by risk-averse behaviors, prioritizing short-term safety and minimizing redundant inputs. V-JEPA, for instance, shows strategic capacity in utilizing terrain features for evasion. However, a critical limitation shared across these prediction-based methods is susceptibility to passivity. When value estimates become uncertain or immediate feedback is lacking, these models often collapse into inaction—idling or outputting zero vectors rather than initiating exploration. Additionally, they can develop biased policies that over-rely on simple heuristics, such as persistently moving in a single direction, failing to adapt when dynamic hazards require complex, non-linear responses.

5.4. Reasoning-based Methods

Large Vision-Language Models (VLMs), such as GPT-4o (Figure 14), GPT-5 (Figure 15), and Qwen3-VL-30B-A3B (Figure 16), introduce strong semantic reasoning capabilities in the control loop. These agents demonstrate proficiency in spatial navigation and target acquisition, successfully executing calculated jumps and neutralizing aerial threats through accurate planning. Despite these strategic strengths, they struggle with real-time situational awareness and reaction latency. The case studies reveal persistent weakness in handling fast-moving dynamic entities, particularly those approaching from behind or requiring rapid reflexes. This suggests that despite competent reasoning, a perception–action latency gap undermines their performance in high-speed adversarial settings.

5.5. Interactive Physical Reasoner

The Interactive Physical Reasoner (IPR) (Figure 17) agent distinguishes itself by predictive imagination. Unlike the purely reactive RL agents or the prediction-based models, the IPR agent can simulate the trajectories of falling hazards and incoming projectiles, allowing for precise evasion and dynamic maneuvering. However, this imaginative capability is computationally constrained. While effective in one-on-one interactions, the agent reveals vulnerabilities in high-density adversarial environments. When the visual scene becomes cluttered with multiple simultaneous threats, the agent's capacity to "imagine" can lead to failure. Although there are limitations, IPR can reach high performance through its imagination ability and interactive physical reasoning.

References

[1] Rayna Andreeva, Katharina Limbeck, Bastian Rieck, and Rik Sarkar. Metric space magnitude and generalisation in neural networks. In *Proceedings of 2nd Annual Workshop on Topology, Algebra, and Geometry in Machine Learning (TAG-ML)*, pages 242–253, 2023. 5

[2] Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, et al. V-jepa 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025. 1, 8, 11, 17

[3] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond. *arXiv preprint arXiv:2308.12966*, 2023. 9

[4] Philip J. Ball, Jakob Bauer, Frank Belletti, Bethanie Brownfield, Ariel Ephrat, Shlomi Fruchter, Agrim Gupta, Kris-

Figure 7. **ACT Case Study.** The figure highlights four representative behaviors of ACT: (1) Line 1 shows that ACT can solve difficult segments by leveraging human demonstrations and extracting effective strategies; (2) Line 2 illustrates that imitation enables high scores on tasks with stable, low-variance dynamics; (3) Line 3 reveals that ACT also absorbs human failure patterns, reproducing suboptimal attempted actions; and (4) Line 4 shows that its behavior is highly sensitive to environmental noise, often leading to unstable or inconsistent actions.

Figure 8. **Qwen-BC Case Study.** The figure illustrates four characteristic behaviors of the BC-trained Qwen agent: (1) Line 1 shows that the agent can faithfully reproduce high-difficulty actions; (2) Line 2 demonstrates its strong temporal stability and highly consistent action repetition; (3) Line 3 reveals its poor generalization to novel or perturbed situations; and (4) Line 4 shows its tendency to collapse into passive idling or a single repeated action when the action distribution is imbalanced.

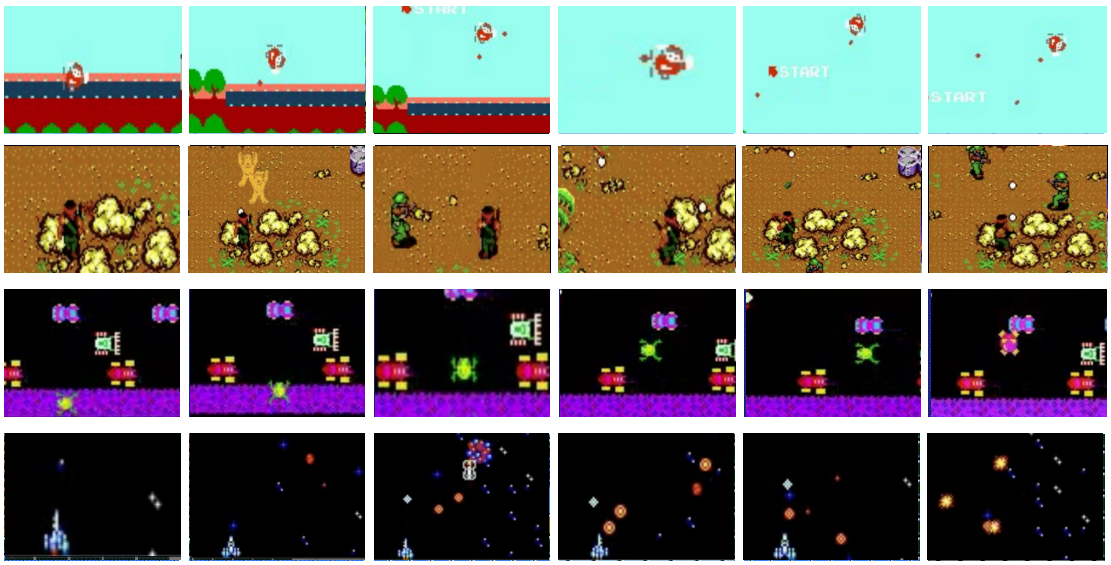

Figure 9. **PPO Case Study.** The figure presents four typical behaviors of the PPO agent: (1) Line 1 demonstrates that PPO can learn effective action sequences, enabling the agent to simultaneously shoot while dodging bullets through rolling maneuvers; (2) Line 2 illustrates its capacity to not only acquire efficient key-press strategies but also identify primary movement directions that drive game progression; (3) Line 3 reveals that due to poorly shaped rewards, PPO may exploit design flaws by repeatedly moving forward and backward to farm progression rewards in a degenerate manner; and (4) Line 4 shows that as the exploration rate decays in later training stages, the agent nearly halts exploration, leading to a performance plateau and frequent failures at identical game states.

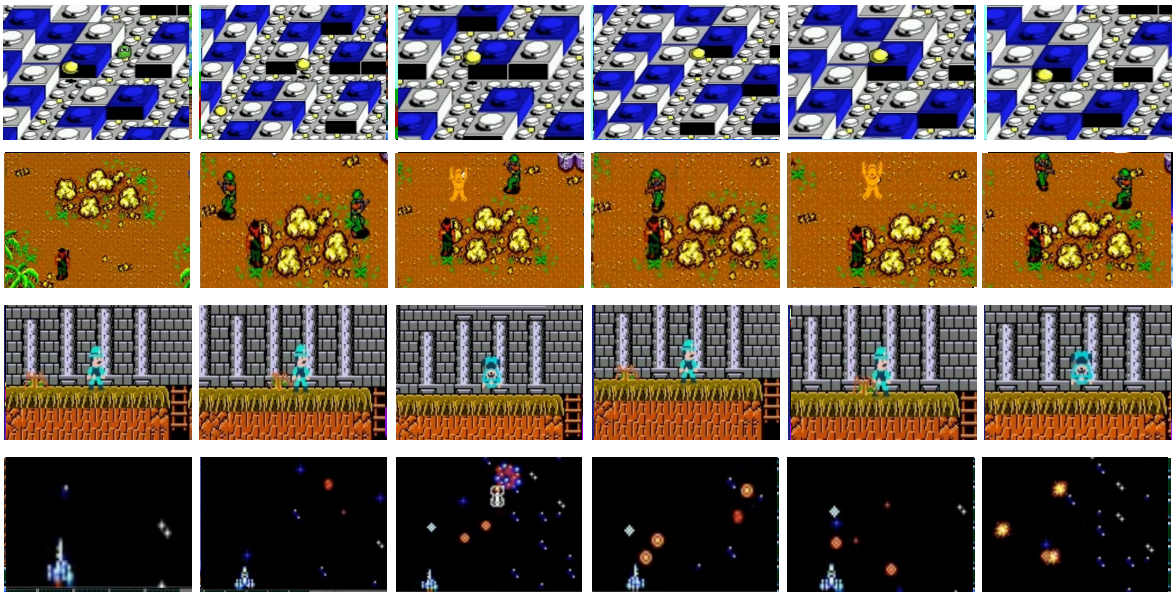

Figure 10. **DQN Case Study.** The figure presents four typical behaviors of the DQN agent: (1) Line 1 shows that it can correctly identify when specific actions should be executed; (2) Line 2 illustrates its ability to detect and exploit advantageous environmental features (*e.g.*, using rocks as cover); (3) Line 3 reveals that poorly shaped rewards can lead the agent to adopt degenerate strategies, such as repeatedly "dying on purpose" when death yields positive reward; and (4) Line 4 shows that the agent may fall into meaningless repetitive actions, such as continuous jumping without purpose.

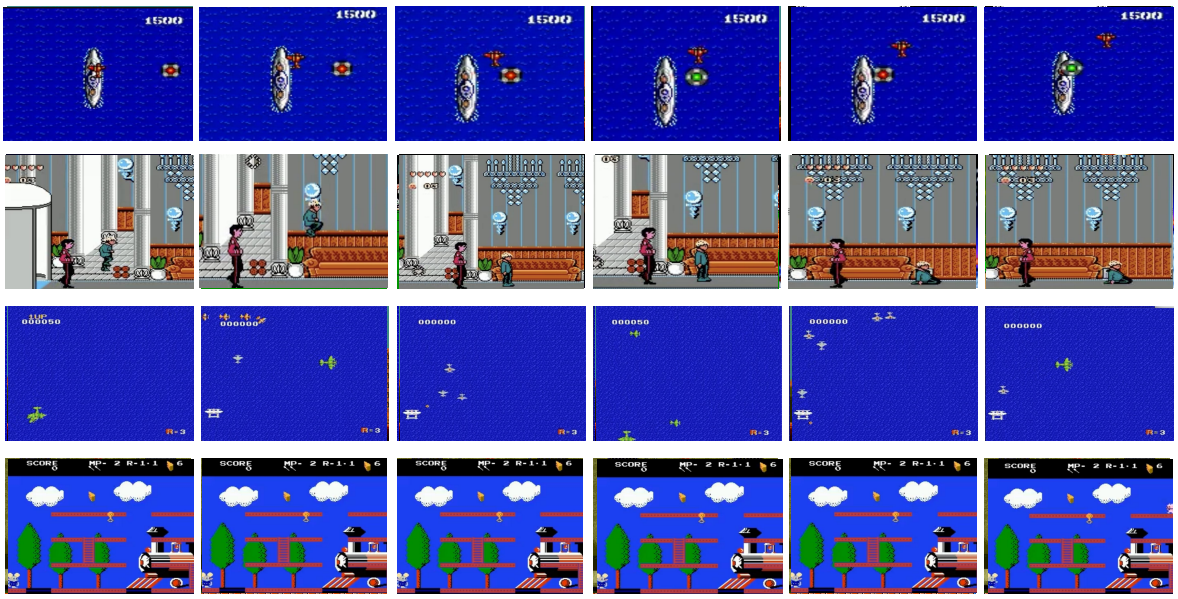

Figure 11. **DreamerV3 Case Study.** The figure illustrates four characteristic behaviors of the Dreamer agent: (1) Line 1 shows that Dreamer reliably exhibits risk-avoiding behavior and tends to choose actions that maximize short-term safety; (2) Line 2 demonstrates its strong temporal stability, often producing highly repetitive and consistent action sequences; (3) Line 3 reveals a biased policy that over-relies on a single heuristic—such as persistently moving left to evade enemies, and (4) Line 4 shows that the agent can easily collapse into inaction, remaining stationary when its value estimates become uncertain.

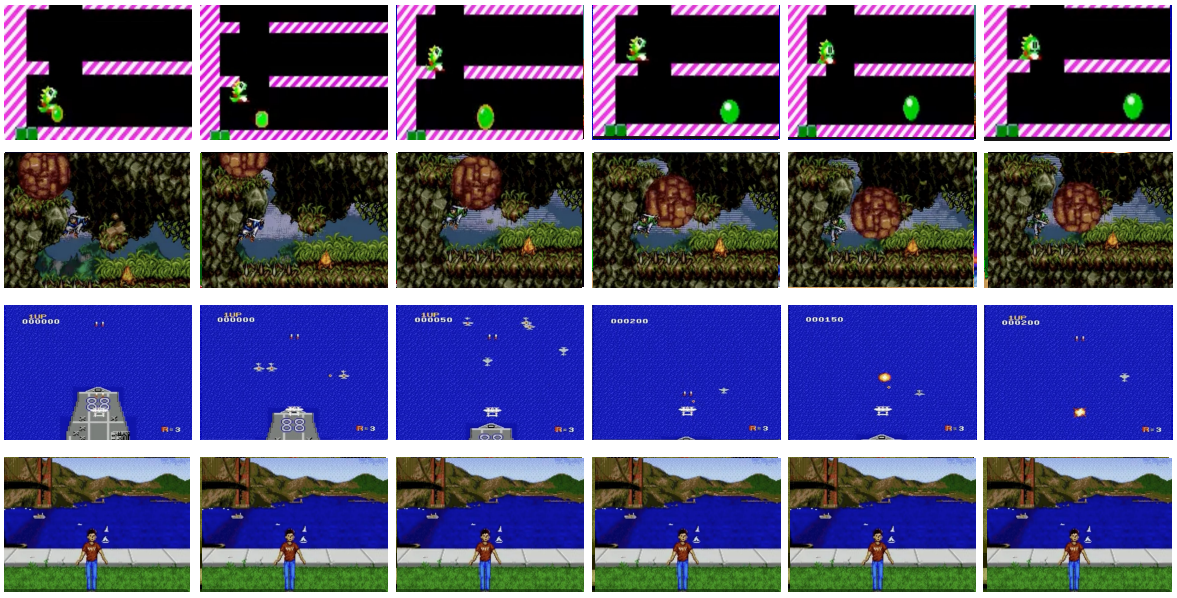

Figure 12. **V-JEPA2 Case Study.** The figure illustrates four representative behaviors of the V-JEPA agent: (1) Line 1 shows that the agent maintains high action efficiency with minimal redundancy, avoiding the ineffective key combinations often observed in other models; (2) Line 2 demonstrates its capacity for strategic environmental exploitation, such as utilizing terrain features (*e.g.*, rocks) to evade hazards; (3) Line 3 reveals its vulnerability to collapse into passive idling by outputting zero vectors; and (4) Line 4 indicates that this susceptibility to inaction persists when immediate feedback is lacking, resulting in a failure to initiate necessary exploration.

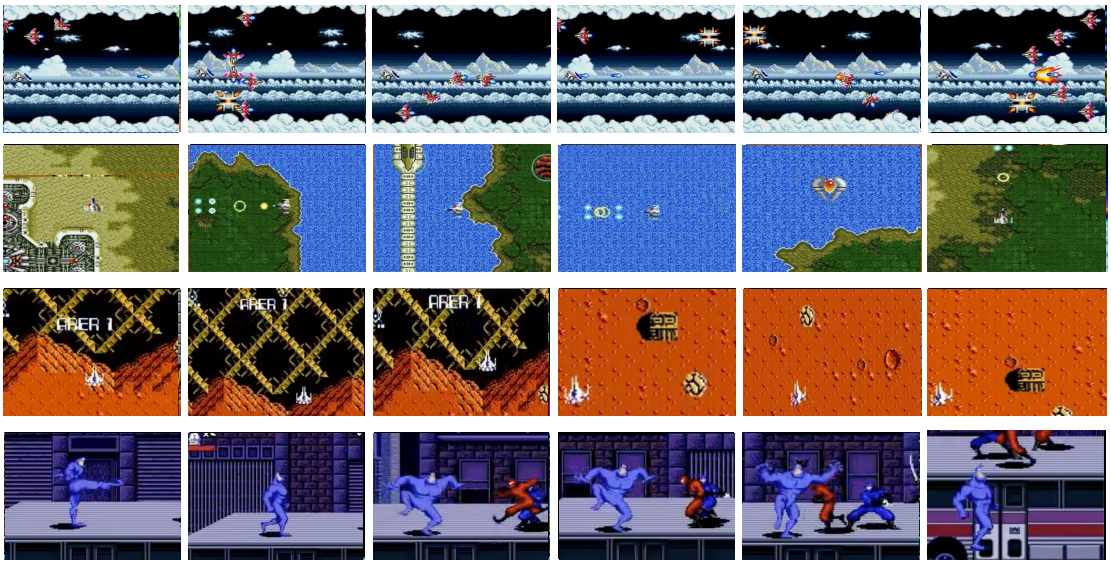

Figure 13. **Genie Case Study.** The figure presents four key capabilities and limitations of our Genie-based world model: (1) Line 1 demonstrates enhanced motion trajectory prediction, enabling the agent to execute preemptive evasion maneuvers; (2) Line 2 reveals the emergence of strategic path planning, where the agent learns systematic navigation paths beyond reactive bullet avoidance; (3) Line 3 illustrates a critical model limitation: in environments with poor bullet reconstruction fidelity, the agent fails to develop shooting behaviors and defaults to purely defensive evasion strategies; (4) Line 4 highlights spatial reasoning deficiencies, where prediction inaccuracies in distance estimation lead to occasional falls from platform edges.

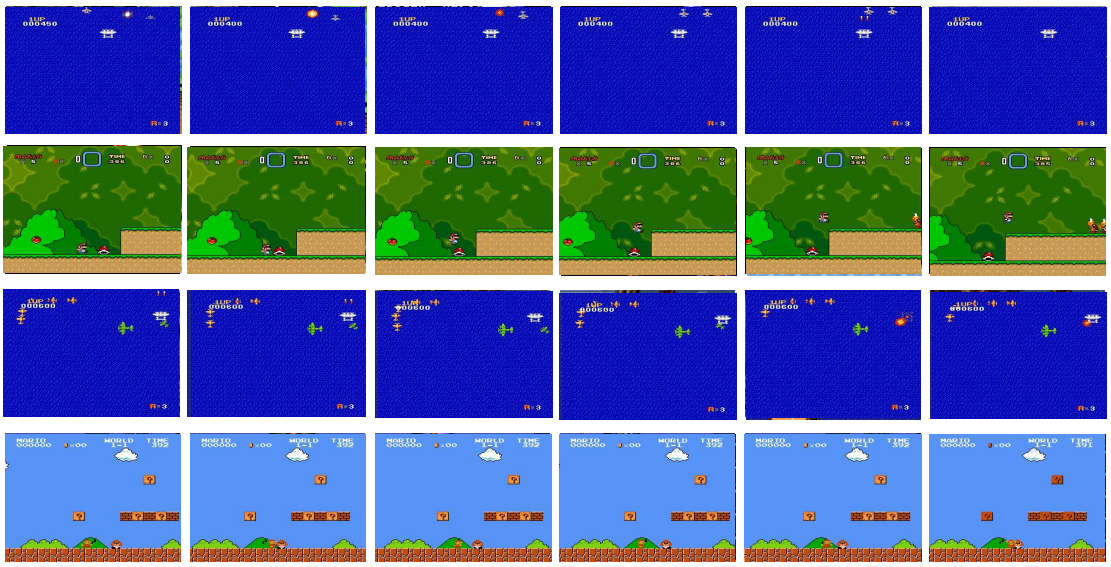

Figure 14. **GPT-4o Case Study.** The figure illustrates four char- acteristic behaviors of the GPT-4o agent. (1) Line 1 shows that the agent demonstrates effective target engagement and reaction speed, discharging projectiles to neutralize an aerial threat; (2) Line 2 highlights its proficiency in precise spatial navigation, executing a controlled jump to successfully land on the target platform; (3) Line 3 reveals a blind spot in situational awareness regarding rear-approaching entities, where the agent fails to evade the trailing aircraft, resulting in a fatal collision; and (4) Line 4 indicates a fundamental failure in basic obstacle avoidance, where the agent initiates a direct collision with a visible ground enemy rather than executing an evasive maneuver.

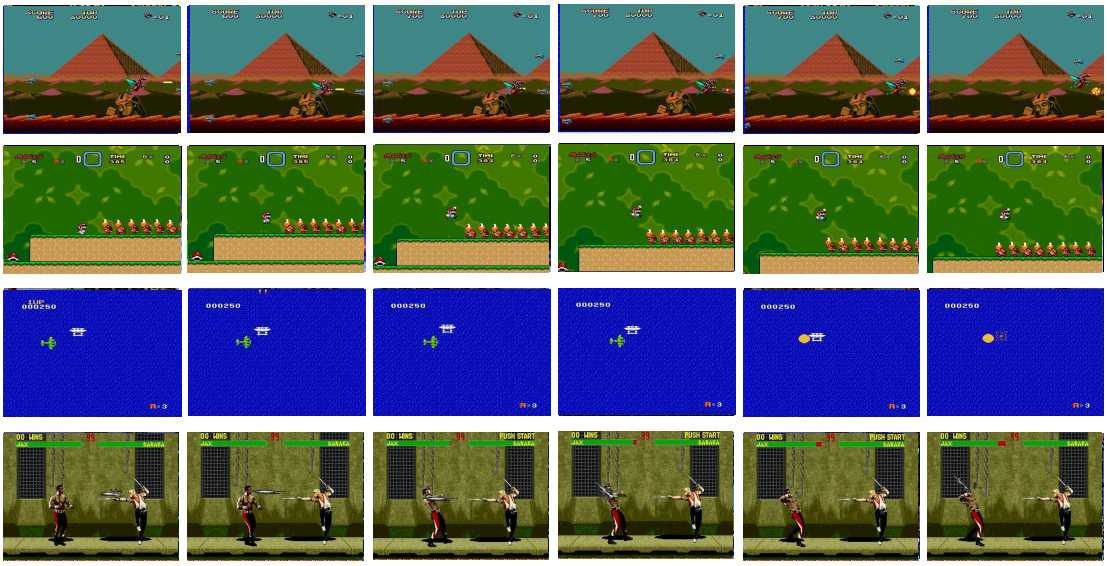

Figure 15. **GPT-5 Case Study.** The figure illustrates four characteristic behaviors of the GPT-5 agent. (1) Line 1 shows that the agent demonstrates accurate target acquisition and offensive capability, intercepting an aerial enemy to clear the path; (2) Line 2 highlights its proficiency in precision platforming and spatial navigation, executing a calculated jump to skip the enemies; (3) Line 3 reveals limitation in situational awareness regarding trailing threats, where the agent fails to evade a collision with an enemy approaching from the rear; and (4) Line 4 indicates susceptibility to delayed reaction times in combat scenarios, resulting in a failure to dodge or block an incoming projectile attack.

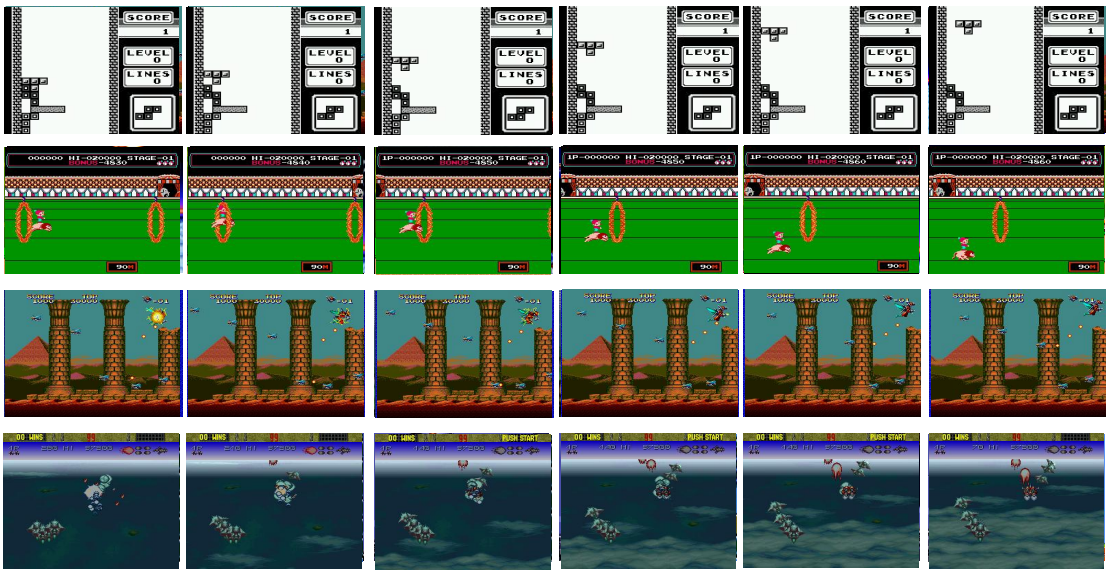

Figure 16. **Qwen3-VL-30B-A3B Case Study.** The figure illustrates four representative behaviors of the Qwen3-VL-30B-A3B agent: (1) Line 1 shows that the agent demonstrates spatial reasoning and planning, rotating and tucking the tetromino into a precise gap to maintain clean board; (2) Line 2 highlights its proficiency in high-frequency temporal control, executing a timed jump to pass the obstacle (the fire ring); (3) Line 3 reveals a limitation in processing dense visual clutter, where the agent fails to distinguish between terrain features and enemy projectiles, resulting in a fatal collision; and (4) Line 4 indicates a susceptibility to policy degradation in open environments, where the agent exhibits passive drifting behavior rather than actively targeting enemies or dodging incoming formations.

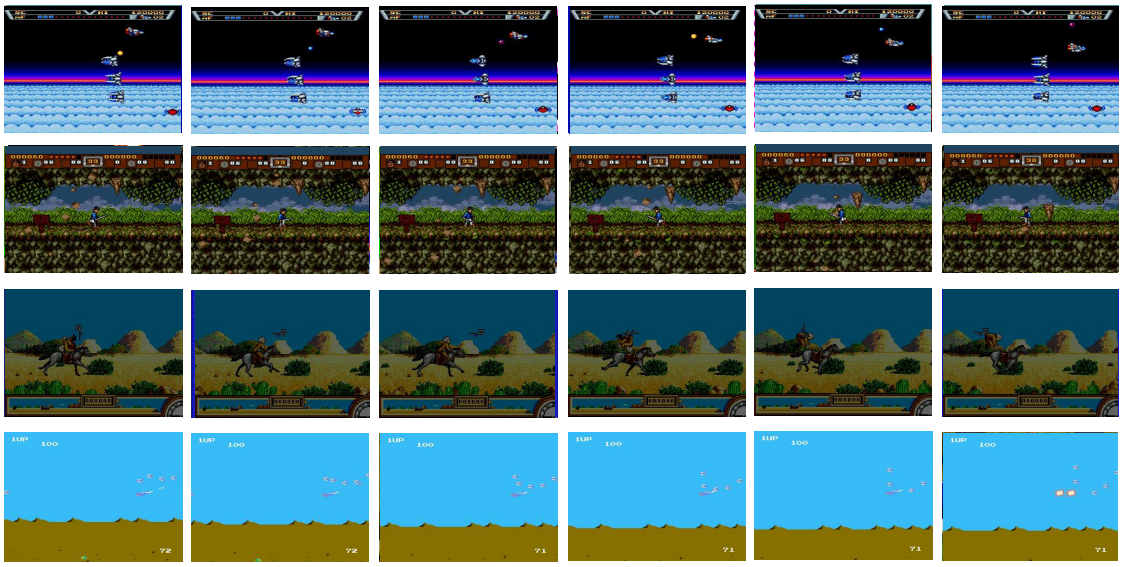

Figure 17. **IPR Case Study.** The figure illustrates four representative behaviors of the IPR agent: (1) Line 1 shows that the agent demonstrates precise reactive control, maneuvering to evade incoming projectiles; (2) Line 2 highlights its proficiency in dynamic environmental perception, allowing it to anticipate and dodge falling hazards (rocks); (3) Line 3 reveals vulnerability in rapid collision avoidance, where the agent fails to react to aerial threats (bats) (4) Line 4 indicates limitation in handling high-density adversarial environments, leading to an inability to evade when confronted by multiple simultaneous enemies.

tian Holsheimer, Aleksander Holynski, Jiri Hron, Christos Kaplanis, Marjorie Limont, Matt McGill, Yanko Oliveira, Jack Parker-Holder, Frank Perbet, Guy Scully, Jeremy Shar, Stephen Spencer, Omer Tov, Ruben Villegas, Emma Wang, Jessica Yung, Cip Baetu, Jordi Berbel, David Bridson, Jake Bruce, Gavin Buttimore, Sarah Chakera, Bilva Chandra, Paul Collins, Alex Cullum, Bogdan Damoc, Vibha Dasagi, Maxime Gazeau, Charles Gbadamosi, Woohyun Han, Ed Hirst, Ashyana Kachra, Lucie Kerley, Kristian Kjems, Eva Knoepfel, Vika Koriakin, Jessica Lo, Cong Lu, Zeb Mehring, Alex Moufarek, Henna Nandwani, Valeria Oliveira, Fabio Pardo, Jane Park, Andrew Pierson, Ben Poole, Helen Ran, Tim Salimans, Manuel Sanchez, Igor Saprykin, Amy Shen, Sailesh Sidhwani, Duncan Smith, Joe Stanton, Hamish Tomlinson, Dimple Vijaykumar, Luyu Wang, Piers Wingfield, Nat Wong, Keyang Xu, Christopher Yew, Nick Young, Vadim Zubov, Douglas Eck, Dumitru Erhan, Koray Kavukcuoglu, Demis Hassabis, Zoubin Gharamani, Raia Hadsell, Aäron van den Oord, Inbar Mosseri, Adrian Bolton, Satinder Singh, and Tim Rocktäschel. Genie 3: A new frontier for world models. 2025. 1

[5] Adrien Bardes, Quentin Garrido, Jean Ponce, Xinlei Chen, Michael Rabbat, Yann LeCun, Mahmoud Assran, and Nicolas Ballas. Revisiting feature prediction for learning visual representations from video, 2024. 1

[6] Jake Bruce, Michael D Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, et al. Genie: Generative interactive environments. In *Forty-first International Conference on Machine Learning*, 2024. 1

[7] Peng Chen, Pi Bu, Yingyao Wang, Xinyi Wang, Ziming Wang, Jie Guo, Yingxiu Zhao, Qi Zhu, Jun Song, Siran Yang, Jiamang Wang, and Bo Zheng. Combatvla: An efficient vision-language-action model for combat tasks in 3d action role-playing games, 2025. 1

[8] Andreas Doerr, Christian Daniel, Martin Schiegg, Nguyen-Tuong Duy, Stefan Schaal, Marc Toussaint, and Trimpe Sebastian. Probabilistic recurrent state-space models. In *International conference on machine learning*, pages 1280–1289. PMLR, 2018. 11

[9] David Ha and Jürgen Schmidhuber. World models. 2018. 1

[10] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination, 2020.

[11] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models, 2022.

[12] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, 2024. 11

[13] Danijar Hafner, Wilson Yan, and Timothy Lillicrap. Training agents inside of scalable world models, 2025. 1

[14] Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, Danny Driess, Michael Equi, Adnan Esmail, Yunhao Fang, Chelsea Finn, Catherine Glossop, Thomas Godden, Ivan Goryachev,

Lachy Groom, Hunter Hancock, Karol Hausman, Gashon Hussein, Brian Ichter, Szymon Jakubczak, Rowan Jen, Tim Jones, Ben Katz, Liyiming Ke, Chandra Kuchi, Marinda Lamb, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Yao Lu, Vishnu Mano, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Charvi Sharma, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, Will Stoeckle, Alex Swerdlow, James Tanner, Marcel Torne, Quan Vuong, Anna Walling, Haohuan Wang, Blake Williams, Sukwon Yoo, Lili Yu, Ury Zhilinsky, and Zhiyuan Zhou. $\pi^*_{0.6}$: a vla that learns from experience, 2025. 2

[15] Rahima Khanam and Muhammad Hussain. Yolov11: An overview of the key architectural enhancements, 2024. 3

[16] Katharina Limbeck, Rayna Andreeva, Rik Sarkar, and Bastian Rieck. Metric space magnitude for evaluating the diversity of latent representations. *Advances in Neural Information Processing Systems*, 37:123911–123953, 2024. 5

[17] OpenAI. Gpt-4o system card, 2024. 9

[18] Jack Parker-Holder, Philip Ball, Jake Bruce, Vibhavari Dasagi, Kristian Holsheimer, Christos Kaplanis, Alexandre Moufarek, Guy Scully, Jeremy Shar, Jimmy Shi, Stephen Spencer, Jessica Yung, Michael Dennis, Sultan Kenjeyev, Shangbang Long, Vlad Mnih, Harris Chan, Maxime Gazeau, Bonnie Li, Fabio Pardo, Luyu Wang, Lei Zhang, Frederic Besse, Tim Harley, Anna Mitenkova, Jane Wang, Jeff Clune, Demis Hassabis, Raia Hadsell, Adrian Bolton, Satinder Singh, and Tim Rocktäschel. Genie 2: A large-scale foundation world model. 2024. 1

[19] Mathieu Poliquin. Stable retro: A maintained fork of openai's gym-retro. https://github.com/Farama-Foundation/stable-retro, 2025. 2

[20] Nedko Savov, Naser Kazemi, Mohammad Mahdi, Danda Pani Paudel, Xi Wang, and Luc Van Gool. Exploration-driven generative interactive environments, 2025. 8

[21] Weihao Tan, Wentao Zhang, Xinrun Xu, Haochong Xia, Ziluo Ding, Boyu Li, Bohan Zhou, Junpeng Yue, Jiechuan Jiang, Yewen Li, Ruyi An, Molei Qin, Chuqiao Zong, Longtao Zheng, Yujie Wu, Xiaoqiang Chai, Yifei Bi, Tianbao Xie, Pengjie Gu, Xiyun Li, Ceyao Zhang, Long Tian, Chaojie Wang, Xinrun Wang, Börje F. Karlsson, Bo An, Shuicheng Yan, and Zongqing Lu. Cradle: Empowering foundation agents towards general computer control, 2024. 1

[22] Weihao Tan, Xiangyang Li, Yunhao Fang, Heyuan Yao, Shi Yan, Hao Luo, Tenglong Ao, Huihui Li, Hongbin Ren, Bairen Yi, Yujia Qin, Bo An, Libin Liu, and Guang Shi. Lumine: An open recipe for building generalist agents in 3d open worlds, 2025. 1

[23] SIMA Team. Sima 2: An agent that plays, reasons, and learns with you in virtual 3d worlds, 2025. 1

[24] SIMA Team, Maria Abi Raad, Arun Ahuja, Catarina Barros, Frederic Besse, Andrew Bolt, Adrian Bolton, Bethanie Brownfield, Gavin Buttimore, Max Cant, Sarah Chakera, Stephanie C. Y. Chan, Jeff Clune, Adrian Collister, Vikki Copeman, Alex Cullum, Ishita Dasgupta, Dario de Cesare, Julia Di Trapani, Yani Donchev, Emma Dunleavy, Martin Engelcke, Ryan Faulkner, Frankie Garcia, Charles Gbadamosi, Zhitao Gong, Lucy Gonzales, Kshitij Gupta, Karol Gregor, Arne Olav Hallingstad, Tim Harley, Sam Haves, Felix Hill, Ed Hirst, Drew A. Hudson, Jony Hudson, Steph Hughes-Fitt, Danilo J. Rezende, Mimi Jasarevic, Laura Kampis, Rosemary Ke, Thomas Keck, Junkyung Kim, Oscar Knagg, Kavya Kopparapu, Rory Lawton, Andrew Lampinen, Shane Legg, Alexander Lerchner, Marjorie Limont, Yulan Liu, Maria Loks-Thompson, Joseph Marino, Kathryn Martin Cussons, Loic Matthey, Siobhan Mcloughlin, Piermaria Mendolicchio, Hamza Merzic, Anna Mitenkova, Alexandre Moufarek, Valeria Oliveira, Yanko Oliveira, Hannah Openshaw, Renke Pan, Aneesh Pappu, Alex Platonov, Ollie Purkiss, David Reichert, John Reid, Pierre Harvey Richemond, Tyson Roberts, Giles Ruscoe, Jaume Sanchez Elias, Tasha Sandars, Daniel P. Sawyer, Tim Scholtes, Guy Simmons, Daniel Slater, Hubert Soyer, Heiko Strathmann, Peter Stys, Allison C. Tam, Denis Teplyashin, Tayfun Terzi, Davide Vercelli, Bojan Vujatovic, Marcus Wainwright, Jane X. Wang, Zhengdong Wang, Daan Wierstra, Duncan Williams, Nathaniel Wong, Sarah York, and Nick Young. Scaling instructable agents across many simulated worlds, 2024. 1

[25] Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, Wanjun Zhong, Yining Ye, Yujia Qin, Yuwen Xiong, Yuxin Song, Zhiyong Wu, Aoyan Li, Bo Li, Chen Dun, Chong Liu, Daoguang Zan, Fuxing Leng, Hanbin Wang, Hao Yu, Haobin Chen, Hongyi Guo, Jing Su, Jingjia Huang, Kai Shen, Kaiyu Shi, Lin Yan, Peiyao Zhao, Pengfei Liu, Qinghao Ye, Renjie Zheng, Shulin Xin, Wayne Xin Zhao, Wen Heng, Wenhao Huang, Wenqian Wang, Xiaobo Qin, Yi Lin, Youbin Wu, Zehui Chen, Zihao Wang, Baoquan Zhong, Xinchun Zhang, Xujing Li, Yuanfan Li, Zhongkai Zhao, Chengquan Jiang, Faming Wu, Haotian Zhou, Jinlin Pang, Li Han, Qi Liu, Qianli Ma, Siyao Liu, Songhua Cai, Wenqi Fu, Xin Liu, Yaohui Wang, Zhi Zhang, Bo Zhou, Guoliang Li, Jiajun Shi, Jiale Yang, Jie Tang, Li Li, Qihua Han, Taoran Lu, Woyu Lin, Xiaokang Tong, Xinyao Li, Yichi Zhang, Yu Miao, Zhengxuan Jiang, Zili Li, Ziyuan Zhao, Chenxin Li, Dehua Ma, Feng Lin, Ge Zhang, Haihua Yang, Hangyu Guo, Hongda Zhu, Jiaheng Liu, Junda Du, Kai Cai, Kuanye Li, Lichen Yuan, Meilan Han, Minchao Wang, Shuyue Guo, Tianhao Cheng, Xiaobo Ma, Xiaojun Xiao, Xiaolong Huang, Xinjie Chen, Yidi Du, Yilin Chen, Yiwen Wang, Zhaojian Li, Zhenzhu Yang, Zhiyuan Zeng, Chaolin Jin, Chen Li, Hao Chen, Haoli Chen, Jian Chen, Qinghao Zhao, and Guang Shi. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning, 2025. 1

[26] Zihao Wang, Xujing Li, Yining Ye, Junjie Fang, Haoming Wang, Longxiang Liu, Shihao Liang, Junting Lu, Zhiyong Wu, Jiazhan Feng, Wanjun Zhong, Zili Li, Yu Wang, Yu Miao, Bo Zhou, Yuanfan Li, Hao Wang, Zhongkai Zhao, Faming Wu, Zhengxuan Jiang, Weihao Tan, Heyuan Yao, Shi Yan, Xiangyang Li, Yitao Liang, Yujia Qin, and Guang Shi. Game-tars: Pretrained foundation models for scalable generalist multimodal game agents, 2025. 1

[27] Alex L. Zhang, Thomas L. Griffiths, Karthik R. Narasimhan,

and Ofir Press. Videogamebench: Can vision-language models complete popular video games?, 2025. 9